

Ressourcesinformatiques



+ QUIZ

Version en ligne

OFFERTE !

pendant 1 an

Intelligence artificielle vulgarisée

Le Machine Learning
et le Deep Learning par la pratique

En téléchargement



le code source des
différents cas pratiques

Aurélien VANNIEUWENHUYZE



Les éléments à télécharger sont disponibles à l'adresse suivante :

<http://www.editions-eni.fr>

Saisissez la référence ENI de l'ouvrage **RIIAVUL** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

1. Un souhait de vulgarisation des concepts liés à l'intelligence artificielle	17
2. Un mot sur l'auteur	18
3. À qui s'adresse cet ouvrage ?	19
4. Structure du livre	20
5. Aspects pratiques	21
6. Remerciements	22

Chapitre 1

Vous avez dit intelligence artificielle ?

1. Ce que nous allons découvrir et les prérequis	23
2. L'intelligence artificielle, ce n'est pas nouveau !	23
3. Quelques dates et périodes clés	25
4. Mais qu'est-ce que l'intelligence artificielle ?	27
5. Intelligence artificielle, Machine Learning et Deep Learning.	28
6. Les différents types d'apprentissage	29
7. L'intelligence artificielle fait peur.	30
7.1 La singularité technologique	30
7.2 Des emplois menacés	31
7.3 Des intelligences artificielles détournées.	31
7.4 Des boîtes noires qui font peur	32
7.5 Et la vie privée dans tout ça ?	32

2 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

8. Créer une intelligence artificielle chez soi c'est possible!	34
---	----

Chapitre 2

Les fondamentaux du langage Python

1. Ce que nous allons découvrir et les prérequis	35
2. Pourquoi Python?	36
3. Installation de Python	36
4. Une rapide découverte du langage Python	39
4.1 Python, un langage interprété	40
4.2 Les opérations de base	40
4.2.1 Affectation et affichage d'une variable	40
4.2.2 Affectations et affichage de plusieurs variables et éventuellement de types différents	41
4.2.3 Création de plusieurs variables de même type et de même valeur	41
4.3 Manipulation de chaînes de caractères	42
4.3.1 Création d'une chaîne de caractères	42
4.3.2 Les concaténations	42
4.3.3 Accès aux caractères d'une chaîne	43
4.3.4 Quelques fonctions utiles	44
4.4 Utilisation des listes	45
4.4.1 L'initialisation	45
4.4.2 Les fonctions de manipulation des listes	45
4.5 Les tuples et les dictionnaires	46
4.6 Les structures conditionnelles et les boucles	47
4.6.1 Les structures conditionnelles	47
4.6.2 Les boucles "tant que"	48
4.6.3 Les boucles "Pour..."	49
5. Installation de PyCharm	50

6. Votre premier script en Python	55
6.1 Création du projet	58
6.2 Création du fichier de script principal	59
6.3 Nos premières lignes de code	60
6.3.1 Un tuple pour le paramétrage	61
6.3.2 Création des zones à l'aide de dictionnaires	61
6.3.3 Regroupement des zones dans une liste	62
6.3.4 Une fonction pour calculer la surface à nettoyer	62
6.3.5 Une deuxième fonction pour coder le temps de nettoyage	64
6.3.6 Le script dans son ensemble	65
7. Conclusion	67

Chapitre 3

Des statistiques pour comprendre les données

1. Ce que nous allons découvrir et les prérequis	69
2. Les statistiques, un outil d'aide à la compréhension des données ...	70
3. Une série de notes en guise d'étude de cas	71
4. Petites notions de vocabulaire avant de commencer	72
4.1 Observations et features	72
4.2 Les types de données	72
5. Et Python dans tout ça?	73
5.1 Des modules dédiés	73
5.2 Une représentation un peu particulière de notre étude de cas ..	74
5.3 Pas de Python, mais Excel en guise d'outil	74
6. Mesure de tendance centrale	75
6.1 Connaître le nombre d'observations et de features	76
6.2 Les valeurs minimales et maximales	76
6.3 La moyenne arithmétique	78

4 Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

6.4	La médiane	80
6.4.1	Cas d'un nombre d'observations impair	80
6.4.2	Cas d'un nombre d'observations pair	81
6.4.3	Retour à notre exemple	81
6.5	Le mode	84
7.	Premières déductions	85
8.	La dispersion	86
8.1	L'étendue	86
8.2	L'écart type (Standard déviation)	87
8.2.1	Calcul de la variance	88
8.2.2	Calcul de l'écart type	89
8.2.3	Interprétation de l'écart type	90
8.3	Les quartiles et interquartile	90
8.3.1	Les quartiles	90
8.3.2	L'interquartile	94
9.	Détection de valeurs extrêmes (outliers en anglais)	94
10.	Traitement des valeurs extrêmes	96
11.	Un peu de visualisation graphique	97
12.	Conclusion sur les données	99
13.	Distribution gaussienne et loi normale	99
13.1	Un exemple pour faire connaissance	100
13.2	Un peu de probabilités	104
14.	Une classe Python pour vous aider à analyser vos données	105
15.	Combien d'observations sont nécessaires pour un bon apprentissage?	107

Chapitre 4**Principaux algorithmes du Machine Learning**

1. Ce que nous allons découvrir et les prérequis	109
2. Supervisé ou non supervisé? Régression ou classification?	110
3. Les algorithmes d'apprentissage supervisés pour la régression (prédiction de valeurs)	110
3.1 La régression linéaire univariée (linear regression)	110
3.2 La régression linéaire multiple (Multiple Linear Regression-MLR)	111
3.3 La méthode de descente de gradient	112
3.4 Régression polynomiale (polynomial regression)	113
3.4.1 Monôme et polynôme	114
3.5 Régression logistique	114
3.6 Arbre de décision (decision tree)	115
3.7 Forêts aléatoires (Random Forest)	116
3.8 Agrégation de modèle : le bagging, le boosting et le Gradient boosting	117
3.8.1 Le bagging	117
3.8.2 Le boosting	118
3.8.3 Gradient Boosting (GBoost) et XGBoost	118
3.9 Machine à vecteurs de support (SVM)	118
3.10 KNN (K-Nearest Neighbours)	120
3.11 Naïve Bayes	121
4. Les algorithmes pour les apprentissages non supervisés	123
4.1 K-Moyennes (KMeans)	123
4.2 Mean-shift	124
4.3 DBSCAN (Density Based Spatial Clustering of Application with Noise)	125
4.4 Mélange gaussien (Gaussian Mixture Models (GMM))	127
5. Et c'est tout?	129

6 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Chapitre 5

Machine Learning et Pokémons : première partie

1. Ce que nous allons découvrir et les prérequis	131
2. L'univers des Pokémons	132
3. Notre mission : choisir le bon Pokémon!	133
4. Des données pour un apprentissage supervisé	133
4.1 Des données basées sur l'expérience	133
4.2 Disposer d'un grand nombre de données d'apprentissage	133
4.3 Des données d'apprentissage et des données de tests.	133
5. Les étapes à réaliser pour mener à bien un projet de Machine Learning	134
5.1 Création et configuration d'un nouveau projet Python.	134
5.1.1 Installation de modules	135
5.1.2 Utilisation des modules dans un script Python	138
5.1.3 Référencement des fichiers de données dans notre projet	139
6. Étape 1 : définir le problème à résoudre	140
7. Étape 2 : acquérir des données d'apprentissage et de tests.	140
8. Étape 3 : préparation des données	141
8.1 De quelles données disposons-nous?	141
8.2 Affichage des dix premières lignes de nos données	144
8.3 Quelles sont les features de catégorisation?	146
8.4 Quelles données sont de type numérique?	147
8.5 Que faut-il penser de la feature LEGENDAIRE?	148
8.6 Manque-t-il des données?	149
8.7 À la recherche des features manquantes	150
8.8 Place aux observations des combats	153
8.9 Assemblage des observations	154
8.9.1 Nombre de combats menés	154
8.9.2 Nombre de combats gagnés	157
8.9.3 Agrégation des données avec le Pokédex.	158

9. Une petite pause s'impose	161
------------------------------------	-----

Chapitre 6

Machine Learning et Pokémons : seconde partie

1. Ce que nous allons découvrir et les prérequis	163
2. Un peu de statistiques	164
2.1 Le nombre de données (count)	164
2.2 La moyenne (mean)	165
2.3 L'écart type (Std pour Standard Deviation)	165
2.4 Les valeurs minimales et maximales	165
2.5 Les quartiles	166
2.6 Description de notre jeu d'observations	167
3. Quels sont les types de Pokémons qu'un dresseur doit posséder? ..	168
4. Les types de Pokémons gagnants et perdants	170
5. Essayons de trouver une corrélation entre les données	172
6. Résumé de nos observations*	174
7. Vérifions nos hypothèses	175
8. Passons à la phase d'apprentissage	176
8.1 Découpage des observations en jeu d'apprentissage et jeu de tests	177
8.2 Algorithme de régression linéaire	181
8.3 L'arbre de décision appliqué à la régression	184
8.4 La random forest	185
8.5 Sauvegarde du modèle d'apprentissage	186
9. Phénomènes de surapprentissage (overfitting) et de sous-apprentissage (underfitting)	186
10. Utiliser le modèle d'apprentissage dans une application	187
11. Fin du cas d'étude	191

8 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Chapitre 7

Bien classifier n'est pas une option

1. Ce que nous allons découvrir et prérequis	193
2. Origines et source du jeu d'observations	194
3. Un problème de classification et algorithmes de prédiction associés	195
4. Démarche de résolution du problème	195
4.1 Définition du problème à résoudre	195
4.2 Acquisition des données d'apprentissage	196
4.3 Préparer et nettoyer les données	196
4.3.1 De quelles données disposons-nous?	196
4.3.2 De combien de données disposons-nous?	197
4.3.3 Affichage des 10 premières observations	198
4.3.4 Transformation de la feature OBJET	200
4.3.5 Manque-t-il des données?	201
4.4 Analyser et explorer les données	203
4.4.1 Combien de mines et combien de rochers?	203
4.4.2 Moyenne, écart type, min, max et quartiles.	203
4.4.3 À la recherche des valeurs extrêmes	204
4.4.4 Traitement des valeurs extrêmes.	209
4.5 Choix d'un modèle de prédiction et résolution du problème. .	209
4.5.1 Des données d'apprentissage et des données de tests. .	209
4.5.2 Test des algorithmes	211
4.5.3 Optimisation	213
4.5.4 Et si on boostait un peu tout ça?	216
4.5.5 Que faire des données extrêmes?	216
5. En résumé	220

Chapitre 8**Opinions et classification de textes**

1. Ce que nous allons découvrir et les prérequis 223
2. Le traitement automatique du langage naturel (TALN) 224
3. Naive Bayes appliqué au TALN 224
 - 3.1 Le théorème 225
 - 3.2 Un exemple : quels mots-clés choisir ? 225
 - 3.2.1 Détermination des probabilités 226
 - 3.2.2 Conclusion 229
4. Naive Bayes pour l'analyse d'opinion 229
 - 4.1 Étape 1 : normalisation des données 230
 - 4.2 Étape 2 : suppression des stops words 231
 - 4.3 Étape 3 : le stemming 231
 - 4.4 Étape 4 : la lemmatisation 232
 - 4.5 Étape 5 : déterminer le nombre d'occurrences de chaque mot . 232
 - 4.6 Étape 6 : déterminer les probabilités pour l'opinion positive . . 233
 - 4.7 Étape 7 : déterminer les probabilités pour
le sentiment positif 235
 - 4.8 Étape 8 : déterminer le sentiment d'une nouvelle phrase . . . 236
5. Cas pratique : croyez-vous au réchauffement climatique ? 237
 - 5.1 Comment obtenir des données ? 237
 - 5.2 Création d'un projet Python 238
 - 5.3 Acquisition des données et préparation des données 239
 - 5.3.1 Chargement du fichier 239
 - 5.3.2 Normalisation 241
 - 5.3.3 Suppression des stop words 242
 - 5.3.4 La stemming 242
 - 5.3.5 La lemmatisation 243

10 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

6. Phases d'apprentissage et de prédiction.	243
6.1 Découpage en jeux de tests et d'apprentissage	243
6.2 Création d'un pipeline d'apprentissage	244
6.3 Apprentissage et analyse des résultats	245
6.4 Classification d'un nouveau message	246
7. L'algorithme SVM (Machine à vecteurs de supports) pour le classement de texte	247
8. L'algorithme SVM plus performant que Naive Bayes?	249

Chapitre 9

Abricots, cerises et clustering

1. Une machine qui apprend seule.	251
2. Acquisition de données d'apprentissage	252
3. Algorithme des K-Means (K-Moyennes).	255
4. Visualiser les données.	256
5. Laisser la machine classifier seule	258
6. Réaliser des classifications	260
7. Des erreurs de classifications	261
8. Algorithme de mélanges gaussiens ou Gaussian Mixture Model (GMM)	263
9. Pour conclure	265

Chapitre 10

Un neurone pour prédire

1. Ce que nous allons découvrir et les prérequis.	267
2. 1957 - Le perceptron	267
2.1 Un peu de biologie	268
2.2 La biologie appliquée au machine learning	270
3. Des données linéairement séparables	271

4. Fonctions d'activation, rétropropagation et descente de gradient	274
4.1 La fonction d'activation	274
4.1.1 La fonction de seuil binaire	274
4.1.2 La fonction sigmoïde	275
4.1.3 La fonction tangente hyperbolique (tanH)	276
4.1.4 La fonction ReLU (Rectified Linear Unit, unité de rectification linéaire)	277
4.1.5 La fonction softMax	278
5. La rétropropagation de l'erreur	279
6. Les fonctions de perte (Loss function)	279
6.1 L'erreur linéaire ou erreur locale	280
6.2 Erreur moyenne quadratique MSE ou erreur globale	280
7. La descente de gradient	281
8. Le biais, un neurone particulier	284
9. Un cas pratique pour comprendre le perceptron	286
9.1 Initialisation du perceptron	287
9.2 Les étapes d'apprentissage	288
9.2.1 Étape 1 : initialisation des poids	288
9.2.2 Étape 2 : chargement des données de la première observation	289
9.2.3 Étape 3 : préactivation	290
9.2.4 Étape 4 : utilisation d'une fonction d'activation	291
9.2.5 Étape 5 : calcul de l'erreur linéaire commise lors de l'apprentissage	292
9.2.6 Étape 6 : ajustement des poids synaptiques	292
10. Codons notre premier neurone formel "From Scratch"	294
10.1 Les données d'apprentissage	294
10.2 Définition des poids	295
10.3 Gestion des hyperparamètres	295
10.4 Codage de fonctions utiles	295
10.5 Passons à l'apprentissage!	297
10.6 À la recherche du point de convergence	300

12 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

10.7 Tests de prédictions	302
11. Un neurone artificiel avec TensorFlow	304
11.1 Un petit mot sur TensorFlow	305
11.2 Données d'apprentissage et de tests	306
11.3 Paramétrage du neurone	306
11.4 L'apprentissage	308
11.5 Tests de prédictions	310
12. Un premier pas vers le Deep Learning	311

Chapitre 11

Utilisation de plusieurs couches de neurones

1. Ce que nous allons découvrir et les prérequis	313
2. Fonctionnement des réseaux de neurones multicouches	313
3. Le cas du Ou exclusif (XOR)	314
3.1 De combien de couches et de neurones avons-nous besoin? ..	315
3.2 Un exemple chiffré	316
3.2.1 Les données d'apprentissage	316
3.2.2 Initialisation des poids	317
3.2.3 Chargement des données d'entrée	317
3.2.4 Calcul de la préactivation du neurone de sortie	318
3.2.5 Calcul de l'activation	318
3.2.6 Calcul de l'erreur	318
3.2.7 Mise à jour des poids	319
3.3 Place au code avec TensorFlow!	321
4. Le retour des mines et des rochers	326
4.1 De meilleures performances avec plus de neurones sur la couche cachée?	327
4.1.1 Chargement des données d'apprentissage	327
4.1.2 Création des jeux d'apprentissage et de tests	328
4.1.3 Paramétrage du réseau de neurones avec une couche cachée de 24 neurones	328

4.1.4 Réalisation de l'apprentissage	331
4.1.5 Calcul de la précision de l'apprentissage	332
4.1.6 De meilleurs résultats avec une couche cachée composée de 24 neurones?	335
4.1.7 Pouvons-nous obtenir de meilleurs résultats?	337
5. Conclusion	339

Chapitre 12

La classification d'images

1. Ce que nous allons découvrir et les prérequis	341
2. Différence entre détection et classification d'images	341
3. Des réseaux de neurones convolutifs pour classifier des images ...	342
3.1 De nombreuses données d'apprentissage nécessaires	342
3.2 Un outil pour illustrer nos propos	343
3.3 L'image d'entrée	344
3.4 Les caractéristiques	345
3.5 La convolution	346
3.6 Pooling	351
3.7 Plusieurs couches de convolutions	353
3.8 Mise à plat (Flatten)	354
3.9 L'apprentissage	355
3.10 Un schéma global qui résume tout	355
4. Un cas pratique autour de la mode	356
4.1 Présentation de Kaggle	356
4.2 Parlons un peu de Keras	357
4.3 Classifier des robes, pulls et chaussures?	357
4.4 De quelles données disposons-nous?	358
4.5 Préparation des données d'apprentissage	361
4.6 Préparation des données de tests	363

14 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

4.7	Un réseau avec une seule couche de convolution	363
4.7.1	Configuration	363
4.7.2	Compilation, apprentissage et test	365
4.7.3	Conclusion sur l'apprentissage	367
4.7.4	Augmentation du nombre de données	369
4.7.5	Sauvegarde du modèle	373
4.8	Un modèle plus performant	374
5.	Utilisation du modèle avec de nouvelles images	375
6.	Pour conclure ce chapitre	381

Chapitre 13

Votre ordinateur sait lire!

1.	Ce que nous allons découvrir et les prérequis	383
2.	Votre mission	384
2.1	Question n°1 : de quelles données avez-vous besoin ?	384
2.2	Question n°2 : comment utiliser le module Python-Mnist ?	385
2.3	Question n°3 : de quelles données disposez-vous à présent ?	387
2.4	Question n°4 : est-ce un problème de régression ou de classification ?	389
2.5	Question n°5 : quel algorithme allez-vous utiliser ?	389
2.6	Question n°6 : comment allez-vous créer vos jeux d'apprentissage et de tests ?	390
2.7	Question n°7 : les images sont-elles au bon format ?	390
2.8	Question n°8 : qu'est-ce que la catégorisation des libellés en One-Hot et comment procéder pour la réaliser ?	392
2.9	Question n°9 : avez-vous une petite idée des paramètres à utiliser pour créer le réseau de neurones ?	392
2.10	Question n°10 : trouvez-vous le résultat satisfaisant ?	395
2.11	Mission accomplie !	395

3. La reconnaissance de lettres sur une vidéo	398
3.1 Une ardoise en guise de support	398
3.2 OpenCV, un module de traitement d'images	399
3.2.1 Utiliser la webcam	400
3.2.2 Détecter les formes rectangulaires	402
3.2.3 Détecter la zone d'écriture	405
3.2.4 Détecter et extraire la lettre écrite	406
3.2.5 Reconnaître la lettre écrite et la faire lire à votre ordinateur	409
4. Et voilà!	412

Chapitre 14

Hommage au premier ChatBot

1. Introduction	413
2. Eliza	414
2.1 Comment fonctionne Eliza?	414
2.2 Le code d'Eliza	415
3. D'autres ChatBots!	421
4. C'est déjà la fin!	422

Index	425
-------------	-----

Avant-propos

1. Un souhait de vulgarisation des concepts liés à l'intelligence artificielle

À l'heure où nous écrivons cet ouvrage (mai 2019), l'intelligence artificielle est une priorité, tant sur le plan économique qu'éducatif.

Cependant, les différentes approches pédagogiques effectuées par les ouvrages et sites internet dédiés à l'intelligence artificielle restent aujourd'hui, si vous n'avez pas de connaissances suffisantes en mathématiques, très complexes et déroutantes.

L'idée à travers cet ouvrage est de montrer, de façon vulgarisée et par la pratique, la création de projets autour de l'intelligence artificielle en mettant de côté autant que possible les formules mathématiques et statistiques. Ainsi, l'objectif de ce livre est de rendre compréhensibles et applicables les concepts du Machine Learning et du Deep Learning à toute personne âgée entre 15 et 99 ans.

Chaque notion abordée est illustrée à l'aide de cas pratiques écrits en langage Python où de nombreux commentaires sont présents afin de faciliter la compréhension du code. Cette mise en pratique permet entre autres d'offrir une concrétisation à des notions abstraites.

2. Un mot sur l'auteur

Abandonnons quelques instants le "nous" au profit du "Je". Au risque de vous choquer, je ne suis ni datascientist ni chercheur en intelligence artificielle. Cependant, souhaitant aider mes clients dans la compréhension des enjeux de l'intelligence artificielle et la mise en place d'algorithmes liés au machine learning (évolution du métier de développeur oblige), je fus confronté à l'apprentissage - et non sans mal - des différents concepts. Cet ouvrage est donc issu de mon expérience personnelle et du souhait de partager mes connaissances acquises au travers du regard d'une personne souhaitant utiliser et comprendre les concepts du Machine Learning sans pour autant en connaître les moindres détails. Pour visser une vis cruciforme, nous avons besoin d'un tournevis cruciforme, peu importe la façon dont il a été fabriqué.

Passons à présent au curriculum vitae. Ayant commencé ma carrière en informatique en tant que développeur, je suis l'auteur d'ouvrages consacrés au langage de programmation Flex (langage aujourd'hui disparu) puis au framework Scrum lié à l'agilité. Je me forme ensuite aux concepts de la transformation digitale dans une grande école de commerce parisienne. À travers mon entreprise QSTOM-IT (www.qstom-it.com), j'accompagne aujourd'hui mes clients dans la mise en place de l'agilité, dans leur transformation numérique, sans oublier l'accompagnement et la mise en place de projets auteur de l'intelligence artificielle.

En 2017, je crée la startup Junior Makers Place (www.juniormakersplace.fr) dont l'objectif est d'apprendre aux enfants et adolescents à coder et à découvrir les sciences tout en ayant un esprit maker (fabrication).

L'intelligence artificielle est au cœur de la transformation digitale des entreprises et du monde numérique en général. Les enfants d'aujourd'hui sont les futurs consommateurs de cette intelligence, mais cette consommation ne doit pas être passive. Je décide donc de créer des stages et formations pour adolescents et jeunes adultes où leur sont expliqués les différents concepts de l'intelligence artificielle à l'aide de cas pratiques simples et accessibles.

Via QSTOM-IT, j'anime également des conférences pour les grandes entreprises où j'explique de façon pratique comment l'intelligence artificielle peut être intégrée dans les métiers de demain et démystifier la peur qu'elle engendre : quoi de mieux que de coder soi-même sa première intelligence artificielle pour comprendre ce que c'est réellement ? Cet ouvrage est donc issu de ces différents ateliers et conférences où l'objectif est de vulgariser et démocratiser les concepts et le développement de l'intelligence artificielle.

3. À qui s'adresse cet ouvrage ?

Cet ouvrage s'adresse à toute personne intéressée par le domaine du machine learning et **novice en la matière**, souhaitant en comprendre le fonctionnement à l'aide de cas pratiques simples et aux concepts vulgarisés.

Ce livre écrit par un développeur s'adresse, par son approche et son contenu donc, avant tout, aux développeurs.

À l'issue de l'ouvrage, le lecteur possèdera les clés nécessaires à la compréhension et à la mise en œuvre de petits projets liés au Machine Learning et disposera des informations nécessaires à la lecture, d'ouvrages et d'articles spécialisés.

Notons cependant que malgré une simplification des exemples, certaines parties de code réalisées à l'aide du langage Python nécessitent des connaissances dans ce langage. Si vous êtes novice en la matière, nous vous invitons donc à réaliser quelques petites applications à l'aide d'un ouvrage spécialisé afin de vous familiariser avec le langage.

Enfin, malgré une vulgarisation, les concepts évoqués restent tout de même du ressort des mathématiques et des statistiques. Il se peut que vous rencontriez certaines difficultés, n'hésitez donc pas à nous contacter, nous nous ferons un plaisir de vous aider !

4. Structure du livre

L'organisation de cet ouvrage se veut progressive. Nous vous invitons donc à le parcourir de façon chronologique.

Dans le chapitre Vous avez dit intelligence artificielle ?, nous commencerons notre aventure par la définition de l'intelligence artificielle et l'identification des craintes qu'elle suscite, puis nous verrons dans le chapitre suivant le langage de programmation Python qui nous accompagnera tout au long de l'ouvrage dans la réalisation des cas pratiques.

Le chapitre Des statistiques pour comprendre les données sera l'occasion de réviser quelques notions de statistiques qui nous seront utiles dans la compréhension du chapitre Principaux algorithmes du Machine Learning qui se chargera de présenter les algorithmes du machine learning. Les chapitres Machine Learning et Pokémons : première partie et Machine Learning et Pokémons : seconde partie mettront en pratique certains de ces algorithmes dédiés à la régression (prédiction de valeur) illustrés par un cas pratique consacré aux Pokemons. Les chapitres Bien classifier n'est pas une option ! et Opinions et classification de textes nous présenteront l'usage des algorithmes de classification, notamment la classification de texte utilisé dans des applications capables de déterminer les opinions des internautes sur les réseaux sociaux.

Nous poursuivrons ensuite dans le chapitre Abricots, cerises et clustering par la découverte de l'apprentissage non supervisé, puis nous découvrirons l'usage des réseaux de neurones et du Deep Learning grâce aux chapitres Un neurone pour prédire et Utilisation de plusieurs couches de neurones.

Le chapitre La classification d'images nous permettra de découvrir les réseaux de neurones convolutifs offrant la capacité à notre machine de reconnaître des images, puis dans le chapitre Votre ordinateur sait lire ! nous réaliserons un cas pratique mêlant réseaux de neurones convolutifs et reconnaissance sur vidéo. Enfin, dans le chapitre Hommage au premier ChatBot ! nous reviendrons aux sources des agents conversationnels grâce au chatbot Eliza que nous aurons le plaisir de coder.

Un joli programme en perspective qui nous permettra d'avoir une vue d'ensemble des concepts de l'intelligence artificielle et de leur mise en pratique.

5. Aspects pratiques

Le code de chaque cas pratique est disponible en téléchargement sur le site de l'éditeur. Pour chaque script, nous avons indiqué à l'intérieur de celui-ci les modules nécessaires et leur version. Ces versions devant être rigoureusement respectées, car nécessaires au bon fonctionnement du script :

```
# Modules nécessaires :
#   PANDAS 0.24.2
#   KERAS 2.2.4
#   PILOW 6.0.0
#   SCIKIT-LEARN 0.20.3
#   NUMPY 1.16.3
#   MATPLOTLIB : 3.0.3
```

Notez également que les résultats que vous obtiendrez à l'issue des différents apprentissages peuvent sensiblement être différents de ceux que nous avons obtenus lors de la rédaction de cet ouvrage, car l'usage de fonctions de hasard pour l'initialisation de certains paramètres peut influencer sur les résultats.

Enfin, à divers moments de votre lecture, vous verrez des mots en langue anglaise insérés entre parenthèses : test (*train*), précision (*accuracy*)... Ces derniers ont pour but de vous familiariser avec les mots que vous rencontrerez lors de vos lectures sur le web ou dans divers ouvrages spécialisés.

6. Remerciements

Je tiens à remercier toute personne ayant participé de près ou de loin à l'écriture de ce livre. Mes premiers remerciements vont envers mon épouse et mon fils qui ont fait preuve d'indulgence lors de l'écriture. Je remercie également les participants et participantes aux différents stages et conférences sur le thème de l'intelligence artificielle que je mène, qui ont nourri le contenu de cet ouvrage et qui sont également à l'origine de son existence. Je tiens à remercier les Éditions ENI et l'ensemble de l'équipe qui m'a renouvelé sa confiance. Enfin, je vous remercie, vous, lecteur ou lectrice, d'avoir fait l'acquisition de cet ouvrage en espérant que celui-ci sera à la hauteur de vos attentes!

Chapitre 1

Vous avez dit intelligence artificielle ?

1. Ce que nous allons découvrir et les prérequis

Dans ce chapitre, nous allons dans un premier temps découvrir l'histoire de l'intelligence artificielle en retraçant les dates et périodes clés de son développement et nous rendre compte que contrairement à ce que l'on peut croire, ce n'est pas une science nouvelle. Nous nous pencherons ensuite quelques instants sur ce qu'est réellement l'intelligence artificielle et les peurs qu'elle suscite pour terminer sur la possibilité de réaliser une intelligence artificielle chez soi.

■ Remarque

Prérequis nécessaires pour bien aborder ce chapitre : aucun.

2. L'intelligence artificielle, ce n'est pas nouveau !

Lorsque l'on pose la question aux plus jeunes quant à la date de naissance de l'intelligence artificielle, beaucoup indiquent les années 2000, mais il n'en est rien !

C'est lors de l'été 1956 qu'a vu officiellement le jour l'intelligence artificielle au Dartmouth College (New Hampshire, États-Unis) lors d'une université d'été (du 18 juin au 17 août) organisée par John McCarthy, Marvin Minsky, Nathaniel Rochester et Claude Shannon.

24 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Selon ces quatre personnes, la nouvelle discipline académique qu'est l'intelligence artificielle suppose que toutes les fonctions cognitives humaines peuvent être décrites de façon très précise pouvant alors donner lieu à une reproduction de celles-ci sur ordinateur.

Il serait alors possible de créer des systèmes capables d'apprendre, de calculer, de mémoriser, et pourquoi pas de réaliser des découvertes scientifiques ou encore de la créativité artistique !

Mais 1956 est la date de reconnaissance de l'intelligence artificielle en tant que science. Les travaux sur ce sujet ayant débuté bien avant. Nous pouvons remonter dans les années 1940 à 1950 où l'on parlait alors de cybernétique, science modélisant à l'aide de flux d'informations les systèmes biologiques, psychiques, politiques et sociaux. C'est à partir de cette science que fut notamment modélisé le neurone formel que nous aurons l'occasion de découvrir au cours de cet ouvrage.

Citons également le mathématicien Alan Turing qui, en 1948, dans son article *Intelligent Machinery*, décrit mathématiquement des réseaux de neurones connectés aléatoirement et capables de s'auto-organiser. Sans oublier en 1950 le fameux *jeu de l'imitation* décrit dans son article intitulé *Computing Machinery and Intelligence*, que l'on nommera par la suite Test de Turing, se résumant comme suit : si une personne à l'issue de quelques jeux de questions et réponses avec un ordinateur ne sait pas si elle a engagé une conversation avec un être humain ou une machine, on considère que l'ordinateur a donc réussi ce test.

Plus ancien encore, le philosophe Thomas Hobbes (1588 - 1679) formula l'hypothèse que toute pensée résulte d'un calcul. Cette hypothèse reprise par Alan Turing accompagné par Alonzo Church (dans leur thèse Church-Turing) en énonce alors une seconde : celle selon laquelle tout calcul peut être fait par une machine. De ce fait, si la pensée résulte d'un calcul et que le calcul est fait par une machine, la pensée peut donc être simulée sur des machines !

3. Quelques dates et périodes clés

L'intelligence artificielle n'est donc pas un concept nouveau et a connu depuis les années 50 des phases d'essor et de ralentissements. Voici quelques dates clés et périodes de cette science :

1943 : publication par Warren Mc Culloch (États-Unis) et Walter Pitts (États-Unis) d'un article fondateur sur le neurone formel.

1950 : évaluation de l'intelligence d'une machine par le *test de Turing* conçu par Alan Turing (Angleterre).

1951 : premier programme d'intelligence artificielle réalisé par Christopher Strachey (Angleterre) et Dietrich Prinz (Allemagne) sur un Ferranti Mark 1. Ce premier programme permettait de jouer aux dames contre une machine.

1951 : construction de la machine SNARC (Stochastic Neural Analog Reinforcement Calculator) par Marvin Minsky (États-Unis) réalisant, physiquement, avec des tubes à vide, des réseaux de neurones formels capables d'apprendre automatiquement les poids synaptiques en s'inspirant des principes dégagés par Donald Hebb (Psychologue et neuropsychologue).

1956 : l'intelligence artificielle est reconnue comme discipline académique.

1957 : proposition par Franck Rosenblatt (États-Unis) du premier réseau de neurones à couches appelé la perceptron.

1965 : naissance du premier programme interactif créé par Joseph Weizenbaum (Allemagne) nommé Eliza simulant un psychothérapeute. Ce programme étant capable de dialoguer en langage naturel comme le font les chatbots que nous connaissons de nos jours.

1967 : Richard Greenblatt (États-Unis) inventeur du langage LISP, développe un programme capable de jouer aux échecs et de rivaliser des joueurs lors de tournois.

1969 : Marvin Minsky (États-Unis) et Seymour Papert (États-Unis), dans leur ouvrage *Perceptrons*, démontrent les limites des réseaux de neurones. Notamment le fait qu'ils n'étaient pas capables de traiter des problèmes non linéairement séparables. Ces démonstrations sonnent l'arrêt des financements dans la recherche de ce domaine.

26 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

1970 à 1980 : de par les limitations à la fois scientifiques et technologiques, les projets d'intelligence artificielle n'aboutissent pas. Par conséquent, les financements publics furent limités et les industriels se détournèrent de l'intelligence artificielle. Cette période est communément nommée *l'hiver de l'intelligence artificielle* (AI Winter).

1980 à 1990 : cette décennie fut l'âge d'or des systèmes experts. Un système expert étant un programme permettant de répondre à des questions à l'aide de règles et de faits connus, mais se limitant à un domaine d'expertise précis.

1990 : l'intelligence artificielle connaît son second hiver du fait que la maintenance de systèmes experts devenait difficile (leur mise à jour devenait compliquée), que leur coût était élevé et que leur champ d'action limité à un seul domaine précis devenait problématique.

2000 : naissance de la première tête robotique exprimant des émotions créée par Cynthia Breazeal (États-Unis).

2009 : lancement par Google de son projet de voiture autonome.

2011 : Watson, le super calculateur d'IBM conçu pour répondre à des questions formulées en langage naturel est vainqueur du jeu Jeopardy.

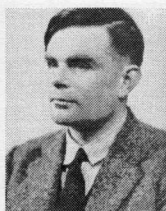
2012 : alors qu'Alex Krizhevsky (États-Unis), Ilya Sutskever (États-Unis) et Geoffrey Hinton (États-Unis) publient leurs résultats de classification d'images sur la base de données *ImageNet* à l'aide d'un réseau de neurones convolutif, l'équipe Google Brain conçoit un réseau de neurones capable de reconnaître les chats sur les vidéos YouTube.

2014 : les équipes de Facebook conçoivent un programme nommé *Deep Face* capable de reconnaître des visages avec seulement 3 % d'erreur.

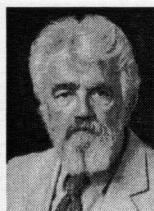
2016 : les équipes de *DeepMind*, une filiale de Google, développent le programme *AlphaGo* mettant en échec Lee Sedol, l'un des meilleurs joueurs de Go.

2017 : *AlphaGo* bat à présent Ke Jie le champion du monde du jeu de Go avec un score de 3 à 0.

2019 : l'intelligence artificielle s'ouvre au grand public à l'aide des enceintes connectées *Alexa* d'Amazon capables d'interpréter et de répondre à des questions courantes.



Alan Turing
1912 - 1954



John McCarthy
1927



Marvin Minsky
1927 - 2016



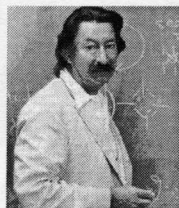
Nathaniel Rochester
1901 - 2001



Claude Shannon
1916 - 2001



Franck Rosenblatt
1928 - 1971



Joseph Weizenbaum
1923 - 2008



Seymour Papert
1928 - 2016



Cynthia Breazeal
1967

Les grandes figures de l'intelligence artificielle

4. Mais qu'est-ce que l'intelligence artificielle?

Comme nous venons de le voir, l'intelligence artificielle n'est pas une science nouvelle. Cependant, qui peut affirmer avoir vu une intelligence artificielle? Personne! Elle est inaudible, inodore et invisible. Les robots, les voitures autonomes ne sont pas ce que l'on peut appeler des intelligences artificielles, ce sont des machines utilisant de cette intelligence.

Comme vous avez sans doute dû le constater ou aurez l'occasion de le faire en lisant cet ouvrage, l'intelligence artificielle n'est autre qu'une série de formules mathématiques donnant naissance à des algorithmes ayant des noms plus étranges les uns des autres. Nous parlons alors de probabilités, de statistiques qui n'ont rien d'intelligent au sens où nous pouvons la qualifier pour les êtres humains.

L'intelligence artificielle se décline en deux parties. La première est le Machine Learning, se basant sur l'utilisation des statistiques pour donner la faculté aux machines "d'apprendre", quant à la seconde partie appelée Deep Learning (apprentissage profond), il s'agit d'algorithmes capables de s'améliorer de façon autonome grâce des modélisations telles que les réseaux de neurones inspirés du fonctionnement du cerveau humain reposant sur un grand nombre de données.

Nous irons donc jusqu'à affirmer que l'intelligence artificielle telle que nous pouvons l'imaginer à l'égal de l'Homme n'existe pas : en effet, avons-nous besoin de trois heures d'apprentissage et de millions de photos pour reconnaître un lapin sur une image ?

5. Intelligence artificielle, Machine Learning et Deep Learning

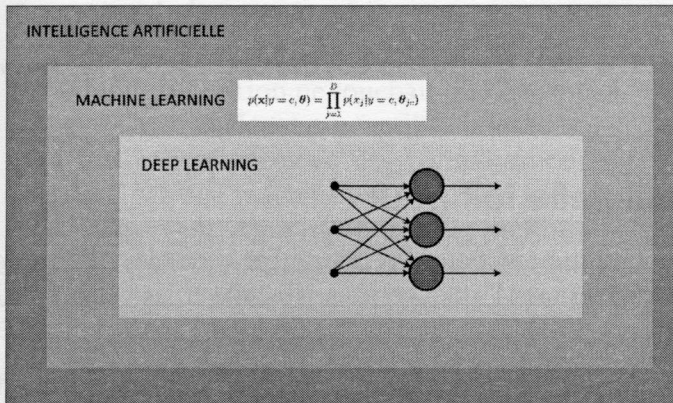
Nous allons à présent nous attarder quelques instants sur la définition des termes autour de l'intelligence artificielle, à savoir le Machine Learning et le Deep Learning car leur sens est parfois mal compris, voire confondu.

L'intelligence artificielle est à notre sens un mot valise défini par Marvin Minsky comme étant *une science dont le but est de faire réaliser par une machine des tâches que l'Homme accomplit en utilisant son intelligence*.

Pour y parvenir, nous avons besoin d'apprendre à notre machine comment réaliser ces tâches par le biais d'algorithmes conçus à partir de modèles statistiques. C'est ce que l'on appelle le Machine Learning.

Le Deep Learning est quant à lui une branche du Machine Learning s'appuyant sur l'usage de neurones artificiels s'inspirant du cerveau humain. Ces neurones sont organisés en couches donnant alors une notion de profondeur (deep) au réseau de neurones.

Par conséquent, lorsque nous parlons d'intelligence artificielle, il est préférable de parler de Machine Learning ou de Deep Learning.



Intelligence artificielle, Machine Learning et Deep Learning

6. Les différents types d'apprentissage

Une machine est capable d'apprendre selon trois formes d'apprentissage. Le premier est l'apprentissage dit supervisé. C'est-à-dire que la machine va apprendre à partir de données labellisées par l'être humain. Dans le cas de reconnaissance d'image entre un chat ou un chien, pour chaque image utilisée dans l'apprentissage nous devons indiquer à la machine s'il s'agit d'un chat ou d'un chien. Cette indication s'appelle une labellisation.

Vient ensuite l'apprentissage non supervisé. Dans ce cas, la machine va apprendre par elle-même. Mais le terme d'apprentissage autonome reste très relatif. Comme nous le verrons, la machine est capable de faire des regroupements et donc de réaliser des classifications, cependant elle n'est pas capable de définir par elle-même les différents libellés, car elle n'a pas conscience des données dont elle a la charge d'en faire l'apprentissage.

Enfin, il existe l'apprentissage par renforcement, consistant pour une machine à apprendre par l'expérience et étant récompensé de façon positive ou négative en fonction des décisions prises.

■ Remarque

Dans cet ouvrage, nous ne traiterons que les cas des apprentissages supervisés et non supervisés.

7. L'intelligence artificielle fait peur

Les paragraphes qui vont suivre ont pour but de mettre à jour les différentes questions et remarques concernant l'intelligence artificielle et les peurs qu'elle engendre.

Nous vous laissons libre de toute pensée et opinion sur ces différents domaines.

7.1 La singularité technologique

Les innovations technologiques se développent à un rythme exponentiel (loi de Moore) et les innovations d'un domaine sont source d'innovation d'un autre domaine. Partant de constat, Raymond Kurzweil (États-Unis) établit que cet enrichissement réciproque allié à la technologie pourrait donner naissance à une "Super intelligence" plus intelligente que l'Homme, et ce dès 2045. Cette super intelligence est appelée "Singularité technologique", car il nous serait alors impossible de prédire ce qui va se passer ensuite. Cette singularité sera-t-elle l'alliée ou ennemie de l'être humain ? Les deux versions s'opposent aujourd'hui, l'une indiquant que cela permettrait à l'être humain d'améliorer son confort de vie (soins des maladies graves...), et l'autre indiquant que cela pourrait entraîner la destruction de l'humanité (Robot destructeur, arme intelligente...).

7.2 Des emplois menacés

Là où l'automatisation ne touchait que les emplois à faible valeur ajoutée, l'intelligence artificielle est susceptible aujourd'hui de réaliser des tâches demandant des compétences et des connaissances spécifiques. La crainte d'être remplacé par une machine est donc très présente dans les esprits. Là encore, les avis divergent. Alors que le cabinet McKinsey Global Institute indique que, d'ici 2030, 45 % du travail accompli actuellement par des humains pourrait être réalisé par l'intelligence artificielle, le forum de Davos a publié un rapport sur ce thème, indiquant que 58 millions d'emplois seraient créés d'ici 2022 par l'Intelligence artificielle, 75 millions seraient reconvertis tandis que 133 millions «de nouveaux rôles peuvent apparaître, plus adaptés à une nouvelle division du travail». Il y a quelques années, le métier de datascientist n'existait pas, montrant de ce fait que l'intelligence artificielle va contribuer sans doute à la création de nouveaux métiers.

L'intelligence artificielle est-elle une opportunité ou une menace pour les emplois ? Si beaucoup de foyers ont un robot aspirateur, beaucoup ont encore un balai !

7.3 Des intelligences artificielles détournées

Tay est une intelligence artificielle développée par Microsoft en 2016, se présentant sous forme d'un agent conversationnel connecté à Twitter. L'objectif de cet agent est d'apprendre à communiquer avec les utilisateurs âgés de 18 à 24 ans à partir des conversations qu'il réalise avec eux sur Twitter. Après 96000 tweets échangés en 8 h, TAY a dû être débranchée par Microsoft, car elle tenait des propos racistes et misogynes suite à ses apprentissages.

En effet, des utilisateurs ont tenté de détourner l'intelligence artificielle en lui apprenant à tenir certains propos, ce à quoi ils sont parvenus.

Cela montre qu'une intelligence artificielle est sensible aux données auxquelles elle se trouve confrontée, car celles-ci sont sa source d'apprentissage. Mais malgré le résultat déplorable et les conséquences désastreuses de cette expérience, cela montre aussi que l'être humain reste maître face à l'intelligence artificielle.

7.4 Des boîtes noires qui font peur

Nous l'avons évoqué en début de chapitre, une intelligence artificielle n'est aucunement perceptible et se base sur des concepts mathématiques obscurs pour la plupart d'entre nous.

Nous lui fournissons des données d'apprentissage et elle réalise des prédictions en sortie avec une suite de traitements qui ressemblent fortement à une boîte noire.

Lorsque nous ne comprenons pas un phénomène, celui-ci nous fait peur. C'est pourquoi à travers cet ouvrage nous avons voulu vulgariser la compréhension et la mise en place de projets d'intelligence artificielle afin que cet aspect de boîte noire soit supprimé.

7.5 Et la vie privée dans tout ça ?

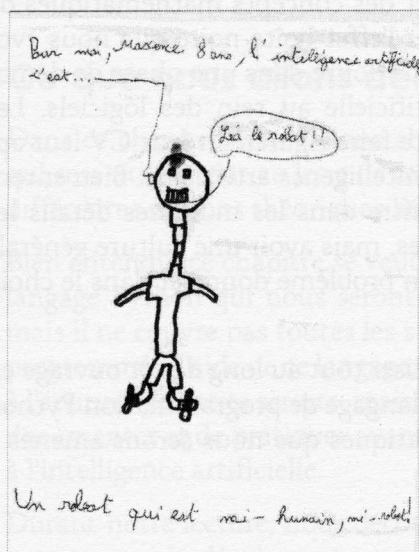
Voici un extrait de conversation que nous avons eu avec un jeune public :

- Question : *"Comment pensez-vous que l'intelligence artificielle de Google soit capable de reconnaître un chat ?"*
- Réponse : *"Grâce aux images"*
- Question : *"Oui, mais comment Google a-t-il obtenu ces images ?"*
- Réponse : *"En faisant des photos avec un téléphone ou un appareil photo"*
- Question : *"Les employés de Google ont donc fait des millions de photos avec leur téléphone ?"*
- Réponse : *"... euh... non ce n'est pas possible !"*
- Question : *"Alors comment ?"*
- Réponse : *"À l'aide des photos que nous publions sur les réseaux sociaux !"*

À ce stade de la conversation, une prise de conscience est tout de suite réalisée, montrant alors l'importance de protéger ses données personnelles.

Nous verrons dans le chapitre Opinions et classification de textes qu'il est aujourd'hui très facile de se procurer des jeux de données issues de Twitter ou de divers échanges de mails afin de donner à notre machine la faculté d'apprendre à réaliser des conversations ou analyser nos sentiments.

D'un autre côté, comment pouvons-nous obtenir une machine intelligente performante si nous ne lui offrons pas une base d'apprentissage solide avec de multiples exemples? Si l'intelligence artificielle est si développée de nos jours c'est sans doute grâce à ces données exploitées par les grands groupes tels que Facebook ou Google. Un juste milieu doit cependant être trouvé en établissant que toute acquisition de donnée doit donner lieu à un consentement de notre part. C'est ce que prône entre autres le Règlement Général sur la Protection des Données (RGPD).



L'intelligence artificielle vue par Maxence âgé, de 8 ans :

"Pour moi, l'intelligence artificielle c'est un robot mi-humain mi-robot"

8. Créer une intelligence artificielle chez soi c'est possible!

Créer une intelligence artificielle nécessite des compétences scientifiques, mais aussi des moyens technologiques. Là où il y a encore quelques années il fallait disposer de solides notions en mathématiques et une machine puissante pour concevoir et développer une intelligence artificielle, aujourd'hui nous disposons d'outils et de matériels capables de minimiser ces prérequis.

En effet, il est aujourd'hui possible de réaliser l'apprentissage d'une machine sur un simple ordinateur portable tout en utilisant des modules d'intelligence artificielle contenant le code informatique et les fonctions mathématiques nécessaires! Néanmoins, cela nécessite tout de même une légère prise en main à la fois des langages de programmation et des concepts mathématiques de l'intelligence artificielle dans le but d'éviter l'effet boîte noire que nous évoquions auparavant. Nous sommes donc sans doute dans une phase de démocratisation de l'usage de l'intelligence artificielle au sein des logiciens. Les développeurs de demain devront sans doute faire figurer sur leur CV leur aptitude à utiliser les différents outils liés à l'intelligence artificielle. Bien entendu, il ne leur sera pas demandé de connaître dans les moindres détails les algorithmes utilisés ainsi que leurs formules, mais avoir une culture générale dans la mise en place de solutions face à un problème donné et dans le choix de celles-ci sera nécessaire.

C'est cet apprentissage que nous allons réaliser tout au long de cet ouvrage en commençant de suite par la découverte du langage de programmation Python qui nous servira dans les différents cas pratiques que nous serons amenés à réaliser.

Chapitre 2

Les fondamentaux du langage Python

1. Ce que nous allons découvrir et les prérequis

Dans ce chapitre, nous allons aborder le langage de programmation Python qui sera utilisé tout au long de cet ouvrage pour illustrer de façon pratique les différentes notions théoriques liées à l'intelligence artificielle.

Bien entendu, ce chapitre se veut être une revue rapide des fondamentaux du langage Python qui nous seront utiles pour mener à bien les cas pratiques, mais il ne couvre pas toutes les subtilités de ce langage. Si vous n'avez jamais programmé à l'aide de ce langage, ce chapitre peut donner les bases nécessaires à l'écriture de petits scripts, mais nous vous conseillons tout de même de vous documenter et de pratiquer avant de vous lancer dans le codage de projets liés à l'intelligence artificielle.

Durant notre lecture, nous découvrirons comment installer Python et l'environnement de développement, puis après avoir vu les notions essentielles, nous terminerons par un mini projet.

Remarque

Prérequis nécessaires pour bien aborder ce chapitre : aucun.

36 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

2. Pourquoi Python?

Il existe une multitude de langages de programmation et Python n'est pas forcément au-dessus des autres pour les applications dédiées à l'intelligence artificielle. C'est cependant ce langage que nous allons utiliser à travers les exemples que nous traiterons tout au long de cet ouvrage. Voyons pourquoi.

Python est un langage ayant une syntaxe simple et précise qui se trouve être en adéquation avec le thème que nous traitons à savoir la vulgarisation de l'intelligence artificielle. Des bibliothèques spécialisées en intelligence artificielle existent ainsi qu'une suite d'outils permettant l'analyse et le traitement des données qui, nous le verrons, sont des phases importantes et non négligeables dans la réalisation d'un projet d'intelligence artificielle.

■ Remarque

Savez-vous d'où vient le nom de Python? Guido Van Rossum, créateur du langage en 1991, aimait tout particulièrement la série Monty Python's flying circus d'où le nom de Python.

3. Installation de Python

L'installation et les différents cas pratiques que nous proposons à travers cet ouvrage sont réalisés dans un environnement **Windows 64 bits**. C'est donc tout naturellement que nous allons détailler l'installation de Python sur cet environnement. Si vous souhaitez utiliser Linux ou bien encore Mac, nous vous laissons le soin de consulter la documentation d'installation relative à votre environnement, disponible sur le site de Python : www.python.org.

Téléchargement et installation de Python

Pour télécharger Python, il convient de se rendre à l'adresse suivante www.python.org et de télécharger la version 3.7.2 pour Windows 64 bits.

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball		Source release	02a75015f7cd845e27b85192bb0ca4cb	22897802	SiG
XZ compressed source tarball		Source release	df6ec36011808205beda239c72f947cb	17042320	SiG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	d8ff07973bc9c0d9d80c269fd7efcca	34405674	SiG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	0fc95e9f6d6b48a1f3ba99da338a9a60	27766090	SiG
Windows help file	Windows		941b7d6279c0d4060a927a65dcab88c4	8092167	SiG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	f51568590be156e5997e63b434664d58	7025085	SiG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	ff258093f0b3953c8b5192dec0f52763	26140976	SiG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	8de2335243d84fe1eeb61ec25858bd82	1362888	SiG
Windows x86 embeddable zip file	Windows		26881045297dc1883a1d61bafeecaf0	6533256	SiG
Windows x86 executable installer	Windows		35156b62c0cbcb03b1dddeb86e6ec3a0f	25365744	SiG
Windows x86 web-based installer	Windows		1ef5c626514b72e21008f9c-d53f945f10	1324648	SiG

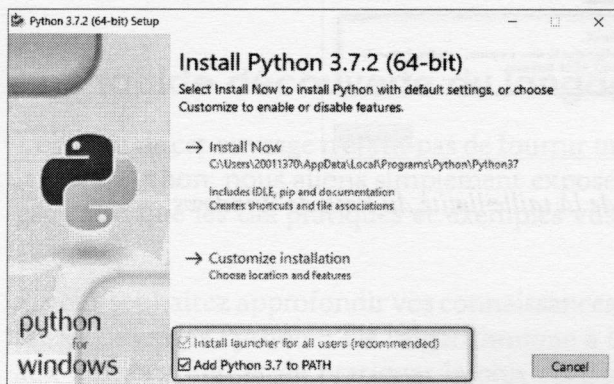
Téléchargement de Python

Remarque

L'ensemble des exemples et cas pratiques présentés dans cet ouvrage est réalisé à l'aide de la version **3.7.2 de Python**. L'ouvrage ayant une période d'édition plus longue que celle des mises à jour du langage de programmation, nous vous invitons à utiliser exclusivement la version 3.7.2 pour vous assurer un bon fonctionnement du code qui sera présenté.

Une fois le téléchargement réalisé, exécuter le programme d'installation et suivre les différentes étapes :

► Insérez le chemin de Python au PATH de Windows.

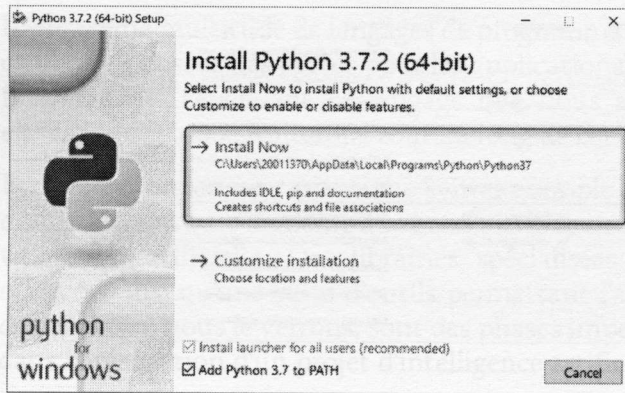


Ajout du chemin de Python au path de Windows

38 — Intelligence Artificielle Vulgarisée

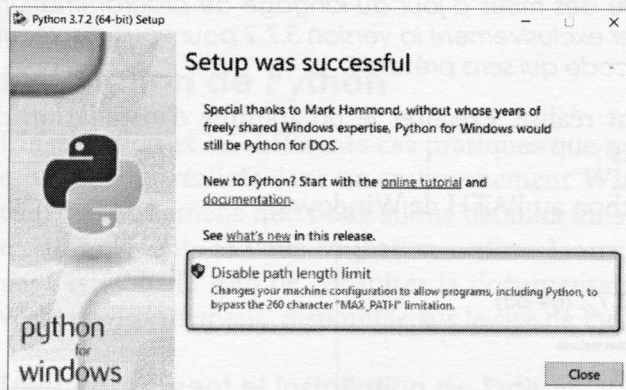
Le Machine Learning et le Deep Learning par la pratique

❑ Installez Python en cliquant sur le bouton **Install Now**.

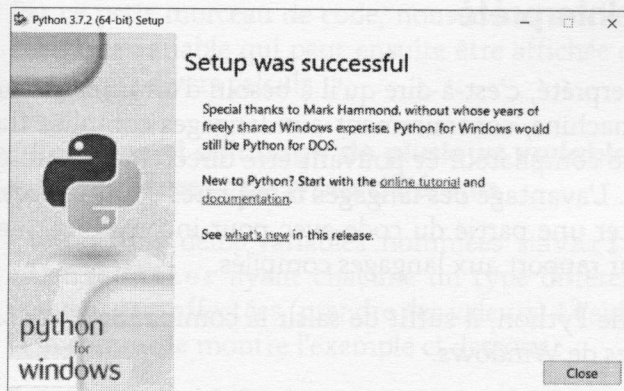


Installation de Python

❑ Autorisez le dépassement de la taille du PATH, limitée par défaut à 260 caractères, en cliquant sur le bouton **Disable path length limit**.



Autorisation de dépassement de la taille limite du Path de Windows



Fin de l'installation

Afin de vérifier que l'installation s'est déroulée correctement, il faut ouvrir une invite de commandes Windows et saisir la commande `python --version`. La version de Python doit alors s'afficher. Si ce n'est pas le cas, le problème vient sans doute du fait que la case **Add Python 3.7 to PATH** n'a pas été cochée lors de la phase d'installation. Dans ce cas, et pour plus de simplicité, il suffit de désinstaller Python et de le réinstaller ensuite.

```
C:\Users\20011370>python --version
Python 3.7.2
```

Visualisation de la version de Python

4. Une rapide découverte du langage Python

L'objectif de cet ouvrage n'étant pas de fournir une description approfondie du langage Python, nous allons simplement exposer les principes de base du langage afin que les cas pratiques et exemples vus plus loin soient compréhensibles.

Si vous souhaitez approfondir vos connaissances, nous vous invitons à consulter les ouvrages Python 3 De l'algorithmique à la maîtrise du langage publiés aux Éditions ENI et de pratiquer le plus possible, car comme le dit l'adage : "c'est en développant que l'on devient développeur".

4.1 Python, un langage interprété

Python est un langage interprété, c'est-à-dire qu'il a besoin d'un interpréteur pour être exécuté sur la machine, contrairement aux langages compilés traduits en code binaire par le compilateur et pouvant être directement utilisés et compris par la machine. L'avantage des langages interprétés étant la possibilité d'exécuter et de tester une partie du code avec pour inconvénient des performances moindres par rapport aux langages compilés.

Pour lancer l'interpréteur de Python, il suffit de saisir la commande python dans l'invite de commandes de Windows.

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC  
v.1916 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more  
information.  
>>>
```

Nous sommes à présent prêts à réaliser les différents exemples décrits ci-dessous et permettant de découvrir le langage Python.

4.2 Les opérations de base

4.2.1 Affectation et affichage d'une variable

Les variables sont fort utiles pour stocker en mémoire des données qui pourraient être utilisées par la suite dans différentes parties du script correspondant au programme ou dans des fonctions permettant de réaliser divers traitements.

■ Créez une variable nommée `maVariable` à laquelle vous affectez la valeur 1234, puis réalisez son affichage à l'aide de la fonction `print`.

```
>>> maVariable = 1234  
>>> print(maVariable)  
1234
```

Par ce petit morceau de code, nous avons stocké en mémoire la valeur 1234 dans une variable qui peut ensuite être affichée ou faire l'objet de manipulations dans divers calculs.

4.2.2 Affectations et affichage de plusieurs variables et éventuellement de types différents

Considérons deux variables nommées `maVariableBooleene` et `maVariableEntier` ayant chacune un type différent (booléen et entier). Elles peuvent être affectées (prendre des valeurs) à l'aide d'une seule ligne d'instruction comme le montre l'exemple ci-dessous :

```
>>> maVariableBooleene, maVariableEntier = False, 4567
>>> print(maVariableBooleene)
False
>>> print(maVariableEntier)
4567
```

■ Remarque

Une variable booléenne est un type de variable prenant pour valeur vrai (True) ou faux (False). Le nom de booléen vient de George Boole fondateur d'une approche algébrique (mathématique) de la logique. Ce type de variable est fréquemment utilisé dans les structures conditionnelles.

4.2.3 Création de plusieurs variables de même type et de même valeur

Le code ci-dessous démontre qu'il est possible de créer trois variables prenant chacune la même valeur. Ces trois variables seront par conséquent de même type (dans notre cas des entiers).

```
>>> a = b = c = 6789
>>> print(a)
6789
>>> print(b)
6789
>>> print(c)
6789
```

4.3 Manipulation de chaînes de caractères

Une chaîne de caractères est une suite de caractères juxtaposés formant alors une chaîne. Voyons comment les manipuler.

4.3.1 Création d'une chaîne de caractères

Pour créer une chaîne de caractères, nous pouvons utiliser des doubles quotes (") ou des simple quote (') :

```
>>> "Une chaîne de caractères"
'Une chaîne de caractères'
>>>
>>> 'Une chaîne de caractères'
'Une chaîne de caractères'
>>>
```

Si votre chaîne de caractères comporte des apostrophes, l'insertion de celles-ci dépend de la façon dont la chaîne de caractères a été créée (avec ou sans double quotes) :

```
>>> "Vulgarisation de l'intelligence artificielle"
"Vulgarisation de l'intelligence artificielle"
>>>
>>> 'Vulgarisation de l\' intelligence artificielle'
"Vulgarisation de l' intelligence artificielle"
```

On note l'usage du caractère d'échappement \ lorsque la chaîne de caractères est créée à l'aide de simples quotes, cela permet d'indiquer à Python que l'apostrophe doit être considérée comme du texte et non comme la fin de la chaîne de caractères.

4.3.2 Les concaténations

Concaténer signifie juxtaposer. La concaténation consiste donc à assembler deux chaînes de caractères pour n'en former qu'une seule. La concaténation se réalise en Python à l'aide du symbole +.

■ Remarque

Attention : il est impossible de concaténer directement une variable numérique à une chaîne de caractères sans la transformer en une autre chaîne de caractères à l'aide de la commande `str`.

```
>>> maChaine = "Blabla"
>>> i=38
>>> print (maChaine + I)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

```
>>> print (maChaine + str(i))
Blabla38
```

4.3.3 Accès aux caractères d'une chaîne

La chaîne de caractères peut être vue comme un tableau de caractères auxquels il est possible d'accéder individuellement grâce à leur position, aussi appelée index.

■ Remarque

Il est important de noter que le premier caractère de la chaîne porte la position 0. Ainsi le 7e caractère de la chaîne "Intelligence artificielle" est la lettre i. Le dernier caractère peut également être récupéré via l'utilisation de la valeur -1.

```
>>> maChaine = "Intelligence artificielle"
>>> print(maChaine[6])
i
>>> print(maChaine[-1])
e
```

Il est également possible d'extraire une partie de la chaîne de caractères en spécifiant une position de début et une position de fin.

44 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

```
>>> print(maChaine[2:])
telligence artificielle
>>> print(maChaine[:3])
Int
>>> print(maChaine[0:12])
Intelligence
```

4.3.4 Quelques fonctions utiles

Voici une liste non exhaustive de fonctions utiles et fréquemment utilisées avec les chaînes de caractères :

- `len` : renvoie la taille de la chaîne de caractères.
- `find` : recherche la position d'une sous-chaîne dans la chaîne.
- `replace` : remplace une chaîne par une autre.
- `split` : découpe une chaîne de caractères.

```
>>> len(maChaine)
25
>>> maChaine.find("artificielle")
13
>>> maChaine.rstrip()
'Intelligence artificielle'
>>> maChaine.replace('artificielle', 'non artificielle')
'Intelligence non artificielle'
>>>
>>>
>>> maChaine = "valeur1,valeur2,valeur3"
>>> a,b,c = maChaine.split(",")
>>> print(a)
valeur1
>>> print(b)
valeur2
>>> print(c)
valeur3
>>>
```

4.4 Utilisation des listes

Comme vous pourrez facilement le constater, la manipulation de listes est une chose très souvent réalisée dans le cadre de la création d'applications liées au Machine Learning. C'est pourquoi il est important de connaître les principes de base de manipulation de ces éléments.

4.4.1 L'initialisation

Une liste est une variable dans laquelle on peut mettre plusieurs variables, tout comme on le ferait pour une liste de courses. Dans l'exemple ci-dessous, la liste `maListe` contient trois entiers.

```
>>> maListe = [1,2,3]
>>> print(maListe)
[1, 2, 3]
```

On note l'usage des crochets lors de la création de la liste et l'utilisation de virgules pour la séparation de chaque élément de la liste.

4.4.2 Les fonctions de manipulation des listes

Pour ajouter un élément à une liste, il convient d'utiliser la fonction `append`. Notons qu'il est possible d'ajouter à une liste des éléments de types différents et il est également possible d'ajouter une liste dans une liste. Dans notre cas, nous ajoutons une chaîne de caractères à notre liste constituée initialement d'entiers.

```
>>> maListe.append("ajout")
>>> print(maListe)
[1, 2, 3, 'ajout']
```

Dans l'exemple ci-dessous, différentes fonctions utiles pour la manipulation de listes sont présentées dont :

- L'accès à un élément de la liste par son numéro d'index
- La suppression d'un élément par son numéro d'index à l'aide de la fonction `del`
- La fonction `remove` permettant de supprimer un élément de la liste par son libellé

46 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

- La fonction `len` permettant de connaître le nombre d'éléments de la liste
- Le comptage du nombre d'occurrences d'un élément dans la liste en utilisant la fonction `count`

```
>>> print(maListe[2])
3
>>>
>>> del maListe[1]
>>> print(maListe)
[1, 3, 'ajout']
>>>
>>> maListe.remove('ajout')
>>> print(maListe)
>>> print(len(maListe))
2
>>>
>>> maListe.append(1)
>>> maListe.append(1)
>>> maListe.append(1)
>>> print(maListe)
[1, 3, 1, 1, 1]
>>> print(maListe.count(1))
4
```

4.5 Les tuples et les dictionnaires

Les tuples sont également des listes, mais celles-ci sont **non modifiables**. La vocation de ce type de liste est d'être utilisé à titre de constante dans l'application.

On note l'usage des parenthèses pour la création de tuples, contrairement à l'utilisation de crochets pour la création de listes

```
>>> monTuple=('a','b','c',1)
>>> print(monTuple)
('a', 'b', 'c', 1)
>>> print(monTuple[2])
c
```

Les dictionnaires, quant à eux, sont des listes modifiables, mais dont les données sont accessibles par des clés. Dans notre exemple, nous avons créé un dictionnaire `monDictionnaire` dont chaque valeur est définie à l'aide des clés, `prenom` et `nom` et accessible par la fonction `get`.

```
>>> monDictionnaire = {"prenom": "monPrenom", "nom": "monNom"}
>>> print(monDictionnaire.get("nom"))
monNom
>>> print(monDictionnaire.get("prenom"))
monPrenom
```

Cependant, contrairement aux listes, les dictionnaires se créent à l'aide d'accolades.

4.6 Les structures conditionnelles et les boucles

Comme tout langage, Python offre la possibilité d'utiliser des structures de contrôles dans nos applications, à savoir les tests conditionnels et les boucles.

4.6.1 Les structures conditionnelles

Les structures conditionnelles s'écrivent sous cette forme :

```
if <condition_1>:
    <blocs d'instructions 1>
elif <condition_2>:
    <blocs d'instructions 2>
else:
    <blocs d'instructions 3>
```

Dans l'exemple ci-dessous, si la variable `age` est inférieure à 18, alors on affiche un message "vous êtes mineur", dans le cas contraire, "vous êtes majeur".

On note l'usage d'indentations devant chaque fonction `print`, réalisées à l'aide la touche [Tab] du clavier. Ces indentations ont pour rôle de définir le début et la fin du bloc de code à exécuter dans chaque condition.

48 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Si mon âge est inférieur à 18, alors :

tab Afficher (vous êtes mineur)

sinon :

tab Afficher (vous êtes majeur)

```
>>> age=8
>>> if age<18:
...     print('vous êtes mineur')
... else:
...     print('vous êtes majeur')
...
vous êtes mineur
```

4.6.2 Les boucles "tant que"

Les boucles de type "Répéter l'action l'action tant que..." ont le formalisme de codage suivant :

```
while <condition 1>:
    <blocs d'instructions 1>
```

À titre d'exemple, nous allons créer un compteur puis afficher sa valeur jusqu'à ce qu'il atteigne la valeur 5. Une attention particulière est à apporter à l'usage des indentations à l'intérieur du bloc d'instructions.

```
>>> monCompteur=0
>>> while monCompteur <=5:
...     print(monCompteur)
...     monCompteur=monCompteur+1
...
0
1
2
3
4
5
```

4.6.3 Les boucles "Pour..."

L'expression "Pour tous les éléments d'une liste ou d'une chaîne faire..." se traduit par l'usage de la fonction `for` ayant le formalisme suivant :

```
for <données> in <objet>:  
    <blocs d'instructions>
```

L'exemple ci-dessous affiche les lettres constituant la chaîne de caractères `maChaine`.

Là encore, les indentations jouent un rôle important dans la bonne interprétation du code par l'interpréteur de Python.

```
>>> maChaine = "I.A"  
>>> for lettre in maChaine:  
...     print(lettre)  
...  
I  
.  
A
```

On peut également utiliser la fonction `break` pour mettre fin à la boucle, lors de conditions particulières. L'exemple ci-dessous montre le parcours, aussi appelé itération, d'une liste de valeurs avec une condition particulière de sortie de boucle.

```
>>> listeAges = [8,5,10,35,20,25]  
>>> for age in listeAges:  
...     if age > 15:  
...         print("Fin de la boucle")  
...         break  
...     print(age)  
...  
1  
5  
10  
35  
Fin de la boucle
```

50 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Enfin, grâce à la fonction `range`, on peut réaliser une itération sur une plage de valeurs définies. Par exemple, afficher les chiffres de 0 à 5.

```
for chiffre in range(0,5):  
    print(chiffre)
```

Par ces quelques exemples, nous venons de découvrir de façon très succincte le langage Python. Afin d'avoir une vision un peu plus claire de l'utilisation de ce langage, nous vous proposons de réaliser un premier programme regroupant les notions que nous venons de découvrir.

Mais avant cela, nous allons avoir besoin d'un outil de développement offrant une meilleure ergonomie de codage et des outils plus élaborés que le simple interpréteur de Python. L'outil que nous vous proposons d'utiliser est PyCharm.

5. Installation de PyCharm

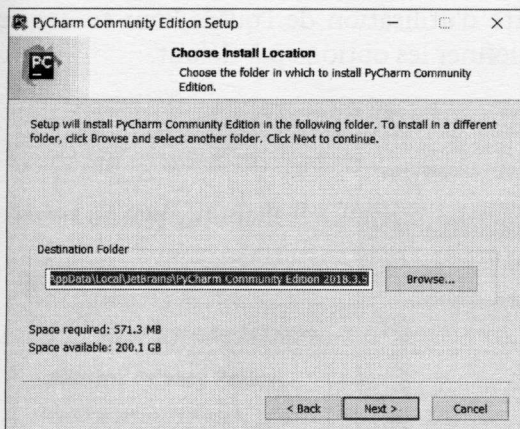
PyCharm est un outil de développement, parmi tant d'autres, dédié au langage Python. Également appelé IDE (*Integrated Development Environment*) ou environnement de développement intégré, PyCharm offre une multitude de fonctionnalités telles que la proposition automatique de code (l'auto-complétion) ou bien encore le débogage, fort utile aux développeurs.

Pour installer PyCharm, il suffit de le télécharger à l'adresse <https://www.jetbrains.com/pycharm/features/> et de suivre les différentes étapes d'installation.

■ Remarque

Vous constaterez que dans le domaine de l'intelligence artificielle, mais aussi dans l'utilisation du langage Python en général, il est souvent fait référence à l'outil Jupyter Notebook. Nous avons fait le choix de vous présenter PyCharm car celui-ci propose des outils plus avancés en termes de développement (Débogueur...), qui vous seront sans doute utiles dans vos futurs projets d'intelligence artificielle.

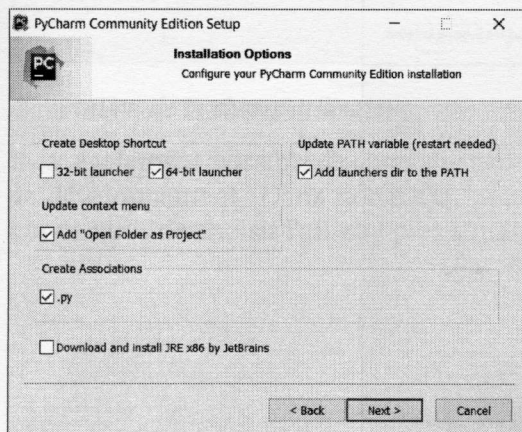
Étape 1: indication du répertoire d'installation



Choix du répertoire d'installation de PyCharm

Étape 2 : choix des différentes options

Dans notre cas, nous allons choisir de faire en sorte que PyCharm soit l'outil permettant d'éditer les fichiers ayant une extension .py propre aux fichiers de scripts Python.

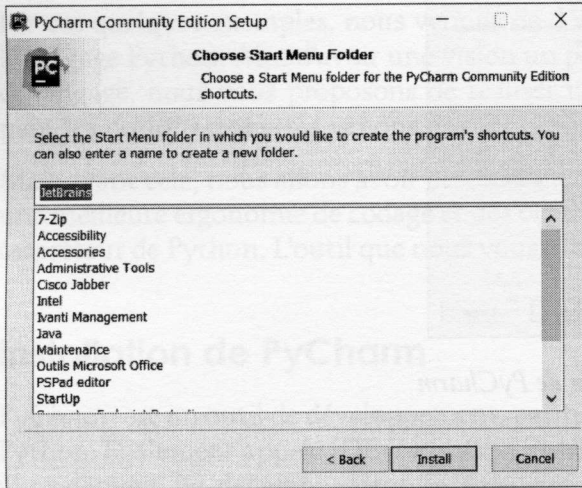


Choix des options d'installation de PyCharm

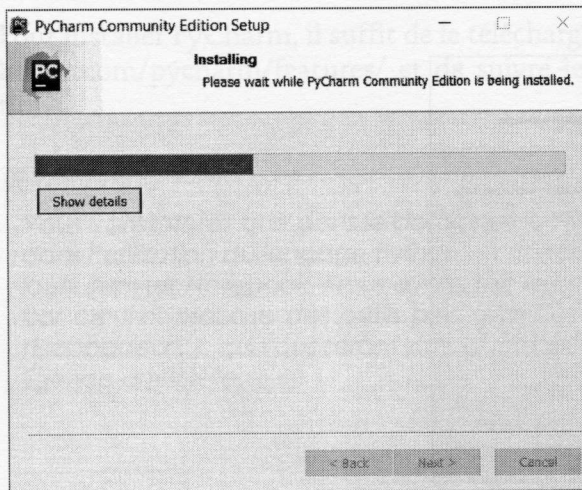
52 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

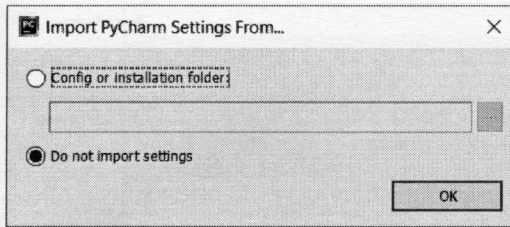
Viennent ensuite les étapes de choix du nom du raccourci dans l'environnement Windows, l'installation proprement dite, l'import de configuration PyCharm et l'acceptation de la charte d'utilisation de l'outil. Pour toutes ces étapes, nous vous invitons à sélectionner les options par défaut.



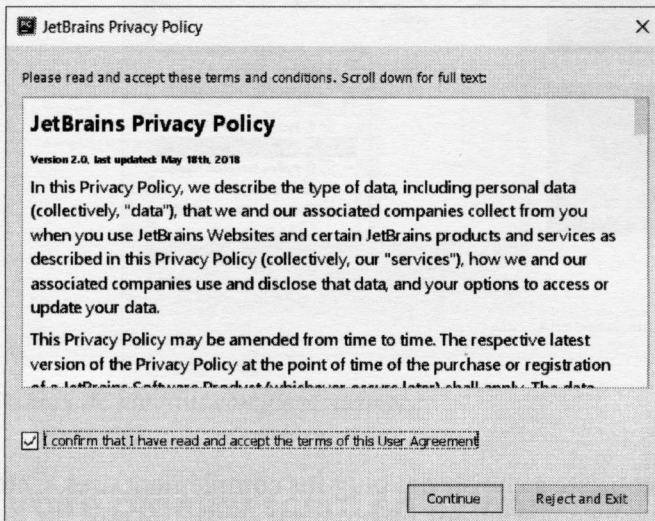
Précision du nom du raccourci de PyCharm dans Windows



Installation de PyCharm

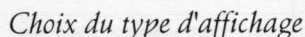


Import de configurations PyCharm existantes

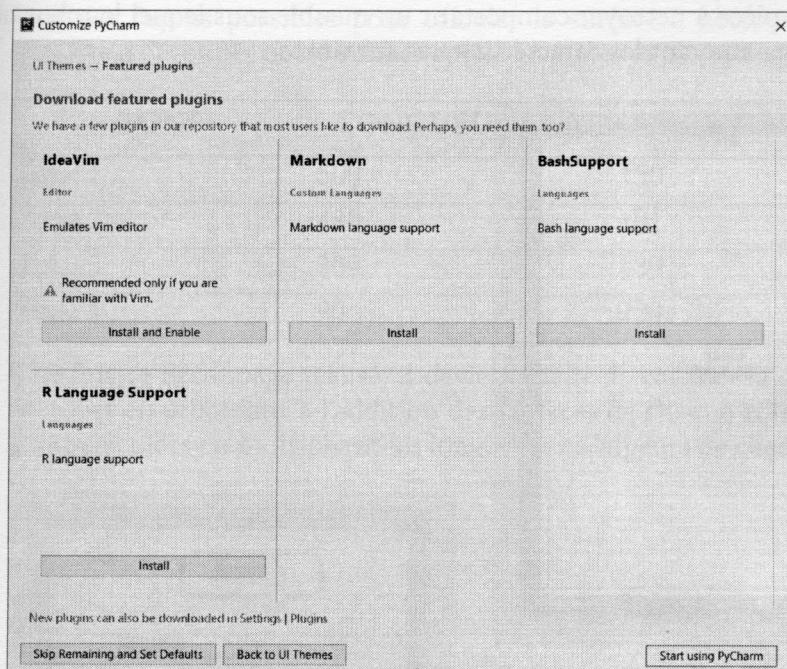


Acceptation de la charte utilisateur

Il est également possible de choisir le type d'affichage de votre environnement de développement. Dans notre cas, nous avons choisi le type Light, mais libre à vous de choisir l'affiche plus sombre si celui-ci vous convient.



Enfin, la dernière étape consiste à choisir des plug-ins complémentaires. Cette étape peut être ignorée, car nous n'aurons besoin d'aucun d'entre eux.



Choix de plug-ins complémentaires

6. Votre premier script en Python

Le projet que nous vous proposons de réaliser consiste en l'écriture d'un script Python permettant à un robot aspirateur de calculer la surface d'une pièce et d'estimer ainsi son temps de nettoyage.

56 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Imaginons la pièce à nettoyer comportant un meuble sous lequel le robot ne peut pas passer et ayant les caractéristiques suivantes :

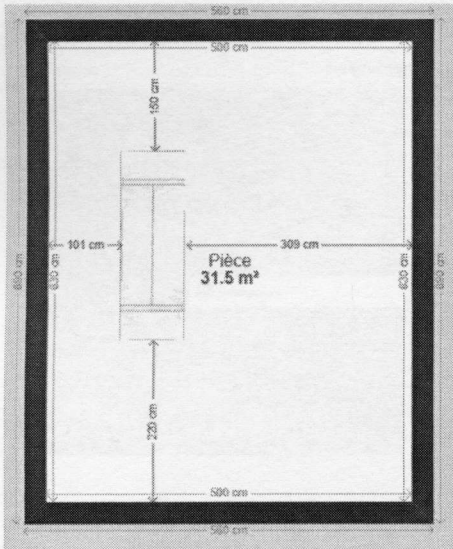
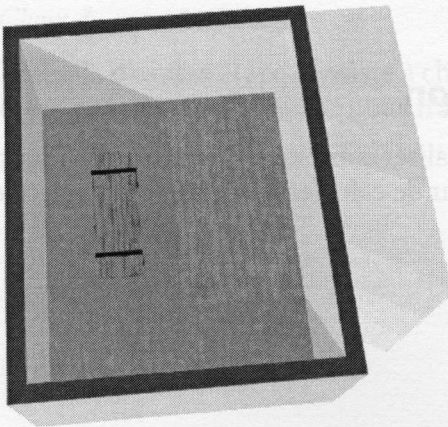


Schéma de la pièce

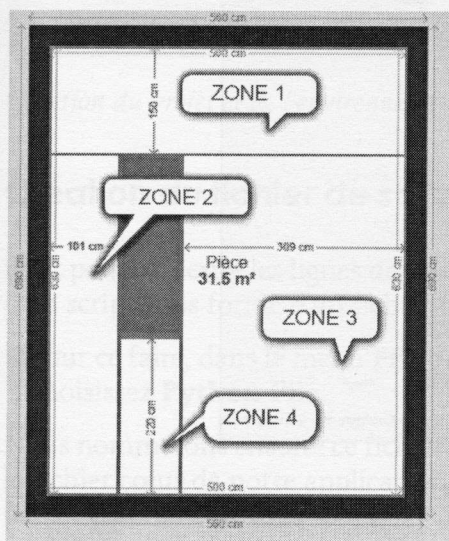


Vue 3D de la pièce

Une des façons possibles pour calculer la surface à nettoyer par le robot consiste à découper notre surface totale en zones exploitables par celui-ci :

ZONE	LONGUEUR (cm)	LARGEUR (cm)
Zone 1	500	150
Zone 2	480	101
Zone 3	309	480
Zone 4	90	220

Une fois ce découpage réalisé, il devient facile de calculer la surface totale à nettoyer en procédant à l'addition des surfaces de chaque zone. Ces surfaces étant calculées en multipliant les longueurs et largeurs de chaque zone.



Zones exploitables par le robot aspirateur

58 — Intelligence Artificielle Vulgarisée

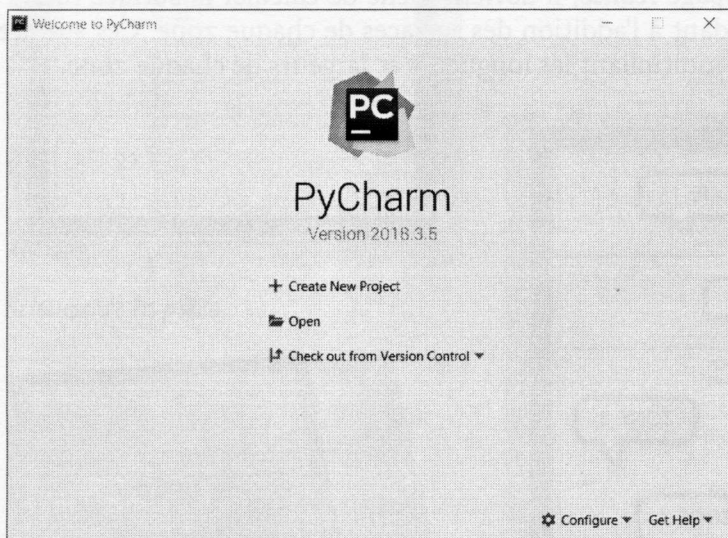
Le Machine Learning et le Deep Learning par la pratique

Maintenant que nous avons défini la façon de procéder, passons à présent au codage du script.

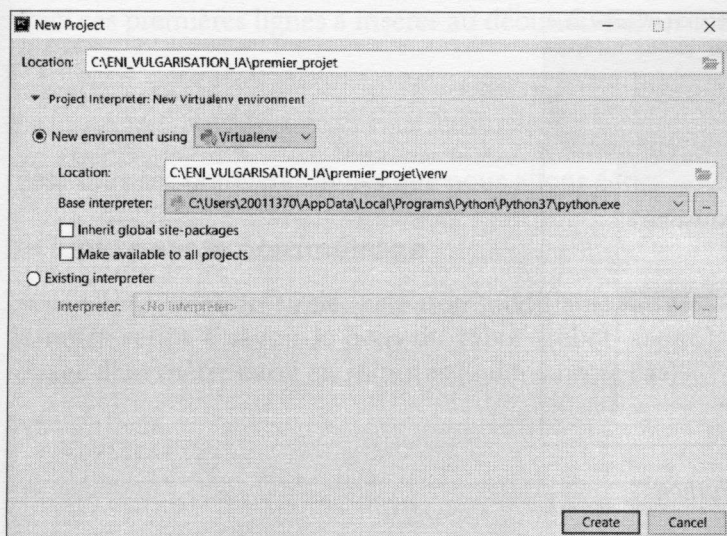
6.1 Création du projet

La première étape consiste à créer un nouveau projet Python à l'aide de l'IDE PyCharm.

Lors de la création de ce projet, un environnement virtuel est créé. Cela permet d'isoler chaque projet et ses dépendances à des modules complémentaires.



Création d'un nouveau projet avec PyCharm



Création du projet et de l'environnement virtuel

6.2 Création du fichier de script principal

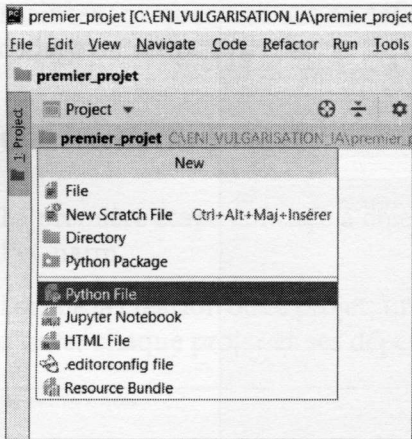
Pour pouvoir écrire les lignes de code de notre programme, nous avons besoin d'un script sous forme d'un fichier portant l'extension `.py`.

■ Pour ce faire, dans le menu **File** de PyCharm, cliquez sur l'option **New** puis choisissez **Python File**.

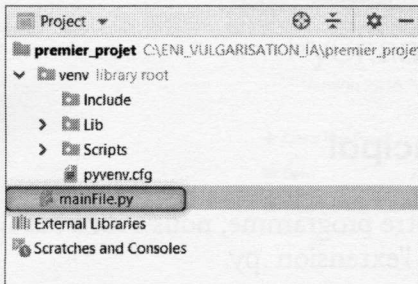
Nous nommerons ensuite ce fichier **mainFile** (fichier principal), celui-ci étant le fichier cœur de notre application. Bien entendu, libre à vous de nommer le fichier comme bon vous semble.

60 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique



Création du script Python



Le fichier mainFile.py

6.3 Nos premières lignes de code

En sélectionnant le fichier `mainFile.py` situé dans la partie gauche de l'éditeur PyCharm, nous avons alors dans la partie droite la possibilité de saisir des lignes de code.

La première chose que nous allons écrire sont des commentaires qui ne seront pas interprétés par le langage Python, mais qui seront d'une grande aide pour comprendre et communiquer à d'autres développeurs ce que nous avons codé. Les commentaires sont symbolisés par l'utilisation du caractère `#`.

Voici ces premières lignes à insérer au début du fichier mainFile.py :

```
#-----  
# APPLICATION  
#-----
```

C'est sous ces premières lignes que nous allons écrire notre script.

6.3.1 Un tuple pour le paramétrage

Nous allons créer un tuple (liste non modifiable) pour définir les paramètres de notre script à savoir le nom du robot (robot_aspiro) et le temps de nettoyage d'un mètre carré en minutes (2 dans notre cas).

```
#-----  
# APPLICATION  
#-----  
  
#Utilisation d'un tuple pour le paramétrage de l'application  
#Nom du robot, temps en minutes pour nettoyer un mètre carré  
parametres = ("robot_aspiro",2)
```

6.3.2 Création des zones à l'aide de dictionnaires

Une zone est définie par sa longueur et sa largeur exprimées en centimètres. Nous allons utiliser des dictionnaires pour créer ces zones :

```
#-----  
# APPLICATION  
#-----  
  
#Utilisation d'un tuple pour le paramétrage de l'application  
#Nom du robot, temps en minutes pour nettoyer un mètre carré  
parametres = ("robot_aspiro",2)  
  
# Utilisation de dictionnaires pour créer les zones  
zone1={"longueur":500,"largeur":150}  
zone2={"longueur":309,"largeur":480}  
zone3={"longueur":101,"largeur":480}  
zone4={"longueur":90,"largeur":220}
```

6.3.3 Regroupement des zones dans une liste

Maintenant que les zones sont définies, nous allons les stocker dans une liste afin de pouvoir accéder à celles-ci plus facilement à partir d'un point unique représenté par la liste.

```
#-----  
# APPLICATION  
#-----  
  
#Utilisation d'un tuple pour le parametrage de l'application  
#Nom du robot, temps en minutes pour nettoyer un metre carré  
parametres = ("robot_aspiro",2)  
  
# Utilisation de dictionnaires pour créer les zones  
zone1={"longueur":500,"largeur":150}  
zone2={"longueur":309,"largeur":480}  
zone3={"longueur":101,"largeur":480}  
zone4={"longueur":90,"largeur":220}  
  
# Utilisation d'une liste permettant de stocker nos  
différentes zones  
zones = []  
zones.append(zone1)  
zones.append(zone2)  
zones.append(zone3)  
zones.append(zone4)
```

6.3.4 Une fonction pour calculer la surface à nettoyer

Une fonction est un ensemble de lignes de code ayant pour but de réaliser une tâche bien définie. Dans notre cas, la fonction que nous allons créer aura pour but de calculer la surface totale à nettoyer. Cette surface sera calculée à partir des caractéristiques des zones qui lui seront passées en paramètres.

Si l'on décrit notre fonction sous forme d'algorithme, cela donne ceci :

Création d'une variable `surfaceTotaleANettoyer` et l'initialiser à 0

Pour chaque zone à nettoyer contenue dans la liste des zones passée en paramètre :

- récupérer la longueur de la zone et la stocker dans une variable
- récupérer la largeur de la zone et la stocker dans une variable
- calculer la surface de la zone en multipliant sa longueur par sa largeur
- ajouter à la surface totale à nettoyer, la surface de la zone que nous venons de déterminer

Renvoyer la `surfaceTotaleANettoyer`

Passons maintenant au codage de cette fonction.

Python interprète les instructions dans l'ordre où il les reçoit. Par conséquent, les fonctions que nous allons utiliser dans notre script doivent être définies en amont de toute ligne de code faisant appel à celle-ci.

Par conséquent, ajoutons ces lignes dans notre script avant le commentaire "Application" :

```
#-----  
# FONCTIONS  
#-----  
  
def calculDeLaSurfaceANettoyer(listeDeZones):  
    surfaceANettoyer = 0  
    for zone in listeDeZones:  
        longueur = zone.get("longueur")/100  
        largeur = zone.get("largeur")/100  
        calcul = longueur*largeur  
        print (str(longueur)+" x "+str(largeur)+"= "+str(calcul))  
        surfaceANettoyer = surfaceANettoyer +calcul  
    return (surfaceANettoyer)  
  
#-----  
# APPLICATION  
#-----
```

64 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

L'instruction `def` permet de définir une fonction portant un nom (`calculDeLaSurfaceANettoyer`) et éventuellement un ou plusieurs paramètres (`listeDeZones` dans notre cas). Notons l'usage des `:` permettant de définir le début de la fonction. Le corps de la fonction sera alors écrit en dessous tout en respectant les indentations (Les tabulations).

Si on analyse le code de notre fonction, on constate la création d'une première variable nommée `surfaceANettoyer` qui est initialisée à 0. Vient ensuite le parcours des zones stockées dans la liste de zones passée en paramètre à l'aide de l'instruction `For`. Chaque zone étant un dictionnaire, on peut donc accéder aux données à l'aide des clés `longueur` et `largeur` pour réaliser le calcul de la surface à nettoyer. Enfin, l'instruction `return` permet de renvoyer la surface calculée.

Notre fonction est à présent codée, il faut maintenant l'utiliser dans notre code comme suit :

```
surfaceANettoyer = calculDeLaSurfaceANettoyer(zones)
print("La surface totale à nettoyer est de :"+str(surfaceANettoyer))
```

6.3.5 Une deuxième fonction pour coder le temps de nettoyage

Nous allons à présent créer une seconde fonction qui permet de calculer le temps de nettoyage en fonction de la surface et du temps de nettoyage pour un mètre carré, passés en paramètre.

Voici le code de cette fonction à ajouter en dessous de la fonction de calcul de la surface :

```
def tempsNettoyageEnMinutes(surfaceANettoyer,
    tempsPourUnMetreCarre):
    return round(surfaceANettoyer*tempsPourUnMetreCarre)
```

Le temps calculé renvoyé par la fonction résulte de l'arrondi (`round`) de la multiplication de la surface à nettoyer par le temps de nettoyage pour un mètre carré.

6.3.6 Le script dans son ensemble

Voici le code complet du script comportant l'utilisation de la fonction que nous venons de créer. Le second paramètre utilisé par notre fonction correspond à la seconde valeur de notre tuple (`parametre[1]`), à savoir le temps de nettoyage de notre robot aspirateur pour un mètre carré.

À noter que nous avons ajouté une structure conditionnelle permettant d'afficher un message en fonction du temps de nettoyage estimé.

```
#-----  
# FONCTIONS  
#-----  
  
def calculDeLaSurfaceANettoyer(listeDeZones):  
    surfaceANettoyer = 0  
    for zone in listeDeZones:  
        longueur = zone.get("longueur")/100  
        largeur = zone.get("largeur")/100  
        calcul = longueur*largeur  
        print (str(longueur)+" x "+str(largeur)+" = "+str(calcul))  
        surfaceANettoyer = surfaceANettoyer +calcul  
    return (surfaceANettoyer)  
  
def tempsNetoyageEnMinutes(surfaceANettoyer,  
    tempsPourUnMetreCarre):  
    return round(surfaceANettoyer*tempsPourUnMetreCarre)  
  
#-----  
# APPLICATION  
#-----  
  
#Utilisation d'un tuple pour le paramétrage de l'application  
#Nom du robot, temps en minutes pour nettoyer un mètre carré  
parametres = ("robot_aspiro",2)  
  
# Utilisation de dictionnaires pour créer les zones  
zone1={"longueur":500,"largeur":150}  
zone2={"longueur":309,"largeur":480}  
zone3={"longueur":101,"largeur":480}  
zone4={"longueur":90,"largeur":220}  
  
# Utilisation d'une liste permettant de stocker nos différentes
```

66 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

```
zones
zones = []
zones.append(zone1)
zones.append(zone2)
zones.append(zone3)
zones.append(zone4)

#Appel de la fonction permettant de calculer la surface à
nettoyer
surfaceANettoyer = calculDeLaSurfaceANettoyer(zones)
print("La surface totale à nettoyer est de
:"+str(surfaceANettoyer)+ " m2")

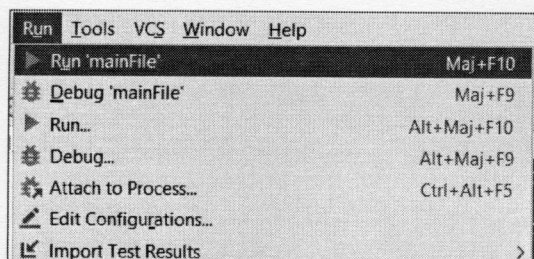
#Appel de la fonction permettant de déterminer le temps de
nettoyage
tempsEstime =
tempsNettoyageEnMinutes(surfaceANettoyer,parametres[1])
print("Le temps estimé est de: "+str(tempsEstime)+" minutes")

#Ajout d'une condition se déclenchant en fonction du temps de
nettoyage
if tempsEstime > 55:
    print(parametres[0]+" dit : Je pense que cela va prendre un
peu de temps !")
```

Il est temps à présent de tester notre code ! Pour cela, il suffit de cliquer sur le menu **Run** puis l'option **Run mainFile** (voir illustration ci-dessous) pour voir apparaître le résultat des différents calculs réalisés par notre programme dans la partie basse de PyCharm.

```
5.0 x 1.5= 7.5
3.09 x 4.8= 14.831999999999999
1.01 x 4.8= 4.848
0.9 x 2.2= 1.9800000000000002
La surface totale à nettoyer est de : 29.16 m2
Le temps estimé est de: 58 minutes
robot_aspiro dit : Je pense que cela va prendre un peu de temps !
```

```
Process finished with exit code 0
```



Exécution de notre script

7. Conclusion

Dans ce chapitre, nous avons abordé rapidement l'utilisation du langage Python et avons appliqué les diverses notions découvertes à travers un premier cas pratique.

Cependant, nous sommes loin d'avoir découvert l'exhaustivité des possibilités de ce langage qui nous servira tout au long de cet ouvrage. Nous aborderons de nouvelles notions au fil des chapitres à venir.

Dans le chapitre suivant, nous allons aborder une tout autre notion, qui peut déplaire à certains ou certaines d'entre vous, mais nécessaire à tout projet de Machine Learning : l'usage des statistiques qui vont nous permettre de comprendre les données que nous allons utiliser dans la phase d'apprentissage du Machine Learning!

Chapitre 3

Des statistiques pour comprendre les données

1. Ce que nous allons découvrir et les prérequis

En ce qui concerne le Machine Learning, tout est mathématique ou plus particulièrement tout est statistique. En effet, les algorithmes d'apprentissages sont fondés sur diverses approches statistiques permettant aux machines "d'apprendre", de réaliser des prédictions et de résoudre des problèmes.

Au-delà de ces prédictions, les statistiques sont également utilisées dans la phase de préparation des données d'apprentissage et de la compréhension de celles-ci.

Dans ce chapitre, nous allons aborder les notions de base des statistiques qui vous permettront de mieux comprendre les jeux de données que vous rencontrerez tout au long de cet ouvrage, mais aussi dans vos différents projets personnels consacrés au Machine Learning. Ces notions permettront également de comprendre le fonctionnement des algorithmes que nous utiliserons à travers les cas pratiques jalonnant ce livre.

Mais ne vous inquiétez pas. Ce chapitre n'a pas pour objectif de vous embêter avec un tas de formules complexes. Nous resterons à un niveau de compréhension accessible à tous et les notions que nous aborderons seront illustrées à l'aide d'exemples.

■ Remarque

Prérequis nécessaires pour bien aborder ce chapitre : avoir lu le chapitre Les fondamentaux du langage Python.

2. Les statistiques, un outil d'aide à la compréhension des données

Dans les chapitres à venir, nous serons amenés à analyser des données d'apprentissage afin de permettre à notre machine d'apprendre à résoudre un problème par le biais de prédictions. Comme nous le verrons, pour que la machine puisse apprendre, il ne suffit pas de choisir une collection d'observations, de choisir un algorithme d'apprentissage et de cliquer sur un bouton en espérant obtenir une bonne prédiction. Nous serons amenés à comprendre par nous-mêmes ce jeu d'observations pour en sélectionner les informations pertinentes et importantes. Pour réaliser cette tâche, les statistiques nous seront d'une grande aide.

Bien entendu, en ce qui concerne la programmation, nous utiliserons par la suite des modules Python spécifiques comportant toutes les fonctionnalités d'analyse nécessaires. Mais dans ce chapitre, nous vous proposons d'agrémenter chaque notion statistique par quelques lignes de code "faites maison", c'est-à-dire dans utilisation de librairie spécifique. Cela permet entre autres de mieux comprendre la logique mathématique qui se cache derrière chaque notion, mais aussi d'approfondir la connaissance et l'usage du langage Python.

■ Remarque

Le code Python peut être utilisé dans un nouveau projet que vous aurez préalablement créé dans PyCharm. Cependant, nous ne détaillerons pas dans ce chapitre toutes les subtilités d'utilisation du code, car nous souhaitons nous focaliser sur les notions essentielles à savoir l'usage des statistiques. Vous pourrez néanmoins retrouver le code complet et un exemple dans les ressources techniques de cet ouvrage.

3. Une série de notes en guise d'étude de cas

Nous allons utiliser un cas d'étude mainte fois utilisé lors de la présentation des notions de base en statistiques : la description et l'analyse d'une série de notes obtenues par un étudiant. À la fois simple et parlant à tous, car se basant sur notre expérience, ce cas d'étude va nous permettre de comprendre les notions nécessaires à la compréhension des données et applicables à des cas plus complexes que nous aurons à traiter dans les chapitres suivants.

Voici le contexte : vous êtes actuellement en formation pour devenir expert(e) en intelligence artificielle et vous venez d'obtenir vos notes trimestrielles que voici :

■ 3, 19, 10, 15, 14, 12, 9, 8, 11, 12, 11, 12, 13, 11, 14, 16

Comme vous pouvez le constater, cette petite série statistique comporte une multitude de notes toutes différentes des unes des autres. Nous allons dans un premier temps procéder à la déduction d'une tendance centrale, c'est-à-dire essayer de déterminer une valeur autour de laquelle se concentre l'ensemble des notes, puis nous étudierons sa dispersion et enfin nous rechercherons l'existence de données aberrantes, c'est-à-dire non représentatives de l'ensemble des notes. Nous avons tous le souvenir d'un examen où nous n'avions pas forcément révisé les notions essentielles ce qui nous a valu l'obtention d'une note non représentative des notes que nous avons obtenues tout au long de l'année.

Ce type d'analyse devra être effectué sur toute série d'observations que vous aurez à utiliser dans le cadre de projets en Machine Learning. En effet, réaliser les mesures de tendance et de dispersion de vos données vous permettra de comprendre leur signification, leur rôle et leur utilité dans la prédiction à effectuer, identifier les données aberrantes vous permettra d'exclure des données non représentatives qui pourraient perturber l'apprentissage.

4. Petites notions de vocabulaire avant de commencer

4.1 Observations et features

Toute étude statistique se réalise sur une **population**, composée d'**individus**. Ces individus peuvent être des personnes ou des choses. Chaque individu dispose de caractères. En Machine Learning, les individus sont appelés des **observations** et les caractères des **features** (caractéristiques).

■ Remarque

Tout au long de cet ouvrage, nous privilégierons le terme de feature à celui de caractéristique, car c'est ce terme que vous rencontrerez le plus dans vos différentes lectures et cas pratiques.

4.2 Les types de données

Il existe deux types de données : les données quantitatives et les données qualitatives.

Les données quantitatives sont des nombres sur lesquels il est possible de faire des opérations mathématiques. Ces données peuvent être classées en deux groupes distincts :

- les données quantitatives continues ayant une valeur infinie et pouvant être énumérée (nombre d'éléments d'une liste...),
- les données quantitatives discrètes ont quant à elles une valeur finie dans un intervalle de valeurs (le nombre de pattes d'un animal...).

Quant aux données qualitatives, celles-ci ont pour but de décrire des qualités propres aux données : est-ce un homme, une femme? Est-ce petit ou grand?

Ces données ne sont pas des nombres et sont également réparties en deux groupes distincts :

- les données qualitatives catégorielles ne pouvant être ordonnées (homme ou femme, chien ou chat...),
- les données qualitatives ordinales pouvant être ordonnées (petit, moyen, grand).

Connaître les différents types de données, permet d'obtenir une première analyse des données et les traitements que nous serons amenés à réaliser.

5. Et Python dans tout ça?

5.1 Des modules dédiés

Bien que nous n'évoquerons pas dans ce chapitre la création d'un projet Python dédié aux statistiques et l'utilisation de modules externes, il est important de savoir que dans les chapitres suivants nous ne serons pas démunis face à l'analyse statistique grâce au langage Python. En effet, vous verrez que nous utiliserons très fréquemment le module **Pandas** pour l'analyse, et pour les manipulations des listes nous utiliserons le module **numpy**. Ces modules offrant tous deux des fonctions très pratiques, nous évitant de les coder par nous même.

Dans le module Pandas (repris sous l'alias `pnd` dans le code ci-dessous), l'ensemble des observations est appelé un `DataFrame`. Pour notre étude de cas, cela se traduit par cette ligne de code :

```
observations =  
pnd.DataFrame({'NOTES':np.array([3,19,10,15,14,12,9,8,11,12,  
11,12,13,11,14,16])})
```

Notre `DataFrame observations` contient une feature `Notes` sous forme d'un tableau (array) contenant l'ensemble des notes.

5.2 Une représentation un peu particulière de notre étude de cas

Pour notre étude de cas, nous disposons d'une unique observation (vous en tant qu'individu) et de 16 features correspondant à chacune de vos notes. Cette observation pouvant se traduire sous cette forme :

NOM	NOTE 1	NOTE 2	NOTE 3	...	NOTE 16
VOTRE NOM	3	19	10	...	16

On constate que la feature Notes est dispatchée en plusieurs colonnes. Cependant, la représentation d'un DataFrame en Python pour une caractéristique donnée se fait sous forme de ligne.

NOTES
3
19
10
...
16

5.3 Pas de Python, mais Excel en guise d'outil

Afin que vous puissiez manipuler les différents concepts, nous vous invitons à utiliser Excel, offrant une souplesse et une facilité dans son utilisation. Dans la première colonne A d'un classeur Excel, nous allons saisir les différentes notes. Dans la cellule A1, nous précisons l'en-tête de la colonne.

	A
1	NOTES
2	3
3	19
4	10
5	15
6	14
7	12
8	9
9	8
10	11
11	12
12	11
13	12
14	13
15	11
16	14
17	16

Listes des notes obtenues saisies dans Excel

Nous pouvons à présent commencer l'analyse des données, par la mesure de tendance centrale.

■ Remarque

Le fichier Excel utilisé pour l'analyse des données de notre exemple est téléchargeable sur le site de l'éditeur.

6. Mesure de tendance centrale

Les mesures de tendance centrale servent à synthétiser la série statistique étudiée au moyen d'un petit nombre de valeurs "caractéristiques". En d'autres termes, nous allons essayer de trouver un certain nombre de valeurs autour desquelles se regroupe l'ensemble des notes que vous avez obtenues lors des divers examens.

6.1 Connaître le nombre d'observations et de features

La première chose à connaître lorsque nous étudions un ensemble d'observations c'est leur nombre.

En effet, en connaissant le nombre d'observations et en le comparant au nombre de features renseignées, il est possible de détecter rapidement si certaines observations ont des caractéristiques manquantes. Connaître le nombre d'observations permet aussi de savoir si nous disposons d'assez de données pour permettre un apprentissage.

Pour connaître le nombre de features que comporte notre jeu d'observations, nous pouvons utiliser la fonction Python `count()`.

```
print("-- NOMBRE D'OBSERVATIONS --")
n = feature.count()
print("Nombre d'observations = " + str(n))
```

Ce qui nous donne une valeur de 16 features dans notre cas, correspondant aux 16 notes.

■ Remarque

Pour connaître le nombre d'observations dans Excel, il suffit d'utiliser la fonction NBVAL(A2:A17) dans une nouvelle cellule de votre classeur.

6.2 Les valeurs minimales et maximales

Connaître la valeur minimale et la valeur maximale d'un ensemble d'observations est toujours intéressant pour établir son étendue.

En ce qui concerne notre cas, on constate après avoir classé nos observations par ordre croissant que la valeur minimale est de 3 et que la valeur maximale est de 19. L'étendue est donc de 19-3 soit 16, ce qui montre une forte dispersion de données. En effet si les notes obtenues étaient celles-ci :

■ 12, 13, 14, 15, 12, 13, 12, 15, 15, 12, 11, 11, 14, 15, 16, 11

- le nombre de notes serait de 16 comme dans notre exemple,
- la note minimale serait de 11,
- la note maximale serait de 16,

L'étendue serait de 5, montrant que la dispersion de la série est faible, c'est-à-dire que les valeurs des notes seraient proches les unes des autres, contrairement à notre étude de cas.

En Python, pour connaître les valeurs minimales et maximales de notre série statistique, nous devons :

- trier dans l'ordre croissant les différentes valeurs,
- réindexer la liste,
- prendre la première valeur pour connaître la valeur minimale,
- prendre la dernière valeur pour connaître la valeur maximale.

```
print ("\n-- MIN --")
valeursTriees = feature.sort_values()
valeursTriees = valeursTriees.reset_index(drop=True)
print("Valeur minimale : "+str(valeursTriees[0]))

print ("\n-- MAX --")
valeursTriees = feature.sort_values()
valeursTriees = valeursTriees.reset_index(drop=True)
print("Valeur maximale : " +
      str(valeursTriees[len(valeursTriees)-1]))
```

La réindexation de la liste est nécessaire, car chaque valeur garde son index d'origine après le tri. En voici l'explication. Le tableau ci-dessous montre la liste d'origine accompagnée des différents index (position de la valeur dans la liste) :

INDEX	0	1	2
VALEUR	8	5	4

78 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Voici à présent la liste et les index après le tri :

INDEX	2	1	0
VALEUR	4	5	8

On constate que les valeurs ont bien été triées dans l'ordre croissant, **mais elles ont gardé leur index d'origine** ! De ce fait, si nous cherchons à récupérer la première valeur de la liste triée comme suit : `listeTrie[0]`, nous aurons pour résultat la valeur 8, alors que nous attendions la valeur 4 ! En réindexant la liste, cela permet de résoudre ce souci.

Voici maintenant les nouveaux index suite à la réindexation :

INDEX	0	1	2
VALEUR	4	5	8

■ Remarque

Dans Excel, les fonctions `Min(A2:A17)` et `MAX(A2:A17)` vous permettront d'obtenir les valeurs minimales et maximales de notre série statistique.

6.3 La moyenne arithmétique

La moyenne est un indicateur qui permet de caractériser une série statistique. Elle permet de connaître la valeur des éléments de notre série statistique si celle-ci était répartie équitablement. Dans notre cas, la moyenne permet de répondre à la question suivante :

"Si nous devons donner la même valeur à chacune de nos 16 notes laquelle serait-elle ?"

Pour répondre à cette question, nous devons :

- calculer la somme de l'ensemble des valeurs,
- diviser cette somme par le nombre d'observations.

Voici une fonction Python permettant de calculer la moyenne :

```
def calculMoyenneArithmetique(feature):

    n = feature.count()
    sommeValeursObservations = 0
    moyenneArithmetique = 0

    if n>0 :
        for valeurObservation in feature:
            sommeValeursObservations = sommeValeursObservations +
            valeurObservation

        moyenneArithmetique = sommeValeursObservations / n

    return moyenneArithmetique
```

Cette fonction reçoit en paramètres les caractéristiques de la série d'observations. Tout en s'assurant que le nombre de features est bien supérieur à 0 (car la division par 0 est impossible), la fonction effectue la somme des valeurs des différentes caractéristiques et la divise par le nombre de caractéristiques.

Dans notre cas, nous obtenons une moyenne de : **11,875**.

Remarque

Voici les formules Excel à utiliser. Cellule D2 = NBVAL(A2:A17) Cellule D3=MIN(A2:A17) Cellule D4=MAX(A2:A17) Cellule D5=D4-D3 Cellule D6=SOMME(A2:A17) Cellule D7=D6/D2.

	A	B	C	D	E
1	NOTES				
2	3		NOMBRE D'OBSERVATIONS :	16	
3	19		MIN	3	
4	10		MAX	19	
5	15		ETENDUE	16	
6	14		SOMME DES NOTES	190	
7	12		MOYENNE	11,875	
8	9				
9	8				
10	11				
11	12				
12	11				
13	12				
14	13				
15	11				
16	14				
17	16				
18					

Calcul de la moyenne

Cela signifie que si nous répartissons de façon équitable toutes les notes, celles-ci auraient pour valeur 11,875. En d'autres termes, si la même performance avait été réalisée pour chaque examen, la note obtenue à chacun d'entre eux aurait été de 11,875.

Malgré de très bonnes notes obtenues (16,19), votre performance moyenne est aux alentours de 12.

6.4 La médiane

La médiane est la valeur qui partage la série d'observations en deux groupes de même effectif. C'est-à-dire que l'on cherche à définir une valeur où il y aurait autant de notes supérieures et autant de notes inférieures à celle-ci.

Pour calculer la médiane, il faut :

- ordonner la liste des observations par ordre croissant de valeurs,
- déterminer si le nombre d'observations est pair ou impair,
- procéder au calcul de la médiane.

6.4.1 Cas d'un nombre d'observations impair

Imaginons la série d'observations suivante comportant sept valeurs :

■ [11, 12, 13, 14, 15, 16, 17]

La médiane de cette série se calcule comme suit :

- définition du rang de la médiane : $(7+1) / 2 = 4$
- recherche de la valeur de la médiane de rang 4 = 14

MÉDIANE AVEC UN NOMBRE IMPAIR D'OBSERVATIONS						
11	12	13	14	15	16	17
Médiane = 14						

Calcul de la médiane (nombre impair d'observations)

La médiane de cette série est donc de 14.

6.4.2 Cas d'un nombre d'observations pair

Imaginons la série d'observations suivante comportant huit valeurs :

■ [11, 12, 13, 14, 15, 16, 17, 18]

La médiane de cette série se calcule comme suit :

- calcul du rang de la médiane : $8/2 = 4$
- prise de la valeur du rang calculé (Rang n°4) = 14
- prise de la valeur du rang calculé + 1 = Rang n° 4+1 = Rang n° 5 = 15
- calcul de la moyenne des deux valeurs = $(15-14)/2 = 0,5$
- ajout de la moyenne à la première valeur = $14+0,5 = 14,5$

MÉDIANE AVEC UN NOMBRE PAIR D'OBSERVATIONS							
11	12	13	14	15	16	17	18
Médiane = $14 + ((15 - 14) / 2) = 14,5$							

Calcul de la médiane (nombre pair d'observations)

La médiane de cette série est donc de 14,5.

6.4.3 Retour à notre exemple

Dans notre exemple, nous disposons de 16 observations. 16 étant un nombre pair, la médiane se trouve entre la 8^e (16/2) et la 9^e ((16/2)+1) observation.

- La valeur de la 8^e observation est de 12.
- La valeur de la 9^e observation est de 12.
- La médiane est donc de $12 + ((12-12)/2) = 12$.

Cela signifie donc que vous avez eu autant de notes au-dessus de 12 que de notes en dessous de 12.

■ Remarque

Attention à bien ordonner les notes avant de se prononcer sur la médiane !

82 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

	A	B	C	D
1	NOTES	NOTES ORDONNEES		
2	3		3 NOMBRE D'OBSERVATIONS :	16
3	19		8 MIN	3
4	10		9 MAX	19
5	15		10 ETENDUE	16
6	14		11 SOMME DES NOTES	190
7	12		11 MOYENNE	11,875
8	9		11 MEDIANE	12
9	8		12	
10	11		12	
11	12		12	
12	11		13	
13	12		14	
14	13		14	
15	11		15	
16	14		16	
17	16		19	

Calcul de la médiane dans Excel

■ Remarque

Pour trouver la médiane d'une série avec Excel, il suffit d'utiliser la fonction comme suit pour notre cas : `MEDIANE(A2:A17)` qui par la même occasion s'occupe d'ordonner par ordre croissant la série.

Voici à présent la fonction de calcul de la médiane en Python :

```
def calculMediane(feature):
    mediane = 0
    feature = feature.sort_values()
    feature = feature.reset_index(drop=True)
    n = self.feature.count()
    pair = False;
    if (n % 2 == 0):
        print("Le nombre d'observations est pair.")
        pair = True

    if pair:
        rang = (n / 2);
        rangPython = rang-1
        valeur1 = feature[rangPython]
        valeur2 = feature[rangPython+1]
        mediane = valeur1 + ((valeur2-valeur1)/2)
    else:
        rang = ((n + 1) / 2)
        rangPython = rang - 1
        mediane = feature[rangPython]

    return mediane
```

En première étape, la série de valeurs est triée et les index des valeurs sont mis à jour (cf. la détermination des valeurs minimales et maximales décrites précédemment).

Nous cherchons ensuite à savoir si le nombre d'observations est pair ou impair à l'aide de la fonction `modulo (%)`. Le modulo consiste à connaître le reste de la division euclidienne. Si le reste de la division du nombre d'observations par 2 est égal à zéro, alors le nombre d'observations est pair, dans le cas contraire il est impair.

On applique ensuite les formules de calcul pour obtenir la médiane en correspondance avec le fait que le nombre d'observations est pair ou impair.

Vous avez sans doute remarqué que nous n'utilisons pas le rang calculé pour obtenir la médiane, mais une variable appelée `rangPython`.

Cela vient du fait que la numérotation des rangs (index) dans une liste Python est différente de la numérotation naturelle, car elle **commence à zéro** !

RANG NATUREL	1	2	3
RANG PYTHON	0	1	2
VALEUR	7	9	18

Si nous extrayons en Python la valeur de rang 2 de cette liste à l'aide de l'instruction `liste[2]`, nous obtenons pour valeur 18 et non 9 comme attendu !

Car en Python la valeur attendue se situe au rang 1. D'où le fait de soustraire 1 à la valeur du rang calculé de la médiane pour obtenir le rang à utiliser sur la liste à l'aide du langage Python.

6.5 Le mode

Le mode d'une série d'observations correspond à la valeur la plus fréquemment présente. On parle donc du nombre maximum de fréquences de présence d'une valeur au sein de la série.

Pour notre cas :

NOTE	12	11	14	3	19	10	15	9	8	13	16
FRÉQUENCE	3	3	2	1	1	1	1	1	1	1	1

Deux notes sont fréquemment présentes dans la série avec une fréquence identique (3 pour la note 11 et la note 12). L'ensemble de la série est donc bimodal (deux modes).

■ Remarque

Lorsqu'une série a pour mode deux valeurs, on dit qu'elle est bimodale, trimodale pour trois et multimodale au-delà de trois. Il se peut également qu'une série ne dispose pas de mode.

Voici à présent le code Python qui utilise la fonction Counter se chargeant de calculer la fréquence de chaque valeur et renvoyer le résultat sous forme d'un dictionnaire :

```
def calculMode(feature):
    mode = Counter(feature)
    return mode
```

```
Counter({12: 3, 11: 3, 14: 2, 3: 1, 19: 1, 10: 1, 15: 1, 9: 1, 8: 1, 13: 1, 16: 1})
```

■ Remarque

Le mode de la série se calcule à l'aide d'Excel grâce à la fonction MODE(A2:A17). Cependant, celle-ci ne nous permet pas d'identifier si la série est bimodale.

7. Premières déductions

Reprenons les différentes mesures de la tendance centrale pour notre jeu d'observations :

- 11,875 pour la moyenne
- 12 pour la médiane
- 11 et 12 pour le mode

La tendance centrale a pour objectif de réduire l'ensemble des valeurs d'une série d'observations par un nombre la résumant. Dans notre cas, ce nombre peut être évalué à 12.

Ce qui signifie que si l'on devait résumer toutes vos notes obtenues aux examens d'intelligence artificielle en une seule, celle-ci s'élèverait à 12, malgré les notes de 15, 16, 19 obtenues, car la majeure partie de vos notes se rassemble autour de cette valeur.

Nous venons donc de synthétiser 16 notes en une seule ! Ce qui permet d'obtenir un aperçu de la valeur de générale série statistique. Dans notre cas, nous n'avons traité que quelques données, mais dans les chapitres suivants nous serons amenés à traiter des exemples comportant plus de 1000 valeurs. Sans les calculs de la moyenne, de la médiane et du mode, il est difficile de se faire une idée de la valeur générale d'une caractéristique.

Prenons l'exemple d'une série regroupant les prix des voitures vendues par un concessionnaire automobile. Au vu du nombre d'options pouvant faire varier le prix d'un véhicule, il est évident que la série va comporter un certain nombre de valeurs différentes. Mais si en tant que consommateur vous souhaitez prévoir un budget pour vous acheter un véhicule chez ce concessionnaire, la moyenne, la médiane et le mode pourront vous aider à prévoir ce budget, car l'ensemble des prix seront synthétisés en une seule valeur.

8. La dispersion

La dispersion d'une série d'observations mesure la variabilité de celle-ci. Avez-vous obtenu beaucoup de notes ayant toutes la même valeur? Ou bien des notes différentes? C'est ce que nous allons découvrir en commençant par calculer l'étendue de la série d'observations.

8.1 L'étendue

L'étendue d'une série d'observations s'effectue en calculant la différence entre la valeur maximale et la valeur minimale de la série.

Si l'étendue est très petite, alors on peut affirmer qu'il y a peu d'écart entre toutes les valeurs de la série. Cette série est dite homogène.

Dans le cas contraire, on peut considérer que l'étendue est forte, mais on ne peut pas considérer qu'elle est hétérogène, car il se peut que cet écart soit dû à une valeur aberrante.

Prenons l'exemple de la série suivante :

8,10,11,12,12,11,9 : L'étendue est de 4 (12-8)

Puis une autre série ayant ces valeurs :

1,10,11,12,12,11,19 : L'étendue est de 18 (19-1)

Malgré une forte étendue, la série est tout de même homogène (se concentrant autour de 12), car les valeurs créant cette forte étendue (19 et 1) peuvent être considérées comme des données extrêmes, voire aberrantes (*outliers* en anglais).

Concernant notre étude de cas, l'étendue est de 16, car la valeur maximale est de 19 et la valeur minimale est de 3 ($19-3 = 16$). On peut donc dire que notre série est étendue.

Voici le code Python permettant de calculer l'étendue :

```
valeursTriees = feature.sort_values()
valeursTriees = valeursTriees.reset_index(drop=True)

print ("Etendue de la série =
"+str(valeursTriees[len(valeursTriees)-1]-valeursTriees[0]))
```

■ Remarque

On note le fait que les valeurs de la feature doivent être obligatoirement triées avant de pouvoir calculer son étendue.

8.2 L'écart type (Standard déviation)

Comment savoir si notre jeu d'observations est fortement dispersé ou non ? C'est grâce à l'écart type que nous allons avoir la réponse.

■ Remarque

Selon l'INSEE, l'écart-type sert à mesurer la dispersion, ou l'étalement, d'un ensemble de valeurs autour de leur moyenne. Plus l'écart-type est faible, plus la population est homogène.

Calculer l'écart-type va donc nous permettre de savoir si les notes que vous avez obtenues aux examens sont toutes concentrées autour de la moyenne ou si au contraire elles s'étalent sur l'intervalle de valeurs compris entre 3 et 19 (les valeurs minimale et maximale de la série).

Mais avant cela, nous devons calculer une mesure préalable au calcul de l'écart-type appelé la variance.

88 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

8.2.1 Calcul de la variance

La variance mesure le degré de dispersion des données. Elle se calcule en prenant la moyenne de l'écart élevé au carré de chaque nombre par rapport à la moyenne d'un ensemble de données. À première vue, cela semble compliqué, mais il n'en est rien.

Pour chaque valeur de la série d'observations, nous allons :

- calculer son écart par rapport à la moyenne,
- mettre cet écart au carré.

Pour ensuite faire la somme de ces écarts élevés au carré et en calculer la moyenne par rapport aux nombres d'observations – 1.

En d'autres termes, il s'agit tout simplement de réaliser une moyenne des écarts élevés au carré.

Voici le tableau qui nous permet de calculer la variance.

VALEUR	MOYENNE	ECART	ECART AU CARRE
3	11,875	-8,875	78,765625
19	11,875	7,125	50,765625
10	11,875	-1,875	3,515625
15	11,875	3,125	9,765625
14	11,875	2,125	4,515625
12	11,875	0,125	0,015625
9	11,875	-2,875	8,265625
8	11,875	-3,875	15,015625
11	11,875	-0,875	0,765625
12	11,875	0,125	0,015625
11	11,875	-0,875	0,765625
12	11,875	0,125	0,015625
13	11,875	1,125	1,265625
11	11,875	-0,875	0,765625
14	11,875	2,125	4,515625
16	11,875	4,125	17,015625
SOMME :			195,75
NOMBRE DE VALEURS :			16
VARIANCE:			13,05

Calcul de la variance

Commençons par la première valeur de la série, à savoir le chiffre 3. Son écart avec la moyenne est de : $3 - 11,875$ soit $-8,875$. Élever l'écart au carré consiste à le multiplier par lui-même : $-8,875 * -8,875 = 78,765625$. Il faut ensuite réaliser les mêmes calculs pour l'ensemble des valeurs de la série pour ensuite réaliser la somme des écarts élevés aux carrés (195,75). Cette somme est ensuite à diviser par le nombre d'observations -1 : $195,75 / (16-1)$, ce qui nous donne une variance de 13,05.

■ Remarque

Attention, le calcul de la variance entraîne un changement d'unité. Comme nous avons réalisé la mise au carré des écarts, l'unité de la variance est le carré de celle du caractère de la valeur. À titre d'exemple, si le caractère de la valeur est en mètre (m), sa moyenne est en mètre (m), mais sa variance est en mètre carré (m²).

8.2.2 Calcul de l'écart type

Comme nous venons de le voir, la variance a une unité différente de celle des valeurs de la série et par conséquent différente également de celle de la moyenne. Comme la dispersion d'une série de valeurs s'effectue par rapport à la moyenne, il faut que l'indicateur chargé de mesurer cette dispersion (la variance) soit de la même unité que celle de la moyenne. Sinon, cela revient à mélanger des choux et des carottes (phrase préférée des professeurs de mathématiques).

Pour y parvenir, il suffit d'effectuer la racine carrée de la variance. En faisant cela, nous venons par la même occasion de calculer l'écart type !

Écart type = racine carrée de la variance = racine carrée de 13,05 = 3,61.

■ Remarque

Dans Excel, l'écart type se calcule soit en utilisant la fonction RACINE() si nous disposons déjà de la variance ou en utilisant la fonction ECARTYPE.STANDARD(A2:A17) pour notre exemple.

8.2.3 Interprétation de l'écart type

Plus la valeur de l'écart type est petite, plus la dispersion est faible. C'est-à-dire que la majeure partie des valeurs de la série d'observations se trouve autour de la moyenne.

Dans notre cas, un écart type de 3,61 signifie que la série n'est pas très dispersée. **Cela montre que les notes que vous avez obtenues sont autour de la moyenne.** Par conséquent, il peut exister des valeurs non représentatives de l'ensemble des notes obtenues dans la série d'observations faisant augmenter l'écart-type (les notes 3 et 19 par exemple). Mais nous nous attarderons sur ce point un peu plus loin dans ce chapitre.

8.3 Les quartiles et interquartile

Pour mesurer la dispersion d'une série d'observations, nous pouvons également nous appuyer sur les quartiles et l'interquartile qui vont en complément nous permettre de mettre en évidence les valeurs aberrantes aussi appelées outliers.

8.3.1 Les quartiles

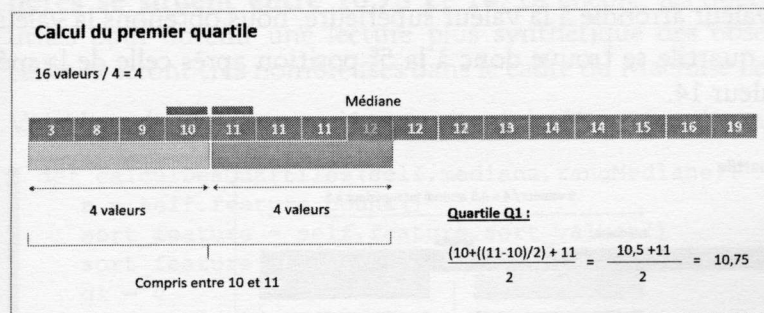
Les quartiles consistent à découper la série d'observations ordonnée par ordre croissant en quatre classes d'effectifs égaux (en quatre groupes contenant le même nombre de données).

Notre série d'observations comporte 16 valeurs. Ce qui représente 4 classes de 4 notes ($16/4 = 4$). La répartition est donc un nombre entier.

Pour trouver la valeur du premier quartile (nommé Q1), nous comptons le nombre de valeurs comprises entre la première note et la médiane que nous divisons ensuite par 2.

Dans notre série d'observations, il y a 8 valeurs entre la première note et la médiane. Donc $8/2 = 4$. C'est un nombre entier, ce qui signifie que nous allons devoir choisir entre deux valeurs.

La figure ci-après illustre ce qui vient d'être énoncé et montre que nous allons devoir déterminer une valeur comprise entre 10 et 11 de notre série de notes.



Calcul du premier quartile

Dans ce cas, une des premières façons de calculer la valeur du quartile est de réaliser la moyenne des valeurs 10 et 11 soit $(11-10) / 2$ pour obtenir la valeur de 10,5.

L'autre option qui est retenue par les outils tels qu'Excel ou le module Pandas de Python est la suivante :

$$\blacksquare \quad ((10 + ((11-10)/2) + 11) / 2 = 10,75$$

C'est donc celle-ci que nous allons retenir tout au long de cet ouvrage afin de garder une cohérence entre les calculs manuels et ceux réalisés par des outils utilisés de façon internationale et sur lesquels s'appuient de nombreux scientifiques.

Nous pouvons donc dire que le premier quartile a pour valeur 10,75. Nous verrons un peu plus tard ce que cela signifie. La valeur du second quartile (nommé Q2) correspond à la médiane de la série, soit la note 12. Pour le troisième quartile (nommé Q3), nous comptons le nombre de valeurs comprises entre la médiane et la dernière valeur de la série puis nous divisons par 2.

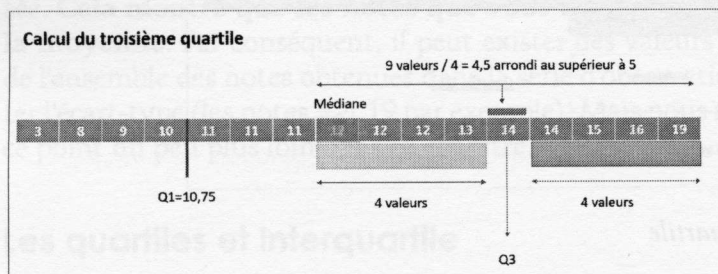
Si le nombre obtenu est un entier, la valeur du quartile devra être calculée entre deux valeurs de la série. Dans le cas contraire, nous arrondirons à la valeur supérieure la valeur obtenue, ce qui correspondra à la position de la valeur du quartile.

92 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Dans notre cas, il y a 9 valeurs entre la médiane et la dernière valeur de la série. Nous divisons ce nombre par 2 et obtenons une valeur non entière ($9/2 = 4,5$).

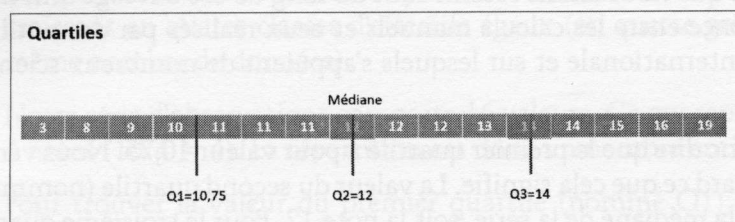
Une fois cette valeur arrondie à la valeur supérieure, nous obtenons la valeur 5. La valeur du quartile se trouve donc à la 5^e position après celle de la médiane, soit la valeur 14.



Calcul des quartiles

Ces étapes nous permettant de définir les quartiles suivants :

- Q1 = 10,75
- Q2 = 12
- Q3 = 14



Quartiles calculés

L'obtention de la valeur des différents quartiles nous permet une lecture complémentaire de notre jeu d'observations :

- **25 % des observations ont une valeur inférieure à 10,75**
- **50 % des observations ont une valeur inférieure à 12**
- **75 % des observations ont une valeur inférieure à 14**

En d'autres termes, 25 % des notes que vous avez obtenues sont inférieures à 10,75 et 25 % des notes sont supérieures à 14, ce qui signifie que **50 % des notes se situent entre 10,75 et 14**. Là encore, les quartiles nous seront utiles pour obtenir une lecture plus synthétique des observations lorsque celles-ci seront très nombreuses dans le cadre du Machine Learning.

Voici le code Python associé permettant de déterminer les quartiles :

```
def calculDesQuartiles(self, mediane, rangMediane):
    n = self.feature.count()
    sort_feature = self.feature.sort_values()
    sort_feature = sort_feature.reset_index(drop=True)
    q1 = 0
    q2 = mediane
    q3 = 0

    #Calcul Q1
    resteDivision = rangMediane%2
    if (resteDivision != 0):
        q1 = sort_feature[((rangMediane/2)+1)-1]
    else:
        valeurMin = sort_feature[((rangMediane/2)-1)]
        valeurMax = sort_feature[(rangMediane/2)]
        q1 = (valeurMin + ((valeurMax - valeurMin) / 2) +
valeurMax) / 2

    # Calcul Q3
    nbdonnees = len(sort_feature)+1
    nbDonneesDepuisMediane = nbdonnees - rangMediane
    resteDivision = nbDonneesDepuisMediane % 2
    if (resteDivision != 0):
        q3 =
sort_feature[(rangMediane+ceil(nbDonneesDepuisMediane/2))-1]
    else:
        valeurMinQ3 =
sort_feature[(rangMediane+(nbDonneesDepuisMediane/2))-1]
        valeurMaxQ3 =
sort_feature[(rangMediane+(nbDonneesDepuisMediane/2))]
        q3 = (valeurMin + ((valeurMax - valeurMin) / 2) +
valeurMax) / 2

    return ([q1, q2, q3])
```

8.3.2 L'interquartile

Alors que l'écart-type mesure la dispersion des données par rapport à la moyenne, l'interquartile mesure la dispersion de données par rapport à la médiane (le 2^e quartile).

Il se calcule en faisant la différence entre le 3^e quartile et le 1^{er} quartile. Dans notre cas, l'interquartile est égal à $14 - 10,75$, soit $3,25$. Plus l'interquartile est important, plus la série est dispersée. Dans notre cas, l'interquartile correspond à l'écart de notes entre les 25 % les plus élevées et les 25 % les plus basses, soit $3,25$ points.

9. Détection de valeurs extrêmes (outliers en anglais)

Outre le fait d'exprimer la dispersion des données autour de la médiane, l'interquartile nous permet également de détecter les valeurs extrêmes de la série d'observations grâce à la méthode de Tuckey.

Les boîtes à moustaches de John Wilder Tuckey

Il existe plusieurs méthodes de détection des valeurs extrêmes d'une série, mais nous avons choisi de vous présenter celle de Tuckey, simple à mettre en œuvre.

■ Remarque

John Wilder Tukey (16 juin 1915 - 26 juillet 2000) est l'un des plus importants statisticiens américains du XX^e siècle. Il a créé et développé de nombreuses méthodes statistiques. Il publia en 1977 son livre le plus diffusé, *Exploratory Data Analysis*, traitant de méthodes d'analyse descriptive et de représentation graphique des données. Il y présente entre autres le principe de la boîte à moustaches (ou diagramme de quartiles), mais aussi les arbres et feuilles (en) (stem-and-leaf), une variante des histogrammes (source Wikipédia).

La méthode consiste à déterminer les valeurs des bornes inférieures et supérieures d'une boîte (appelée aussi boîte à moustaches) comme suit :

- la valeur de la borne inférieure est égale à la valeur du premier quartile - $(1,5 \times 1^{\text{er}} \text{interquartile})$,
- la borne supérieure est quant à elle égale à la valeur du 3e quartile + $(1,5 \times 1^{\text{er}} \text{interquartile})$.

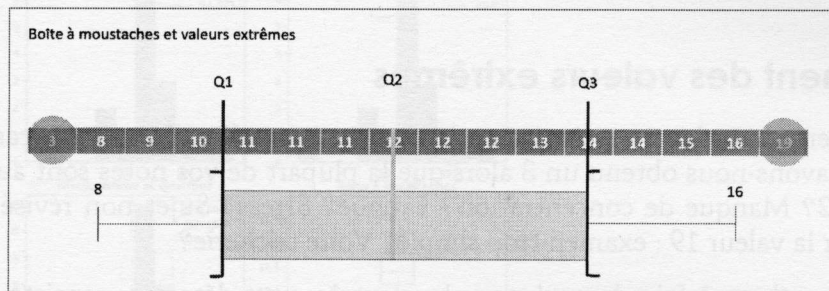
Dans notre cas :

- la valeur de la borne inférieure est égale à 5,125 : $(10 - (1,5 \times 3,25))$,
- la valeur de la borne supérieure est égale à 18,75 : $(14 + (1,5 \times 3,25))$.

Les valeurs de la série d'observations se trouvant à l'extérieur de ces bornes peuvent être considérées comme extrêmes. Dans notre cas, les valeurs 3 et 19 semblent extrêmes.

La figure suivante montre la représentation de la boîte à moustaches pour notre série. On note cependant que les moustaches s'arrêtent aux valeurs 8 et 16, alors que nous venons de calculer les valeurs 5,125 et 18,75 pour chacune des bornes.

Cela vient du fait que les valeurs calculées n'existent pas dans la série d'observations. Nous prenons donc les valeurs les plus proches. On constate également que les valeurs 3 et 19 sont en dehors des bornes de la boîte à moustaches, nous pouvons donc les considérer comme des outliers.



Boîte à moustaches

Remarque

Ne pas confondre valeur extrême et valeur aberrante. Une valeur aberrante est une valeur fausse. Une valeur extrême exprime un évènement dans la série d'observations.

Voici la fonction Python permettant de détecter les valeurs extrêmes :

```
def critereDeTukey(self, premierQuartile, troisiemeQuartile):

    valeursExtremesInferieures = []
    valeursExtremesSuperieures = []
    feature = self.feature.sort_values()
    interquartile = troisiemeQuartile - premierQuartile
    print("Inter-quartile = "+str(interquartile))
    borneInferieure = premierQuartile - (1.5 * interquartile)
    borneSuperieure = troisiemeQuartile + (1.5 * interquartile)

    for valeurObservation in feature:
        if valeurObservation < borneInferieure:
            valeursExtremesInferieures.append(valeurObservation)

        if valeurObservation > borneSuperieure:
            valeursExtremesSuperieures.append(valeurObservation)

    valeursExtremes = valeursExtremesInferieures +
    valeursExtremesSuperieures

    return (valeursExtremes)
```

10. Traitement des valeurs extrêmes

Bien souvent, les valeurs extrêmes expriment des anomalies. Dans notre cas, pourquoi avons-nous obtenu un 3 alors que la plupart de vos notes sont autour de 12? Manque de concentration? Fatigue? Stress? Sujet non révisé? Idem pour la valeur 19 : examen trop simple? Voire tricherie?

La première chose à faire lorsqu'une valeur extrême est détectée, consiste à s'assurer qu'il n'y a pas eu d'erreur de saisie dans les valeurs de la série d'observations.

S'il n'y a pas d'erreur de saisie, peut-on alors expliquer formellement les valeurs extrêmes (appareil de mesure mal réglé...) ? Si oui, la valeur extrême peut être éliminée.

Dans le cas où l'on ne peut pas expliquer clairement la raison de ces valeurs extrêmes, nous pouvons alors proposer des hypothèses qui devront être vérifiées avant toute mise à l'écart des valeurs.

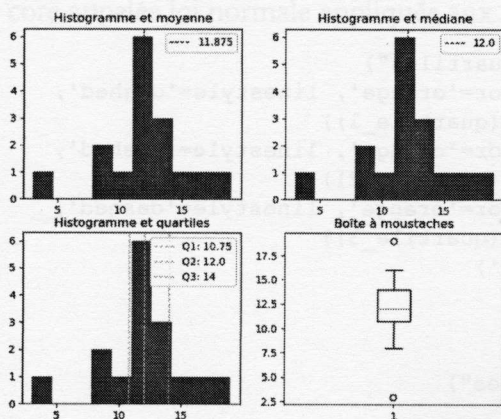
La suppression des valeurs extrêmes n'est pas à prendre à la légère. Supprimer une valeur peut avoir une incidence forte sur la signification, l'interprétation et l'utilisation par la suite des données de la série d'observations.

Cependant, il n'existe pas de formule magique permettant d'affirmer ou non le droit de suppression de la valeur extrême. Il faudra vous fier à votre intuition et au contexte du sujet.

11. Un peu de visualisation graphique

Une des étapes importantes lorsque nous sommes amenés à analyser une série d'observations est d'en réaliser une représentation graphique.

Cela permet notamment de nous rendre compte rapidement des spécificités de celle-ci (dispersion, valeur aberrante) ce qui permet entre autres de valider nos hypothèses de calcul et d'interprétation.



Représentation graphique à l'aide du module Python Matplotlib

98 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

On constate aisément que la valeur 3 peut être considérée comme une valeur extrême, car elle se trouve détachée de l'histogramme général, mais il n'en est pas de même pour la valeur 19. On peut aussi constater que les notes obtenues se situent autour de la moyenne et de la médiane (soit la valeur de 12). La boîte à moustaches montre les valeurs aberrantes.

Pour réaliser la visualisation des données en Python, nous utilisons le module matplotlib utilisé sous l'alias plt.

```
def
visualisation(self, moyenne, mediane, quartile_1, quartile_2, quartile
_3):

    plt.subplot(2, 2, 1)
    plt.hist(self.feature)
    plt.title("Histogramme et moyenne")
    plt.axvline(moyenne, color='red', linestyle='dashed',
linewidth=1, label = str(moyenne))
    plt.legend(loc='upper right')

    plt.subplot(2, 2, 2)
    plt.hist(self.feature)
    plt.title("Histogramme et mediane")
    plt.axvline(mediane, color='green', linestyle='dashed',
linewidth=1, label = str(mediane))
    plt.legend(loc='upper right')

    plt.subplot(2, 2, 3)
    plt.hist(self.feature)
    plt.title("Histogramme et quartiles")
    plt.axvline(quartile_1, color='orange', linestyle='dashed',
linewidth=1, label = "Q1: "+str(quartile_1))
    plt.axvline(quartile_2, color='orange', linestyle='dashed',
linewidth=1, label = "Q2: "+str(quartile_2))
    plt.axvline(quartile_3, color='orange', linestyle='dashed',
linewidth=1, label = "Q3: "+str(quartile_3))
    plt.legend(loc='upper right')

    plt.subplot(2, 2, 4)
    plt.boxplot(self.feature)
    plt.title("Boîte à moustaches")
    plt.show()
```

12. Conclusion sur les données

Malgré une forte dispersion calculée à l'aide de l'écart-type, les notes obtenues se concentrent tout de même autour de la moyenne une fois les valeurs non représentatives écartées. Par conséquent, si nous devons interpréter vos notes, nous dirions que celles-ci se situent autour de 12.

Qu'est-ce que cela signifie? Cela dépend du contexte du problème!

Si nous devons prédire le fait que vous deveniez expert(e) en intelligence artificielle se joue sur les notes obtenues à cette matière, nous devrions étudier les notes obtenues par les autres étudiants et trouver une **corrélation** entre les notes obtenues et le fait de devenir expert(e).

Si nous constatons le fait que la plupart des experts(e)s ont obtenu une valeur moyenne de 12 à leurs examens cela pourrait signifier que vous avez de grandes chances de devenir expert(e). Dans le cas où cette note est de 18, cela minimiserait vos chances.

13. Distribution gaussienne et loi normale

Nous allons quelque peu nous écarter de l'analyse descriptive de nos données pour aborder un point important qui vous sera utile dans la compréhension de certains algorithmes du Machine Learning : la distribution gaussienne ou encore appelée loi normale appliquée aux statistiques et aux probabilités.



Les avis correspondent à nos valeurs observées notées X_i et le nombre de votes est assimilé au nombre de fois où la valeur observée X_i a été votée par les spectateurs. On parle alors de fréquence de choix que l'on note N_i .

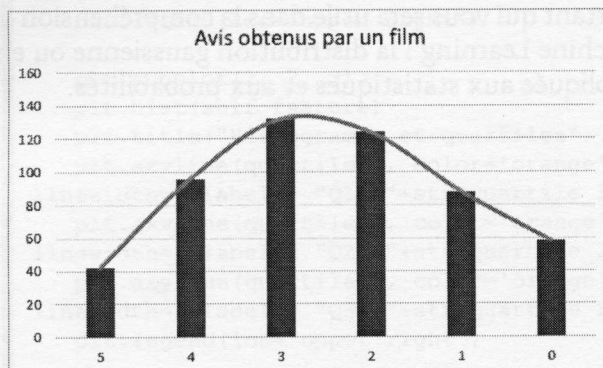
13.1 Un exemple pour faire connaissance

Pour faire connaissance avec la distribution gaussienne, laissons de côté les notes obtenues à l'examen et concentrons-nous sur les critiques de films.

Voici les avis (notés de 0 à 5) obtenus par un film. 5 étant la meilleure note que peut obtenir le film : les fameuses 5 étoiles que l'on retrouve sur l'ensemble des sites de critiques de films.

Avis	Nombre de votants
5	42
4	96
3	132
2	124
1	88
0	58

Si nous réalisons une représentation graphique de ces données, nous obtenons une forme particulière : une cloche.



Courbe de Gauss

Lorsque nous sommes en présence de ce type de graphique, nous pouvons affirmer que la série d'observations suit une loi mathématique appelée loi normale ou loi de Gauss (du nom de Karl Friederich Gauss (1777-1855)).

En statistique et en probabilité, la loi normale permet de représenter beaucoup de phénomènes aléatoires naturels. Lorsqu'une série d'observations obéit à la loi normale, on peut affirmer que :

- 50 % des observations sont au-dessus de la moyenne,
- 50 % des observations sont en dessous de la moyenne,
- 68 % des observations sont comprises dans l'intervalle allant de la moyenne - l'écart type à la moyenne + l'écart type,
- 95 % des observations sont comprises dans l'intervalle allant de la moyenne - 2* l'écart type à la moyenne + 2* l'écart type,
- 99,7 % des observations sont comprises dans l'intervalle allant de la moyenne - 3* l'écart type à la moyenne + 3* l'écart type.

Nous allons à présent réaliser quelques calculs qui nous permettront par la même occasion de voir comment utiliser la notion de fréquence dans les calculs de moyenne et d'écart type.

Avis (Xi)	Nombre de votants (Ni)
5	40
4	99
3	145
2	133
1	96
0	40

Les avis correspondent à nos valeurs observées nommées Xi et le nombre de votants est assimilé au nombre de fois où la valeur observée a été choisie par les spectateurs. On parle alors de fréquence de choix que l'on nomme Ni.

102 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Pour calculer la moyenne de cette série d'observations, il faut réaliser pour chaque observation le produit des avis par le nombre de votants :

Avis (X_i)	Nombre de votants (N_i)	Produits ($N_i * X_i$)
5	40	200
4	99	396
3	145	435
2	133	266
1	96	96
0	40	0

Puis faire la somme de ces produits :

$$200 + 396 + 435 + 266 + 96 + 0 = 1393$$

Et réaliser ensuite la somme des fréquences :

$$40 + 99 + 145 + 133 + 96 + 40 = 553$$

La moyenne se calcule en réalisant le rapport entre ces deux valeurs, soit : $1393/553 = 2,51$.

Passons à présent au calcul de la variance en réalisant pour chaque observation le produit de la fréquence par la différence élevée au carré entre la valeur observée et la moyenne.

Par exemple, pour la première observation nous avons :

$$40 * ((5 - 2,51)^2) = 246,21$$

Ce qui nous donne le tableau suivant :

Avis (Xi)	Nombre de votants (Ni)	Produits (Ni * Xi)	Ni*((Xi-moyenne) ²)
5	40	200	246,21
4	99	396	217,14
3	145	435	33,54
2	133	266	35,82
1	96	96	221,50
0	40	0	253,81

Ce qui nous permet de calculer la variance en faisant la somme de la colonne que nous venons de créer divisée par la somme des fréquences :

$$\text{Variance} = (246,21 + 217,14 + 33,54 + 35,82 + 221,50 + 253,81) / 553 = 1,82$$

Et enfin, nous pouvons terminer par l'écart type en calculant la racine carrée de la variance :

$$\text{Ecart type} = \sqrt{1,82}$$

Ce qui nous donne une valeur de 1,35 pour l'écart type.

Nous vous laissons à présent le soin d'observer la répartition des observations en fonction des écarts entre la moyenne et l'écart type permettant de définir les 68 %, 95 % et 97 % de répartitions.

À titre d'exemple, nous pouvons vérifier que 68 % des observations sont bien comprises dans l'intervalle $[1,3]$. Les bornes de l'intervalle ayant été déterminées par la soustraction de l'écart type à la moyenne pour la borne inférieure et l'ajout de l'écart type à la moyenne pour la borne supérieure. Ce qui nous donne :

- nombre d'observations total : 553
- nombre d'observations comprises entre 1 et 3 = $145 + 133 + 96 = 374$
- soit un pourcentage de $374 / 553 = 67,63 \%$

13.2 Un peu de probabilités

Comme nous venons de l'évoquer, une distribution gaussienne est caractérisée par la moyenne et l'écart type. Ces valeurs sont également utilisées pour calculer des probabilités à partir du jeu d'observations.

Nous avons effectué un relevé de 100 poids d'oranges chez notre vendeur de fruits et légumes, et il s'avère que toutes les mesures suivent une loi normale.

Voici les informations essentielles à considérer à l'issue de nos mesures :

- nombre d'observations : 100
- poids moyen : 200 g
- écart type : 30 g

Quelle est la probabilité qu'une orange prise au hasard fasse moins de 130 grammes?

Tout d'abord, on calcule l'écart par rapport à la moyenne : $130 - 200 = -70$.

On divise ensuite par l'écart type : $-70/30 = -2,3333$.

On lit ensuite la valeur de la probabilité dans la table suivante correspondant à la valeur $2,3333 = 0,9901$.

VALEUR	MOYENNE	ECART	ECART AU CARRE
3	11,875	-8,875	78,765625
19	11,875	7,125	50,765625
10	11,875	-1,875	3,515625
15	11,875	3,125	9,765625
14	11,875	2,125	4,515625
12	11,875	0,125	0,015625
9	11,875	-2,875	8,265625
8	11,875	-3,875	15,015625
11	11,875	-0,875	0,765625
12	11,875	0,125	0,015625
11	11,875	-0,875	0,765625
12	11,875	0,125	0,015625
13	11,875	1,125	1,265625
11	11,875	-0,875	0,765625
14	11,875	2,125	4,515625
16	11,875	4,125	17,015625
SOMME :			195,75
NOMBRE DE VALEURS :			16
VARIANCE :			13,05

Table de probabilité

Ce qui nous permet de calculer la probabilité comme suit : $1 - 0,9901 = 0,0099$, soit 0,99 %, autant dire aucune chance qu'une orange prise au hasard fasse moins de 130 grammes !

Nous n'irons pas plus loin dans l'explication de la loi normale ou distribution gaussienne. Sachez néanmoins que cette distribution sera utilisée dans certains algorithmes d'apprentissage non supervisés sur lesquels nous aurons l'occasion de revenir dans le chapitre suivant.

14. Une classe Python pour vous aider à analyser vos données

Le code Python présent dans chaque partie de ce chapitre fait l'objet de la création d'une classe vous permettant d'analyser vos données. Grâce à cette classe, vous retrouverez l'analyse que nous venons d'effectuer ainsi que la représentation graphique d'une série d'observations.

Vous pouvez télécharger cette classe sur le site de l'éditeur. Son utilisation est assez simple, car il suffit de lui passer une caractéristique d'un DataFrame pour en obtenir son analyse !

Remarque

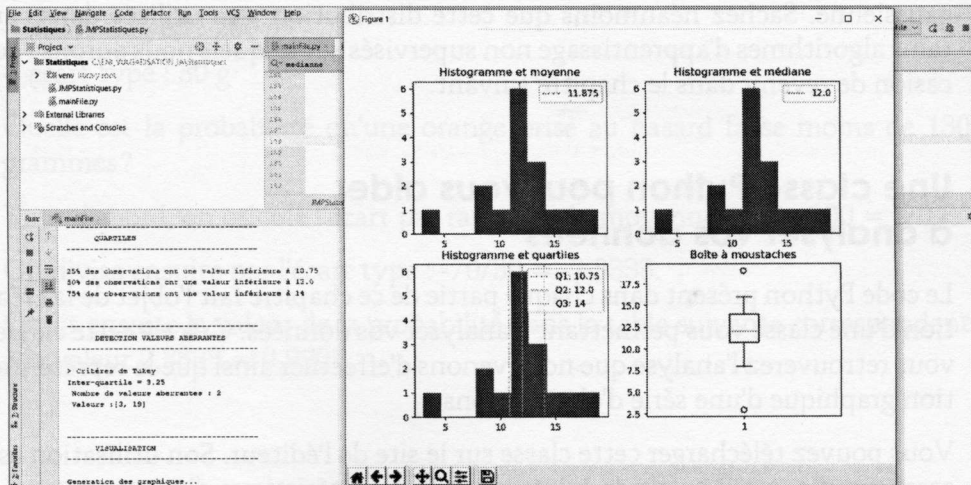
Les classes sont un moyen de réunir des données et des fonctionnalités.
(source Python.org).

Nous aurons l'occasion d'utiliser cette classe dans un prochain chapitre, mais un exemple d'utilisation reprenant les données que nous venons d'utiliser est proposé dans le code fourni. Il convient alors de créer un nouveau projet Python et d'importer les modules nécessaires comme indiqué dans les commentaires :

```
import pandas as pnd
import JMPStatistiques as jmp
import numpy as np

#--- CREATION D'UN DATAFRAME ---
observations =
pnd.DataFrame({'NOTES':np.array([3,19,10,15,14,12,9,8,11,12,11,12,
13,11,14,16])})

#--- ANALYSE D'UNE FEATURE ---
stats = jmp.JMPStatistiques(observations['NOTES'])
stats.analyseFeature()
```



Utilisation de la classe JMPStatistiques

15. Combien d'observations sont nécessaires pour un bon apprentissage ?

En guise de conclusion, nous allons répondre à une question fréquemment posée : "Combien de données sont nécessaires pour un bon apprentissage ?" Ce à quoi nous répondrons : cela dépend du cas d'étude, de la précision de prédiction à réaliser et de l'algorithme utilisé.

Si vous souhaitez une très forte précision dans vos prédictions, vous devez alors proposer à votre algorithme d'apprentissage un panel d'observations le plus large possible en termes de représentativité, ce qui nécessite un grand nombre de données.

Vient ensuite le type d'algorithme utilisé (parfois en lien direct avec le point précédent), certains algorithmes ont besoin de quelques milliers d'observations pour être suffisamment efficaces là où d'autres en ont besoin de plusieurs milliers, notamment dans le cas d'apprentissage dans le domaine de la classification d'image.

Une question complémentaire à se poser est : de quel budget disposons-nous ? En effet, comme vous pourrez le constater au fur et à mesure de votre avancée dans l'aventure de l'intelligence artificielle, les données nécessaires pour réaliser de bons apprentissages se font rares. Et comme ce qui est rare est cher, disposer de données de qualité a un coût.

Comme vous pouvez le constater, il n'existe donc pas de réponse toute faite à la question posée. Si nous nous basons sur le module `SciKit-Learn` de Python permettant d'utiliser les algorithmes dédiés au Machine Learning que nous aurons bientôt l'occasion de découvrir, en dessous de 50 observations, le Machine Learning est impossible.

Chapitre 4

Principaux algorithmes du Machine Learning

1. Ce que nous allons découvrir et les prérequis

Dans le chapitre précédent, nous avons découvert ou redécouvert les fondamentaux de l'analyse statistique descriptive qui, nous le verrons par la pratique, nous permettront de comprendre et de préparer nos données avant l'apprentissage. Nous allons à présent faire connaissance avec les principaux algorithmes du Machine Learning qui vont nous permettre de réaliser cet apprentissage.

Attention, notre objectif en écrivant cet ouvrage est de vulgariser les concepts de l'intelligence artificielle. Par conséquent, **nous n'y aborderons pas les explications théoriques et mathématiques de chaque algorithme d'apprentissage.**

Nous nous contenterons d'une explication la plus explicite possible illustrée par un ou plusieurs exemples le cas échéant. Si nous devons faire un parallèle avec le monde du bricolage, nous allons vous présenter les différents outils à utiliser en fonction du travail à réaliser, mais nous ne vous expliquerons pas comment ils ont été fabriqués.

Nous vous conseillons de considérer ce chapitre comme un aide-mémoire dans lequel vous pourrez venir vous référer au fur et à mesure de votre lecture afin de comprendre pourquoi nous utilisons tel ou tel algorithme et en comprendre son fonctionnement dans les grandes lignes.

■ Remarque

Prérequis nécessaires pour bien aborder ce chapitre : avoir lu le chapitre Des statistiques pour comprendre les données

2. Supervisé ou non supervisé ? Régression ou classification ?

Réaliser un apprentissage supervisé consiste à fournir à la machine des données étiquetées (labellisées) et propices à l'apprentissage. C'est-à-dire que nous allons analyser et préparer les données et leur donner une signification. C'est à partir de cette signification que la machine va réaliser son apprentissage. L'objectif étant d'indiquer à la machine que pour une série de données et pour une observation précise, la valeur à prédire est un chat, un chien ou bien une autre valeur.

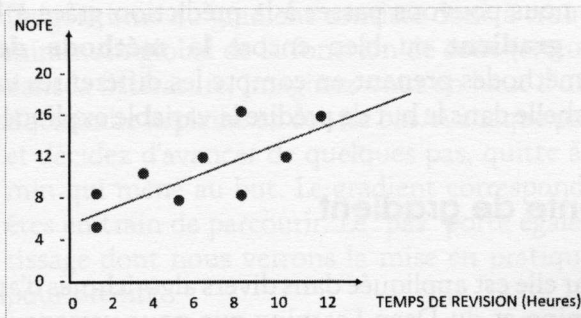
Lorsqu'il s'agit de prédire une valeur, nous parlerons alors de **régression**, dans le cas contraire, nous parlerons de **classification**. Prédire le pourcentage de réussite d'une équipe de football lors d'un match est une régression, prédire que la photo affichée est un chat ou un chien est une classification.

3. Les algorithmes d'apprentissage supervisés pour la régression (prédiction de valeurs)

3.1 La régression linéaire univariée (linear regression)

Cet algorithme cherche à établir, sous forme d'une droite, une relation entre une variable expliquée et une variable explicative. Par exemple, prédire une note à un examen (variable expliquée) en fonction du nombre d'heures de révisions (variable explicative).

En d'autres termes, les données d'une série d'observations sont représentées sous forme d'un nuage de points et l'on cherche à trouver une droite passant au plus près de ces points.



Régression linéaire univariée

Ainsi, connaissant le nombre d'heures de révisions, il nous est possible de prédire approximativement la note que l'on obtiendra au prochain examen.

3.2 La régression linéaire multiple (Multiple Linear Regression-MLR)

Là où nous utilisons une seule variable explicative pour expliquer une autre variable (une note en fonction d'un temps de révision), dans la régression linéaire multivariée nous allons utiliser plusieurs variables explicatives.

Par exemple, nous allons chercher à prédire le temps que va mettre un cycliste pour remporter une étape du tour de France, en fonction de son âge, du temps qu'il a réalisé à la précédente étape, de son classement dans le peloton...

Une étape importante lors de l'utilisation de multiples variables explicatives est leur **normalisation** (mise à l'échelle). Dans notre exemple, le temps réalisé en minutes lors de la précédente étape peut éventuellement varier entre 160 à 200, la position dans le peloton entre 1 et 80 en fonction du nombre de participants au tour de France. Nous ne sommes donc pas sur la même échelle pour chacune des variables explicatives (160 à 200 vs 1 à 80).

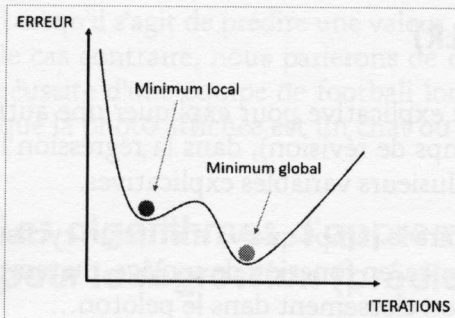
La mise à l'échelle (*scaling*) va donc consister à faire en sorte que la moyenne de chaque série d'observations soit égale à 0, que la variance et l'écart-type soient égaux à 1. Cette méthode est également appelée centrage de réduction.

Une fois cette étape réalisée, nous pouvons passer à la prédiction grâce à la **méthode de descente de gradient** ou bien encore la **méthode des moindres carrés**. Ces deux méthodes prenant en compte les différentes variables explicatives mises à l'échelle dans le but de prédire la variable expliquée.

3.3 La méthode de descente de gradient

Cette notion est essentielle, car elle est appliquée dans divers algorithmes d'apprentissage du Machine Learning et du Deep Learning que nous verrons un peu plus loin dans cet ouvrage.

Lorsqu'un système est en phase d'apprentissage, il commet des erreurs. Le taux d'erreur diminue au fur et à mesure de l'apprentissage, mais il se peut qu'à un moment donné l'erreur augmente pour à nouveau rediminuer et atteindre un niveau d'erreur plus bas que le précédent qui est le niveau optimal d'apprentissage.



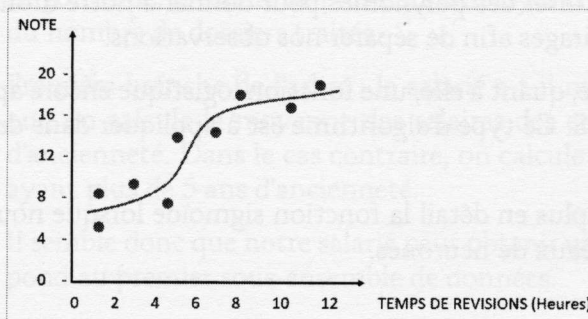
La descente de gradient

Sur la figure précédente, on constate qu'en début d'apprentissage, l'erreur diminue progressivement pour ensuite remonter. Nous aurions donc tendance à dire que le niveau optimal d'apprentissage a été atteint puisque de nouvelles erreurs apparaissent. Cependant, on peut s'apercevoir qu'après de nouvelles itérations d'apprentissage, l'erreur continue de diminuer pour atteindre un niveau plus bas que le précédent appelé minimum global ! Le niveau optimal d'apprentissage n'était donc pas atteint.

L'algorithme du gradient consiste donc à trouver par itérations successives le minimum global de la fonction de coût (erreur). Par analogie souvent reprise dans la littérature, imaginez-vous en haut d'une montagne avec pour objectif d'atteindre la plaine en contre bas. À chaque pas, vous analysez votre situation et décidez d'avancer de quelques pas, quitte à remonter pour prendre le chemin qui mène au but. Le gradient correspondant à la pente du sol que vous êtes en train de parcourir. Le "pas" porte également le nom de taux d'apprentissage dont nous verrons la mise en pratique dans le chapitre Un neurone pour prédire.

3.4 Régression polynomiale (polynomial regression)

Il est parfois difficile de trouver une droite pouvant passer parmi les points de la série d'observations de façon optimale. Cependant, il est parfois possible de trouver un lien entre les variables à l'aide d'une courbe. C'est ce que permet la régression polynomiale en ajoutant des plis à la courbe à l'aide d'éléments appelés polynômes.



Régression polynomiale

3.4.1 Monôme et polynôme

Un monôme est une expression mathématique s'exprimant sous cette forme :

$$\alpha x^n$$

Où

- α (alpha) est un nombre réel ou complexe appelé coefficient du monôme.
- n est un entier naturel représentant le degré du monôme.

Ainsi, $5x^2$ est un monôme de coefficient 5 et de degré 2.

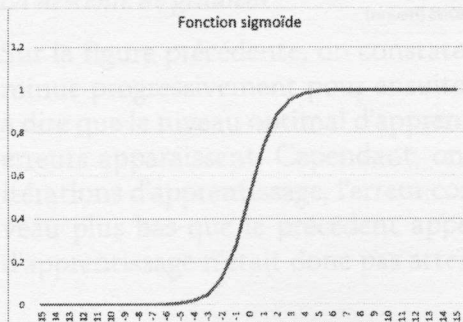
Un polynôme est une somme de monômes. On peut donc dire que $5x^2 + 2x$ est un polynôme.

3.5 Régression logistique

Comme nous venons de le voir, lorsque les données ne sont pas linéairement séparables, il est possible d'utiliser des polynômes pour donner à notre droite la possibilité de réaliser des virages afin de séparer nos observations.

La régression logistique utilise, quant à elle, une fonction logistique encore appelée sigmoïde ou courbe en S. Ce type d'algorithme est à appliquer dans des problèmes de classification.

À noter que nous croiserons plus en détail la fonction sigmoïde lorsque nous traiterons en pratique les réseaux de neurones.



Fonction sigmoïde ou courbe en S

3.6 Arbre de décision (decision tree)

Cet outil d'aide à la décision ou d'exploration de données permet de représenter un ensemble de choix sous la forme graphique d'un arbre.

Il s'agit donc d'une suite de tests réalisés dans le but de prédire un résultat, on retrouve les décisions à chaque extrémité des branches de l'arbre.

Pour chaque test, on partage l'ensemble des données en sous-ensembles et on effectue la moyenne des valeurs de ces sous-ensembles en guise de prédiction.

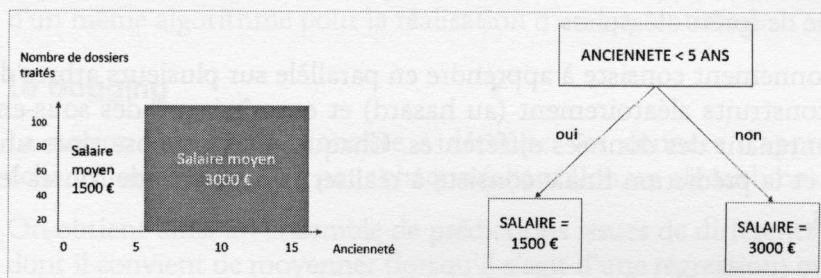
Prenons l'exemple de la prédiction du salaire d'un employé dans une entreprise de vente en ligne. Chaque vente concrétisée fait l'objet d'un dossier. Le salaire de l'employé est calculé en fonction de son ancienneté dans l'entreprise et du nombre de dossiers qu'il traite dans l'année.

Quel sera le salaire de Monsieur X si son ancienneté est de 3 ans et qu'il traite 40 dossiers durant l'année?

Comme nous sommes dans un apprentissage supervisé, nous avons fourni à l'algorithme, le salaire de plusieurs employés en fonction de leur ancienneté et du nombre de dossiers traités.

Première branche de l'arbre : le salarié a-t-il moins de 5 ans d'ancienneté? Si oui, on calcule la moyenne des salaires des employés ayant moins de 5 ans d'ancienneté. Dans le cas contraire, on calcule le salaire moyen des employés ayant plus de 5 ans d'ancienneté.

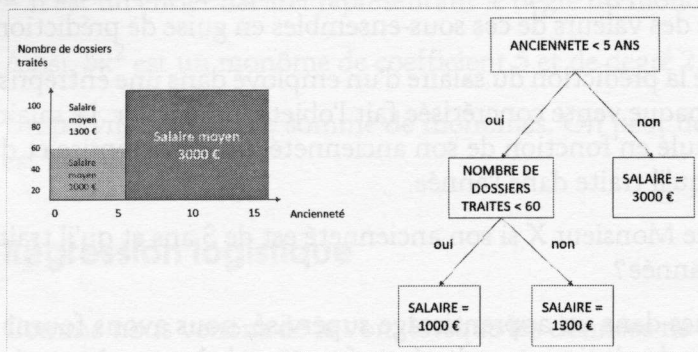
Il semble donc que notre salarié peut obtenir un salaire de 1500 €, car il correspond au premier sous-ensemble de données.



Arbre de décision

Sur le même principe, ajoutons le critère du nombre de dossiers traités. Cet ajout de critères complémentaires divise en deux sous-ensembles l'ensemble des résultats précédents. Nous avons à présent un sous-ensemble ayant un salaire moyen de 1300 € pour plus de 60 dossiers traités et un autre ayant un salaire de 1000 € pour moins de 60 dossiers traités.

À la lecture de notre arbre de décision illustré par la figure précédente, Monsieur X peut donc prétendre à un salaire de 1000 €.



Arbre de décision

Simple à interpréter et rapides à entraîner, les arbres de décisions sont utilisés aussi bien dans des apprentissages supervisés de régression que de classification.

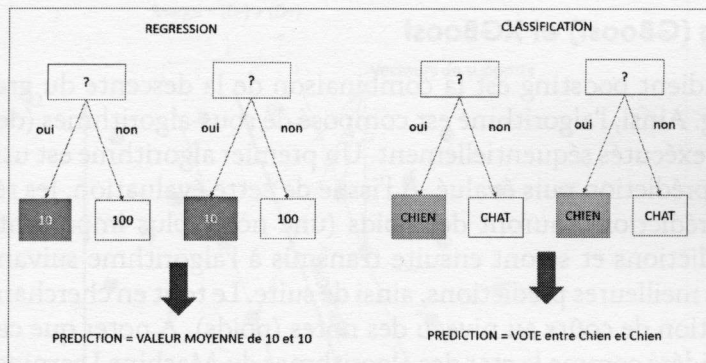
3.7 Forêts aléatoires (Random Forest)

Comme chacun le sait, une forêt est constituée d'arbres. Il en va de même pour l'algorithme de forêts aléatoires.

Son fonctionnement consiste à apprendre en parallèle sur plusieurs arbres de décisions construits aléatoirement (au hasard) et entraînés sur des sous-ensembles contenant des données différentes. Chaque arbre propose alors une prédiction et la prédiction finale consiste à réaliser la moyenne de toutes les prédictions.

À noter que cet algorithme est aussi utilisé dans le cadre de la classification. Au lieu de faire une moyenne des prédictions, un vote est réalisé parmi les propositions de classification.

La complexité dans l'utilisation de cet algorithme est de trouver le bon nombre d'arbres à utiliser pouvant aller jusqu'à plusieurs centaines!



Forêts aléatoires

3.8 Agrégation de modèle : le bagging, le boosting et le Gradient boosting

Découvrons à présent une notion que vous rencontrerez sans doute tout au long de votre parcours dédié au Machine Learning et dont nous avons déjà évoqué l'existence lorsque nous avons évoqué l'algorithme du Random Forest : l'agrégation de modèles. C'est-à-dire l'utilisation de plusieurs modèles au sein d'un même algorithme pour la réalisation d'un apprentissage.

3.8.1 Le bagging

La notion de bagging consiste à découper les données d'apprentissage en échantillons et d'utiliser pour chaque échantillon un algorithme différent.

On obtient ainsi un ensemble de prédictions issues de différents algorithmes dont il convient de moyenner (lorsqu'il s'agit d'une régression) ou de faire voter (pour une classification).

3.8.2 Le boosting

Comme le bagging, il s'agit là encore d'utiliser divers algorithmes pour réaliser une prédiction. Les différents algorithmes sont notés selon leur prédiction. Plus l'algorithme prédit une bonne valeur, plus il obtient une bonne note. L'un des algorithmes de boosting fonctionnant sur ce principe est Adaboost.

3.8.3 Gradient Boosting (GBoost) et XGBoost

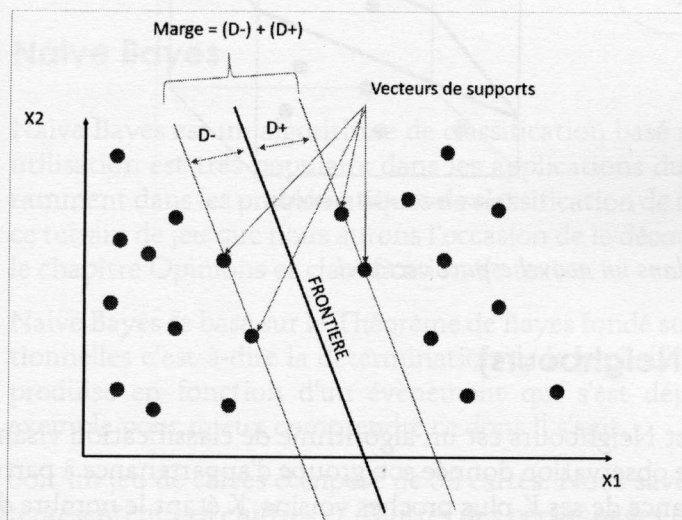
L'algorithme du gradient boosting est la combinaison de la descente du gradient et du boosting. Ainsi, l'algorithme est composé de sous-algorithmes (des arbres de décisions) exécutés séquentiellement. Un premier algorithme est utilisé pour réaliser la prédiction puis évalué. À l'issue de cette évaluation, les résidus (mauvaises prédictions) auront des poids (une note) plus importants que les bonnes prédictions et seront ensuite transmis à l'algorithme suivant chargé de réaliser de meilleures prédictions, ainsi de suite. Le tout en cherchant à minimiser la fonction de coûts au niveau des notes (poids). À noter que cet algorithme est considéré comme la star des algorithmes du Machine Learning, mais son paramétrage complexe ainsi que son temps d'exécution sont le revers de la médaille. On note cependant l'existence de l'algorithme XGBoost (Extreme Gradient Boosting) qui offre la possibilité d'exécuter les sous-algorithmes en parallèle et offre également la possibilité d'utiliser d'autres algorithmes que les arbres de décisions.

3.9 Machine à vecteurs de support (SVM)

SVM est un algorithme puissant utilisé aussi bien dans les cas de classification que de régression. Son objectif est de déterminer une frontière afin de séparer les observations en groupes distincts tout en maximisant la marge de séparation.

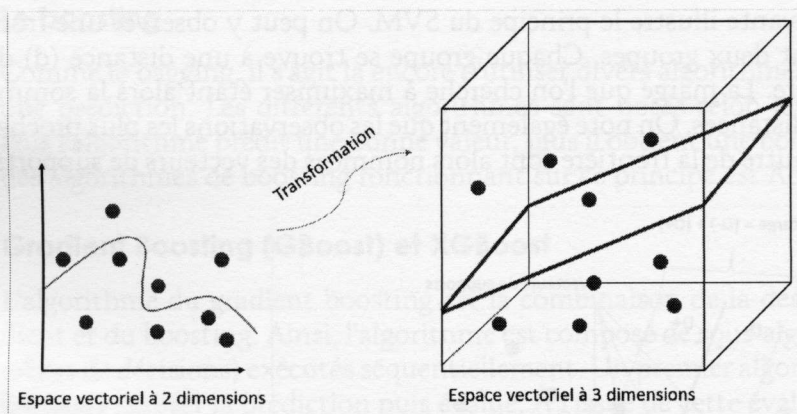
Les observations les plus proches de la frontière sont appelées des vecteurs de supports.

La figure suivante illustre le principe du SVM. On peut y observer une frontière séparant deux groupes. Chaque groupe se trouve à une distance (d) de cette frontière. La marge que l'on cherche à maximiser étant alors la somme de ces deux distances. On note également que les observations les plus proches de part et d'autre de la frontière sont alors nommées des vecteurs de supports.



Machine à vecteurs de supports

Ce type d'algorithme est particulièrement efficace lorsque les données sont linéairement séparables (séparables par une droite), ce qui en pratique est rarement le cas. Une technique consiste donc à projeter les données dans un espace vectoriel de plus grande dimension à l'aide d'un élément appelé noyau, permettant alors une création de frontière.

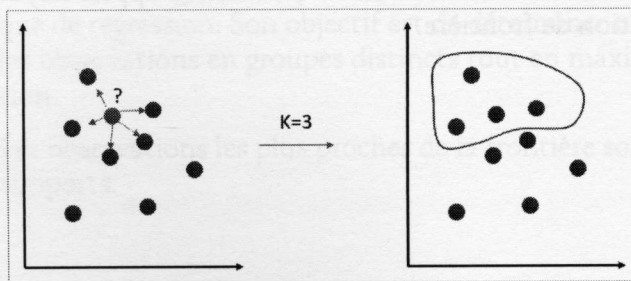


Projection des données dans un nouvel espace vectoriel

3.10 KNN (K-Nearest Neighbours)

L'algorithme k-Nearest Neighbours est un algorithme de classification visant à déterminer pour une observation donnée son groupe d'appartenance à partir du groupe d'appartenance de ses K plus proches voisins. K étant le nombre de voisins à considérer.

La figure suivante illustre ce principe. Imaginons une observation pour laquelle nous devons définir son groupe d'appartenance. Si nous choisissons pour paramètre $K=3$, cela signifie que le groupe choisi sera celui des trois plus proches voisins ayant le même groupe d'appartenance.



Algorithme des K plus proches voisins

Le coût en calcul est assez important, car pour une observation donnée, il faut déterminer les distances par rapport aux autres données. Qui plus est, le fait de devoir calculer ces distances nécessite de garder en mémoire toutes les observations d'apprentissage. Cet algorithme est donc performant sur un petit nombre d'observations.

3.11 Naive Bayes

Naive Bayes est un algorithme de classification basé sur les probabilités. Son utilisation est très populaire dans les applications du Machine Learning notamment dans les problématiques de classification de texte. C'est d'ailleurs sur ce terrain de jeu que nous aurons l'occasion de le découvrir plus en détail dans le chapitre Opinions et classification de textes.

Naive Bayes se base sur le Théorème de Bayes fondé sur les probabilités conditionnelles c'est-à-dire la détermination de la probabilité qu'un évènement se produise en fonction d'un évènement qui s'est déjà produit. Prenons un exemple pour mieux comprendre ce dont il s'agit.

Soit un jeu de cartes composé de 52 cartes. Nous savons que certaines cartes représentent des chiffres et d'autres des personnages (Roi, Reine, Valet).

Essayons de déterminer la probabilité qu'une carte tirée au hasard soit une reine, sachant que la carte tirée est un personnage.

- L'élément connu, noté A, est le fait que la carte soit un personnage.
- L'élément à prédire, noté B, est le fait que la carte soit une reine.

Le théorème de Bayes s'exprime sous la formule suivante :

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Soit pour notre cas :

```
P(la carte est un personnage ET la carte est une reine) = (P(la
carte est une reine ET la carte est un personnage) * P(la carte
est un personnage)) / (P(la carte est une reine))
```

Nous savons par expérience que :

Il n'y a que 4 reines dans un jeu de cartes, soit 4 chances sur 52 que la carte tirée soit une reine. La probabilité que la carte soit une reine se traduit comme suit :

$$P(B) = P(\text{La carte est une reine}) = \frac{4}{52}$$

Toutes les cartes ayant une reine sont des cartes personnage. La probabilité que la carte soit un personnage ET une reine est de 100 % se traduisant comme suit :

$$P(B|A) = P(\text{La carte est un personnage} | \text{La carte est une reine}) = 1$$

Il y a trois possibilités de personnages (roi, reine, valet) pour quatre couleurs (pique, trèfle, cœur et carreau), ce qui nous donne la probabilité que la carte soit un personnage suivante :

$$P(A) = P(\text{La carte est un personnage}) = \frac{3 * 4}{52} = \frac{12}{52}$$

Il ne nous reste plus qu'à remplacer ces éléments dans la formule du théorème de Bayes :

$$P(\text{La carte est une reine ET la carte est un personnage}) = P(A|B)$$

$$P(A|B) = \frac{1 * \frac{12}{52}}{\frac{4}{52}} = 1/3$$

La probabilité que la carte tirée au hasard soit une reine alors que nous savons que la carte est un personnage est de 1/3

Par ce petit exemple, nous avons illustré le principe de fonctionnement de l'algorithme Naive Bayes. Ce type d'algorithme est utilisé dans la classification de texte et son application la plus connue est la classification des emails en tant que Spam ou non-Spam.

■ Remarque

Le théorème de Naive Bayes s'appuie sur une hypothèse forte que chaque variable est indépendante, d'où le terme de naive. Dans le cadre d'une classification de texte à l'aide de l'algorithme de Naive Bayes, celle-ci se base sur le nombre d'occurrences d'un mot dans une phrase pour en déterminer sa catégorie. Chaque mot est alors pris de façon indépendante dans l'analyse.

4. Les algorithmes pour les apprentissages non supervisés

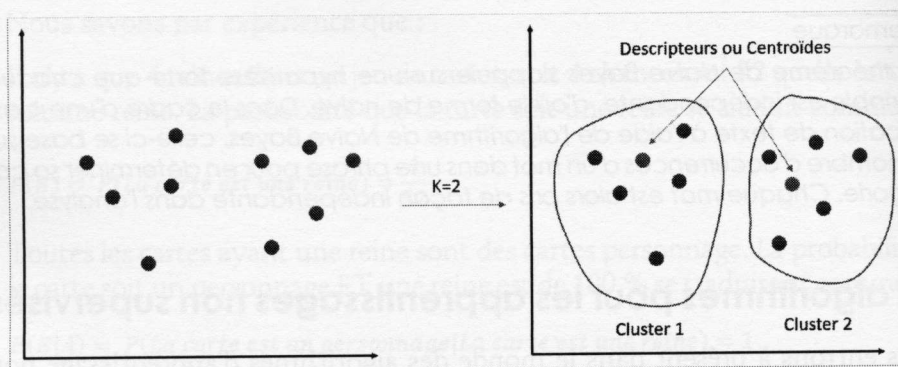
Nous entrons à présent dans le monde des algorithmes d'apprentissage non supervisés. C'est-à-dire que lors de la phase d'apprentissage, nous n'indiquons pas à la machine si l'observation étudiée appartient à un groupe précis comme nous le faisons lors d'un apprentissage supervisé. Charge donc à la machine de déterminer ce groupe d'appartenance (appelé cluster) toute seule.

4.1 K-Moyennes (KMeans)

L'algorithme des K-Moyennes (K-means) est l'un des algorithmes de clustering les plus utilisés.

Son principe de fonctionnement est relativement simple, car après avoir indiqué à l'algorithme le nombre de clusters à trouver, celui-ci tente par itérations successives de déterminer des centroïdes (un par cluster) autour desquels il est possible de regrouper les données. Ces regroupements s'effectuant en calculant la distance de chaque observation par rapport à un point central de regroupement appelé centroïde et permettant ainsi de classer les observations en plusieurs groupes de façon automatique.

Dans la figure suivante, nous avons indiqué à l'algorithme de classer les observations en deux clusters. Il y est parvenu en trouvant de façon automatique deux centroïdes permettant de diviser les observations en deux groupes distincts.



Algorithme des K-Moyennes

4.2 Mean-shift

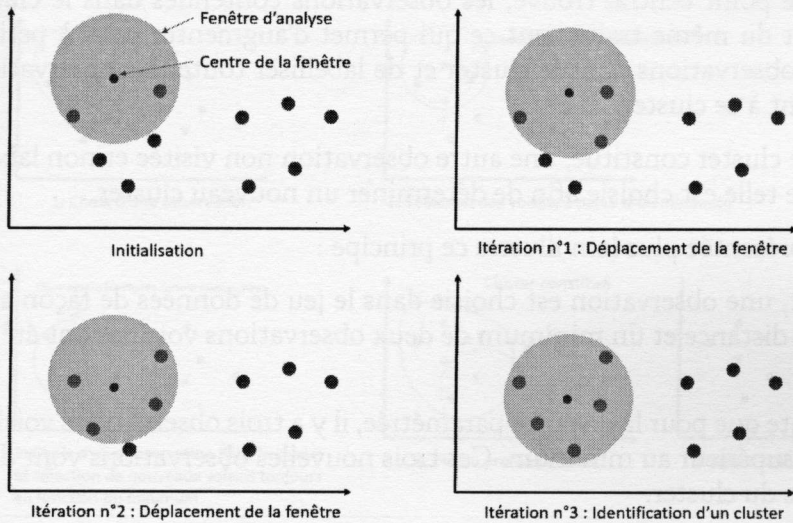
Mean-shift est un algorithme basé sur une notion de "fenêtre coulissante" parcourant le jeu d'observations à la recherche de zones ayant une densité d'observations concentrée autour du centre de la fenêtre que l'on nommera centroïde.

Voici son fonctionnement dans un jeu d'observations :

On définit une fenêtre circulaire ayant pour centre un point choisi au hasard et un rayon d'analyse précisé, appelé Kernel. L'algorithme analyse ensuite les observations contenues dans son rayon d'analyse et déplace la fenêtre vers des régions de densité plus élevée. Ce déplacement se réalise en positionnant le point central de la fenêtre vers la moyenne des points qu'elle contient.

Ce processus est répété jusqu'à ce que le nombre d'observations (la densité) contenues dans la fenêtre d'analyse soit fixe, définissant ainsi un cluster.

La principale différence avec l'algorithme du K-Mean vu précédemment est que nous n'avons pas besoin d'indiquer à la machine combien de clusters elle doit trouver. Ces groupes étant créés de façon automatique par la notion de densité.



Algorithme Mean-shift

4.3 DBSCAN (Density Based Spatial Clustering of Application with Noise)

Tout comme l'algorithme Mean-shift, DBSCAN utilise également la notion de densité d'observations pour déterminer les différents clusters d'un jeu d'observations.

Son analyse commence par le choix d'une observation de façon arbitraire dans le jeu d'observations. DBSCAN extrait ensuite les observations voisines de cette observation en fonction d'une distance définie. Si le nombre d'observations voisines correspond à un minimum préalablement défini, il considère alors l'observation choisie en première étape comme point central d'un nouveau cluster. Dans le cas contraire, il considère les observations voisines comme des "bruits". Enfin, l'observation choisie est considérée comme étant "visitée" et labellisée comme telle, et ce qu'elle soit considérée comme point central d'un cluster ou non.

Une fois le point central trouvé, les observations contenues dans le cluster font l'objet du même traitement ce qui permet d'augmenter petit à petit le nombre d'observations dans le cluster et de labelliser toutes les observations appartenant à ce cluster.

Une fois le cluster constitué, une autre observation non visitée et non labellisée comme telle est choisie afin de déterminer un nouveau cluster.

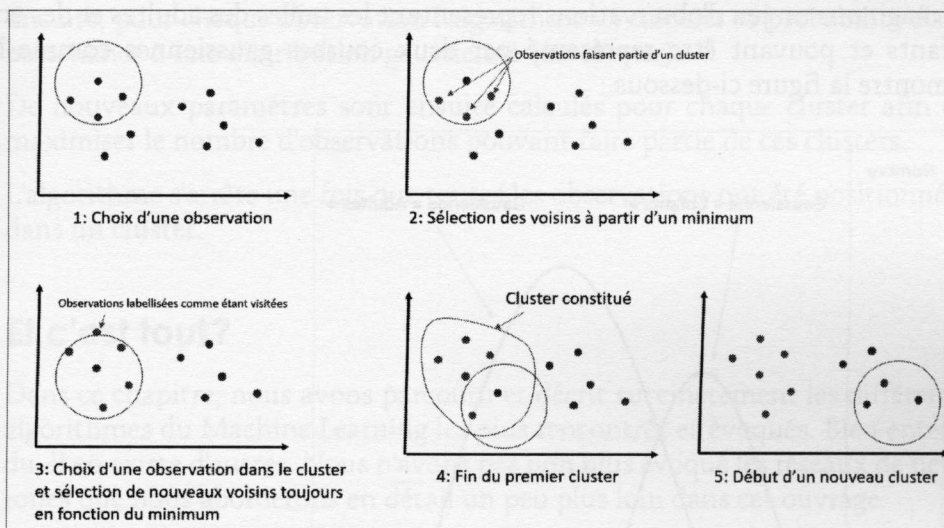
La figure présentée plus loin illustre ce principe :

En étape 1, une observation est choisie dans le jeu de données de façon aléatoire. Une distance et un minimum de deux observations voisines ont été paramétrés.

On constate que pour la distance paramétrée, il y a trois observations voisines ce qui est supérieur au minimum. Ces trois nouvelles observations vont donc faire partie du cluster.

L'observation choisie au départ est labellisée comme "visitée" et une observation faisant partie du cluster est choisie à son tour. On constate que pour la distance donnée il existe encore des observations voisines candidates qui feront partie du cluster. La nouvelle observation est à son tour labellisée comme visitée et une autre observation faisant partie du cluster est alors choisie pour réitérer le processus. À noter que toutes les observations du cluster doivent être analysées par l'algorithme afin d'être considérées comme visitées et appartenant à ce cluster.

On constate en étape 4 que toutes les observations du cluster ont été labellisées comme visitées et qu'il n'est plus possible de trouver d'autres observations candidates au cluster, car le nombre de voisins n'est pas atteint. L'algorithme choisit donc une autre observation non labellisée et recommence le processus afin de déterminer un autre cluster.



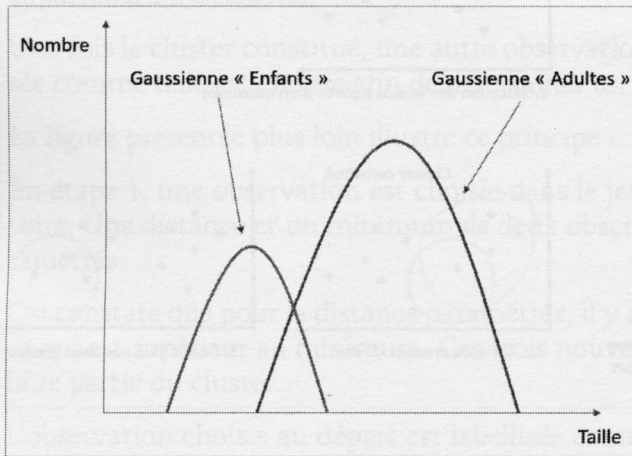
Algorithme DBSCAN

4.4 Mélange gaussien (Gaussian Mixture Models (GMM))

Souvenez-vous, dans le chapitre précédent, nous vous avons présenté la loi normale encore appelée distribution gaussienne représentée graphiquement sous la forme d'une cloche.

Voyons à présent comment cette distribution gaussienne est utilisée dans l'apprentissage non supervisé.

Imaginons un jeu d'observations représentant les tailles des adultes et des enfants et pouvant être représenté par deux courbes gaussiennes comme le montre la figure ci-dessous :



Tailles des adultes et des enfants

L'objectif de l'algorithme GMM va être de déterminer pour une observation donnée son appartenance à une distribution gaussienne. Chaque distribution gaussienne trouvée par l'algorithme fera office de cluster dont le centroïde sera la moyenne de la distribution.

La première étape va être de spécifier à l'algorithme le nombre de clusters à trouver.

L'algorithme initialise ensuite de façon aléatoire pour chaque cluster à trouver une moyenne et un écart type qui, rappelons-le, sont les éléments principaux de la distribution gaussienne.

Ensuite, pour chaque observation l'algorithme va calculer sa probabilité d'appartenance à une distribution gaussienne (un cluster).

De nouveaux paramètres sont ensuite calculés pour chaque cluster afin de maximiser le nombre d'observations pouvant faire partie de ces clusters.

L'algorithme s'arrête une fois que toutes les observations ont été positionnées dans un cluster.

5. Et c'est tout?

Dans ce chapitre, nous avons parcouru et décrit succinctement les différents algorithmes du Machine Learning les plus rencontrés et évoqués. Bien entendu, il en existe d'autres. Nous n'avons pas non plus évoqué les réseaux de neurones que nous aborderons en détail un peu plus loin dans cet ouvrage.

Ce chapitre vous permet d'avoir une vue d'ensemble des algorithmes, leur fonctionnement et leur cas d'utilisation en fonction du problème que vous aurez à traiter : régression ou classification dans le cadre d'un apprentissage supervisé et clustering dans le cas d'apprentissages non supervisés. Le tableau ci-dessous reprend les différents algorithmes que nous avons abordés. Libre à vous de le compléter lors de vos futures découvertes et essais d'algorithmes.

Algorithme	Apprentissage supervisé - régression	Apprentissage supervisé - catégorisation	Apprentissage non supervisé
Régression linéaire univariée	X		
Régression linéaire multiple	X		
Régression polynomiale	X		
Régression logistique		X	
Arbres de décision	X	X	
Forêt aléatoire	X	X	

130 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Algorithme	Apprentissage supervisé - regression	Apprentissage supervisé - catégorisation	Apprentissage non supervisé
Machine à vecteurs de support		X	
KNN		X	
Naive Bayes		X	
K-moyennes			X
Mean-shift			X
DBSCAN			X
Mélange gaussien			X

La partie théorique et présentation est à présent terminée. Nous allons dans le chapitre suivant démarrer notre première étude de cas du Machine Learning à travers un exemple assez étonnant utilisant l'univers des Pokemons.

Chapitre 5

Machine Learning et Pokémons : première partie

1. Ce que nous allons découvrir et les prérequis

Dans ce chapitre, nous allons découvrir à travers un cas pratique les premières étapes pour mener à bien un projet de Machine Learning.

L'exemple que nous allons traiter est un cas d'école et il y est souvent fait référence dans la littérature avec plus ou moins de facilités de compréhension pour les néophytes en matière d'analyse de données. Malgré cette récurrence d'explications, nous avons choisi de vous le présenter, car il permet d'aborder un large panel de techniques propres au Machine Learning et à l'analyse de données. Bien souvent présenté sous un angle scientifique et mathématique, nous allons lui apporter une description vulgarisée vous permettant de bien comprendre les rouages du Machine Learning.

■ Remarque

Prérequis nécessaires pour bien aborder ce chapitre : avoir lu les chapitres Les fondamentaux du langage Python, Des statistiques pour comprendre les données et Principaux algorithmes du Machine Learning.

2. L'univers des Pokémons

Un Pokémon est un animal issu du monde des jeux vidéo. Chaque Pokémon possède des caractéristiques qui lui sont propres, à savoir son type (Pokémon d'herbe, Pokémon de feu...) lui donnant alors des facultés spéciales. Ainsi, le Pokémon nommé Dracofeu, de type Feu est capable de cracher du feu. De plus, on note l'existence de Pokémons légendaires très rares et très puissants faisant référence à un mythe en relation avec la création et l'organisation du monde (Pokémon diamant, Pokémon platine...).

Les Pokémons sont également de différentes générations faisant référence à l'évolution de l'univers des Pokémons. Les Pokémons de première génération sont ceux ayant pris naissance au tout début de la création de l'univers (création de 151 Pokémons) pour arriver à nos jours à la huitième génération avec plus de 800 Pokémons!

Dans cet univers viennent ensuite les dresseurs de Pokémons. Les dresseurs les collectionnent et les élèvent dans le but de les faire combattre entre eux jusqu'au KO. Lors d'un combat de Pokémons, opposant deux dresseurs, l'un choisit l'animal qui sera le premier à entrer dans l'arène. En fonction de ce premier Pokémon, le second dresseur choisit un animal dans sa collection qui sera en mesure de battre le premier Pokémon. Si le premier Pokémon présent dans l'arène est de type feu, il y a de fortes chances que le Pokémon choisi par l'adversaire soit de type eau afin de contrer les attaques du premier.

Cela nécessite pour le dresseur d'avoir une bonne connaissance de chaque Pokémon pour choisir celui qui pourra devenir vainqueur du combat. Pour l'aider dans son choix, les caractéristiques des Pokémons sont répertoriées dans un document appelé Pokédex.

3. Notre mission : choisir le bon Pokémon!

En tant que dresseurs de Pokémon du 21^e siècle, nous allons faire appel au Machine Learning pour nous aider à faire le bon choix de Pokémon et ainsi gagner nos combats. Pour cela, nous allons nous appuyer sur des données en provenance du Pokédex, mais aussi sur des données issues des différents combats afin de trouver le bon modèle de prédiction qui pourra alors nous assurer une victoire lors de chaque bataille!

4. Des données pour un apprentissage supervisé

4.1 Des données basées sur l'expérience

Le Machine Learning est basé sur l'utilisation de données afin de permettre à notre ordinateur d'apprendre et de pouvoir réaliser des prédictions. Ces données doivent être en relation avec la mission qui nous est confiée et basée sur l'expérience.

Dans notre cas, l'expérience consiste à connaître les issues de combats de Pokémon.

4.2 Disposer d'un grand nombre de données d'apprentissage

Une machine n'est pas capable d'apprendre sur un petit jeu de données, car elle doit pouvoir étudier toutes les possibilités pour réaliser ses prédictions. Par conséquent, plus le nombre de cas d'études pour résoudre un problème est important, plus les prédictions seront précises.

4.3 Des données d'apprentissage et des données de tests

Tout comme l'être humain, il est nécessaire de valider l'apprentissage de la machine afin de pouvoir corriger les écarts d'apprentissage (appelés biais) et ajuster ou modifier le modèle d'apprentissage. Pour cela, nous avons besoin de données d'apprentissage et de données de tests.

5. Les étapes à réaliser pour mener à bien un projet de Machine Learning

Mener à bien un projet de Machine Learning consiste à réaliser six étapes consécutives :

- 1 - Définition du problème à résoudre
- 2 - Acquisition des données d'apprentissages et de tests
- 3 - Préparer et nettoyer les données
- 4 - Analyser, explorer les données
- 5 - Choisir un modèle d'apprentissage
- 6 - Visualiser les résultats, et ajuster ou modifier le modèle d'apprentissage

La phase de préparation des données est la plus importante dans un projet de Machine Learning, car en tant qu'humains, nous devons essayer de trouver les données les plus intéressantes qui nous permettront de répondre au problème donné.

Bien plus qu'une simple analyse des données, il faut déterminer comment il nous est possible de résoudre manuellement le problème, à partir des informations dont nous disposons, avant de le confier à la machine.

Ainsi, un même jeu de données peut être exploité différemment en fonction du problème donné.

5.1 Création et configuration d'un nouveau projet Python

Avant d'aller plus loin, nous devons créer un nouveau projet Python.

■ Pour ce faire, il convient d'ouvrir l'éditeur PyCharm puis dans le menu **file**, de choisir l'option **New Project**.

Nous vous laissons le soin de choisir le nom du projet. À titre indicatif, nous l'avons appelé Pokemon.

5.1.1 Installation de modules

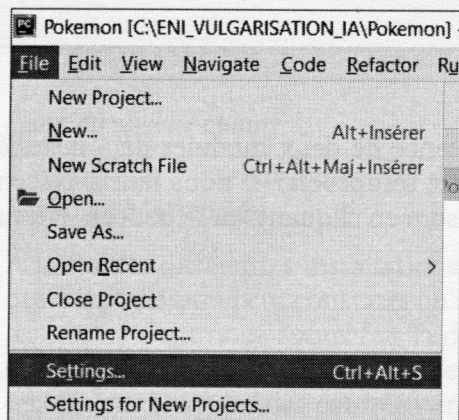
Un module peut être assimilé à une boîte à outils offrant des fonctions complémentaires dans un domaine dédié, nous évitant ainsi de les coder nous-mêmes. Concernant notre sujet, il existe des modules spécialisés dans l'analyse de données et l'intelligence artificielle, c'est donc tout naturellement que nous allons utiliser certains d'entre eux pour notre projet.

En voici la liste et leur rôle :

Module	Rôle
Numpy	Permet la manipulation de tableaux et matrices
Pandas	Permet la manipulation et l'analyse de données

Pour pouvoir faire référence à ces modules et les utiliser dans notre script, nous devons les importer dans l'environnement virtuel de notre projet. Voici comment procéder :

▣ Dans le menu **file**, choisissez l'option **Settings**.



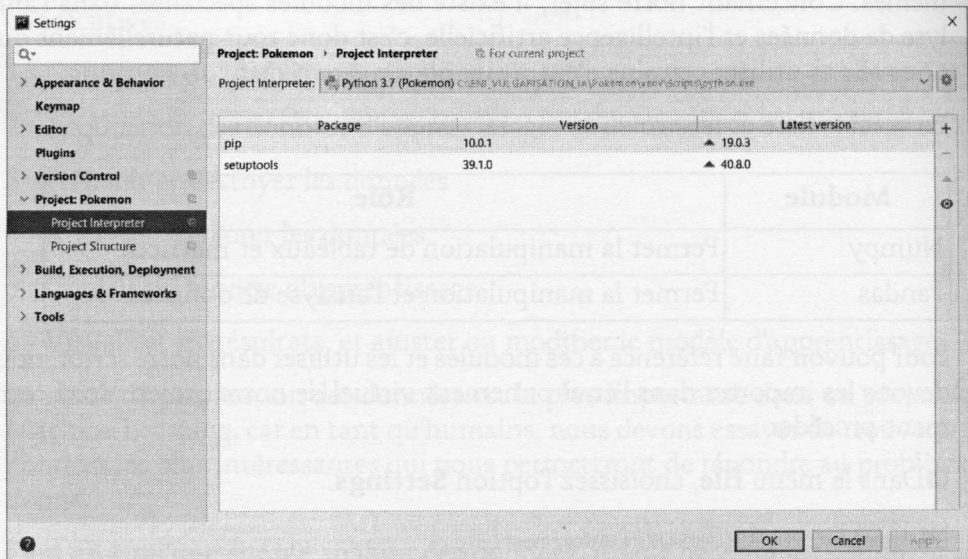
Menu de paramétrage du projet

136 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Une fenêtre s'ouvre alors à l'écran.

- Positionnez-vous dans le menu **Project: Pokemon** et choisissez le sous-menu **Project Interpreter**.

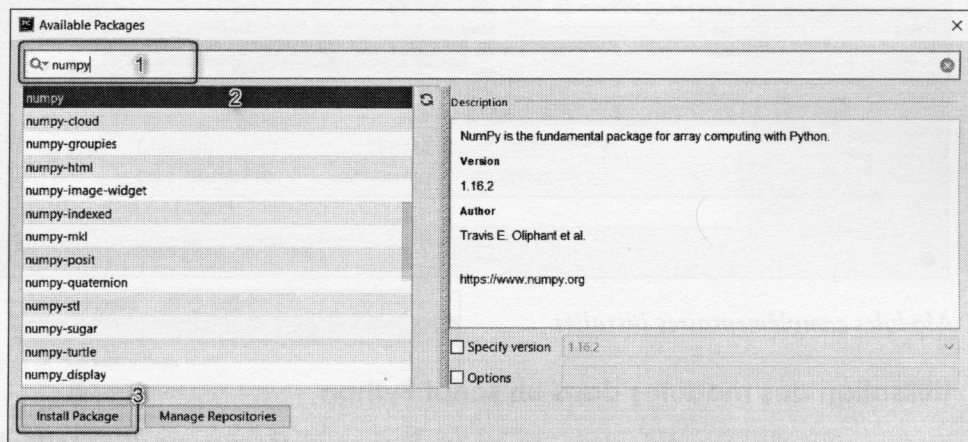


Liste des modules présents pour notre projet

Dans cette fenêtre, on constate la présence de deux modules déjà installés dans notre environnement virtuel (pip et setuptools). Il nous faut à présent ajouter les modules dont nous avons besoin en cliquant sur le bouton + situé en haut à droite.

Une nouvelle fenêtre s'ouvre alors, où il nous est possible de rechercher et d'installer un module. À titre d'exemple, la copie d'écran ci-dessous montre comment ajouter le module numpy.

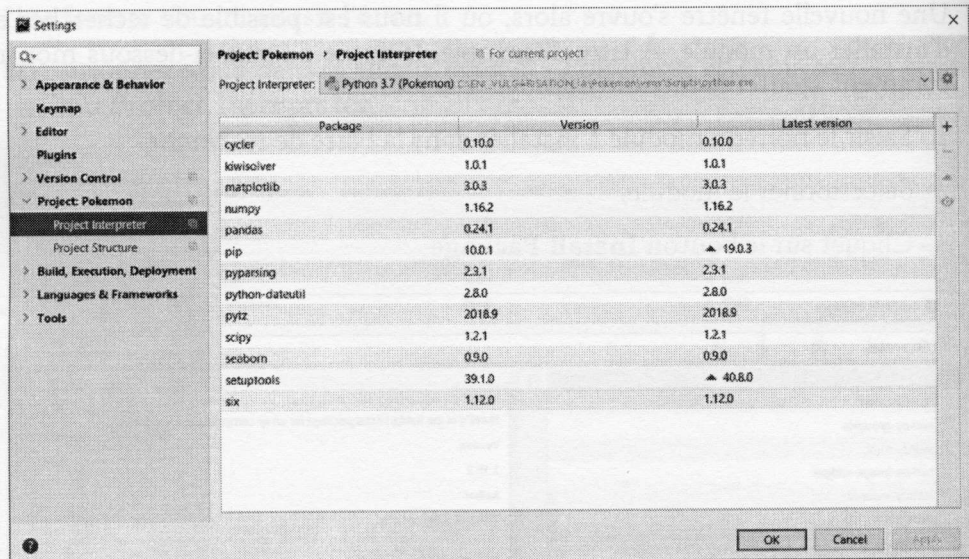
- 1 - Saisir le nom du module à installer dans la barre de recherche.
- 2 - Sélectionner le module.
- 3 - Cliquer sur le bouton **Install Package**.



Ajout du module numpy

Il nous reste à présent à procéder de la même façon pour les modules Pandas, Seaborn, Matplotlib.

À la fin des différentes installations, si l'on observe la fenêtre listant les modules installés, on peut constater qu'il y a davantage de nouveaux modules que les quatre dont nous avons fait la demande d'installation. Cela vient du fait que nos modules ont besoin d'autres modules pour pouvoir fonctionner correctement. Les modules complémentaires sont donc installés de façon automatique.



Modules complémentaires installés

5.1.2 Utilisation des modules dans un script Python

L'utilisation d'un module dans un script Python s'effectue par l'usage de la commande `import` comme le montre le code ci-dessous :

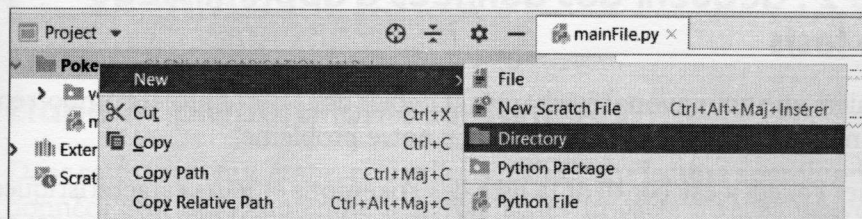
```
import numpy as nmp
```

Cette ligne de code peut être interprétée comme *"Importer le module numpy et l'utiliser sous le nom nmp"*. Ce nom est ce que l'on appelle un alias faisant référence au module importé. C'est cet alias que nous utiliserons par la suite dans notre script pour accéder aux fonctions proposées par le module.

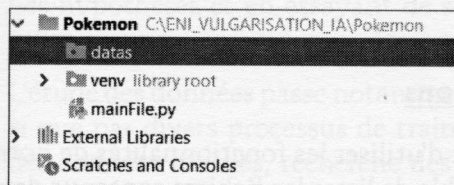
5.1.3 Référencement des fichiers de données dans notre projet

La dernière étape de création du projet consiste à importer les fichiers de données d'apprentissage et de tests.

- Pour ce faire, créez un nouveau répertoire dans la structure du projet en faisant un clic droit sur le nom du projet puis choisissez l'option **New** et la sous-option **Directory**. Donnez-lui ensuite le nom de **datas**.



Ajout d'un nouveau répertoire



Répertoire datas précédemment créé

- Téléchargez ensuite les différents fichiers de données disponibles sur le site de l'éditeur. Déposez à présent dans le répertoire **datas** nouvellement créé les trois fichiers, via un simple glissé-déplacé.

Notre projet est à présent créé et correctement configuré. Nous allons pouvoir commencer l'analyse des données.

6. Étape 1 : définir le problème à résoudre

Comme évoqué en début de chapitre, la problématique à laquelle nous devons répondre consiste à préconiser au dresseur de Pokémon l'animal à utiliser lors d'un combat afin d'être le vainqueur.

7. Étape 2 : acquérir des données d'apprentissage et de tests

Les fichiers que nous avons téléchargés et copiés dans notre projet comportent les données nécessaires à la résolution de notre problème.

Le fichier Pokedex.csv contient la liste des Pokémon et leurs caractéristiques. Le fichier Combats.csv est notre base de connaissance et d'apprentissage, car il contient une liste de combats de Pokémon et le résultat de chacun d'entre eux. Enfin, le fichier tests.csv nous permettra de valider le modèle d'apprentissage.

Listing des fichiers dont nous disposons

À l'aide du module OS nous permettant d'utiliser les fonctionnalités de notre système d'exploitation, il nous est possible de lister les fichiers contenus dans le répertoire datas de notre projet, comme le montre le code ci-dessous :

```
#-----  
# IMPORT DES MODULES  
#-----  
import os #Utilisation du module OS (operating system)  
  
#-----  
# ANALYSE DES DONNEES  
#-----  
  
#Récupération des fichiers contenus dans le répertoire datas  
#de notre projet  
listeDeFichiers = os.listdir("datas")  
  
#Quel est le nom de chaque fichier ?  
for fichier in listeDeFichiers:
```

```
print(fichier)
```

Si nous exécutons à présent notre script, il en résulte une liste exhaustive des fichiers dont nous disposons :

```
combats.csv  
pokedex.csv  
tests.csv
```

```
Process finished with exit code 0
```

8. Étape 3 : préparation des données

Dans cette troisième étape, nous allons réaliser une lecture approfondie de nos données afin de comprendre leur rôle et les impacts qu'elles peuvent avoir dans l'objectif de prédiction que nous nous sommes fixé. Nous allons en quelque sorte essayer de résoudre le problème "manuellement" en formulant des hypothèses et en essayant de sélectionner les données qui répondront à celles-ci.

L'étude des données passe notamment par leur description (nom, type...), ainsi que par divers processus de traitement tels que le nettoyage (suppression des données inutiles, recherche des données manquantes) et enfin la combinaison entre elles, aussi appelée agrégation, dans le but de disposer d'un jeu de connaissances (observations) utilisables et appropriées à l'apprentissage et à l'atteinte de notre objectif.

8.1 De quelles données disposons-nous ?

Les fichiers de données dont nous disposons portent l'extension CSV (*Comma Separated Value*). C'est-à-dire que les données contenues dans ces fichiers sont séparées par des virgules. Pour vous donner une petite idée du contenu de ces fichiers, vous pouvez les ouvrir à l'aide du logiciel Excel ou via un simple éditeur de texte tel que Notepad ou bien encore directement dans le logiciel PyCharm.

Dans notre exemple, les fichiers sont de petite taille, pouvant être ouverts et exploités par des logiciels. Mais dans la plupart des cas liés au Machine Learning, les fichiers contiennent énormément de données, ne pouvant alors plus être exploités par des logiciels traditionnels.

La démarche de préparation des données sera donc réalisée en supposant que les fichiers dont nous disposons sont trop volumineux pour être ouverts dans Excel ou dans l'outil NotePad de Windows. Nous travaillerons donc à l'aveugle et procéderons donc à la découverte progressive de nos données.

Nous allons utiliser le module Pandas, disposant d'une fonction de lecture de fichiers CSV (`read_csv`), capable de découper le contenu du fichier et de le stocker dans un tableau, appelé `DataFrame`, afin de pouvoir facilement analyser et manipuler les données.

■ Remarque

Un `DataFrame` est, au sens Python du terme, un dictionnaire dont les clés sont les noms des features et les valeurs sont les données contenues dans chaque caractéristique.

```
#-----  
# IMPORT DES MODULES  
#-----  
  
#Utilisation du module Pandas  
import pandas as pnd  
  
#-----  
# ANALYSE DES DONNEES  
#-----  
  
#Chargement des données des Pokémons dans un  
#Dataframe nommé nosPokemons  
nosPokemons = pnd.read_csv("datas/pokedex.csv")
```

Maintenant que les données sont stockées dans un Dataframe, il est important de connaître les libellés des colonnes pour avoir une idée des données dont nous disposons et de leur sémantique :

```
#-----
# ANALYSE DES DONNEES
#-----

#Chargement des données des Pokémons dans un
#Dataframe nommé nosPokemons
nosPokemons = pnd.read_csv("datas/pokedex.csv")

#Affichage des colonnes du Dataframe
print(nosPokemons.columns.values)
```

Ce script donnant le résultat suivant, soit au total 12 colonnes :

```
['NUMERO' 'NOM' 'TYPE_1' 'TYPE_2' 'POINTS_DE_VIE'
 'NIVEAU_ATTAQUE'
 'NIVEAU_DEFENSE' 'NIVEAU_ATTAQUE_SPECIALE'
 'NIVEAU_DEFENSE_SPECIALE'
 'VITESSE' 'GENERATION' 'LEGENDAIRE']
```

Colonne	Signification
NUMERO	Identifiant du Pokémon
NOM	Nom du Pokémon
TYPE_1	Type primaire (Herbe, Feu, Acier...)
TYPE_2	Type secondaire (Herbe, Feu, Acier...)
POINTS_DE_VIE	Points de vie du Pokémon
NIVEAU_ATTAQUE	Le niveau d'attaque du Pokémon
NIVEAU_DEFENSE	Le niveau de défense du Pokémon
NIVEAU_ATTAQUE_SPECIALE	Le niveau d'attaque spéciale du Pokémon
NIVEAU_DEFENSE_SPECIALE	Le niveau de défense spéciale du Pokémon
VITESSE	Vitesse du Pokémon
GENERATION	Numéro de génération à laquelle appartient le Pokémon

Colonne	Signification
LEGENDAIRE	Le Pokémon est-il légendaire? Il s'agit d'une donnée booléenne (Vrai ou Faux)

8.2 Affichage des dix premières lignes de nos données

Nous connaissons à présent les types de données propres à nos Pokémon, il nous faut à présent visualiser leur contenu.

Pour afficher les dix premières lignes de nos données, nous allons utiliser la fonction Head du Dataframe `nosPokemons` que nous avons créé.

Il faut également désactiver la limitation du nombre de colonnes affichées dans PyCharm, car par défaut seules 5 colonnes sont présentées et notre jeu de données en comporte 12.

```
#-----
# IMPORT DES MODULES
#-----

#Utilisation du module Pandas
import pandas as pd

#Désactivation du nombre maximum de colonnes du DataFrame à
afficher
pnd.set_option('display.max_columns',None)

#-----
# ANALYSE DES DONNEES
#-----

#Chargement des données des Pokémon dans un
#Dataframe nommé nosPokemons
nosPokemons = pnd.read_csv("datas/pokedex.csv")

#Affichage des colonnes du Dataframe
print(nosPokemons.columns.values)

#Affichage des 10 premières lignes du Dataframe
print(nosPokemons.head(10))
```

Voici le résultat de ce script :

	NUMERO	NOM	TYPE_1	TYPE_2	POINTS_DE_VIE
	NIVEAU_ATTAQUE \				
0	1	Bulbizarre	Herbe	Poison	45
49					
1	2	Herbizarre	Herbe	Poison	60
62					
2	3	Florizarre	Herbe	Poison	80
82					
3	4	Mega Florizarre	Herbe	Poison	80
100					
4	5	Salameche	Feu	NaN	39
52					
5	6	Reptincel	Feu	NaN	58
64					
6	7	Dracaufeu	Feu	Vol	78
84					
7	8	Mega Dracaufeu X	Feu	Dragon	78
130					
8	9	Mega Dracaufeu Y	Feu	Vol	78
104					
9	10	Carapuce	Eau	NaN	44
48					

NIVEAU_DEFFENCE		NIVEAU_ATTAQUE_SPECIALE	
NIVEAU_DEFENSE_SPECIALE		VITESSE \	
0		49	65
65	45		
1		63	80
80	60		
2		83	100
100	80		
3		123	122
120	80		
4		43	60
50	65		
5		58	80
65	80		
6		78	109
85	100		
7		111	130
85	100		
8		78	159

115	100		
9		65	50
64	43		

GENERATION LEGENDAIRE

0	1	FAUX
1	1	FAUX
2	1	FAUX
3	1	FAUX
4	1	FAUX
5	1	FAUX
6	1	FAUX
7	1	FAUX
8	1	FAUX
9	1	FAUX

Il est à présent possible d'analyser plus en détail les données.

Remarque

En Data Science, chaque ligne de notre fichier est appelée une observation et chaque colonne une feature (caractéristique).

8.3 Quelles sont les features de catégorisation ?

Des features dites de catégorisation permettent de classer les données dans différents groupes à l'aide de caractéristiques communes.

Ainsi, les features pouvant nous aider à catégoriser ou à classer nos Pokémons dans des groupes différents sont :

- TYPE_1
- TYPE_2
- LEGENDAIRE
- GENERATION

En effet, il est possible de regrouper les Pokémons par type, par génération et par le fait qu'ils soient légendaires ou non. Les autres features ne peuvent pas être utilisées dans la catégorisation, car elles sont différentes par Pokémon.

Il est encore possible de classer ces features de catégorisation, en tant que feature de catégorisation catégorielle ou ordinale.

Une donnée de catégorisation ordinale est une donnée chiffrée. Dans notre cas, il s'agit de la génération du Pokémon. Pour les autres données de catégorisation, il s'agit de données de catégorisation catégorielle.

DONNEES DE CATEGORISATION	
CATEGORIELLE	ORDINALE
TYPE_1	GENERATION
TYPE_2	
LEGENDAIRE	

Connaître les données de catégorisation nous permet de définir des données qui peuvent nous servir dans la résolution de notre problème. Par exemple, il se peut que si le Pokémon est de génération 2 ou bien de type Herbe ou bien encore légendaire, il est alors le vainqueur du combat. Cela permet donc d'optimiser l'algorithme de prise de décision.

De par leur importance, lors de la phase de nettoyage des données, ces features ne devront pas être supprimées.

8.4 Quelles données sont de type numérique ?

Dans le traitement des données, il est difficile de s'appuyer sur des données non numériques. Par conséquent, il est important de connaître les données numériques qui pourront être le socle de nos analyses.

En observant les dix premières données, on peut constater que les données numériques sont :

- NUMERO
- POINT_DE_VIE

- NIVEAU_ATTACHE
- NIVEAU_DEFENSE
- NIVEAU_ATTACHE_SPECIALE
- NIVEAU_DEFENSE_SPECIALE
- VITESSE

Là encore, il est possible de classer les données numériques en deux types : Les données numériques discrètes et les données numériques continues.

Une donnée numérique discrète est une donnée ayant une valeur dénombrable et énumérable (le nombre de pattes d'un Pokémon...).

Une donnée numérique continue, est une donnée ayant un nombre infini de valeurs formant un ensemble continu. Par exemple, le temps d'un combat de Pokémon peut être compris entre 1 et 60 minutes et peut prendre la valeur de 1 minute et 30 secondes, 3 minutes et 34 secondes... La taille et le poids d'un Pokémon sont également des données numériques continues, car un Pokémon peut peser 10,5 kg 10,6 kg, 12 kg...

Dans notre jeu d'observation, les données numériques dont nous disposons sont de type numérique discret.

8.5 Que faut-il penser de la feature LEGENDAIRE?

Nous avons vu que la feature LEGENDAIRE est de type booléen et qu'être un Pokémon légendaire est une chose rare et permet de disposer de force supplémentaire.

Cependant, si nous utilisons la feature dans son état actuel, elle sera considérée comme une chaîne de caractères et ne sera pas exploitable, car n'oublions pas que seules les données numériques sont appréciées du Machine Learning.

Il nous faut alors transformer cette donnée pour qu'elle devienne utilisable. Si la feature possède la valeur VRAI, elle sera alors transformée en 1, sinon elle prendra la valeur 0.

Pour ce faire, utilisons l'instruction suivante :

```
#Transformation de la colonne LEGENDAIRE en entier 0= FAUX et
1=VRAI
nosPokemons['LEGENDAIRE'] =
(nosPokemons['LEGENDAIRE']=='VRAI').astype(int)

print(nosPokemons['LEGENDAIRE'].head(800))
```

8.6 Manque-t-il des données?

Maintenant que nous avons classé nos observations et transformé la feature LEGENDAIRE, la question à se poser est "*Combien avons-nous d'observations?*"

Pour cela, nous allons utiliser la fonction shape du Dataframe :

```
#Comptage du nombre d'observations et de features
print(nosPokemons.shape)
```

Voici le résultat :

```
(800,12)
```

Cela signifie que notre jeu de données comporte 800 observations (Pokémons) comportant 12 caractéristiques (correspondant au nombre de colonnes déterminées dans la partie "*De quelles données disposons-nous?*" dans ce chapitre).

Regardons à présent si des observations sont manquantes. Pour cela, nous pouvons utiliser la fonction info() du Dataframe nous donnant, entre autres, le nombre d'éléments par feature :

```
#Informations sur notre jeu de données
print(nosPokemons.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 12 columns):
NUMERO      800 non-null int64
NOM         799 non-null object
TYPE_1      800 non-null object
TYPE_2      413 non-null object
POINTS_DE_VIE 800 non-null int64
```

```

NIVEAU_ATTAQUE          800 non-null int64
NIVEAU_DEFFENCE          800 non-null int64
NIVEAU_ATTAQUE_SPECIALE  800 non-null int64
NIVEAU_DEFENSE_SPECIALE  800 non-null int64
VITESSE                  800 non-null int64
GENERATION                800 non-null int64
LEGENDAIRE                800 non-null int32
dtypes: int64(8), object(4)
memory usage: 75.1+ KB
None

```

À la lecture de ce résultat, nous pouvons constater :

- Que nous disposons de 800 observations (800 entries).
- Que nous disposons de 12 features.
- Qu'il manque 1 nom (799 éléments sur 800 éléments attendus).
- Qu'il manque 387 Type_2 (413 éléments sur 800 éléments attendus).

On note en complément de ces informations que le type des différentes données est spécifié. Ainsi, les données numériques sont précisées par l'indication `int64` (int signifiant entier). Pour les autres, ce sont des données de type `Object` pouvant alors prendre la forme de chaîne de caractères (cas des features `NOM`, `TYPE_1` et `TYPE_2`).

8.7 À la recherche des features manquantes

Comme nous venons de le constater, il manque quelques données. En ce qui concerne la feature `TYPE_2`, cela peut s'expliquer par le fait qu'un Pokémon peut ne pas avoir de type secondaire. Les manques pour cette feature sont donc normaux et acceptables.

Concentrons-nous à présent sur le nom manquant. Pour le moment, nous ne savons pas si le nom du Pokémon a une incidence forte sur la résolution de notre problème. Est-ce que le fait de s'appeler Dracofeu fait que l'on gagne à chaque fois les combats? Ne pouvant pas répondre à cette question de façon positive ou négative, nous ne pouvons donc pas laisser cette observation manquante.

La première chose à faire est de trouver l'observation ayant le nom manquant.

L'instruction ci-dessous permet de rechercher et d'afficher l'observation du Dataframe `nosPokemon`, ayant la feature `NOM` non renseignée (`isNull()`):

```
print(nosPokemon[nosPokemon['NOM'].isNull()])
```

Ce qui nous permet de déterminer qu'il s'agit de l'observation 62 :

	NUMERO	NOM	TYPE_1	TYPE_2	POINTS_DE_VIE	NIVEAU_ATTAQUE
62	63	NaN	Combat	NaN	65	105

Il existe deux façons de traiter les observations manquantes :

- La suppression pure et simple de l'observation,
- Le remplacement de la valeur par une valeur artificielle, c'est-à-dire calculée, ou par une valeur réelle.

Comme nous l'avons évoqué, l'ensemble des Pokémons et leurs caractéristiques sont référencés dans le Pokédex. Ce document est accessible à cette adresse :

https://www.pokepedia.fr/Liste_des_Pok%C3%A9mon_dans_l'ordre_du_Pok%C3%A9dex_National.

Les Pokémons contenus dans notre fichier sont classés dans l'ordre de ce Pokédex. Il nous suffit donc de connaître le nom du Pokémon précédent et le nom du Pokémon suivant pour ensuite pouvoir réaliser une recherche dans le Pokédex et trouver le nom du Pokémon manquant.

Le Pokémon ayant le numéro d'observation 62 se trouve entre les observations 61 et 63, soit les Pokémons Férosinge et Caninos :


```
print(nosPokemon['NOM'][61])  
print(nosPokemon['NOM'][63])
```

```
Férosinge  
Caninos
```

152 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

En consultant à présent le Pokédex, on peut en déduire que le nom du Pokémon manquant est Colossinge!

056		Ferosinge	Mankey	Menki	マンキー (Mankī) Markey	COMBAT
057		Colossinge	Primeape	Rasaff	オコリザル Okorizaru	COMBAT
058		Caninos	Growlithe	Fukano	ガーディ (Gādī) Gardie	FEU

Recherche du nom du Pokémon manquant à l'aide du Pokédex

Il ne nous reste plus qu'à renseigner la feature NOM de l'observation n°62 comme suit :

```
nosPokemons['NOM'][62] = "Colossinge"
```

Remarque

Cette modification peut être à l'origine d'un avertissement de type "A value is trying to be set on a copy of a slice from a Dataframe" lors de l'exécution du script. Cet avertissement peut être ignoré.

Nous pouvons vérifier la bonne prise en compte de cette modification à l'aide de l'instruction :

```
print(nosPokemons['NOM'][62])  
Colossinge
```

Attention, la modification apportée par cette instruction ne concerne que le Dataframe et ne met pas à jour le fichier de données original. La mise à jour du fichier Pokedex.csv situé dans le répertoire Datas de notre projet peut se faire en réalisant un double clic sur le fichier ayant pour action d'ouvrir le fichier dans l'éditeur, nous permettant alors de renseigner la feature manquante.

Avant la modification :

```
63,,Combat,,65,105,60,60,70,95,1,FAUX
```

Après la modification :

```
63,Colossinge,Combat,,65,105,60,60,70,95,1,FAUX
```

8.8 Place aux observations des combats

Nous venons d'analyser les observations relatives aux Pokémons. Nous devons à présent nous concentrer sur celles des combats, répertoriées dans le fichier `combats.csv` en procédant par la même démarche d'analyse :

- Chargement du fichier
- Affichage des features
- Affichage des dix premières observations
- Classement des features : donnée catégorielle, numérique continue ou numérique discrète
- Comptage du nombre d'observations et recherche d'éventuelles données absentes

```
#Chargement des données des combats
combats = pd.read_csv("datas/combats.csv")

#Affichage des colonnes du Dataframe
print(combats.columns.values)

#Affichage des 10 premières lignes du Dataframe
print(combats.head(10))

#Comptage du nombre de lignes et de colonnes
print (combats.shape)

#Informations sur notre jeu de données
print (combats.info())
```

Voici le résultat de cette analyse :

- Le jeu de données comporte 50 000 observations et 3 features.
- Les features sont "Premier_Pokemon", "Second_Pokemon" et "Pokemon_Gagnant".
- La caractéristique "Pokemon_Gagnant" contient le numéro du Pokémon gagnant à l'issue du combat.
- Chaque feature est de type numérique.
- Il ne manque aucune information.

8.9 Assemblage des observations

Nous disposons de deux jeux d'observations. L'un consacré aux caractéristiques des Pokémons, l'autre aux combats. Il nous faut à présent réunir ces informations afin de savoir pour chaque Pokémon le nombre de combats menés ainsi que le nombre de victoires.

Pour avoir cette vision, qui nous permettra de connaître les Pokémons ayant une probabilité de victoire plus importante que les autres, nous devons assembler (agréger) les deux Dataframe Pokedex et Combats.

8.9.1 Nombre de combats menés

Avant de réaliser cette agrégation, nous allons réaliser quelques actions sur le Dataframe combats, afin de calculer le nombre de combats menés par chaque Pokémon.

Lors d'un combat, le Pokémon peut être utilisé en première position ou en seconde position :

COMBAT	PREMIER POKEMON	SECOND POKEMON
1	BULBIZARRE	DRACAUFEU
2	DRACAUFEU	COLOSSINGE
3	DRACAUFEU	PIKACHU

Donnant alors le nombre de combats par Pokémon :

POKEMON	NOMBRE DE COMBATS
BULBIZARRE	1
DRACAUFEU	3
COLOSSINGE	1
PIKACHU	1

Pour connaître le nombre de combats menés par un Pokémon, il faut donc connaître le nombre de fois où il a été en première position auquel on ajoute le nombre de fois où il a été en seconde position.

Mettons cela en pratique, en utilisant l'instruction suivante :

```
nbFoisPremierePosition =
combats.groupby('Premier_Pokemon').count()
print(nbFoisPremierePosition)
```

Si l'on reprend notre exemple, les données que nous obtenons seraient :

POKEMON	NOMBRE PREMIERE_POSITION
BULBIZARRE	1
DRACAUFEU	2
COLOSSINGE	0
PIKACHU	0

Il s'agit donc de compter le nombre de fois où le Pokémon est en première position via l'instruction `count()` puis de regrouper le résultat via l'instruction `groupBy()`.

La méthode `count()` permet de compter le nombre de caractéristiques par observation contenues dans un Dataframe en excluant les caractéristiques ayant des valeurs de type NA (Non Available). Dans l'exemple ci-dessous, `taillesDataframe.count()` reverra 4, car la seconde valeur vaut NA.

```
taillesDataframe = pnd.dataframe({"taille": [175, np.nan, 160,
170, 180]})
```

Exécutons le code saisi et observons le résultat :

PREMIER_POKEMON	SECOND POKEMON	POKEMON_GAGNANT
1	70	70
2	55	55
3	68	68
...

À la première lecture, ce tableau est difficile à comprendre et semble peu cohérent.

En effet, on s'attendrait à obtenir en intitulé de première colonne "NUMERO" puis une autre colonne "NOMBRE_DE_FOIS_EN_PREMIERE_POSITION". Mais cela n'est pas le cas.

La première colonne est notre critère de regroupement, c'est-à-dire la feature "PREMIER_POKEMON". Les données correspondent au numéro de Pokémon.

Lorsque l'on réalise un regroupement à l'aide de la fonction `groupBy()`, l'ensemble des features du jeu de données prennent la valeur de ce regroupement. C'est donc pour cela que les features `SECOND_POKEMON` et `POKE-MON_GAGNANT` ont la même valeur.

Si nous voulons donner plus de sens au résultat, nous pourrions le faire en changeant le nom des colonnes, les deuxième et troisième colonnes ayant le même intitulé :

NUMERO DU POKEMON	NOMBRE DE FOIS EN PREMIERE POSITION	NOMBRE DE FOIS EN PREMIERE POSITION
1	70	70
2	55	55
3	68	68
...

Procédons à présent de la même façon pour connaître le nombre de fois où chaque Pokémon a combattu en seconde position.

```
nbFoisSecondePosition = combats.groupby('Second_Pokemon').count()
print(nbFoisSecondePosition)
```

Puis réalisons la somme des deux calculs pour connaître le nombre total de combats.

```
nombreTotalDeCombats = nbFoisPremierePosition +
nbFoisSecondePosition
print(nombreTotalDeCombats)
```

Là encore, ne pas tenir compte des noms de colonnes affichés dans la réponse. La première correspond au numéro de Pokémon, la seconde au nombre de combats menés.

■ Remarque

Remarquez la force du module Panda, qui par un simple `+` a réussi à agréger l'ensemble des deux tableaux. Si nous avions codé cette fonctionnalité, nous aurions dû parcourir la liste des Pokémon tout en parcourant les deux tableaux de résultats afin de faire la somme du nombre de combats menés en première et seconde position.

POKEMON	NOMBRE DE COMBATS MENES
1	37
2	46
3	89
4	70
...	...
228	150

8.9.2 Nombre de combats gagnés

Passons à présent au nombre de combats gagnés par chaque Pokémon. Connaître cette information nous permettra de savoir si certains Pokémon n'ont jamais gagné de combats, ce qui nous permettra déjà de savoir qu'il faudra éviter de les utiliser lors d'un combat :

```
nombreDeVictoires = combats.groupby('Pokemon_Gagnant').count()
print(nombreDeVictoires)
```

POKEMON	NOMBRE DE COMBATS GAGNES
1	37
2	46
3	89
4	70

POKEMON	NOMBRE DE COMBATS GAGNES
...	...
228	104

En agrégeant les deux tableaux que nous venons de constituer, on constate que le Pokémon 228 a gagné 104 combats, alors qu'il en a mené 150 (cf. tableau précédent) alors que les quatre premiers Pokémon ont gagné tous leurs combats.

En utilisant la fonction `info()`, nous constatons que nous disposons de 783 observations. Cela signifie qu'il existe des Pokémon n'ayant jamais gagné! Charge à présent de les découvrir.

8.9.3 Agrégation des données avec le Pokédex

Le moment est maintenant venu d'agréer nos données avec le Pokédex. Cette agrégation va nous permettre d'avoir une vue centralisée des données et de permettre une analyse plus approfondie.

Dans un premier temps, nous allons créer une liste comportant :

- Les numéros des Pokémon
- Le nombre de combats menés
- Le nombre de combats gagnés
- Le pourcentage de victoires par rapport au nombre de combats menés

```
#On crée une liste à partir d'une extraction pour obtenir la
liste des Pokémon, que l'on trie par numéro
#Cette liste de numéros nous permettra de réaliser l'agrégation
des données
listeAAgreger = combats.groupby('Pokemon_Gagnant').count()
listeAAgreger.sort_index()

#On ajoute le nombre de combats
listeAAgreger['NBR_COMBATS'] =
nbFoisPremierePosition.Pokemon_Gagnant +
nbFoisSecondePosition.Pokemon_Gagnant

#On ajoute le nombre de victoires
listeAAgreger['NBR_VICTOIRES'] =
```

```

nombreDeVictoires.Premier_Pokemon

#On calcule le pourcentage de victoires
listeAAgreger['POURCENTAGE_DE_VICTOIRES']=
nombreDeVictoires.Premier_Pokemon/(nbFoisPremierePosition.Pokemon
_Gagnant + nbFoisSecondePosition.Pokemon_Gagnant)

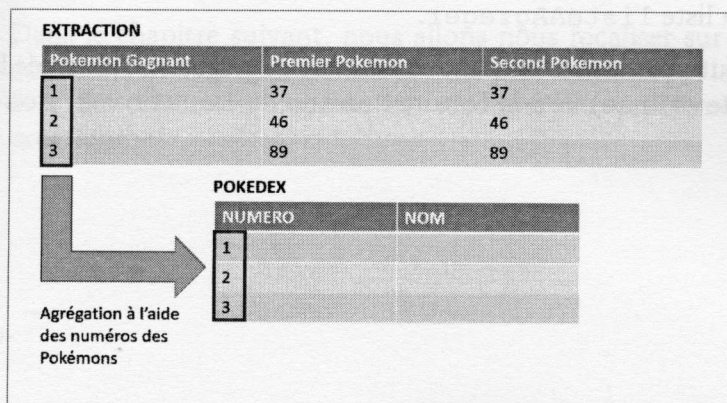
#On affiche la nouvelle liste
print(listeAAgreger)

```

La première étape du code consiste à créer une liste dont le contenu sera une extraction qui nous permet d'obtenir les numéros des Pokémons.

Pour cette extraction, nous avons choisi celle indiquant le nombre de victoires par Pokémon, mais nous aurions pu également choisir celle indiquant le nombre de fois où le Pokémon se trouve en première position, ou bien encore celle précisant le nombre de fois où il a combattu en seconde position. L'important est que nous obtenions la liste des numéros de Pokémon qui nous servira de clé permettant l'agrégation avec le Pokédex (cf. figure suivante).

Notons également l'usage de la fonction `sort_index()` qui permet de trier par ordre croissant les numéros des Pokémons, correspondant à l'ordre du Pokédex.



Agrégation du Pokédex afin d'obtenir le nombre de victoires par Pokémon

Une fois le code exécuté, voici la nouvelle liste que nous obtenons :

Pokemon_Gagnant	Premier_Pokemon	Second_Pokemon	NBR_COMBÂTS	NBR_VICTOIRES	POURCENTAGE_DE_VICTOIRES
1	37	37	133	37	0.278195
2	46	46	121	46	0.380165
3	89	89	132	89	0.674242
4	70	70	125	70	0.560000
...

Il ne nous reste plus qu'à agréger cette liste avec le Dataframe `nosPokemons` représentant le Pokédex. Cela se fait à l'aide la fonction `merge()` du Dataframe.

```
nouveauPokedex = nosPokemons.merge(listeAAgreger,  
left_on='NUMERO', right_index = True, how='left')  
  
print(nouveauPokedex)
```

Le résultat de ce code est la création d'un nouveau Dataframe que nous avons nommé `nouveauPokedex`, issu de l'agrégation du Dataframe `nosPokemons` avec la liste `listeAAgreger`.

Cette agrégation fut possible en utilisant comme clé d'agrégation l'index de la liste (`right_index=True`) lié à la feature `Numero` du Dataframe `nosPokemons`.

9. Une petite pause s'impose

Avant d'aller plus loin, revenons quelques instants sur ce que nous avons appris.

Dans ce chapitre, nous avons découvert la mise en place des premières étapes permettant de mener à bien un projet de machine learning, à savoir :

- La définition du problème à résoudre
- L'acquisition de données
- La préparation des données

Comme vous pouvez le constater, le temps de préparation des données est assez important. Et c'est ainsi pour tout projet de Machine Learning. En effet, comme nous l'avons indiqué, sans données, pas de Machine Learning. Sans oublier que ces données doivent être de bonne **qualité** due à une bonne préparation de celles-ci.

Nous avons également abordé quelques fonctions du module Pandas de Python, permettant d'interroger les données afin de mieux les comprendre, mais aussi à les modifier et à les agréger en vue d'obtenir un jeu de données qualitatif.

Dans le chapitre suivant, nous allons nous focaliser sur la visualisation des données, l'analyse approfondie de celle-ci et terminer par la recherche d'un modèle de prédiction capable de répondre au problème donné. Tout un programme !

Chapitre 6

Machine Learning et Pokémons : seconde partie

1. Ce que nous allons découvrir et les prérequis

Dans le chapitre précédent, nous avons constitué un jeu d'observations qui va nous permettre d'entamer la démarche d'analyse et de recherche de solution à notre problème, à savoir déterminer le Pokémon à utiliser lors d'un combat dans le but de le gagner.

Nous allons commencer ce chapitre par un peu de statistiques, mais n'ayez crainte, cela reste simple de compréhension.

■ Remarque

Prérequis nécessaires pour bien aborder ce chapitre : avoir lu les chapitres Les fondamentaux du langage Python, Des statistiques pour comprendre les données, Principaux algorithmes du machine learning, et avoir réalisé les différentes étapes du chapitre Machine Learning et Pokémons - première partie.

2. Un peu de statistiques

Avant d'entrer dans le vif du sujet, nous allons faire un peu de statistiques sur notre jeu d'observations à l'aide de la fonction `describe()` du module `Pandas`.

```
print(nouveauPokedex.describe())
```

Cette fonction a pour but d'expliquer statistiquement chaque feature de notre ensemble d'observations dans le but d'en dégager quelques caractéristiques intéressantes.

À titre d'exemple, prenons le résultat de l'analyse statistique de la feature `NIVEAU_ATTAQUE`.

Module	Définition	Valeur
Count	Nombre d'observations	800
Mean	La moyenne des observations	79.001250
Std	L'écart type	32.457366
min	La valeur minimale des observations	5
25 %		55
50 %		75
75 %		100
Max	La valeur maximale des observations	190

2.1 Le nombre de données (count)

On constate que 800 observations ont été renseignées pour la feature `NIVEAU_ATTAQUE`. Cela signifie qu'il ne manque pas de données.

2.2 La moyenne (mean)

Le niveau d'attaque moyen des Pokémons est de 79, c'est-à-dire qu'en dessous de cette moyenne on peut considérer que le Pokémon a un niveau faible, et au-dessus qu'il a un niveau fort, pouvant alors aider à gagner le combat. Nous disposons donc d'une première information intéressante.

2.3 L'écart type (Std pour Standard Deviation)

L'écart type est une valeur statistique, permettant de montrer la répartition des données autour de la moyenne. Plus sa valeur est petite, plus les données sont proches de la moyenne, dans le cas contraire, elles sont éloignées.

Dans notre cas, l'écart type est de 32, signifiant que les données de la feature NIVEAU_ATTAQUE semblent être assez éloignées de la moyenne. Cela signifie qu'il existe de nombreux cas différents de niveaux d'attaque et que les observations ne sont pas homogènes. C'est un point que nous devrons vérifier lors de la visualisation des données.

2.4 Les valeurs minimales et maximales

Les attributs min et max correspondent à la valeur minimale et à celle maximale des observations de la feature.

Ainsi, on note l'existence d'un ou plusieurs Pokémons ayant un niveau d'attaque à 5 et un ou plusieurs autres ayant un niveau d'attaque à 190.

Là encore, cette information peut nous servir dans notre choix du Pokémon, car plus son niveau maximal d'attaque est fort, plus il a de chances de remporter le combat.

2.5 Les quartiles

Les quartiles que nous avons découverts dans le chapitre Des statistiques pour comprendre les données sont utilisés en statistiques pour séparer les observations en quatre groupes homogènes.

Pour comprendre la notion de quartiles, prenons l'ensemble des observations de notre feature NIVEAU_ATTACHE et découpons-les en quatre parties égales :

LES QUARTILES				
1 ^{er} quartile Q1	2 ^{ème} quartile Q2	3 ^{ème} quartile Q3	4 ^{ème} quartile Q4	
25 %	50 %	75 %	100 %	
5	55	75	100	190
(Min)				(Max)

Les quartiles

En observant la figure, on constate que :

- 25 % des Pokémon ont un niveau d'attaque inférieur à 55,
- 50 % des Pokémon ont un niveau d'attaque inférieur à 75,
- 75 % des Pokémon ont un niveau d'attaque inférieur à 100.

Ces informations nous permettent d'affirmer que peu de Pokémon ont un taux d'attaque supérieur à 100 (seulement 25 %). Là encore, cette donnée est importante, car si nous voulons gagner un combat, il faut que nous utilisions un Pokémon présent dans ce dernier quartile.

2.6 Description de notre jeu d'observations

À présent que nous venons d'analyser une caractéristique précise, nous devons réaliser l'analyse complète des observations afin d'avoir une vue globale des données tout en gardant en tête le problème à résoudre.

L'analyse statistique nous permet-elle d'identifier des données singulières?

Tout d'abord, on constate que les résultats des combats se basent sur les données numériques (VITESSE, NIVEAU_ATTAQUE...), nous pouvons donc exclure les features NOM, TYPE_1, TYPE_2.

Est-il important de s'intéresser au numéro du Pokémon? Non, car il s'agit d'un identifiant, par conséquent cela n'a pas de lien dans le fait de gagner ou non un combat.

Certains Pokémon n'ont qu'un point de vie! Ce sont les Pokémon que nous devons éviter d'utiliser lors d'un combat.

25 % des Pokémon ont une vitesse supérieure à 90 (dernier quartile). Être rapide lors d'un combat peut être un avantage!

On constate un écart type de 11 pour la feature Combat. Cet écart type est le plus bas de l'ensemble des données. Cela signifie que les Pokémon ont tous réalisé un nombre de combats proche de la moyenne, c'est-à-dire proche de 127.

Cette valeur nous permet de dire que les observations contenues dans le jeu de données sont assez qualitatives, car il n'y a pas de Pokémon ayant davantage combattu par rapport aux autres, ce qui ne nous permettrait pas d'avoir un jeu d'apprentissage reflétant la réalité.

En effet, si seuls les Pokémon Bulbizarre et Dracaufeu avaient réalisé des combats, nous n'aurions aucune information sur ceux réalisés par Pikachu et nous aurions peut-être considéré que Bulbizarre ou Dracaufeu sont plus forts que Pikachu, alors que ce n'est peut-être pas le cas.

■ Remarque

Dans le chapitre *Des statistiques pour comprendre les données*, nous avons fait mention d'une classe Python permettant de réaliser une étude statistique pour une feature donnée. Si vous le souhaitez, vous pouvez l'utiliser sur la feature de votre choix en important cette classe dans votre projet et en lui passant en paramètre la feature.

3. Quels sont les types de Pokémons qu'un dresseur doit posséder ?

Répondre à notre problématique, c'est se mettre à la place d'un dresseur de Pokémons. Afin de maximiser les chances de gagner, il faut que le dresseur de Pokémons dispose des principaux Pokémons du Pokédex dans sa collection.

En effet, en ayant les Pokémons les plus fréquemment rencontrés, cela signifie que les adversaires ont de grandes chances de posséder les mêmes. Sachant que deux Pokémons de même type peuvent contrer les attaques, cela peut donc éviter de perdre le combat (<https://pokemondb.net/type/dual>).

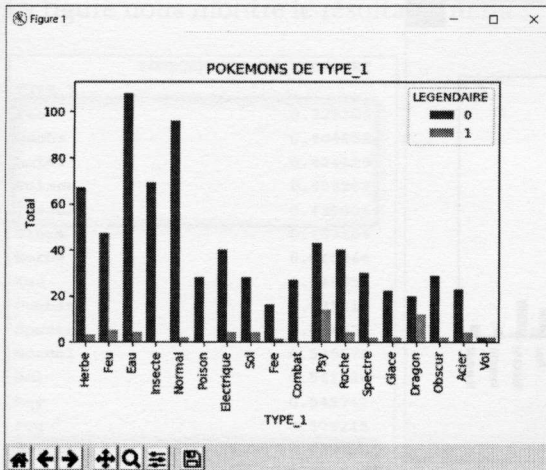
Pour connaître ces Pokémons indispensables à tout dresseur, nous allons utiliser un graphique qui nous permettra de visualiser rapidement leur nombre en fonction de leur type. Voici le code à utiliser :

■ Remarque

On note l'import du module `matplotlib`, et l'utilisation du module `seaborn`, tous deux nécessaires à la représentation graphique des données.

```
import matplotlib.pyplot as plt
import seaborn as sns

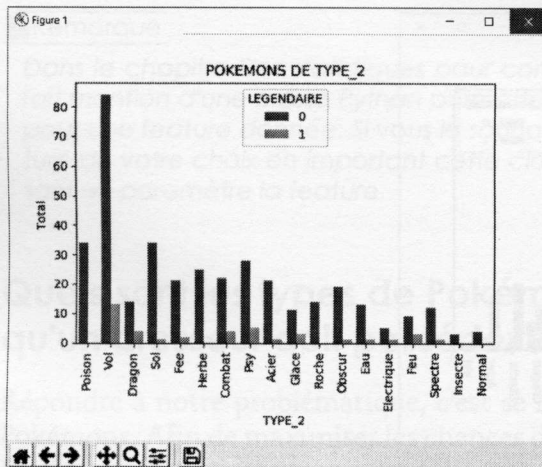
axe_X = sns.countplot(x="TYPE_1", hue="LEGENDAIRE",
data=nouveauPokedex)
plt.xticks(rotation= 90)
plt.xlabel('TYPE_1')
plt.ylabel('Total ')
plt.title("POKEMONS DE TYPE_1")
plt.show()
```



Visualisation du nombre de Pokémons de type 1

Idem pour les Pokémons de type 2 :

```
axe_X = sns.countplot(x="TYPE_2", hue="LEGENDAIRE",
data=nouveauPokedex)
plt.xticks(rotation= 90)
plt.xlabel('TYPE_2')
plt.ylabel('Total ')
plt.title("POKEMONS DE TYPE_2")
plt.show()
```



Visualisation du nombre de Pokémons de type 2

Ce qui nous permet d'affirmer qu'un dresseur doit posséder dans sa collection des Pokémons ayant pour premier type "Herbe", "Eau", "Insecte" et "Normal". Puis "Vol", "Poison" et "Sol" en second type pour pouvoir contrer les attaques, car ce sont ces Pokémons qui ont une barre d'histogramme plus importante que les autres.

Voilà une première information intéressante!

Maintenant, essayons de déterminer les Pokémons ayant un taux de victoire supérieur aux autres.

4. Les types de Pokémons gagnants et perdants

Il est à notre sens important de connaître les types de Pokémons gagnants. En effet, en s'assurant de les avoir dans sa collection et de les utiliser dans les combats, le dresseur a de fortes chances de gagner.

Pour obtenir cette information, on calcule la moyenne des pourcentages de victoires de chaque Pokémon. Cette moyenne, calculée et groupée par type de Pokémon, est ensuite triée par ordre croissant :

```
print(nouveauPokedex.groupby('TYPE_1').agg({"POURCENTAGE_VICTOIRE":
"mean"}).sort_values(by = "POURCENTAGE_VICTOIRE"))
```

La figure nous montre le résultat obtenu :

POURCENTAGE_DE_VICTOIRE	
TYPE 1	
Fee	0.329300
Roche	0.404852
Acier	0.424529
Poison	0.433262
Insecte	0.439006
Glace	0.439604
Herbe	0.440364
Eau	0.469357
Combat	0.475616
Spectre	0.484027
Normal	0.535578
Sol	0.541526
Psy	0.545747
Feu	0.579215
Obscur	0.629726
Electrique	0.632861
Dragon	0.633587
Vol	0.765061

Classement des Pokémons par le pourcentage moyen de leur victoire

On peut donc en déduire que les Pokémons gagnants sont de type :

- Obscur
- Électrique
- Dragon
- Vol

Nous pouvons également affirmer que les types de Pokémons suivants sont souvent voués à perdre leur combat :

- Fée
- Eau
- Roche
- Acier
- Poison

Peut-on déjà en déduire quelque chose? Si l'on prend le Pokémon, héros de la série, nommé Pikachu, de type électrique et qu'on le compare à un Pokémon Fée, on peut en déduire que Pikachu a de grandes chances de remporter le combat.

Si vous jouez personnellement aux Pokémon, possédez-vous la majorité des Pokémon du Pokédex? Avez-vous également des Pokémon de types électrique, obscur, dragon et vol?

Si ce n'est pas le cas, il faut d'urgence vous en procurer. Ce sont les statistiques qui l'affirment!

5. Essayons de trouver une corrélation entre les données

Une corrélation entre deux données signifie que celles-ci ont un lien fort entre elles. Exemple le statut d'adulte et l'âge de 18 ans, le fait de savoir voler et d'être pourvu d'ailes...

Dans notre cas, il s'agit de déterminer l'existence de features ayant un lien fort avec la capacité à gagner un combat. Faut-il être rapide? Faut-il avoir un grand niveau d'attaque? C'est ce que nous allons découvrir.

Avant toute chose, nous devons nous questionner sur l'utilité de l'ensemble des données en nous posant la question : "Cette feature peut-elle avoir un impact sur le fait de gagner ou non le combat?"

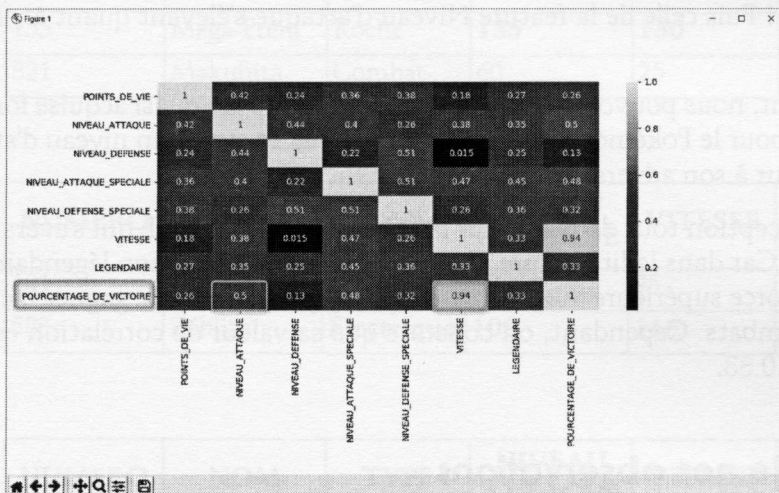
- Le numéro : non, car différent pour chaque Pokémon.
- Le nom : non, car différent pour chaque Pokémon.
- Le type_1 : oui, car nous venons de voir que certains types de Pokémon sont propices à la victoire.
- Le type_2 : non, car tous les Pokémon n'ont pas de second type.
- Les points de vie : oui, car plus on a de points de vie, plus on a de chance de gagner.
- Les différents niveaux (attaque, défense, attaque spéciale, défense spéciale) : oui, car ce sont des caractéristiques propres au combat.
- La vitesse : oui, car c'est aussi une caractéristique de combat.

- Légendaire : oui, car un Pokémon légendaire disposerait d'une force supplémentaire.

Pour permettre de visualiser la corrélation entre les features, nous allons utiliser un graphique appelé HeatMap se construisant comme suit :

- Calcul de la corrélation des différentes features
- Création du graphique à partir de la corrélation calculée

```
corr =
nouveauPokedex.loc[:, ['TYPE_1', 'POINTS_DE_VIE', 'NIVEAU_ATTAQUE',
'NIVEAU_DEFENSE', 'NIVEAU_ATTAQUE_SPECIALE', 'NIVEAU_DEFENSE_SPECIALE',
'VITESSE', 'LEGENDAIRE', 'POURCENTAGE_DE_VICTOIRE']].corr()
sns.heatmap(corr,
             xticklabels=corr.columns,
             yticklabels=corr.columns, annot=True)
plt.show()
```



HeatMap de corrélation des features

Remarque

Si vous rencontrez un problème d'exécution du script à cette étape, supprimez la colonne POURCENTAGE_DE_VICTOIRE dans le code que vous venez de saisir, puis après avoir exécuté une première fois le code, vous pouvez à nouveau l'ajouter. Le souci provient du fait que la colonne n'est pas correctement indexée.

À la lecture de ce graphique créé à l'aide du module Seaborn, on constate que la feature `Type_1` n'est pas présente. Cela vient du fait que ce n'est pas une valeur numérique et elle se trouve donc exclue !

Comment lire ce graphique ?

La corrélation se faisant entre deux features, il faut prendre la feature située à gauche du graphique (axe des ordonnées) et la croiser avec celle située en bas (axe des abscisses).

Le chiffre se situant à l'intersection des deux features correspond au pourcentage de corrélation. Plus il se rapproche de 1, plus la corrélation est forte.

Si l'on considère la feature qui nous intéresse, à savoir le pourcentage de victoires, le taux de corrélation est égal à 1 lorsqu'elle est comparée à elle-même ce qui est tout à fait normal. S'en suit la feature Vitesse montrant une corrélation de 0,94 ! Puis celle de la feature Niveau d'attaque s'élevant quant à elle à 0.50.

Par conséquent, nous pouvons en déduire que la victoire est quasi acquise lors d'un combat pour le Pokémon le plus rapide des deux et ayant un niveau d'attaque supérieur à son adversaire.

Une petite déception tout de même sur la feature **LEGENDAIRE** qui s'avérerait prometteuse. Car dans la littérature, il est indiqué qu'un Pokémon légendaire possède une force supérieure aux autres, ce qui lui permettrait de gagner facilement les combats. Cependant, on constate que sa valeur de corrélation est seulement de 0.33.

6. Résumé de nos observations

Si l'on résume les observations que nous avons effectuées, nous pouvons dire que pour remporter la victoire lors d'un combat, le dresseur de Pokémon doit :

- Posséder des Pokémon de type herbe, eau, insecte et normal, ce qui lui permet de contrer les attaques,
- Posséder des Pokémon de type obscur, électrique, dragon et vol, car ce sont eux qui ont un taux supérieur de victoire,

- Utiliser un Pokémon ayant une grande vitesse,
- Utiliser un Pokémon ayant un bon niveau d'attaque.

7. Vérifions nos hypothèses

Il nous faut à présent vérifier nos hypothèses en sélectionnant quelques combats dans le fichier combats.csv.

155, 321, 155
101, 583, 583
404, 32, 32

NUMERO	NOM	TYPE	NIVEAU ATTAQUE	VITESSE	VAINQUEUR
155	Méga-Ptéra	Roche	135	150	OUI
321	Makuhita	Combat	60	25	

NUMERO	NOM	TYPE	NIVEAU ATTAQUE	VITESSE	VAINQUEUR
101	Spectrum	Spectre	50	95	
583	Zéblitz	Électrique	100	116	OUI

NUMERO	NOM	TYPE	NIVEAU ATTAQUE	VITESSE	VAINQUEUR
275	Jungko	Herbe	85	120	
155	Méga-Ptéra	Roche	135	150	OUI

NUMERO	NOM	TYPE	NIVEAU ATTAQUE	VITESSE	VAINQUEUR
404	Rosabyss	Eau	84	52	
32	Raichu	Électrique	90	110	OUI

Au vu de ces résultats, nos hypothèses semblent donc vérifiées.

8. Passons à la phase d'apprentissage

Le problème que nous devons résoudre consiste à déterminer si lors d'un combat, un Pokémon a de grandes chances de gagner.

Comme nous disposons de données d'apprentissage, nous sommes donc dans un cas de Machine Learning dit "supervisé". C'est-à-dire que la machine va apprendre en fonction de ce qu'on lui fournit en entrée.

Dans ce type de cas, nous disposons alors de deux types d'algorithmes : ceux dédiés à la classification ou ceux dédiés à la régression.

La classification permet d'organiser les prédictions en groupe, quant à la régression elle permet de définir une valeur.

Dans notre cas, nous devons prédire le pourcentage de victoire, c'est donc une valeur et c'est naturellement que nous utiliserons les algorithmes de **régression** que nous avons découverts dans le chapitre Principaux algorithmes du machine learning, à savoir :

- La régression linéaire
- Les arbres de décisions
- Les forêts aléatoires

8.1 Découpage des observations en jeu d'apprentissage et jeu de tests

La première étape avant tout apprentissage est de découper les observations dont nous disposons en un jeu d'apprentissage avec lequel la machine va réaliser son apprentissage et en jeu de tests avec lequel la machine va évaluer son apprentissage. Pour cela, nous allons utiliser le module Scikit-Learn que nous vous invitons à ajouter à votre projet.

Afin d'éviter d'exécuter les différentes tâches d'analyse de données et de préparation de celles-ci avant chaque apprentissage, nous avons sauvegardé le Dataframe dans un fichier nommé dataset.csv avec pour séparateur des tabulations (sep='\\t').

```
#Sauvegarde du Dataframe Pokedex
dataset = nouveauPokedex
dataset.to_csv("datas/dataset.csv", sep='\\t')
```

Dans un nouveau fichier de script Python, nous allons donc dans un premier temps charger ce fichier et supprimer les lignes pour lesquelles des valeurs sont manquantes comme à la ligne 12 du fichier :

A	I	J	K	L	M	N	O	P	Q	R
Colonnes	NIVEAU_ATTAQUE_SPECIALE	NIVEAU_DEFENSE_SPECIALE	VITESSE	GENERATION	LEGENDAIRE	Premier_Pokemon	Second_Pokemon	PBR_COURANTS	NBR_VICTOIRES	POURCENTAGE_DE_VICTOIRE
0	85	85	45	1	0.37.0	37.0	113.0	37.0	0.2781954807218045	
1	80	80	60	1	0.46.0	46.0	121.0	46.0	0.38016528925619836	
2	100	100	80	1	0.88.0	88.0	112.0	88.0	0.5742624242424242	
3	122	120	89	1	0.70.0	70.0	125.0	70.0	0.56	
4	60	50	65	1	0.55.0	55.0	112.0	55.0	0.49107142857142855	
5	80	65	80	1	0.64.0	64.0	118.0	64.0	0.5423728813559322	
6	109	85	100	1	0.115.0	115.0	112.0	115.0	0.8646616541553184	
7	130	85	100	1	0.119.0	119.0	119.0	119.0	0.8561155079136991	
8	159	115	100	1	0.114.0	114.0	135.0	114.0	0.8444444444444444	
9	50	64	43	1	0.19.0	19.0	117.0	19.0	0.1623931623931624	
10	65	80	58	1	0.59.0	59.0	141.0	59.0	0.41843971631205673	
11	85	105	78	1	0					
12	135	115	78	1	0.83.0	83.0	144.0	83.0	0.5761888888888888	

Données manquantes dans le fichier dataset.csv

```
#Chargement du dataset avec pour séparateur des tabulations
dataset = pnd.read_csv("datas/dataset.csv", delimiter='\\t')

#Suppression des lignes ayant des valeurs manquantes
dataset = dataset.dropna(axis=0, how='any')
```

Puis nous allons ensuite extraire les features qui seront source d'apprentissage (données X), c'est-à-dire les colonnes :

```
POINTS_ATTAQUE;POINTS_DEFFENCE;POINTS_ATTAQUE_SPECIALE;
POINT_DEFENSE_SPECIALE;POINTS_VITESSE;NOMBRE_GENERATIONS.
```

```
#X = on prend toutes les données, mais uniquement les features 5
à 12 (ne pas hésiter à ouvrir le fichier afin de bien visualiser
les features concernées)
#
POINTS_ATTAQUE;POINTS_DEFFENCE;POINTS_ATTAQUE_SPECIALE;POINT_
DEFENSE_SPECIALE;POINTS_VITESSE;NOMBRE_GENERATIONS
X = dataset.iloc[:, 5:12].values
```

Et les features que nous devons prédire à partir de ces données d'apprentissage, c'est-à-dire les valeurs de la colonne POURCENTAGE_DE_VICTOIRE.

```
#y = on prend uniquement la feature POURCENTAGE_DE_VICTOIRE
(17 ème feature)
y = dataset.iloc[:, 17].values
```

En d'autres termes, nous allons apprendre à la machine à prédire un pourcentage de victoire d'un Pokémon en fonction de ses points d'attaque (standards et spéciaux), de sa vitesse, de ses points de vie, de ses points de défense (standards et spéciaux), et de son nombre de générations. Nous venons donc de définir les variables prédictives (X) et la variable à prédire (Y).

Chaque variable sera ensuite découpée en un groupe d'apprentissage (Train) et en un groupe de test (Test). Nous aurons donc un groupe X_APPRENTISSAGE, Y_APPRENTISSAGE permettant à la machine d'apprendre et un groupe X_VALIDATION, Y_VALIDATION lui permettant de valider son apprentissage.

Les tableaux ci-dessous vous permettront sans doute de mieux comprendre ces différents découpages.

Le jeu d'observation complet (réduit à dix observations pour l'exemple) :

NOM	POINTS_DE_VIE	NIVEAU_ATTAQUE	[...]	VITESSE	POURCENTAGE_DE_VICTOIRE
Bulbizarre	45	49	[...]	45	0.2781954887218045
Herbizarre	60	62	[...]	60	0.38016528925619836
Florizarre	80	82	[...]	80	0.6742424242424242
Mega-Florizarre	80	100	[...]	80	0.56

NOM	POINTS_DE_VIE	NIVEAU_ATTACHE	[...]	VITESSE	POURCENTAGE_DE_VICTOIRE
Salameche	39	52	[...]	65	0.49107142857142855
Reptincel	58	64	[...]	80	0.5423728813559322
Dracaufeu	78	84	[...]	100	0.8646616541353384
Mega-Dracaufeu X	78	130	[...]	100	0.8561151079136691
Mega-Dracaufeu Y	78	104	[...]	100	0.8444444444444444
Carapuce	44	48	[...]	43	0.1623931623931624

Découpage en variables explicatives (X) et une variable expliquée (Y) :

X					Y
NOM	POINTS_DE_VIE	NIVEAU_ATTACHE	[...]	VITESSE	POURCENTAGE_DE_VICTOIRE
Bulbizarre	45	49	[...]	45	0.2781954887218045
Herbizarre	60	62	[...]	60	0.38016528925619836
Florizarre	80	82	[...]	80	0.6742424242424242
Mega-Florizarre	80	100	[...]	80	0.56
Salameche	39	52	[...]	65	0.49107142857142855
Reptincel	58	64	[...]	80	0.5423728813559322
Dracaufeu	78	84	[...]	100	0.8646616541353384
Mega-Dracaufeu X	78	130	[...]	100	0.8561151079136691
Mega-Dracaufeu Y	78	104	[...]	100	0.8444444444444444
Carapuce	44	48	[...]	43	0.1623931623931624

180 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

On prend 80 % des variables explicatives et des variables expliquées pour créer le jeu d'apprentissage (Train) :

X_APPRENTISSAGE (80 %)					Y_APPRENTISSAGE (80 %)
NOM	POINTS_DE_VIE	NIVEAU_ATTACHE	[...]	VITESSE	POURCENTAGE_DE_VICTOIRE
Bulbizarre	45	49	[...]	45	0.2781954887218045
Herbizarre	60	62	[...]	60	0.38016528925619836
Florizarre	80	82	[...]	80	0.6742424242424242
Mega-Florizarre	80	100	[...]	80	0.56
Salameche	39	52	[...]	65	0.49107142857142855
Reptincel	58	64	[...]	80	0.5423728813559322
Dracaufeu	78	84	[...]	100	0.8646616541353384
Mega-Dracaufeu X	78	130	[...]	100	0.8561151079136691

Les 20 % restant des variables explicatives et des variables expliquées constitueront le jeu de tests permettant de valider l'apprentissage :

X_VALIDATION (20 %)					Y_VALIDATION (20 %)
NOM	POINTS_DE_VIE	NIVEAU_ATTACHE	[...]	VITESSE	POURCENTAGE_DE_VICTOIRE
Mega-Dracaufeu Y	78	104	[...]	100	0.8444444444444444
Carapuce	44	48	[...]	43	0.1623931623931624

Enfin, voici le code associé utilisant la fonction `train_test_split` du module Scikit-Learn, permettant de créer les jeux de données d'apprentissage et de tests :

```
#X = on prend toutes les données, mais uniquement les features 4
à 11
#
POINTS_ATTAQUE;POINTS_DEFFENCE;POINTS_ATTAQUE_SPECIALE;POINT_
DEFENSE_SPECIALE;POINTS_VITESSE;NOMBRE_GENERATIONS
X = dataset.iloc[:, 4:11].values

#y = on prend uniquement la colonne POURCENTAGE_DE_VICTOIRE (16 ème
feature) les : signifiant "Pour toutes les observations"
y = dataset.iloc[:, 16].values

#Construction du jeu d'entrainement et du jeu de tests
from sklearn.model_selection import train_test_split
X_APPRENTISSAGE, X_VALIDATION, Y_APPRENTISSAGE, Y_VALIDATION =
train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Notons l'usage de la fonction `iloc` qui permet de sélectionner les features par leur numéro d'index et la valeur 0.2 utilisée dans la fonction `train_test_split` indiquant que nous consacrons 20 % des observations au jeu de tests.

8.2 Algorithme de régression linéaire

Comme nous l'avons vu dans le chapitre Des statistiques pour comprendre les données (encore lui), l'algorithme de régression linéaire modélise les relations entre une variable prédictive et une variable cible.

Dans notre cas, les variables prédictives sont :

- les points de vie
- le niveau d'attaque
- le niveau de défense
- le niveau d'attaque spéciale
- le niveau de défense spécial
- la vitesse et la génération du Pokémon

Et la variable cible (la prédiction) est :

- le pourcentage de victoire

Dans notre cas, nous avons plusieurs variables prédictives pour une prédiction. Nous sommes donc dans le cas d'une **régression linéaire multiple**. Voici comment utiliser l'algorithme de régression linéaire pour réaliser notre apprentissage à l'aide du module Scikit-Learn :

```
#---- ALGORITHME 1: REGRESSION LINEAIRE ----  
from sklearn.metrics import r2_score  
from sklearn.linear_model import LinearRegression  
  
#Choix de l'algorithme  
algorithme = LinearRegression()  
  
#Apprentissage à l'aide de la fonction fit  
#Apprentissage effectué sur le jeu de données d'apprentissage  
algorithme.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)  
  
#Realisation de prédictions sur le jeu de tests (validation)  
predictions = algorithme.predict(X_VALIDATION)  
  
#Calcul de la précision de l'apprentissage à l'aide de la  
#fonction r2_score en comparant les valeurs prédites  
#(predictions) et les valeurs attendues (Y_VALIDATION)  
  
precision = r2_score(y_test, predictions)  
  
print(">> ----- REGRESSION LINEAIRE -----")  
print(">> Precision = "+str(precision))  
print("-----")
```

Le calcul de la précision d'apprentissage (accuracy) se réalise sur les prédictions réalisées à partir des données de tests. En effet, nous souhaitons valider le niveau d'efficacité de l'algorithme. Pour cela, il faut le mesurer sur des données qu'il ne connaît pas. Notons qu'il est parfois également intéressant de calculer la précision sur les données d'apprentissage et de la comparer avec celle obtenue sur les données de tests afin de déterminer si nous sommes en présence d'un surapprentissage ou non. Mais nous aurons l'occasion de revenir sur ce point.

Pour mesurer cette précision (aussi appelé score), nous utilisons la méthode `r2_score` comme le préconise la documentation du module SciKit-Learn dans le cas de problèmes de régression (https://scikit-learn.org/stable/modules/model_evaluation.html) :

Scoring	Function	Comment
Classification		
'accuracy'	<code>metrics.accuracy_score</code>	
'balanced_accuracy'	<code>metrics.balanced_accuracy_score</code>	
'average_precision'	<code>metrics.average_precision_score</code>	
'brier_score_loss'	<code>metrics.brier_score_loss</code>	
'f1'	<code>metrics.f1_score</code>	for binary targets
'f1_micro'	<code>metrics.f1_score</code>	micro-averaged
'f1_macro'	<code>metrics.f1_score</code>	macro-averaged
'f1_weighted'	<code>metrics.f1_score</code>	weighted average
'f1_samples'	<code>metrics.f1_score</code>	by multilabel sample
'neg_log_loss'	<code>metrics.log_loss</code>	requires <code>predict_proba</code> support
'precision' etc.	<code>metrics.precision_score</code>	suffixes apply as with 'f1'
'recall' etc.	<code>metrics.recall_score</code>	suffixes apply as with 'f1'
'jaccard' etc.	<code>metrics.jaccard_score</code>	suffixes apply as with 'f1'
'roc_auc'	<code>metrics.roc_auc_score</code>	
Clustering		
'adjusted_mutual_info_score'	<code>metrics.adjusted_mutual_info_score</code>	
'adjusted_rand_score'	<code>metrics.adjusted_rand_score</code>	
'completeness_score'	<code>metrics.completeness_score</code>	
'fowlkes_mallows_score'	<code>metrics.fowlkes_mallows_score</code>	
'homogeneity_score'	<code>metrics.homogeneity_score</code>	
'mutual_info_score'	<code>metrics.mutual_info_score</code>	
'normalized_mutual_info_score'	<code>metrics.normalized_mutual_info_score</code>	
'v_measure_score'	<code>metrics.v_measure_score</code>	
Regression		
'explained_variance'	<code>metrics.explained_variance_score</code>	
'max_error'	<code>metrics.max_error</code>	
'neg_mean_absolute_error'	<code>metrics.mean_absolute_error</code>	
'neg_mean_squared_error'	<code>metrics.mean_squared_error</code>	
'neg_mean_squared_log_error'	<code>metrics.mean_squared_log_error</code>	
'neg_median_absolute_error'	<code>metrics.median_absolute_error</code>	
'r2'	<code>metrics.r2_score</code>	

Documentation du module Scikit-Learn concernant les fonctions à utiliser pour mesurer la précision d'un algorithme.

Nous laisserons pour le moment de côté les autres indicateurs de mesure, notamment la MSE (*Mean Square Error*) dont l'utilité sera expliquée dans le chapitre Un neurone pour prédire.

La compréhension du code permettant l'apprentissage et l'évaluation de l'algorithme est assez simple. Les commentaires présents dans le code vous ont sans doute permis de comprendre qu'il s'agissait en premier lieu de choisir un algorithme en concordance avec notre problème à résoudre (dans notre cas un problème de régression), puis de réaliser l'apprentissage à l'aide de la méthode `fit` tout en veillant à passer en paramètres de cette fonction les données d'apprentissage (`X_APPRENTISSAGE` et `Y_APPRENTISSAGE`).

Des prédictions sont ensuite réalisées sur les données de tests (`X_TEST`) et sont comparées à celles attendues (`Y_TEST`) à l'aide la méthode `r2_Score` chargée de calculer la précision de l'algorithme.

Ce qui nous permet d'en déduire qu'à l'aide de la régression linéaire multivariée, nous obtenons un score de 90 %, ce qui est plutôt bon.

```
>> ----- REGRESSION LINEAIRE -----
>> Precision = 0.9043488485570964
```

8.3 L'arbre de décision appliqué à la régression

Passons à présent à l'utilisation des arbres de décisions en faisant attention à bien utiliser l'algorithme destiné aux problèmes de régression : `DecisionTreeRegressor` et non celui destiné aux problèmes de classification : `DecisionTreeClassifier`.

```
#Choix de l'algorithme
from sklearn.tree import DecisionTreeRegressor
algorithme = DecisionTreeRegressor()

algorithme.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)

predictions = algorithme.predict(X_VALIDATION)

precision = r2_score(Y_VALIDATION, predictions)

print(">> ----- ARBRES DE DECISIONS -----")
```

```
print(">> Precision = "+str(precision))
print("-----")
```

On constate qu'en utilisant l'arbre de décisions le résultat est moins bon...

```
>> ----- ARBRE DE DECISIONS -----
>> Precision = 0.8812980947158535
```

8.4 La random forest

Testons à présent l'algorithme des forêts aléatoires (toujours appliqué à la régression) :

```
#Choix de l'algorithme
from sklearn.ensemble import RandomForestRegressor
algorithme = RandomForestRegressor()

algorithme.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)

predictions = algorithme.predict(X_VALIDATION)

precision = r2_score(Y_VALIDATION, predictions)

print(">> ----- FORETS ALEATOIRES -----")
print(">> Precision = "+str(precision))
print("-----")
```

Et nous obtenons cette fois-ci une précision de 93 %!

```
>> ----- FORETS ALEATOIRES -----
>> Precision = 0.9328013851258918
```

Nous avons donc trouvé notre challenger. **Cependant, nous tenons à préciser que les algorithmes sont ici utilisés dans leur forme la plus simple, c'est-à-dire sans optimisation de leurs paramètres.** Il se peut qu'en les torturant un peu nous obtenions d'autres résultats, peut-être encore plus prometteurs, avec l'un des trois algorithmes.

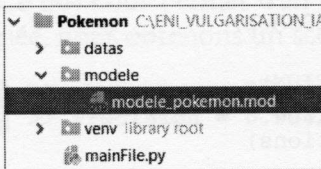
8.5 Sauvegarde du modèle d'apprentissage

Afin de pouvoir utiliser le modèle de prédiction fraîchement entraîné dans une application sans devoir réitérer la phase d'apprentissage, nous devons le sauvegarder.

Pour ce faire, après avoir créé un nouveau répertoire `modele` dans notre projet, nous allons utiliser le sous-module `joblib` du module Scikit-learn pour sauvegarder notre modèle d'apprentissage.

```
from sklearn.externals import joblib
fichier = 'modele/modele_pokemon.mod'
joblib.dump(algorithme, fichier)
```

Le fichier porte l'extension `.mod` pour modèle. Mais libre à vous d'utiliser l'extension que vous souhaitez.



Modèle d'apprentissage sauvegardé

9. Phénomènes de surapprentissage (overfitting) et de sous-apprentissage (underfitting)

Lorsque l'on cherche une solution à un problème en Machine Learning, nous souhaitons que celle-ci soit généralisable.

C'est-à-dire que la solution trouvée doit être applicable sur des données inconnues de celles utilisées lors de l'apprentissage. Pour cela, il faut que les données utilisées lors de la phase d'apprentissage soient les plus proches possible de la réalité et du problème à résoudre.

Le surapprentissage (*overfitting*) est un phénomène se traduisant par le fait que la solution est trop adaptée aux données d'apprentissage et ne se généralise pas à de nouvelles données qui lui sont inconnues. Ainsi, si pour un algorithme nous obtenons une précision 99 % sur les données d'apprentissage et que nous obtenons une valeur de 20 % sur les données de tests, il y a fort à parier que nous sommes en présence d'un surapprentissage. C'est pourquoi il est conseillé de mesurer la précision à la fois sur les données d'apprentissage et sur les données de validation :

```
predictions = algorithme.predict(X_VALIDATION)
precision_apprentissage =
algorithme.score(X_APPRENTISSAGE, Y_APPRENTISSAGE)
precision = r2_score(Y_VALIDATION, predictions)
```

Quant au phénomène de sous-apprentissage, ce dernier se produit lorsque l'algorithme n'arrive pas à trouver une corrélation entre les données d'apprentissage et n'arrive donc pas à réaliser de bonnes prédictions.

La notion d'*overfitting* est souvent rencontrée lors de l'utilisation des réseaux de neurones. En effet, lors de l'apprentissage nous allons chercher à minimiser les erreurs de prédictions en réalisant un certain nombre de boucles d'apprentissage (itérations) où, à chaque boucle, l'algorithme va apprendre de ses erreurs et se corriger.

Plus nous réalisons d'itérations, plus l'algorithme apprend et moins il se trompe. Si trop d'itérations sont réalisées, l'algorithme aura une très bonne précision sur les données d'apprentissage, mais il se sera spécialisé sur celles-ci et aura une mauvaise précision sur les données de tests. La généralisation n'est donc plus possible.

10. Utiliser le modèle d'apprentissage dans une application

Nous disposons d'un modèle d'apprentissage capable de nous prédire le pourcentage de victoire de chaque Pokémon. Nous allons à présent créer une application utilisant ce modèle d'apprentissage et ayant pour objectif de prédire le vainqueur d'un combat opposant deux Pokémon choisis dans le Pokédex.

La première étape consiste à créer un nouveau fichier Python dans notre projet que l'on nommera `quiSeraLeVainqueur.py`.

Dans ce fichier, nous allons tout d'abord importer les modules dont nous avons besoin.

```
#Module de lecture de fichiers CSV
import csv
```

```
#Module de chargement du modèle d'apprentissage
from sklearn.externals import joblib
```

Nous allons ensuite écrire une première fonction qui sera chargée de chercher les informations d'un Pokémon dans le Pokédex à partir de son numéro et utiles à notre modèle de prédiction.

```
def rechercheInformationsPokemon(numPokemon, Pokedex):
    infosPokemon = []
    for pokemon in Pokedex:
        if (int(pokemon[0])==numPokemon):
            infosPokemon =
[pokemon[0],pokemon[1],pokemon[4],pokemon[5],pokemon[6],pokemon[7],
pokemon[8],pokemon[9],pokemon[10]]
            break
    return infosPokemon
```

On crée tout d'abord une liste (`infosPokemon`) permettant de stocker toutes les informations du Pokémon nécessaires à la prédiction de sa victoire ou de sa défaite. On parcourt ensuite le Pokédex à la recherche du Pokémon ayant le même numéro que celui que nous cherchons. Il est important de noter que le numéro du Pokémon correspondant à la première colonne du Pokédex est transformé en entier (`int (pokemon [0])`) afin de pouvoir le comparer avec le numéro du Pokémon recherché, passé en paramètre de la fonction.

Si lors de la boucle de parcours du Pokédex, le numéro lu correspond au numéro du Pokémon recherché, on extrait alors les informations nécessaires pour pouvoir ensuite les renvoyer à la fonction appelante de notre fonction de recherche.

Le tableau ci-dessous reprend l'information nécessaire à la prédiction de victoire ou de défaite et sa position dans le Pokédex (le numéro se trouve dans la première colonne (pokemon[0]), le nom dans la deuxième (pokemon[1])...).

INFORMATION	INDICE DE COLONNE DANS LE POKEDEX
Numéro	0
Nom	1
Points de vie	4
Niveau d'attaque	5
Niveau de défense	6
Niveau d'attaque spécial	7
Niveau de défense spécial	8
Vitesse	9
Génération	10

Vient ensuite la fonction principale de notre application, permettant de prédire qui des deux Pokémons combattants sera le vainqueur.

```
def prediction (numeroPokemon1, numeroPokemon2, Pokedex):
    pokemon1 = rechercheInformationsPokemon (numeroPokemon1,
    Pokedex)
    pokemon2 = rechercheInformationsPokemon (numeroPokemon2,
    Pokedex)
    modele_prediction = joblib.load('modele/modele_pokemon.mod')
    prediction_Pokemon_1 =
    modele_prediction.predict([[pokemon1[2], pokemon1[3], pokemon1[4],
    pokemon1[5], pokemon1[6], pokemon1[7], pokemon1[8]]])
    prediction_Pokemon_2 =
    modele_prediction.predict([[pokemon2[2], pokemon2[3],
    pokemon2[4], pokemon2[5], pokemon2[6], pokemon2[7],
    pokemon2[8]]])
    print ("COMBAT OPPOSANT : (" + str(numeroPokemon1) + ")
    "+ pokemon1[1] + " à (" + str(numeroPokemon2) + ") " + pokemon2[1])
    print ("    " + pokemon1[1] + ": " + str(prediction_Pokemon_1[0]))
    print("    " + pokemon2[1] + ": " +
    str(prediction_Pokemon_2[0]))
    print ("")
```

```

if prediction_Pokemon_1 > prediction_Pokemon_2:
    print(pokemon1[1].upper() + " EST LE VAINQUEUR !")
else:
    print(pokemon2[1].upper() + " EST LE VAINQUEUR !")

```

Cette fonction prend en paramètres :

- le numéro du premier Pokémon
- le numéro du second Pokémon
- le Pokédex

La première action de notre fonction est de rechercher les informations sur les deux Pokémon.

Viennent ensuite le chargement du modèle et la prédiction pour chaque Pokémon :

```

modele_prediction = joblib.load('modele/modele_pokemon.mod')

modele_prediction.predict([[pokemon1[2], pokemon1[3], pokemon1[4],
pokemon1[5], pokemon1[6], pokemon1[7], pokemon1[8]]])

```

On compare ensuite le pourcentage de victoire prédit pour en déduire le vainqueur du combat.

Vient enfin le moment de saisir les dernières lignes de code, en chargeant le fichier Pokédex, en excluant sa première ligne contenant les noms des différentes colonnes (next (pokedex)) et en appelant la fonction de prédiction.

```

#Chargement du Pokédex et lancement d'un combat
with open("datas/pokedex.csv", newline='') as csvfile:
    pokedex = csv.reader(csvfile)
    next(pokedex)
    prediction(368, 598, pokedex)

```

Voici le résultat de ce script :

```

COMBAT OPPOSANT : (368) Mangriff à (598) Crapustule
Mangriff: 0.7008019808073136
Crapustule: 0.5924562022360302

MANGRIFF EST LE VAINQUEUR !

```

Il vous est à présent possible de prédire le vainqueur de chaque combat de Pokémons ! N'hésitez pas à utiliser le contenu du fichier `combat.csv` comme source d'inspiration et surtout de vérification de la bonne prédiction.

11. Fin du cas d'étude

Nous voici à présent au terme du cas d'étude dédié au Pokémon. Celui-ci nous a permis dans sa première partie de découvrir les étapes de préparation de données pour la résolution d'un problème lié au Machine Learning supervisé.

Dans la seconde partie, nous nous sommes attardés sur l'analyse plus fine des données afin de pouvoir déterminer celles ayant une incidence forte (corrélation) sur la résolution de notre problème. Une fois cette étape réalisée, nous avons émis des hypothèses sur la prédiction de la victoire d'un Pokémon par rapport à son adversaire et vérifié celles-ci.

Nous avons ensuite testé divers algorithmes de prédiction, liés à la régression (car nous cherchions à prédire une valeur) afin de déterminer celui qui nous permettra d'obtenir un modèle de prédiction fiable. Modèle que nous avons ensuite utilisé dans une application.

Ce qu'il faut retenir de cette expérience est que la préparation et l'analyse des données est une phase, si ce n'est la phase la plus importante d'un projet de Machine Learning. Des données bien préparées et de bonne qualité permettront de réaliser de bonnes prédictions.

Gardez également en tête les notions de surapprentissage et de sous-apprentissage qui vous seront utiles dans vos recherches d'algorithmes pouvant répondre à vos problèmes de Machine Learning. Un bon algorithme est un algorithme qui produit des résultats généralisables.

Nous allons à présent découvrir l'application des algorithmes de classification à travers un cas pratique... explosif !

Chapitre 7

Bien classer n'est pas une option

1. Ce que nous allons découvrir et prérequis

Dans l'étude de cas précédente consacrée à la régression, nous avons évoqué l'univers imaginaire des Pokémons. À présent retour à la réalité. Le cas d'étude que nous allons suivre tout au long de ce chapitre va nous permettre de classer un signal d'un sonar embarqué à bord d'un navire afin de déterminer si l'objet détecté est une mine ou bien un rocher. Autant dire que nous n'avons pas le droit à l'erreur !

■ Remarque

Prérequis nécessaires pour bien aborder ce chapitre : avoir lu les chapitres Les fondamentaux du langage Python, Des statistiques pour comprendre les données et Principaux algorithmes du Machine Learning.

2. Origines et source du jeu d'observations

Le jeu d'observations que nous allons utiliser est issu des travaux de recherche de Terry Sejnowski et R. Paul Gorman en 1988 visant à déterminer le paramétrage d'un réseau de neurones capable de déterminer si un objet sondé par un sonar est une mine ou bien un rocher.

Dans ce chapitre, nous n'allons pas encore aborder le sujet passionnant des réseaux de neurones. Nous allons dans un premier temps utiliser des algorithmes du Machine Learning pour essayer d'établir les meilleures classifications possible.

Avant d'entrer dans le vif du sujet, il est important de rappeler le fonctionnement d'un sonar afin de comprendre les données que nous utiliserons dans notre prédiction.

Le fonctionnement d'un sonar est assez simple. Dans le cas d'un sonar sous-marin actif (il existe des sonars passifs uniquement en écoute), l'antenne du sonar envoie, dans l'eau, une énergie électrique convertie en onde sonore à une fréquence donnée. Lorsque cette onde sonore rencontre un objet, elle rebondit sur celui-ci puis est captée à nouveau par l'antenne du sonar qui se charge de transformer l'onde sonore reçue en énergie électrique (signal électrique).

Le jeu d'observations que nous allons utiliser contient pour chaque prédiction (mine ou rocher), 60 mesures. Chaque mesure correspondant à l'intensité du signal électrique reçu dans une bande de fréquences sonores particulière.

Pour obtenir ces informations, un cylindre métallique, en guise de mine, et des rochers, ont été bombardés par les ondes d'un sonar dans différents angles et différentes conditions et différentes fréquences. Chaque onde sonore de réflexion, transformée dès sa réception par le sonar en signal électrique, est ensuite enregistrée selon le couple (fréquence, intensité du signal).

3. Un problème de classification et algorithmes de prédiction associés

Notre problème consiste à prédire si l'objet repéré par le sonar est un rocher ou une mine. Il n'y a donc que deux options possibles nous permettant de classer notre prédiction dans l'une ou l'autre catégorie. Il s'agit donc bien d'un problème de classification.

Nous pouvons à présent lister les algorithmes de prédiction candidats à la résolution de problème de classification évoqués dans le chapitre Principaux algorithmes du Machine Learning :

- La régression logistique
- Les arbres de décision
- Les forêts aléatoires
- Les K plus proches voisins
- Les machines à vecteurs de supports

4. Démarche de résolution du problème

4.1 Définition du problème à résoudre

Le problème que nous devons résoudre a été identifié en début de chapitre et consiste à déterminer si l'objet détecté par un sonar embarqué à bord d'un navire est une mine ou un rocher.

Création d'un nouveau projet Python

Il va de soi que nous allons avoir besoin d'un nouveau projet Python pour répondre à notre problème. Nous vous laissons le soin de choisir le nom pour ce nouveau projet ainsi que le nom du fichier de script qui contiendra toutes les instructions.

En complément, nous vous invitons à télécharger et installer les modules suivants (comme nous avons procédé dans les chapitres précédents) :

- Pandas
- Numpy
- Matplotlib
- Scikit-learn

4.2 Acquisition des données d'apprentissage

Le jeu d'observation que nous allons utiliser est bien entendu téléchargeable sur le site de l'éditeur.

Une fois le jeu d'observation obtenu, il convient à présent de référencer ce fichier dans notre projet en créant un nouveau répertoire nommé `datas` et en plaçant le fichier dans ce nouveau répertoire.

Via les instructions ci-dessous, nous sommes alors en capacité de charger ce fichier dans un `Dataframe` du module `Pandas` en vue de l'analyser par la suite :

```
import pandas as pnd
observations = pnd.read_csv("datas/sonar.all-data.csv")
```

4.3 Préparer et nettoyer les données

4.3.1 De quelles données disposons-nous ?

Pour connaître les données dont nous disposons et leur signification, nous allons utiliser ces quelques lignes :

```
print(observations.columns.values)
```

Ayant pour résultat :

```
['0.0200' '0.0371' '0.0428' '0.0207' '0.0954' '0.0986' '0.1539' '0.1601'
'0.3109' '0.2111' '0.1609' '0.1582' '0.2238' '0.0645' '0.0660' '0.2273'
'0.3100' '0.2999' '0.5078' '0.4797' '0.5783' '0.5071' '0.4328' '0.5550'
'0.6711' '0.6415' '0.7104' '0.8080' '0.6791' '0.3857' '0.1307' '0.2604'
'0.5121' '0.7547' '0.8537' '0.8507' '0.6692' '0.6097' '0.4943' '0.2744']
```

```
'0.0510' '0.2834' '0.2825' '0.4256' '0.2641' '0.1386' '0.1051' '0.1343'
'0.0383' '0.0324' '0.0232' '0.0027' '0.0065' '0.0159' '0.0072' '0.0167'
'0.0180' '0.0084' '0.0090' '0.0032' 'R']
```

Cela nous montre que notre jeu d'observations ne contient pas d'en-tête ! Car la première observation est directement affichée. À nous donc de supposer les libellés des différentes features.

Par déduction, on peut déjà affirmer que la dernière feature correspond à l'objet détecté (R pour "Rocher", M pour "Mine"), pour les autres features, celles-ci correspondent aux 60 signaux électriques.

4.3.2 De combien de données disposons-nous ?

Pour connaître le nombre d'observations et le nombre de features, nous allons utiliser la fonction `shape` de notre Dataframe :

```
print (observations.shape)
```

Nous pouvons en déduire à la lecture du résultat issu de cette commande que nous disposons de 208 observations comportant 62 features :

```
(207, 61)
```

Remarque

Nous devons ajouter 1 au nombre d'observations affiché, car la numérotation des numéros d'observations commence à la valeur 0.

Ce qui confirme notre hypothèse sur la nature de chaque feature. Nous pouvons alors leur donner un intitulé en modifiant la fonction d'ouverture du fichier CVS et son stockage dans le Dataframe :

```
observations = pd.read_csv("datas/sonar.all-data.csv",
names=["F1", "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9",
F10", "F11", "F12", "F13", "F14", "F15", "F16", "F17", "F18", "F19",
"F20", "F21", "F22", "F23", "F24", "F25", "F26", "F27", "F28", "F29",
"F30", "F31", "F32", "F33", "F34", "F35", "F36", "F37", "F38", "F39",
"F40", "F41", "F42", "F43", "F44", "F45", "F46", "F47", "F48", "F49",
```

```
"F50", "F51", "F52", "F53", "F54", "F55", "F56", "F57", "F58", "F59",
"F60", "OBJET"])
```

Nous avons choisi de nommer chaque fréquence par la lettre F associée à un numéro. Quant à la dernière colonne, nous l'avons nommée OBJET.

Notons que nous disposons d'un petit jeu d'observations (208) comparé à celui que nous avons utilisé pour le cas d'étude sur les Pokémons (50000).

Cela peut avoir un impact sur la précision de la classification, car le jeu d'observations est assez faible, mais en contrepartie, cela permet une rapidité d'apprentissage.

4.3.3 Affichage des 10 premières observations

En affichant les 10 premières observations, nous allons pouvoir définir la nature des différentes features : sont-elles de nature catégorielle ou numérique ? Ce qui nous permettra d'identifier les données susceptibles d'être importantes pour notre apprentissage.

Étant donné que nous disposons de 61 caractéristiques pour chaque observation et que Pandas n'en affiche que 5 par défaut, nous allons donc lui supprimer sa limite d'affichage :

```
#Désactivation du nombre maximum de colonnes du Dataframe à
afficher
pnd.set_option('display.max_columns', None)
```

Une fois cette étape réalisée, nous pouvons alors procéder à l'affichage des 10 premières observations :

```
#Affichage des 10 premières observations
print(observations.head(10))
```

	F1	F2	F3	F4	F5	F6	F7	F8	F9	\
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	
5	0.0286	0.0453	0.0277	0.0174	0.0384	0.0990	0.1201	0.1833	0.2105	
6	0.0317	0.0956	0.1321	0.1408	0.1674	0.1710	0.0731	0.1401	0.2083	

7	0.0519	0.0548	0.0842	0.0319	0.1158	0.0922	0.1027	0.0613	0.1465
8	0.0223	0.0375	0.0484	0.0475	0.0647	0.0591	0.0753	0.0098	0.0684
9	0.0164	0.0173	0.0347	0.0070	0.0187	0.0671	0.1056	0.0697	0.0962

	F10	F11	F12	F13	F14	F15	F16	F17	F18 \
0	0.2111	0.1609	0.1582	0.2238	0.0645	0.0660	0.2273	0.3100	0.2999
1	0.2872	0.4918	0.6552	0.6919	0.7797	0.7464	0.9444	1.0000	0.8874
2	0.6194	0.6333	0.7060	0.5544	0.5320	0.6479	0.6931	0.6759	0.7551
3	0.1264	0.0881	0.1992	0.0184	0.2261	0.1729	0.2131	0.0693	0.2281
4	0.4459	0.4152	0.3952	0.4256	0.4135	0.4528	0.5326	0.7306	0.6193
5	0.3039	0.2988	0.4250	0.6343	0.8198	1.0000	0.9988	0.9508	0.9025
6	0.3513	0.1786	0.0658	0.0513	0.3752	0.5419	0.5440	0.5150	0.4262
7	0.2838	0.2802	0.3086	0.2657	0.3801	0.5626	0.4376	0.2617	0.1199
8	0.1487	0.1156	0.1654	0.3833	0.3598	0.1713	0.1136	0.0349	0.3796
9	0.0251	0.0801	0.1056	0.1266	0.0890	0.0198	0.1133	0.2826	0.3234

	F19	F20	F21	F22	F23	F24	F25	F26	F27 \
0	0.5078	0.4797	0.5783	0.5071	0.4328	0.5550	0.6711	0.6415	0.7104
1	0.8024	0.7818	0.5212	0.4052	0.3957	0.3914	0.3250	0.3200	0.3271
2	0.8929	0.8619	0.7974	0.6737	0.4293	0.3648	0.5331	0.2413	0.5070
3	0.4060	0.3973	0.2741	0.3690	0.5556	0.4846	0.3140	0.5334	0.5256
4	0.2032	0.4636	0.4148	0.4292	0.5730	0.5399	0.3161	0.2285	0.6995
5	0.7234	0.5122	0.2074	0.3985	0.5890	0.2872	0.2043	0.5782	0.5389
6	0.2024	0.4233	0.7723	0.9735	0.9390	0.5559	0.5268	0.6826	0.5713
7	0.6676	0.9402	0.7832	0.5352	0.6809	0.9174	0.7613	0.8220	0.8872
8	0.7401	0.9925	0.9802	0.8890	0.6712	0.4286	0.3374	0.7366	0.9611
9	0.3238	0.4333	0.6068	0.7652	0.9203	0.9719	0.9207	0.7545	0.8289

	F28	F29	F30	F31	F32	F33	F34	F35	F36 \
0	0.8080	0.6791	0.3857	0.1307	0.2604	0.5121	0.7547	0.8537	0.8507
1	0.2767	0.4423	0.2028	0.3788	0.2947	0.1984	0.2341	0.1306	0.4182
2	0.8533	0.6036	0.8514	0.8512	0.5045	0.1862	0.2709	0.4232	0.3043
3	0.2520	0.2090	0.3559	0.6260	0.7340	0.6120	0.3497	0.3953	0.3012
4	1.0000	0.7262	0.4724	0.5103	0.5459	0.2881	0.0981	0.1951	0.4181
5	0.3750	0.3411	0.5067	0.5580	0.4778	0.3299	0.2198	0.1407	0.2856
6	0.5429	0.2177	0.2149	0.5811	0.6323	0.2965	0.1873	0.2969	0.5163
7	0.6091	0.2967	0.1103	0.1318	0.0624	0.0990	0.4006	0.3666	0.1050
8	0.7353	0.4856	0.1594	0.3007	0.4096	0.3170	0.3305	0.3408	0.2186
9	0.8907	0.7309	0.6896	0.5829	0.4935	0.3101	0.0306	0.0244	0.1108

	F37	F38	F39	F40	F41	F42	F43	F44	F45 \
0	0.6692	0.6097	0.4943	0.2744	0.0510	0.2834	0.2825	0.4256	0.2641
1	0.3835	0.1057	0.1840	0.1970	0.1674	0.0583	0.1401	0.1628	0.0621
2	0.6116	0.6756	0.5375	0.4719	0.4647	0.2587	0.2129	0.2222	0.2111
3	0.5408	0.8814	0.9857	0.9167	0.6121	0.5006	0.3210	0.3202	0.4295
4	0.4604	0.3217	0.2828	0.2430	0.1979	0.2444	0.1847	0.0841	0.0692
5	0.3807	0.4158	0.4054	0.3296	0.2707	0.2650	0.0723	0.1238	0.1192
6	0.6153	0.4283	0.5479	0.6133	0.5017	0.2377	0.1957	0.1749	0.1304
7	0.1915	0.3930	0.4288	0.2546	0.1151	0.2196	0.1879	0.1437	0.2146
8	0.2463	0.2726	0.1680	0.2792	0.2558	0.1740	0.2121	0.1099	0.0985
9	0.1594	0.1371	0.0696	0.0452	0.0620	0.1421	0.1597	0.1384	0.0372

	F46	F47	F48	F49	F50	F51	F52	F53	F54 \
0	0.1386	0.1051	0.1343	0.0383	0.0324	0.0232	0.0027	0.0065	0.0159
1	0.0203	0.0530	0.0742	0.0409	0.0061	0.0125	0.0084	0.0089	0.0048
2	0.0176	0.1348	0.0744	0.0130	0.0106	0.0033	0.0232	0.0166	0.0095
3	0.3654	0.2655	0.1576	0.0681	0.0294	0.0241	0.0121	0.0036	0.0150
4	0.0528	0.0357	0.0085	0.0230	0.0046	0.0156	0.0031	0.0054	0.0105
5	0.1089	0.0623	0.0494	0.0264	0.0081	0.0104	0.0045	0.0014	0.0038
6	0.0597	0.1124	0.1047	0.0507	0.0159	0.0195	0.0201	0.0248	0.0131
7	0.2360	0.1125	0.0254	0.0285	0.0178	0.0052	0.0081	0.0120	0.0045
8	0.1271	0.1459	0.1164	0.0777	0.0439	0.0061	0.0145	0.0128	0.0145
9	0.0688	0.0867	0.0513	0.0092	0.0198	0.0118	0.0090	0.0223	0.0179

	F55	F56	F57	F58	F59	F60 OBJET
0	0.0072	0.0167	0.0180	0.0084	0.0090	0.0032 R
1	0.0094	0.0191	0.0140	0.0049	0.0052	0.0044 R
2	0.0180	0.0244	0.0316	0.0164	0.0095	0.0078 R
3	0.0085	0.0073	0.0050	0.0044	0.0040	0.0117 R
4	0.0110	0.0015	0.0072	0.0048	0.0107	0.0094 R
5	0.0013	0.0089	0.0057	0.0027	0.0051	0.0062 R
6	0.0070	0.0138	0.0092	0.0143	0.0036	0.0103 R
7	0.0121	0.0097	0.0085	0.0047	0.0048	0.0053 R
8	0.0058	0.0049	0.0065	0.0093	0.0059	0.0022 R
9	0.0084	0.0068	0.0032	0.0035	0.0056	0.0040 R

La première colonne correspond au numéro d'observation. Viennent ensuite les 60 mesures qui sont de type numérique continu. Concernant la feature OBJET, celle-ci est une chaîne de caractères et semble correspondre à notre hypothèse de départ à savoir, catégoriser l'observation : R faisant référence à un Rocher et nous le verrons plus loin, M correspondant à une Mine.

4.3.4 Transformation de la feature OBJET

Comme évoqué depuis le début de cet ouvrage, l'analyse des données de type texte reste compliquée. Par conséquent, nous allons devoir transformer la caractéristique OBJET de nos observations en établissant arbitrairement la règle suivante :

- Si l'objet est une mine, alors il prendra la valeur 1
- Si l'objet est un rocher, alors il prendra la valeur 0

Cette transformation s'opère en utilisant la ligne de code suivante :

```
observations['OBJET'] = (observations['OBJET']=='M').astype(int)
```

Ainsi, si la feature OBJET a pour valeur M, alors elle prend la valeur 1 (astype(int)).

4.3.5 Manque-t-il des données?

Après avoir transformé la caractéristique OBJET de nos observations, nous devons nous assurer qu'il ne manque aucune donnée.

Pour ce faire, nous allons utiliser la fonction `info()` de notre Dataframe :

```
print(observations.info())
```

Nous obtenons le résultat suivant :

```
RangeIndex: 208 entries, 0 to 207
```

```
Data columns (total 61 columns):
```

```
F1      208 non-null float64
```

```
F2      208 non-null float64
```

```
F3      208 non-null float64
```

```
F4      208 non-null float64
```

```
F5      208 non-null float64
```

```
F6      208 non-null float64
```

```
F7      208 non-null float64
```

```
F8      208 non-null float64
```

```
F9      208 non-null float64
```

```
F10     208 non-null float64
```

```
F11     208 non-null float64
```

```
F12     208 non-null float64
```

```
F13     208 non-null float64
```

```
F14     208 non-null float64
```

```
F15     208 non-null float64
```

```
F16     208 non-null float64
```

```
F17     208 non-null float64
```

```
F18     208 non-null float64
```

```
F19     208 non-null float64
```

```
F20     208 non-null float64
```

```
F21     208 non-null float64
```

```
F22     208 non-null float64
```

```
F23     208 non-null float64
```

```
F24     208 non-null float64
```

```
F25     208 non-null float64
```

```
F26     208 non-null float64
```

```
F27     208 non-null float64
```

```
F28     208 non-null float64
```

```
F29     208 non-null float64
```

```
30      208 non-null float64
```

```
F31     208 non-null float64
```

```
F32     208 non-null float64
```

```

F33      208 non-null float64
F34      208 non-null float64
F35      208 non-null float64
F36      208 non-null float64
F37      208 non-null float64
F38      208 non-null float64
F39      208 non-null float64
F40      208 non-null float64
F41      208 non-null float64
F42      208 non-null float64
F43      208 non-null float64
F44      208 non-null float64
F45      208 non-null float64
F46      208 non-null float64
F47      208 non-null float64
F48      208 non-null float64
F49      208 non-null float64
F50      208 non-null float64
F51      208 non-null float64
F52      208 non-null float64
F53      208 non-null float64
F54      208 non-null float64
F55      208 non-null float64
F56      208 non-null float64
F57      208 non-null float64
F58      208 non-null float64
F59      208 non-null float64
F60      208 non-null float64

```

```
OBJET    208 non-null int32
```

```
dtypes: float64(60), int32(1)
```

```
memory usage: 98.4 KB
```

```
None
```

Nous disposons donc au total de 208 observations (0 to 207 entries). En analysant le nombre d'observations par feature (F1, F2...), on peut en déduire qu'il ne manque aucune donnée. C'est donc une bonne nouvelle. Nous pouvons à présent passer à la phase d'analyse et à l'exploration des données.

4.4 Analyser et explorer les données

4.4.1 Combien de mines et combien de rochers?

Essayons de répondre à cette question : *Disposons-nous d'un jeu d'observation ayant une distribution équilibrée, à savoir autant de mines que de rochers?*

Ci-dessous le code comptant le nombre d'éléments groupés par type (0 ou 1) de la caractéristique OBJET, qui nous permettra de répondre à cette question :

```
print(observations.groupby("OBJET").size())
```

La réponse obtenue est la suivante :

```
OBJET
0      97
1     111
dtype: int64
```

On constate donc que nous avons 111 mines (type 1) et 97 rochers (type 0).

Il n'y a donc pas une grande différence entre le nombre de mines et de rochers pouvant impacter le modèle d'apprentissage. En effet, si nous avons un nombre prédominant de mines ou de rochers, l'apprentissage n'aurait pas pu se faire dans de bonnes conditions.

4.4.2 Moyenne, écart type, min, max et quartiles

Calculons à présent les différents indicateurs qui vont nous permettre de mieux comprendre nos données.

Pour cela, utilisons la fonction `describe()` du module Pandas :

```
print(observations.describe())
```

	F1	F2	F3	F4	F5	F6 \
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000
mean	0.029164	0.038437	0.043832	0.053892	0.075202	0.104570
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300

	F7	F8	F9	F10	F11	F12 \
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000
mean	0.121747	0.134799	0.178003	0.208259	0.236013	0.250221
std	0.061788	0.085152	0.118387	0.134416	0.132705	0.140072
min	0.003300	0.005500	0.007500	0.011300	0.028900	0.023600
25%	0.080900	0.080425	0.097025	0.111275	0.129250	0.133475
50%	0.106950	0.112100	0.152250	0.182400	0.224800	0.249050
75%	0.154000	0.169600	0.233425	0.268700	0.301650	0.331250
max	0.372900	0.459000	0.682800	0.710600	0.734200	0.706000

Nous avons à présent toutes les mesures de moyenne, d'écart type, de minimum et de maximum pour l'ensemble des caractéristiques. Cependant, il faut être honnête, cela ne nous aide pas beaucoup dans la compréhension et l'analyse des données. Car cela reste une multitude de chiffres...

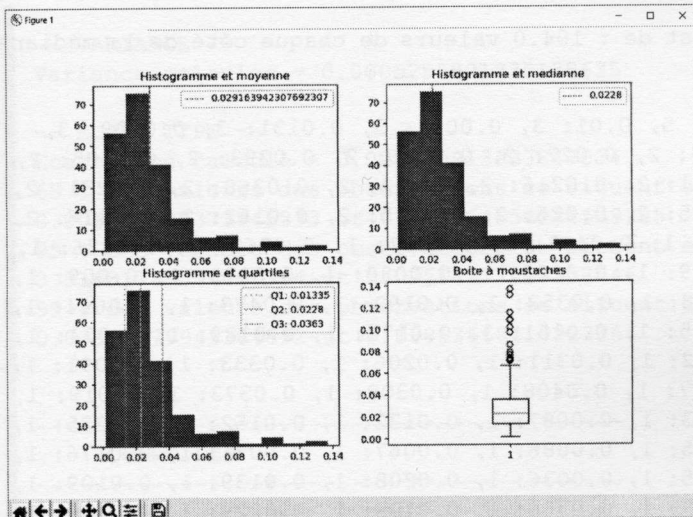
4.4.3 À la recherche des valeurs extrêmes

Comme nous l'avons vu dans le chapitre Des statistiques pour comprendre les données (eh oui, encore lui!), l'utilisation des indicateurs de tendance permet d'identifier les différentes données extrêmes, pouvant après analyse être identifiées comme des données aberrantes.

Cependant, nous venons de voir que les données obtenues précédemment sont complexes à analyser. Nous allons donc faire usage de graphiques qui pourront nous aider dans la détection de ces valeurs extrêmes et également mieux comprendre nos données.

Nous allons dans un premier temps utiliser la librairie `JMPStatistiques` (vue également dans le chapitre Des statistiques pour comprendre les données). En effet, grâce à cette librairie, nous pouvons comprendre comment les calculs sont effectués. Pour l'utiliser, nous devons importer le fichier `JMPStatistiques.py` dans le projet et l'utiliser comme suit dans notre code pour déterminer les valeurs extrêmes de la fréquence n°1 :

```
#Recherche des valeurs extrêmes à l'aide de la librairie
JMPStatistiques
import JMPStatistiques as jmp
stats = jmp.JMPStatistiques(observations['F1'])
stats.analyseFeature()
```



Utilisation de la classe JMPStatistiques pour rechercher les valeurs extrêmes

MESURE DE TENDANCE CENTRALE

-- NOMBRE D'OBSERVATIONS --

Nombre d'observations = 208

-- MIN --

Valeur minimale : 0.0015

-- MAX --

Valeur maximale : 0.1371

-- MOYENNE --

Moyenne arithmétique calculée = 0.029163942307692307

> Observations : Si les observations avaient toute la même valeur (répartition équitable) celle-ci serait de 0.029163942307692307

-- MEDIANE --

Le nombre d'observations est pair.

RANG = 104.0

Mediane calculée = 0.0228

> Observations : La valeur se trouvant au milieu des observations

est de :0.0228

La répartition est de : 104.0 valeurs de chaque côté de la médiane

-- MODE --

```
Counter({0.0201: 5, 0.01: 3, 0.0094: 3, 0.0131: 3, 0.0209: 3,
0.0164: 2, 0.0123: 2, 0.027: 2, 0.0126: 2, 0.0293: 2, 0.0195: 2,
0.0093: 2, 0.0211: 2, 0.0216: 2, 0.013: 2, 0.0368: 2, 0.0231: 2,
0.0217: 2, 0.0235: 2, 0.026: 2, 0.0335: 2, 0.0162: 2, 0.0116: 2,
0.0228: 2, 0.02: 1, 0.0453: 1, 0.0262: 1, 0.0762: 1, 0.0286: 1,
0.0317: 1, 0.0519: 1, 0.0223: 1, 0.0039: 1, 0.0079: 1, 0.009: 1,
0.0124: 1, 0.0298: 1, 0.0352: 1, 0.0192: 1, 0.0473: 1, 0.0664: 1,
0.0099: 1, 0.0115: 1, 0.0151: 1, 0.0177: 1, 0.0189: 1, 0.024: 1,
0.0084: 1, 0.0442: 1, 0.0311: 1, 0.0206: 1, 0.0333: 1, 0.0091: 1,
0.0068: 1, 0.0257: 1, 0.0408: 1, 0.0308: 1, 0.0373: 1, 0.019: 1,
0.0119: 1, 0.0353: 1, 0.0087: 1, 0.0132: 1, 0.0152: 1, 0.0225: 1,
0.0125: 1, 0.0135: 1, 0.0086: 1, 0.0067: 1, 0.0071: 1, 0.0176: 1,
0.0265: 1, 0.0065: 1, 0.0036: 1, 0.0208: 1, 0.0139: 1, 0.0109: 1,
0.0202: 1, 0.0239: 1, 0.0336: 1, 0.0108: 1, 0.0229: 1, 0.0409: 1,
0.0378: 1, 0.0365: 1, 0.0188: 1, 0.0856: 1, 0.0274: 1, 0.0253: 1,
0.0459: 1, 0.0025: 1, 0.0291: 1, 0.0181: 1, 0.0491: 1, 0.1313: 1,
0.0629: 1, 0.0587: 1, 0.0307: 1, 0.0331: 1, 0.0428: 1, 0.0599: 1,
0.0264: 1, 0.021: 1, 0.053: 1, 0.0454: 1, 0.0283: 1, 0.0114: 1,
0.0414: 1, 0.0363: 1, 0.0261: 1, 0.0346: 1, 0.0249: 1, 0.0388: 1,
0.0715: 1, 0.0374: 1, 0.1371: 1, 0.0443: 1, 0.115: 1, 0.0968: 1,
0.079: 1, 0.1083: 1, 0.1088: 1, 0.043: 1, 0.0731: 1, 0.0412: 1,
0.0707: 1, 0.0526: 1, 0.0516: 1, 0.0299: 1, 0.0721: 1, 0.1021: 1,
0.0654: 1, 0.0712: 1, 0.0207: 1, 0.0233: 1, 0.0117: 1, 0.0047: 1,
0.0107: 1, 0.0258: 1, 0.0305: 1, 0.0072: 1, 0.0163: 1, 0.0221: 1,
0.0411: 1, 0.0137: 1, 0.0015: 1, 0.0134: 1, 0.0179: 1, 0.018: 1,
0.0329: 1, 0.0191: 1, 0.0294: 1, 0.0635: 1, 0.0197: 1, 0.0394: 1,
0.031: 1, 0.0423: 1, 0.0095: 1, 0.0096: 1, 0.0269: 1, 0.034: 1,
0.0089: 1, 0.0158: 1, 0.0156: 1, 0.0315: 1, 0.0056: 1, 0.0203: 1,
0.0392: 1, 0.0129: 1, 0.005: 1, 0.0366: 1, 0.0238: 1, 0.0272: 1,
0.0187: 1, 0.0323: 1, 0.0522: 1, 0.0303: 1})
```

> Observations : le mode permet de déterminer les valeurs les plus souvent observées

MESURE DE DISPERSION

-- ETENDUE --

Etendue de la série = 0.1356

```
-- VARIANCE --
Variance calculée = 0.0005285821235135635

-- ECART TYPE --
Ecart type calculé = 0.022990913933847074
68 % des valeurs des observations se situent entre
0.006173028373845233 et 0.052154856241539385
95 % des valeurs des observations se situent entre -
0.01681788556000184 et 0.07514577017538646
99 % des valeurs des observations se situent entre -
0.03980879949384892 et 0.09813668410923354
```

QUARTILES

```
25% des observations ont une valeur inférieure à 0.01335
50% des observations ont une valeur inférieure à 0.0228
75% des observations ont une valeur inférieure à 0.0363
```

DETECTION VALEURS EXTREMES

```
> Critères de Tukey
Inter-quartile = 0.022949999999999998
Nombre de valeurs extremes : 14
Valeurs : [0.0712, 0.0715, 0.0721, 0.0731, 0.0762, 0.079, 0.0856,
0.0968, 0.1021, 0.1083, 0.1088, 0.115, 0.1313, 0.1371]
```

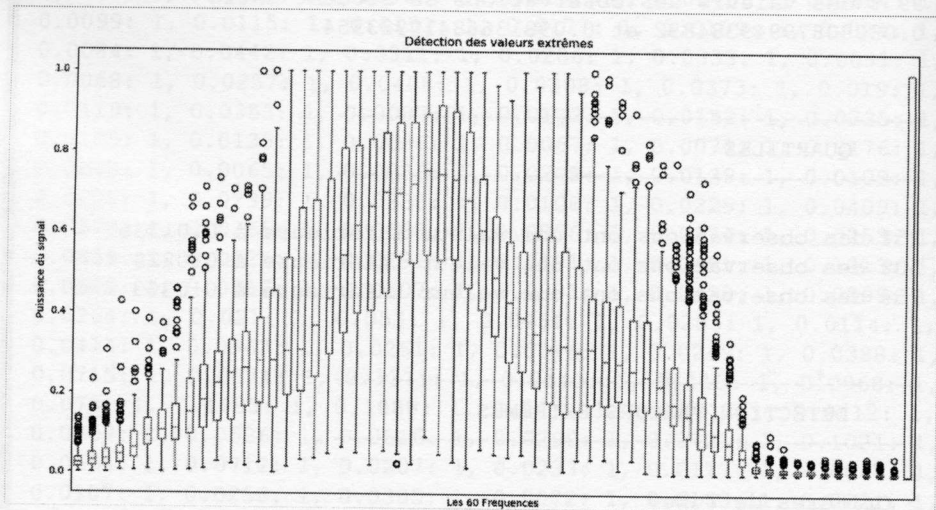
On constate donc que la fréquence 1 nommée F1 de notre jeu d'observations comporte des valeurs extrêmes, car 50 % de ses valeurs sont comprises entre 0.01335 et 0,0363 MHz. Il faut à présent analyser les 59 autres fréquences !

Bien entendu, utiliser notre librairie s'avère fastidieux, car nous allons devoir réaliser 59 fois la même opération. Pour pallier cela, nous allons utiliser la fonction `box` du module `Matplotlib` :

```
#Import du module Matplotlib
from matplotlib import pyplot as plt
```

```
#Création du graphique boîtes à moustaches
#XTicks [] permet de ne pas afficher les libellés des différentes
fréquences : "F1,F2...."
observations.plot.box(figsize=(20,10), xticks=[])

#Information concernant le graphique
plt.title('Détection des valeurs extrêmes')
plt.xlabel('Les 60 Frequences')
plt.ylabel('Puissance du signal')
plt.show()
```



Détection des valeurs extrêmes à l'aide du module Matplotlib

La figure précédente montre que, pour certaines fréquences, il existe des valeurs extrêmes, symbolisées par les points noirs situés en dehors des boîtes à moustaches, qu'il nous faudra alors traiter.

4.4.4 Traitement des valeurs extrêmes

Que faut-il faire des valeurs extrêmes? En effet, ces données peuvent entraîner de mauvaises prédictions. Cependant, les supprimer n'est pas la solution. Une fois le modèle d'apprentissage choisi et l'apprentissage réalisé, il faut en effet constater les erreurs commises et valider l'existence d'un lien entre ces erreurs et les valeurs aberrantes. Si un lien évident apparaît, il faut alors mettre des actions de traitement de ces valeurs extrême. Nous traiterons donc ce point une fois l'apprentissage réalisé.

4.5 Choix d'un modèle de prédiction et résolution du problème

Nous voici à présent arrivés à la dernière étape : le choix d'un modèle de prédiction adapté à la résolution de notre problème.

4.5.1 Des données d'apprentissage et des données de tests

La première chose à faire est de scinder nos observations en données d'apprentissage et en données de tests qui permettront de valider le bon apprentissage de notre modèle.

Pour cela, nous allons utiliser la fonction `train_test_split` du module `Scikit-Learn`, qui va nous permettre de diviser de façon aléatoire nos données en 80 % de données d'apprentissage et 20 % de données de validation. Pour plus d'explications sur ce point, nous vous invitons à reparcourir le chapitre précédent dans lequel nous avons détaillé les différentes étapes de ce découpage.

```
from sklearn.model_selection import train_test_split
array = observations.values

#Conversion des données en type decimal
X = array[:,0:-1].astype(float)

#On choisit la dernière colonne comme feature de prédiction
Y = array[:, -1]

#Création des jeux d'apprentissage et de tests
```

```
percentage_donnees_test = 0.2
X_APPRENTISSAGE, X_VALIDATION, Y_APPRENTISSAGE, Y_VALIDATION =
train_test_split(X, Y, test_size=percentage_donnees_test,
random_state=42)
```

On note également la transformation des données d'apprentissage en type décimal et le choix de la dernière feature de notre jeu d'observation comme donnée de la labellisation.

L'usage du paramètre `random_state` dans la fonction de création des jeux de données permet quant à lui d'obtenir les mêmes données d'apprentissage et de tests à chaque exécution du script.

Voici un exemple permettant de mieux comprendre ce paramètre :

```
Apprentissage = [[1],[2],[3],[4]]
Validation = [[10],[11],[12],[13]]
print ("Apprentissage = "+str(Apprentissage))
print ("Validation = "+str(Validation))
print (" Sans random_state : ")
print(train_test_split(Apprentissage,Validation,test_size=0.2))
print (" Sans random_state : ")
print(train_test_split(Apprentissage,Validation,test_size=0.2))
print (" Avec random_state : ")
print(train_test_split(Apprentissage,Validation,test_size=0.2,
random_state=42))
print (" Avec random_state : ")
print(train_test_split(Apprentissage,Validation,test_size=0.2,
random_state=42))
```

Ce qui nous donne pour résultat :

```
Apprentissage = [[1], [2], [3], [4]]
Validation = [[10], [11], [12], [13]]

Sans random_state :
[[[1], [2], [3]], [[4]], [[10], [11], [12]], [[13]]]
Sans random_state :
[[[3], [1], [2]], [[4]], [[12], [10], [11]], [[13]]]

Avec random_state :
[[[4], [1], [3]], [[2]], [[13], [10], [12]], [[11]]]
Avec random_state :
[[[4], [1], [3]], [[2]], [[13], [10], [12]], [[11]]]
```

On constate que lorsqu'on n'utilise pas la propriété `random_state`, les jeux de données sont différents. Le chiffre 42, quant à lui, est choisi arbitrairement.

4.5.2 Test des algorithmes

Nous allons à présent tester les algorithmes évoqués en début de ce chapitre, liés à la résolution de problèmes de classification.

Pour chaque algorithme, nous ferons appel à la fonction `fit` réalisant l'apprentissage. Contrairement au chapitre précédent où, pour chaque algorithme de régression, nous avons utilisé la fonction `r2_score` pour calculer la précision de l'apprentissage (accuracy) sur les données de tests, pour les algorithmes de classification, nous devons utiliser la fonction `accuracy_score` comme le préconise la documentation Scikit-Learn (https://scikit-learn.org/stable/modules/model_evaluation.html).

Scoring	Function	Comment
Classification		
'accuracy'	<code>metrics.accuracy_score</code>	
'balanced_accuracy'	<code>metrics.balanced_accuracy_score</code>	
'average_precision'	<code>metrics.average_precision_score</code>	
'brier_score_loss'	<code>metrics.brier_score_loss</code>	
'f1'	<code>metrics.f1_score</code>	for binary targets
'f1_micro'	<code>metrics.f1_score</code>	micro-averaged
'f1_macro'	<code>metrics.f1_score</code>	macro-averaged
'f1_weighted'	<code>metrics.f1_score</code>	weighted average
'f1_samples'	<code>metrics.f1_score</code>	by multilabel sample
'neg_log_loss'	<code>metrics.log_loss</code>	requires <code>predict_proba</code> support
'precision' etc.	<code>metrics.precision_score</code>	suffixes apply as with 'f1'
'recall' etc.	<code>metrics.recall_score</code>	suffixes apply as with 'f1'
'jaccard' etc.	<code>metrics.jaccard_score</code>	suffixes apply as with 'f1'
'roc_auc'	<code>metrics.roc_auc_score</code>	
Clustering		
'adjusted_mutual_info_score'	<code>metrics.adjusted_mutual_info_score</code>	
'adjusted_rand_score'	<code>metrics.adjusted_rand_score</code>	
'completeness_score'	<code>metrics.completeness_score</code>	
'fowlkes_mallows_score'	<code>metrics.fowlkes_mallows_score</code>	
'homogeneity_score'	<code>metrics.homogeneity_score</code>	
'mutual_info_score'	<code>metrics.mutual_info_score</code>	
'normalized_mutual_info_score'	<code>metrics.normalized_mutual_info_score</code>	
'v_measure_score'	<code>metrics.v_measure_score</code>	
Regression		
'explained_variance'	<code>metrics.explained_variance_score</code>	
'max_error'	<code>metrics.max_error</code>	
'neg_mean_absolute_error'	<code>metrics.mean_absolute_error</code>	
'neg_mean_squared_error'	<code>metrics.mean_squared_error</code>	
'neg_mean_squared_log_error'	<code>metrics.mean_squared_log_error</code>	
'neg_median_absolute_error'	<code>metrics.median_absolute_error</code>	
'r2'	<code>metrics.r2_score</code>	

Choix de la fonction à utiliser pour connaître la précision (accuracy) de l'apprentissage des algorithmes de classification utilisés

```
#Import des algorithmes de type classifieur (classification)
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

#REGRESSION LOGISTIQUE
regression_logistique = LogisticRegression()
regression_logistique.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = regression_logistique.predict(X_VALIDATION)
print("Regression logistique: "+str(accuracy_score(predictions,
Y_VALIDATION)))

#ARBRE DE DECISION
arbre_decision = DecisionTreeClassifier()
arbre_decision.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = arbre_decision.predict(X_VALIDATION)
print("Arbre de décision: "+str(accuracy_score(predictions,
Y_VALIDATION)))

#FORETS ALEATOIRES
foret_aleatoire= RandomForestClassifier()
foret_aleatoire.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = foret_aleatoire.predict(X_VALIDATION)
print("Forêt aléatoire: "+str(accuracy_score(predictions,
Y_VALIDATION)))

#K PLUS PROCHES VOISINS
knn = KNeighborsClassifier()
knn.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = knn.predict(X_VALIDATION)
print("K plus proches voisins: "+str(accuracy_score(predictions,
Y_VALIDATION)))

#MACHINE VECTEURS DE SUPPORTS
SVM = SVC(gamma='auto')
SVM.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = SVM.predict(X_VALIDATION)
print("Machine vecteurs de supports:
"+str(accuracy_score(predictions, Y_VALIDATION)))
```

On constate que les deux algorithmes KNN et la régression logistique offrent de bons résultats. Ils se démarquent donc vis-à-vis des autres. Cependant, ces algorithmes ont été utilisés dans leur forme la plus simple c'est-à-dire sans aucun paramétrage.

Regression logistique: 0.8571428571428571

Arbre de décision: 0.6904761904761905

Forets aléatoire: 0.7142857142857143

K plus proches voisins: 0.8571428571428571

Machine vecteurs de supports: 0.8333333333333334

4.5.3 Optimisation

Nous allons à présent voir qu'avec un peu d'optimisation d'un des hyperparamètres de l'algorithme SCV (Machine à vecteurs de supports), nous pouvons changer la donne!

■ Remarque

Un paramètre est une donnée interne à l'algorithme et dont la valeur peut être mise à jour lors des différents apprentissages (par exemple, les poids dans un réseau de neurones). Un hyperparamètre est une donnée externe à l'algorithme ne pouvant être mise à jour lors des apprentissages (par exemple, le nombre voisin à considérer dans l'algorithme KNN).

Souvenez-vous, nous avons vu que l'algorithme machine à vecteurs de supports consistait à déterminer une frontière afin de séparer les observations en groupes distincts tout en maximisant la marge de séparation. Le but étant de trouver une marge la plus large possible avec un minimum d'erreurs de classification.

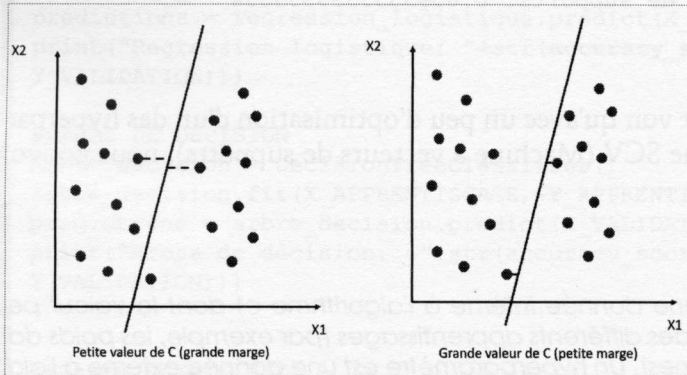
Voici le principe :

- Si on augmente la marge, on augmente le nombre d'erreurs possible de classification.
- Si on diminue la marge, on diminue le nombre d'erreurs de classifications.

Cependant, un nombre peu important d'erreurs sur les données d'apprentissage ne signifie pas forcément un nombre peu important d'erreurs sur le jeu de données de tests (rappel des notions de surapprentissage vues dans le chapitre précédent). Il faut donc chercher à généraliser ce nombre d'erreurs.

Pour cela, il existe pour l'algorithme SVM (Machine à Vecteurs de supports) un hyperparamètre nommé C qui pénalise les termes d'erreurs. L'algorithme SVM va chercher le meilleur compromis entre la maximisation de la marge et la minimisation des erreurs en utilisant ce paramètre afin de généraliser ses prédictions.

- Si la valeur C est importante, la marge est réduite.
- Si la valeur C est faible, la marge est large.



Utilisation de l'hyperparamètre C dans l'algorithme SVM

Pour trouver la valeur de l'hyperparamètre optimal pour notre exemple, nous allons utiliser la fonction `GridSearchCV` qui va tester un ensemble d'hyperparamètres et trouver celui comme étant le plus optimal :

```
from sklearn.model_selection import GridSearchCV

#Définition d'une plage de valeurs à tester
penalite = [{'C': range(1,100)}]

#Tests avec 5 échantillons de Validation Croisée
recherche_optimisations = GridSearchCV(SVC(), penalite, cv=5)
recherche_optimisations.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)

print("Le meilleur paramètre est :")
print()
print(recherche_optimisations.best_params_)
print()
```

Tout d'abord, on définit une plage de valeurs pour l'hyperparamètre C allant de 1 à 100. 1 étant la valeur par défaut de l'hyperparamètre utilisé par l'algorithme SVC.

On utilise ensuite la fonction `GridSearchCV` prenant en paramètres :

- L'algorithme à tester
- Les hyperparamètres à optimiser
- Le nombre de validations croisées

La fonction `GridSearchCV` va ensuite utiliser les données d'apprentissage pour trouver la meilleure optimisation des hyperparamètres en utilisant la méthode de validation croisée. C'est-à-dire qu'elle va, dans notre cas, découper les données d'apprentissage en 5 groupes ($CV=5$), déterminer une optimisation sur les quatre premiers groupes et tester sur le cinquième.

Une fois le script exécuté, nous avons pour réponse :

```
Le meilleur hyperparamètre est :  
{'C': 65}
```

Nous pouvons à présent saisir ces quelques lignes de code utilisant la valeur déterminée de notre hyperparamètre et exécuter à nouveau le script :

```
#MACHINE VECTEURS DE SUPPORTS OPTIMISEE  
SVM = SVC(C=65, gamma='auto')  
SVM.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)  
predictions = SVM.predict(X_VALIDATION)  
print("Machine à vecteurs de supports optimisée: "+str(accuracy_score(predictions, Y_VALIDATION)))
```

Ce qui nous donne après exécution du code le résultat suivant :

```
Machine à vecteurs de supports optimisée: 0.8809523809523809
```

On constate à présent que l'algorithme de Machine à vecteurs de supports surpasse les algorithmes de régression et des K plus proches voisins donnés vainqueurs précédemment !

4.5.4 Et si on boostait un peu tout ça ?

Terminons à présent en testant l'algorithme GradientBoosting afin de vérifier s'il offre de meilleures performances.

```
from sklearn.ensemble import GradientBoostingClassifier

#GRADIENT BOOSTING
gradientBoosting = GradientBoostingClassifier()
gradientBoosting.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = SVM.predict(X_VALIDATION)
print("GRADIENT BOOSTING: "+str(accuracy_score(predictions,
Y_VALIDATION)))
```

GRADIENT BOOSTING: 0.8809523809523809

Sans spécification d'hyperparamètres, on constate que le Gradient Boosting offre de bonnes prédictions et, dans notre cas, a des résultats identiques au SVM optimisé. Sa réputation de "Super algorithme" est donc tout à fait respectée.

4.5.5 Que faire des données extrêmes ?

Jusqu'à présent, nous avons utilisé un jeu d'observations contenant des données extrêmes pour réaliser notre apprentissage. Bien souvent, la première réaction est de les supprimer. Mais voyons si cela est bien raisonnable.

Nous allons reprendre nos données afin d'exclure les lignes contenant plus de sept caractéristiques ayant des valeurs supérieures ou inférieures à 1,5 fois leur interquartile (cf. chapitre Des statistiques pour comprendre les données).

Le code ci-dessous, accompagné des commentaires, nous explique la marche à suivre :

```
#Pour chaque caractéristique on cherche les numéros de lignes
correspondant à une donnée extrême
import numpy as np

#On crée une liste chargée de contenir les numéros de lignes
correspondant à une valeur extrême
```

```

num_lignes = []

#On parcourt toutes les 60 caractéristiques
for caractéristique in observations.columns.tolist():
    #Pour une caractéristique : calcul des percentiles
    Q1 = np.percentile(observations[caractéristique],25)
    Q3 = np.percentile(observations[caractéristique],75)
    #Calcul de la borne
    donnee_extreme = 1.5*(Q3-Q1)
    #Si la donnée est inférieure ou supérieure à la borne on
    récupère son numéro de ligne et on l'ajoute à la liste
    liste_donnees_extremes =
observations[(observations[caractéristique]<Q1-donnee_extreme) |
(observations[caractéristique]>Q3+donnee_extreme)].index
    num_lignes.extend(liste_donnees_extremes)

#On ordonne la liste par ordre croissant
num_lignes.sort()

#On crée une liste contenant les numéros de lignes à supprimer
num_lignes_a_supprimer=[]

#On parcourt l'ensemble des numéros de lignes
for ligne in num_lignes :
    #Pour une ligne, on récupère son numéro
    num_ligne = ligne
    #On calcule le nombre de fois où apparait ce numéro de ligne
    #dans l'ensemble des numéros de lignes
    nbr_valeurs_extremes = num_lignes.count(num_ligne)

    #Si le nombre d'erreurs est supérieur à 7 alors on ajoute le
    numéro de la
    #ligne à la liste des lignes à supprimer
    if (nbr_valeurs_extremes>7):
        num_lignes_a_supprimer.append(num_ligne)

#On supprime les doublons
num_lignes_a_supprimer = list(set(num_lignes_a_supprimer))

```

```
#On supprime ensuite les lignes dans le Dataframe
print(num_lignes_a_supprimer)
print("Nombre de lignes à supprimer =
"+str(len(num_lignes_a_supprimer)))
observations = observations.drop(num_lignes_a_supprimer,axis=0)
print()
print()

#Import des algorithmes et de la fonction de calcul du précision
#accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

#Suppression des erreurs de type warning
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

#REGRESSION LOGISTIQUE
regression_logistique = LogisticRegression()
regression_logistique.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = regression_logistique.predict(X_VALIDATION)
print("Regression logistique: "+str(accuracy_score(predictions,
Y_VALIDATION)))

#ARBRE DE DECISION
arbre_decision = DecisionTreeClassifier()
arbre_decision.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = arbre_decision.predict(X_VALIDATION)
print("Arbre de décision: "+str(accuracy_score(predictions,
Y_VALIDATION)))

#FORETS ALEATOIRES
foret_aleatoire= RandomForestClassifier()
foret_aleatoire.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = foret_aleatoire.predict(X_VALIDATION)
print("Forets aléatoires: "+str(accuracy_score(predictions,
Y_VALIDATION)))
```

```
#K PLUS PROCHES VOISINS
knn = KNeighborsClassifier()
knn.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = knn.predict(X_VALIDATION)
print("K plus proche voisins: "+str(accuracy_score(predictions,
Y_VALIDATION)))

#MACHINE VECTEURS DE SUPPORTS
SVM = SVC(gamma='auto')
SVM.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = SVM.predict(X_VALIDATION)
print("Machine vecteurs de supports:
"+str(accuracy_score(predictions, Y_VALIDATION)))

from sklearn.model_selection import GridSearchCV

#Définition d'une plage de valeurs à tester
penalite = [{'C': range(1,100)}]

#Tests avec 5 échantillons de Validation Croisée
recherche_optimisations = GridSearchCV(SVC(), penalite, cv=5)
recherche_optimisations.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)

print("Le meilleur paramètre est :")
print()
print(recherche_optimisations.best_params_)
print()

#MACHINE A VECTEURS DE SUPPORTS OPTIMISEE
SVM = SVC(C=98, gamma='auto')
SVM.fit(X_APPRENTISSAGE, Y_APPRENTISSAGE)
predictions = SVM.predict(X_VALIDATION)
print("Machine à vecteurs de supports optimisée:
"+str(accuracy_score(predictions, Y_VALIDATION)))
```

Nous pouvons à présent tester à nouveau nos algorithmes et nous rendre compte que nous obtenons un meilleur résultat pour l'algorithme Machine à Vecteurs de supports optimisée :

■ **Machine à vecteurs de supports optimisée (C=98) : 0.8974358974358975**

Si nous supprimons à présent toutes les valeurs extrêmes :

```
#ligne à la liste des lignes à supprimer
if (nbr_valeurs_extremes>7):
    num_lignes_a_supprimer.append(num_ligne)
```

Devient :

```
#ligne à la liste des lignes à supprimer
if (nbr_valeurs_extremes>1):
    num_lignes_a_supprimer.append(num_ligne)
```

Les résultats sont satisfaisants, mais dans une moindre mesure :

```
Regression logistique: 0.7
Arbre de décision: 0.7333333333333333
Forets aléatoires: 0.7666666666666667
K plus proche voisins: 0.8
Machine vecteurs de supports: 0.6
Machine vecteur de supports optimisée (C=60): 0.8333333333333334
```

Ceci pour prendre conscience que même si les données semblent aberrantes, elles peuvent tout de même jouer un rôle important dans la prédiction. Il ne faut donc pas les supprimer sans avoir déterminé ce rôle.

5. En résumé

Dans ce chapitre, à travers un jeu d'observations un peu particulier, nous avons pu découvrir la démarche commune à toute résolution de problème lié au Machine Learning, à savoir : la définition du problème, l'acquisition et la préparation des données, l'analyse des données et enfin le choix de l'algorithme.

Nous avons également abordé l'utilisation des algorithmes liés à la classification, sans paramétrage particulier pour chacun d'entre eux. Puis nous avons modifié un simple paramètre de l'algorithme de machine à vecteurs de supports ce qui a fait pencher la balance dans le choix de l'algorithme à utiliser.

Cela veut donc dire que la phase d'amélioration des différents algorithmes utilisés est primordiale. Sortant du cadre de cet ouvrage dédié à la vulgarisation de l'utilisation des différents algorithmes, les étapes d'optimisation et d'améliorations (*tuning*) ne seront pas abordées. Cependant, vous pourrez trouver une multitude d'exemples sur Internet et nous vous invitons à vous référer à la page dédiée à l'optimisation des algorithmes proposée par le module Scikit-Learn et consultable à cette adresse :

https://scikit-learn.org/stable/modules/grid_search.html

Enfin, nous avons vu que les données extrêmes doivent bénéficier d'une étude particulière, car elles peuvent avoir une importance dans la prédiction.

La question à laquelle nous devons répondre est notre degré de satisfaction sur la qualité de notre classification avec une précision de 89 %. En effet, il reste une marge d'erreur de 11 % qui dans notre cas d'étude peut coûter la vie à des marins. Alors que faut-il en penser ? Peut-être existe-t-il une autre méthode qui nous permettrait d'obtenir de meilleurs résultats ? C'est ce que nous découvrirons un peu plus loin.

En attendant, dans le chapitre suivant nous allons voir comment le Machine Learning peut s'appliquer aux données textuelles et s'immiscer dans les réseaux sociaux.

Chapitre 8

Opinions et classification de textes

1. Ce que nous allons découvrir et les prérequis

Dans le chapitre précédent, nous avons vu l'utilité de la classification appliquée à des chiffres en nous permettant de déterminer si l'objet détecté par un sonar était une mine ou un rocher. Nous allons à présent découvrir l'utilité de la classification pour le texte, qui est peut-être moins évidente, sauf si nous évoquons le fait que ce type de classification permet de vous proposer des publicités ou des articles ayant un contenu en relation avec les différentes recherches que vous avez effectuées sur Internet ou les différentes conversations que vous avez réalisées sur les réseaux sociaux !

Nous verrons également que la classification de texte peut être appliquée à la détermination de l'opinion des internautes sur les réseaux sociaux sur un sujet donné. Ce qui peut, nous en convenons, être un peu dérangeant, et susciter les craintes que nous avons évoquées au début de cet ouvrage.

■ Remarque

Prérequis nécessaires pour bien aborder ce chapitre : avoir lu les chapitres Les fondamentaux du langage Python, Des statistiques pour comprendre les données, Principaux algorithmes du Machine Learning et Bien classifier n'est pas une option !

2. Le traitement automatique du langage naturel (TALN)

Le traitement automatique du langage naturel (TALN ou *Natural Language Processing* - NLP) est une discipline à la frontière entre l'informatique, la linguistique et l'intelligence artificielle.

L'un des objectifs de cette discipline est la compréhension du langage naturel (celui que nous utilisons) afin d'en déduire le sens et de pouvoir interagir avec nous. Les champs d'applications possibles étant bien entendu orientés autour du langage et de la langue, nous pouvons donc citer les applications de :

- Traduction automatique
- Corrections orthographiques
- Communication homme-machine (chatbot)
- Synthèse et reconnaissance de la parole
- Analyse des opinions

Pour y parvenir, nous pouvons utiliser une première approche appelée méthode linguistique dont l'objectif est de décrire l'information à extraire en utilisant des règles linguistiques (grammaires, formulations) ou bien d'utiliser le Machine Learning. C'est bien entendu la seconde méthode que nous allons privilégier.

3. Naive Bayes appliqué au TALN

Lorsque que nous évoquons la classification de texte, l'algorithme incontournable permettant de réaliser cette tâche est Naive Bayes se basant sur la fréquence d'apparition des mots. Pour les différents exemples qui vont suivre, nous vous invitons à laisser quelques instants de côté votre ordinateur pour vous munir d'une feuille de papier, d'un crayon et d'une calculatrice.

3.1 Le théorème

Comme nous l'avons déjà évoqué et illustré dans le chapitre Principaux algorithmes du Machine Learning, l'algorithme de Naive Bayes se base sur le Théorème de Bayes fondé sur les probabilités conditionnelles c'est-à-dire la détermination de la probabilité qu'un évènement se produise en fonction d'un évènement qui s'est déjà produit. Les évènements devant être indépendants l'un de l'autre.

Le théorème de Bayes s'exprime sous la formule suivante :

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

3.2 Un exemple : quels mots-clés choisir?

Pour illustrer à l'aide d'un exemple concret le fonctionnement de l'algorithme, nous avons pris le cas d'un site proposant des ateliers de codage pour enfants et adolescents. Nous allons chercher à déterminer si le fait d'utiliser les mots-clés "Atelier enfant" et "stage IA" dans les publicités sur les réseaux sociaux incite les internautes ayant été ciblés par cette publicité à inscrire davantage leurs enfants aux stages.

Pour cela, nous disposons des données de 10 internautes ayant fait ou non une inscription aux stages après avoir vu une publicité comportant uniquement le mot-clé "Atelier enfant", une autre uniquement le mot-clé "Stage IA", une troisième comportant les deux mots-clés et une quatrième ne comportant aucun de ces mots-clés.

internaute	atelier_enfant	stage_ia	inscription
1	Non	Oui	Non
2	Non	Oui	Oui
3	Oui	Non	Non
4	Oui	Non	Oui

internaute	atelier_enfant	stage_ia	inscription
5	Non	Oui	Non
6	Non	Oui	Non
7	Oui	Non	Oui
8	Non	Oui	Oui
9	Non	Non	Oui
10	Oui	Non	Oui

Voici quelques lectures de ce tableau :

- Le premier internaute a visionné la publicité contenant le mot-clé "Stage IA" mais n'a pas réalisé d'inscription.
- L'internaute n°4 a visionné la publicité contenant le mot-clé "Atelier enfant" et a réalisé une inscription.
- L'internaute n°9 a visionné la publicité ne contenant aucun des mots-clés "Atelier enfant" et "Stage IA" et il a réalisé une inscription.

3.2.1 Détermination des probabilités

Pour déterminer les différentes probabilités nécessaires à l'application du théorème de Naive Bayes, répondons à ces différentes questions :

- *Combien y a-t-il d'inscriptions ?* : 6 (6 oui dans la colonne inscription).
- *Quelle est la probabilité qu'il y ait une inscription ? Probabilité que nous nommerons $P(\text{inscription}=\text{oui})$:* $6/10 = 0,6$. 6 étant le nombre d'inscriptions et 10 le nombre d'observations.
- *Combien y a-t-il de visites sans inscription ?* : 4 (4 non dans la colonne inscription).
- *Quelle est la probabilité qu'il y ait une inscription ? Probabilité que nous nommerons $P(\text{inscription}=\text{non})$:* $4/10 = 0,4$. 4 étant le nombre de non-inscriptions et 10 le nombre d'observations.
- *Combien y a-t-il d'inscriptions avec le mot-clé "Atelier enfant" :* 3 (les internautes n°4, 7, 10, car ils ont oui dans la colonne atelier_enfant et oui dans la colonne inscription).

- *Quelle est la probabilité qu'il y ait une inscription avec le mot-clé "Atelier enfant"? Probabilité que nous noterons $P(\text{atelier_enfant} | \text{inscription} = \text{oui}) : 3/6 = 0,5$. 3 étant le nombre d'inscriptions réalisées avec le mot-clé, calculé précédemment, et 6 étant le nombre d'inscriptions.*
- *Combien y a-t-il d'inscriptions avec le mot-clé "Stage IA" : 2 (les internautes n°2, 8, car ils ont oui dans la colonne stage_ia et oui dans la colonne inscription).*
- *Quelle est la probabilité qu'il y ait une inscription avec le mot-clé "Stage IA"? Probabilité que nous noterons $P(\text{stage_ia} | \text{inscription} = \text{oui}) : 2/6 = 0,3333$. 2 étant le nombre d'inscriptions réalisées avec le mot-clé calculé précédemment, et 6 étant le nombre d'inscriptions.*
- *Combien y a-t-il de **non-inscriptions** avec le mot-clé "Atelier enfant" : 1 (l'internaute n°3, car il a oui dans la colonne atelier_enfant et non dans la colonne inscription).*
- *Quelle est la probabilité qu'il n'y ait pas d'inscription avec le mot-clé "Atelier enfant"? Probabilité que nous noterons $P(\text{atelier_enfant} | \text{inscription} = \text{non}) : 1/4 = 0,25$. 1 étant le nombre d'inscriptions non réalisées avec le mot-clé calculé précédemment, et 4 étant le nombre de non-inscriptions.*
- *Combien y a-t-il de **non-inscriptions** avec le mot-clé "Stage IA" : 3 (les internautes n°1, 8, 9, car ils ont oui dans la colonne stage_ia et non dans la colonne inscription).*
- *Quelle est la probabilité qu'il n'y ait pas d'inscription avec le mot-clé "Stage IA"? Probabilité que nous noterons $P(\text{stage_ia} | \text{inscription} = \text{non}) : 3/4 = 0,75$. 3 étant le nombre d'inscriptions non réalisées avec le mot-clé calculé précédemment, et 4 étant le nombre de non-inscriptions.*
- *Quelle est la probabilité de l'utilisation du mot-clé "Atelier enfant"? Probabilité que nous nommerons $P(\text{atelier_enfant}) : 4/10 = 0,4$.*
- *Quelle est la probabilité de l'utilisation du mot-clé "Stage IA"? Probabilité que nous nommerons $P(\text{stage_ia}) : 5/10 = 0,5$.*

Nous disposons à présent des différentes probabilités reprises dans le tableau ci-dessous :

Probabilité	Valeur
$P(\text{inscription}=\text{oui})$	0,6
$P(\text{inscription}=\text{non})$	0,4
$P(\text{atelier_enfant} \text{inscription}=\text{oui})$	0,5
$P(\text{stage_ia} \text{inscription}=\text{oui})$	0,3333
$P(\text{atelier_enfant} \text{inscription}=\text{non})$	0,25
$P(\text{stage_ia} \text{inscription}=\text{non})$	0,75
$P(\text{atelier_enfant})$	0,4
$P(\text{stage_ia})$	0,5

Nous allons pouvoir utiliser ces probabilités afin de réaliser les prédictions suivantes à l'aide du théorème de Naive Bayes :

- Quelle est la probabilité qu'une personne inscrive son enfant après avoir visionné une publicité contenant uniquement le mot-clé "Atelier enfant"? Probabilité que nous noterons $P(\text{inscription}|\text{atelier_enfant})$.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

$$P(\text{inscription}|\text{atelier_enfant}) = \frac{P(\text{atelier enfant}|\text{inscription}) * P(\text{inscription})}{P(\text{atelier_enfant})}$$

$$P(\text{inscription}|\text{atelier enfant}) = \frac{0,5 * 0,6}{0,4}$$

$$P(\text{inscription}|\text{atelier enfant}) = 0,75$$

Probabilité	Valeur
$P(\text{inscription} \text{atelier_enfant})$	0,75

Probabilité	Valeur
$P(\text{inscription} \text{stage_ia})$	0,39

3.2.2 Conclusion

Au vu des différents résultats, on constate qu'il y a un bon retour sur les publicités ne comportant que le mot-clé "Atelier enfant".

4. Naive Bayes pour l'analyse d'opinion

Comme nous allons à présent le constater, le théorème de Naive Bayes va nous permettre de réaliser l'analyse d'option des internautes en fonction de leurs posts sur les réseaux sociaux.

En analysant les différents messages des utilisateurs laissés sur les réseaux sociaux ou sites internet à propos d'un produit, ou bien d'un sujet en particulier (sport, politique...), nous pouvons déterminer s'ils sont plutôt contents, mécontents, heureux, tristes, en accord, en désaccord et agir en conséquence.

Comment ? En analysant les mots des différents messages et en classant ces messages dans différentes catégories : en accord, en désaccord, content, mécontent. Cette classification permet par exemple d'alerter en direct les responsables de la communication d'un produit en cas de mécontentement. Cette alerte permet de répondre à l'utilisateur très rapidement et évite ainsi la propagation d'une mauvaise image de la marque.

Prenons l'exemple d'une étude de conversations à propos des chats récupérée sur les réseaux sociaux. À partir de l'occurrence des mots dans une phrase, on peut identifier l'opinion de la personne ayant posté le message à savoir : elle aime ou non les chats.

■ Remarque

L'ensemble des observations que nous allons utiliser dans ce chapitre sera basé sur des textes en anglais. La raison en est qu'il n'est malheureusement pas si évident de trouver des jeux de données en français et que la langue anglaise est plus facile à manipuler et traiter de par sa composition.

Message	Opinion
I love cats	Positive
I hate cats	Négative
Cats are beautiful	Positive
Cats are awful	Négative
Cats are intelligent	Positive

Imaginons à présent un nouveau commentaire : "I love awful cats". La personne ayant posté ce commentaire aime-t-elle ou non les chats ? C'est la difficile question à laquelle devra répondre notre algorithme de classement de texte.

4.1 Étape 1 : normalisation des données

La normalisation du texte, va consister à supprimer les accents, les caractères spéciaux et le mettre en minuscule :

Message	Opinion
i love cats	Positive
i hate cats	Négative
cats are beautiful	Positive
cats are awful	Négative
cats are intelligent	Positive

4.2 Étape 2 : suppression des stops words

Les stops words sont les mots d'une langue les plus utilisés et qui n'apportent pas de valeur ajoutée au sens de la phrase. Il convient donc de les supprimer. Nous verrons qu'en Python il existe des fonctions capables de faire cela pour nous.

Voici le résultat de la suppression des stops words qui étaient simplement les mots "I" et "are" :

Message	Opinion
love cats	Positive
hate cats	Négative
cats beautiful	Positive
cats awful	Négative
cats intelligent	Positive

■ Remarque

La liste des stops words de la langue anglaise est disponible à cette adresse :
<https://gist.github.com/sebleier/554280>

4.3 Étape 3 : le stemming

L'étape de stemming consiste à supprimer les suffixes et les préfixes des mots. Dans notre cas, nous pouvons supprimer les "S" aux mots "cats", le suffixe "ful" au mot "beautiful" et "ent" au mot "intelligent".

Message	Opinion
love cat	Positive
hate cat	Négative
cat beauti	Positive
cat awful	Négative

Message	Opinion
cat intellig	Positive

4.4 Étape 4 : la lemmatisation

La lemmatisation prend en compte l'analyse morphologique des mots, afin de n'obtenir que des noms.

Dans notre cas, les données restent inchangées, car nous ne disposons que de noms.

4.5 Étape 5 : déterminer le nombre d'occurrences de chaque mot

Voici une étape relativement simple, car elle consiste à réaliser un tableau ayant pour colonnes la liste complète des mots de l'ensemble des phrases et en ligne les différentes phrases.

Il suffit ensuite, pour un mot donné, de compter le nombre de fois où il apparaît dans chacune des phrases.

Phrase	love	cat	hate	beauti	awful	Intellig	Opinion
love cat	1	1	0	0	0	0	Positive
hate cat	0	1	1	0	0	0	Négative
cat beauti	0	1	0	1	0	0	Positive
cat awful	0	1	0	0	1	0	Négative
cat intellig	0	1	0	0	0	1	Positive

4.6 Étape 6 : déterminer les probabilités pour l'opinion positive

Nous allons à présent calculer les différentes probabilités correspondant aux opinions positives. Pour cela, nous allons prendre uniquement en considération les phrases positives :

Phrase	love	cat	hate	beauti	awful	Intellig	Opinion
love cat	1	1	0	0	0	0	Positive
cat beauti	0	1	0	1	0	0	Positive
cat intellig	0	1	0	0	0	1	Positive

- Probabilité d'opinions positives que nous nommerons $P(+)$ = 3 phrases sur 5, soit $3/5$, soit $= 0,6$
- Nombre de mots : 6 (love, cat, hate, beauti, awful, intellig)
- Nombre total d'occurrences : 6 (somme des 1 dans le tableau)

Pour calculer les probabilités, nous allons utiliser la formule suivante :

$P(\text{mot}|\text{opinion positive})$

$$= \frac{\text{Nombre d'occurrences du mot utilisé dans une phrase ayant une opinion positive} + 1}{\text{Le nombre d'utilisations de ce mot} + \text{le nombre total de mots}}$$

■ Remarque

On exprime ici le fait que, si la probabilité du mot mignon est égale à 0 pour une opinion positive, la phrase « ce chat est mignon » aura également une probabilité de 0. Or nous ne souhaitons pas obtenir de probabilité égale à zéro. Pour éviter cela, nous effectuons une opération de lissage, appelée Lissage Laplace (Laplace smoothing) consistant à ajouter 1 au dividende..

Voici un exemple permettant de calculer la probabilité que l'opinion soit positive si le mot "love" est utilisé :

- Nombre d'occurrences du mot "love", utilisé dans une phrase ayant une opinion positive = 1
- Nombre total de mots = 6
- Nombre total d'occurrences = 6

$$P(\text{love}|\text{opinion positive}) = \frac{1+1}{6+6} = 0,1666$$

Faisons à présent la même démarche pour les autres mots :

P(love positif)	Probabilité d'utilisation du mot "love" pour des opinions positives	$(1+1) / (6+6) = 0,1666$
P(cat positif)	Probabilité d'utilisation du mot "cat" pour des opinions positives	$(3+1) / (6+6) = 0,3333$ (Le mot "cat" est utilisé 3 fois dans les phrases exprimant une opinion positive)
P(hate positif)	Probabilité d'utilisation du mot "hate" pour des opinions positives	$(0+1) / (6+6) = 0,0833$ (Le mot "hate" n'est pas utilisé dans les phrases exprimant une opinion positive)
P(beauti positif)	Probabilité d'utilisation du mot "beauti" pour des opinions positives	$(1+1) / (6+6) = 0,1666$
P(awful positif)	Probabilité d'utilisation du mot "awful" pour des opinions positives	$(0+1) / (6+6) = 0,0833$
P(intellig positif)	Probabilité d'utilisation du mot "intellig" pour des opinions positives	$(1+1) / (6+6) = 0,1666$

4.7 Étape 7 : déterminer les probabilités pour le sentiment positif

Extrayons à présent les phrases exprimant une opinion négative :

Phrase	love	cat	hate	beauti	awful	intellig	Opinion
hate cat	0	1	1	0	0	0	Négative
cat awful	0	1	0	0	1	0	Négative

- Il existe deux phrases sur cinq exprimant une opinion négative. Par conséquent, la probabilité qu'une phrase ait une opinion négative est de $2/5$, soit $P(\text{opinion négative}) = 0,4$
- Nombre de mots : 6 (love, cat, hate, beauti, awful, intellig)
- Nombre total d'occurrences : 4 (somme des 1 dans le tableau)

Calculons ensuite les probabilités des différents mots en fonction d'une phrase exprimant une opinion négative :

$P(\text{love} \text{négatif})$	Probabilité d'utilisation du mot "love" pour des opinions négatives	$(0+1) / (4+6) = 0,1$
$P(\text{cat} \text{négatif})$	Probabilité d'utilisation du mot "cat" pour des opinions négatives	$(2+1) / (4+6) = 0,3$ (Le mot cat est utilisé 2 fois dans les phrases exprimant une opinion négative)
$P(\text{hate} \text{négatif})$	Probabilité d'utilisation du mot "hate" pour des opinions négatives	$(1+1) / (4 + 6) = 0,2$
$P(\text{beauti} \text{négatif})$	Probabilité d'utilisation du mot "beauti" pour des opinions négatives	$(0+1) / (4+6) = 0,1$
$P(\text{awful} \text{négatif})$	Probabilité d'utilisation du mot "awful" pour des opinions négatives	$(1+1) / (4 + 6) = 0,2$

P(intellig nega- tif)	Probabilité d'utilisation du mot "intellig" pour des opi- nions négatives	$(0+1) / (4+6) = 0,1$
---------------------------	---	-----------------------

4.8 Étape 8 : déterminer le sentiment d'une nouvelle phrase

Maintenant que l'ensemble des probabilités est déterminé, nous pouvons à présent passer à l'étape de classification de notre nouvelle phrase, à savoir :

■ "I love awful cats"

N'oublions pas de lui faire subir toutes les phases de normalisation, de suppression des stops words, de stemmisation et de lemmisation avant d'en déterminer le sentiment.

Voici le résultat de ces opérations :

■ love awful cat

Nous pouvons à présent passer à la phase de classification de notre phrase en :

- Calculant la probabilité que la phrase exprime une opinion positive.
- Calculant la probabilité que la phrase exprime une opinion négative.
- Prenant la probabilité ayant une plus forte valeur.

$$P(\text{Opinion Positive}) = P(+)*P(\text{love}|+)*P(\text{awful}|+)*P(\text{cat}|+)$$

$$P(\text{Opinion Positive}) = 0,6 * 0,1666 * 0,0833 * 0,3333$$

$$P(\text{Opinion Positive}) = 0,0027$$

$$P(\text{Opinion négative}) = P(-)*P(\text{love}|-)*P(\text{awful}|-)*P(\text{cat}|-)$$

$$P(\text{Opinion négative}) = 0,4*0,1*0,2*0,3$$

$$P(\text{Opinion négative}) = 0,0024$$

Étant donné que 0,0027 est supérieur à 0,0024, la phrase exprime donc une opinion positive!

En procédant de la sorte, nous venons d'utiliser le théorème de Naive Bayes appliqué à la classification. Pour les plus curieux d'entre vous, voici la formule appliquée :

$$V_{nB} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_{m \in \text{Emots}} P(\text{mot}, v_j)$$

Pour les deux opinions V que nous nommerons de façon individuelle V_j (V_j opinion négative et V_j opinion positive).

La probabilité d'un type d'opinion $P(V_j)$ correspond à la probabilité V_j multipliée par somme des produits des probabilités de chaque mot appartenant à V_j .

$$P(\text{Opinion positive}) = P(+)*P(\text{love}|+)*P(\text{awful}|+)*P(\text{cat}|+)$$

$$P(\text{Opinion négative}) = P(-)*P(\text{love}|-)*P(\text{awful}|-)*P(\text{cat}|-)$$

L'opinion de la phrase est déterminée en prenant la probabilité maximum (arg-Max) des probabilités V_j calculées.

$$\operatorname{argMax}(0,0027; 0,0024) = 0,0027$$

5. Cas pratique : croyez-vous au réchauffement climatique ?

Nous allons à présent procéder à la détermination d'opinions à partir d'un jeu d'observations réel issu du réseau social Twitter. Nous ne nous attarderons pas sur les différentes phases de calcul, mais utiliserons le langage Python pour mener à bien la mission de classification.

5.1 Comment obtenir des données ?

La première chose est d'obtenir un jeu d'observations. Pour cela, il est possible d'utiliser les différents réseaux sociaux, dont Facebook et Twitter, pour récupérer un ensemble de conversations. En effet, les réseaux sociaux sont propices à la classification d'opinions, car les utilisateurs se servent de ces réseaux pour publier leurs avis sur différents sujets.

De plus grâce aux hastags, il est facile de sélectionner les conversations qui concernent un sujet qui nous intéresse. En sélectionnant les conversations ayant le hashtag #intelligenceartificielle, il y a fort à parier que celles-ci traiteront de l'intelligence artificielle.

À titre d'exemple, nous allons étudier les différentes opinions des Américains laissés sur Twitter à propos du réchauffement climatique. Nous essayerons de les classer en deux groupes : ceux ayant conscience du réchauffement et ceux n'y croyant pas du tout. Usage politique, ou bien de sensibilisation, libre à vous de décider l'utilité de cette classification.

5.2 Création d'un projet Python

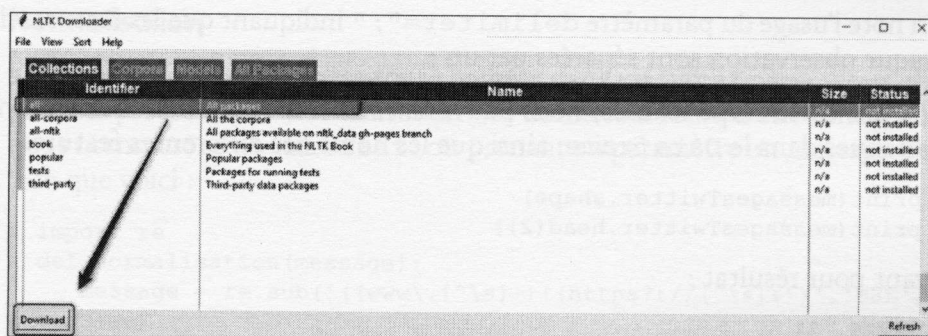
Les modules Python que nous allons utiliser dans ce projet sont relatifs à la manipulation de données textuelles et de tableaux.

C'est pourquoi nous allons utiliser les librairies suivantes :

- NLTK (*Natural Language ToolKit*)
- Numpy
- Pandas
- Scikit-Learn

Ces bibliothèques devront bien entendu être importées dans votre projet Python.

Lors du premier import de NLTK dans votre script Python, une fenêtre s'ouvre vous demandant les modules complémentaires à installer. Il convient alors de sélectionner l'option **All packages** et de cliquer sur le bouton **Download** situé en bas à gauche de l'écran (cf. figure ci après).



Téléchargement des packages complémentaires au module NLTK

Nous vous invitons également à créer le fichier de script Python pour pouvoir y insérer les différentes lignes de code.

5.3 Acquisition des données et préparation des données

Les données que nous allons utiliser sont téléchargeables sur le site de l'éditeur. Afin de vous éviter toutes les phases de préparation des données (normalisation des libellés de classification, suppression des observations sans catégorisation), nous avons déjà réalisé cette tâche pour vous. Cependant, nous avons également proposé au téléchargement le fichier original, vous permettant de vous exercer aux diverses tâches de préparation de ce fichier.

5.3.1 Chargement du fichier

La première étape va consister à charger le fichier contenant les observations que nous aurons préalablement placées dans un répertoire nommé `datas` du projet. Le stockage des données se fera dans un `Dataframe` que nous nommerons `messagesTwitter`.

```
import pandas as pnd

messagesTwitter =
pnd.read_csv("datas/rechauffementClimatique.csv", delimiter=";")
```

On note l'usage du paramètre `delimiter=" ; "` indiquant que les features de chaque observation sont séparées par un ;

Une fois cette étape réalisée, nous pouvons vérifier le nombre d'observations contenues dans le `Dataframe`, ainsi que les noms des différentes features :

```
print(messagesTwitter.shape)
print(messagesTwitter.head(2))
```

Ayant pour résultat :

(4225, 3)

		TWEET	CROYANCE
	CONFIANCE		
0	Global warming report urges governments to act...		Yes
	1.0		
1	Fighting poverty and global warming in Africa ...		Yes
	1.0		

Nous disposons donc de 4226 observations et 3 features portant les noms de TWEET, CROYANCE, CONFIANCE.

- La feature TWEET correspond au message laissé sur Tweeter.
- La feature CROYANCE correspond à l'opinion exprimée par le message. Si la feature est égale à Yes, cela signifie que le message exprime une opinion de croyance au réchauffement climatique.
- La feature CONFIANCE indique le degré de confiance du type d'opinion exprimée : 1 signifiant que nous sommes sûrs à 100 % que l'opinion exprimée correspond à une croyance ou non au réchauffement climatique.

Nous allons modifier le contenu de la feature CROYANCE afin que celle-ci prenne la valeur 1 lorsque la valeur originale est Yes et 0 lorsque la valeur originale est No.

```
messagesTwitter['CROYANCE'] =
(messagesTwitter['CROYANCE']=='Yes').astype(int)

print(messagesTwitter.head(100))
```

5.3.2 Normalisation

Maintenant que les données sont chargées dans un `Dataframe`, nous allons pouvoir procéder à la normalisation des données à l'aide d'une fonction que nous nommerons `normalisation` prenant en paramètre un message. Fonction que voici :

```
import re
def normalisation(message):
    message = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', 'URL',
message)
    message = re.sub('@[^\s]+', 'USER', message)
    message = message.lower().replace("ë", "e")
    message = re.sub('[^a-zA-Za-zA-ZÀ-Я1-9]+', ' ', message)
    message = re.sub(' +', ' ', message)
    return message.strip()
```

Cette fonction utilise ce que l'on appelle des expressions régulières (*regular expressions*) qui permettent de rechercher dans une chaîne de caractères un ensemble de caractères spécifiques. Une fois les caractères trouvés, nous pouvons alors appliquer un traitement sur ceux-ci.

La fonction `sub` du module `Re` (regular expression) est utilisée comme suit :

```
text = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', 'URL', text)
```

Ce qui signifie que l'on remplace tous les caractères `www` par le mot "URL" dans la chaîne de caractères `text`. En d'autres termes, on supprime toute notion de lien internet que l'on remplace par le terme "URL". On note également que l'ensemble des messages est transformé en minuscules à l'aide de la fonction `lower`. Quant à l'usage de la fonction `strip()`, celle-ci a pour fonction de supprimer les tabulations, les espaces et les retours à la ligne contenus dans le message.

Pour réaliser la normalisation, nous devons appeler la méthode que nous venons de créer comme suit :

```
messagesTwitter["TWEET"] =
messagesTwitter["TWEET"].apply(normalisation)
print(messagesTwitter.head(10))
```

Pour chaque message contenu dans la feature `TWEET`, on applique la fonction `normalisation`. Les résultats obtenus peuvent alors être affichés à l'aide de la fonction `head()` du `Dataframe`.

5.3.3 Suppression des stop words

Pour supprimer les stop words de nos messages en anglais, nous allons utiliser ceux connus par le module `nltk.corpus` :

```
from nltk.corpus import stopwords
stopWords = stopwords.words('english')
```

Pour supprimer les stop words dans nos messages, nous allons utiliser une fonction `lambda` (anonyme) qui, pour chaque mot contenu dans les messages, vérifie s'il est connu par la variable `stopWords`.

Cette fonction `lambda` crée une chaîne vide. Si le premier mot du message n'est pas connu de la variable `stopWords`, il est alors gardé et concaténé (`join`) à la chaîne vide. Dans le cas contraire, il est ignoré. L'analyse du mot suivant peut alors commencer. Si le second mot du message n'est pas connu de la variable `stopWords`, il est alors gardé et concaténé (`join`) à la chaîne contenant le premier mot. L'analyse du troisième mot peut alors commencer, etc.

```
messagesTwitter['TWEET'] = messagesTwitter['TWEET'].apply(lambda
message: ' '.join([mot for mot in message.split() if mot not in
(stopWords)]))
print(messagesTwitter.head(10))
```

5.3.4 La stemming

La stemming des messages se fait à l'aide du module `nltk` et de la classe `SnowballStemmer`. Nous utilisons également une fonction `lambda` comme nous l'avons fait pour la suppression des stop words.

```
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer('english')
messagesTwitter['TWEET'] = messagesTwitter['TWEET'].apply(lambda
message: ' '.join([stemmer.stem(mot) for mot in message.split('
')]))
print(messagesTwitter.head(10))
```

5.3.5 La lemmatisation

La lemmatisation des messages se fait également à l'aide du module `nlk` mais cette fois-ci à l'aide de la classe `WordNetLemmatizer` :

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
messagesTwitter['TWEET'] = messagesTwitter['TWEET'].apply(lambda
message: ' '.join([lemmatizer.lemmatize(mot) for mot in
message.split(' ')]))
```

6. Phases d'apprentissage et de prédiction

Les données étant préparées, nous pouvons à présent passer aux phases d'apprentissage et de prédiction.

6.1 Découpage en jeux de tests et d'apprentissage

Comme nous en avons maintenant l'habitude, nous créons un jeu d'apprentissage et un jeu de tests ayant respectivement 80 % et 20 % des observations avec pour information à prédire la feature `CROYANCE`.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(messagesTwitter['TWEET'].values,
messagesTwitter['CROYANCE'].values, test_size=0.2)
```

Remarque

Nous avons fait le choix de garder les termes anglophones (train et Test), car c'est sous cette forme que vous les retrouverez dans les différents cas pratiques présents sur Internet.

6.2 Création d'un pipeline d'apprentissage

Un pipeline d'apprentissage peut être assimilé à une suite d'actions consécutives à réaliser sur le jeu d'observations.

La première action ajoutée au pipeline va consister, à l'aide de la fonction `CountVectorizer()`, à créer la matrice des occurrences des différents mots dans les différentes phrases comme nous avons appris à le faire manuellement.

La matrice d'occurrence des mots sert à déterminer le nombre de fois où apparaît un mot dans un texte. Cependant, cela ne signifie pas qu'un mot ayant un nombre important d'occurrences est déterminant dans la classification du message. En effet, il se peut que dans un message le même mot soit répété dix fois, mais qu'il n'ait été simplement utilisé que dans un seul message. On préfère donner un poids aux mots en fonction de leur fréquence d'apparition dans un message (*Term Frequency*), et à leur nombre d'apparitions dans l'ensemble des messages (*Inverse document frequency*). Ce poids appelé *TF-IDF* sert également à exprimer la rareté du mot dans l'ensemble des messages.

Nous ne nous pencherons pas sur la formule mathématique de détermination de ce poids. Sachez néanmoins que :

- le *TF-IDF* décroît quand un mot est présent dans beaucoup de messages ;
- le *TF-IDF* décroît également quand il est peu présent dans un message ;
- le *TF-IDF* est maximal pour les mots peu fréquents apparaissant beaucoup dans l'ensemble des messages que nous avons à analyser.

Ainsi dans les phrases suivantes :

Le chien joue dehors avec d'autres chiens avec une balle bleue.

Le chien et les autres chiens jouent dehors avec une balle rouge.

- Le mot "chien" apparaît plusieurs fois dans les phrases, son *TF-IDF* sera faible.
- Les mots "bleue" et "rouge" apparaissent peu de fois, leur *TF-IDF* sera faible également.

- Le mot "balle" apparaît certes peu de fois dans les deux phrases, mais au moins une fois dans les deux. Sa fréquence (TF) est donc faible, mais il apparaît dans deux phrases (IDF). Son TF-IDF sera donc élevé.

Le TF_IDF est calculé à l'aide de la fonction `TfidfTransformer()` du module Scikit-Learn que nous ajoutons en tant que deuxième action dans notre pipeline.

La troisième action ajoutée au Pipeline est l'utilisation de l'algorithme Naive Bayes Multimodal (`MultinomialNB()`) qui va permettre de réaliser l'apprentissage :

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB

etapes_apprentissage = Pipeline([('frequence',
                                   CountVectorizer()),
                                  ('tfidf', TfidfTransformer()),
                                  ('algorithme',
                                   MultinomialNB())])
```

6.3 Apprentissage et analyse des résultats

Nous pouvons à présent lancer la phase d'apprentissage et vérifier les résultats de celui-ci sur les données de tests à l'aide de la fonction `classification_report` prenant en paramètres :

- les résultats des observations de tests à prédire ;
- la prédiction et la précision décimale (`digits`) des résultats affichés.

```
modele = etapes_apprentissage.fit(X_train,y_train)

from sklearn.metrics import classification_report
print(classification_report(y_test, modele.predict(X_test),
                           digits=4))
```

	Précision	Recall	F1-score	Support
0	0.8519	0.3067	0.4510	225

	Précision	Recall	F1-score	Support
1	0.7958	0.9806	0.8786	620
Micro avg	0.8012	0.8012	0.8012	845
Macro avg	0.8238	0.6437	0.6648	845
Weight avg	0.8107	0.8012	0.7647	845

Au vu de ces chiffres, nous pouvons affirmer que nous obtenons une précision de classification de 81 %, ce qui n'est pas trop mal.

6.4 Classification d'un nouveau message

Nous pouvons à présent tester notre modèle d'apprentissage en lui proposant le nouveau message suivant :

```
Why should trust scientists with global warming if they didnt
know Pluto wasnt a planet
```

Pourquoi faire confiance aux scientifiques pour le réchauffement climatique s'ils ne savaient pas que Pluton n'était pas une planète?

Bien entendu, il convient de normaliser les phases et procéder aux étapes de suppression des stop words, de stemmisation et de lemmatisation avant de procéder à la classification.

```
phrase = "Why should trust scientists with global warming if they
didnt know Pluto wasnt a planet"
print(phrase)

#Normalisation
phrase = normalisation(phrase)

#Suppression des stops words
phrase = ' '.join([mot for mot in phrase.split() if mot not in
(stopWords)])

#Stemmatisation
phrase = ' '.join([stemmer.stem(mot) for mot in phrase.split(' ')])
```

```
#Lemmatisation
phrase = ' '.join([lemmatizer.lemmatize(mot) for mot in
phrase.split(' ')])
print (phrase)

prediction = modele.predict([phrase])
print (prediction)
if(prediction[0]==0):
    print(">> Ne croit pas au rechauffement climatique...")
else:
    print(">> Croit au rechauffement climatique...")
```

Le résultat de la classification est le suivant :

```
>> Ne croit pas au rechauffement climatique...
```

Ce qui semble cohérent avec le sens de la phrase!

7. L'algorithme SVM (Machine à vecteurs de supports) pour le classement de texte

Nous allons à présent utiliser l'algorithme SVM pour tenter de classifier notre texte. Cet algorithme commence à nous être familier, car les chapitres précédents nous ont permis de découvrir son fonctionnement et son optimisation.

Tout comme pour l'algorithme de Naive Bayes, nous allons utiliser un pipeline pour réaliser les différentes tâches d'analyse des messages avant leur apprentissage :

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.svm import SVC
etapes_apprentissage = Pipeline([('frequence',
CountVectorizer()),
                                ('tfidf', TfidfTransformer()),
                                ('algorithme',
SVC(kernel='linear', C=2))])
```

Réalisons à présent l'apprentissage :

```
modele = etapes_apprentissage.fit(X_train,y_train)
from sklearn.metrics import classification_report
print(classification_report(y_test, modele.predict(X_test),
digits=4))
```

	Précision	Recall	F1-score	F1-score
0	0.7514	0.6376	0.6898	218
1	0.8803	0.9266	0.9266	627
Micro avg	0.8521	0.8521	0.8521	845
Macro avg	0.8158	0.7821	0.7964	845
Weight avg	0.8470	0.8521	0.8479	845

Voyons à présent si nous ne pouvons pas obtenir de meilleurs résultats en optimisant l'algorithme.

Pour cela, nous allons rechercher la meilleure valeur pour le paramètre C à l'aide de la fonction GridSearchCV prenant en paramètre le pipeline créé précédemment.

```
from sklearn.model_selection import GridSearchCV
parameters = {'algorithm__C': (1,2,4,5,6,7,8,9,10,11,12)}

clf = GridSearchCV(etapes_apprentissage, parameters,cv=2)
clf.fit(X_train,y_train)
print(clf.best_params_)
```

Nous obtenons alors la réponse suivante :

```
{'algorithm__C': 1}
```

Remarque

Si le paramètre C de l'algorithme de Machine à Vecteurs de Supports et son optimisation vous sont inconnus, nous vous invitons à vous référer au chapitre Bien classifier n'est pas une option chargé de leurs explications.

Testons à présent ce nouveau paramètre :

```
etapes_apprentissage = Pipeline([('frequence',
CountVectorizer()),
                                ('tfidf', TfidfTransformer()),
                                ('algorithme',
svm.SVC(kernel='linear', C=1))])

modele = etapes_apprentissage.fit(X_train,y_train)
from sklearn.metrics import classification_report
print(classification_report(y_test, modele.predict(X_test),
digits=4))
```

Nous obtenons alors une précision de classification de 85 %!

weighted avg	0.8507	0.8556	0.8501	845
--------------	--------	--------	--------	-----

8. L'algorithme SVM plus performant que Naive Bayes?

Nous pouvons constater que l'algorithme SVM est bien plus performant que Naïves Bayes. Le même constat est également réalisé par les experts du Machine Learning (ouf!). Cela est dû au fait que Naive Bayes considère chaque feature comme étant indépendante alors que SVM cherche à trouver des éléments de corrélation entre elles. Cependant, Naive Bayes offre de bons résultats pour un coût de performance plus faible de l'algorithme SVM. Naive Bayes est donc à privilégier. C'est d'ailleurs pour cela que les filtres antispam l'utilisent, pour sa simplicité de mise en œuvre et pour ses bonnes performances.

Dans ce chapitre quelque peu ardu, nous en conviendrons lors de certains paragraphes, nous avons découvert comment le Machine Learning pouvait intervenir dans la classification de texte et les multiples applications que cela pouvait générer, jusqu'à la détermination de l'opinion exprimée par les utilisateurs des réseaux sociaux. Cela peut vous interloquer et nous en sommes conscients surtout dans la période que nous traversons actuellement. Nous profitons donc de ce chapitre pour vous sensibiliser sur les informations laissées sur les réseaux sociaux et les usages pouvant en être faits. Dans le chapitre suivant, nous allons faire connaissance avec les algorithmes d'apprentissage non supervisé : votre machine est-elle capable d'apprendre seule? C'est ce que nous verrons!

Chapitre 9

Abricots, cerises et clustering

1. Une machine qui apprend seule

L'ensemble des apprentissages que nous avons réalisés jusqu'à présent sont dits supervisés, car nous avons indiqué à la machine la valeur à prédire pour chaque cas d'apprentissage : "Voici l'observation à apprendre et nous nous attendons à ce que tu nous prédises si c'est une mine ou un rocher".

Nous allons à présent découvrir comment il est possible de laisser la machine se débrouiller toute seule pour apprendre à classifier une observation donnée à partir d'une liste d'observations dont elle ne connaît pas le groupe d'appartenance. "Voici une liste d'observations : peux-tu en déduire des groupes de classification? Si oui, voici une observation : à quel groupe appartient-elle?"

Pour illustrer l'apprentissage supervisé à travers un exemple concret, nous allons demander à notre machine d'apprendre à déterminer si, pour un ensemble de données comportant le poids et le diamètre d'un fruit, celui-ci est un abricot ou une cerise.

■ Remarque

Prérequis nécessaires pour bien aborder ce chapitre : avoir lu les chapitres Les fondamentaux du langage Python, Des statistiques pour comprendre les données et Principaux algorithmes du Machine Learning

2. Acquisition de données d'apprentissage

Comme pour tout apprentissage, nous devons disposer de données. La machine devra être capable de faire la distinction entre une cerise et un abricot à partir du poids et de la taille de chaque fruit. La première option qui s'offre à nous est de nous rendre chez un marchand de fruits, de prendre les mesures nécessaires pour chaque fruit et de les répertorier dans un document. Ce travail est assez fastidieux.

Il existe cependant une seconde option. Comme vous le savez sans doute, les caractéristiques des fruits sont "normalisées". Ainsi des normes existent pour définir la taille et le poids minimaux et maximaux d'une cerise, idem pour les abricots. C'est donc à partir de ces normes que nous allons constituer notre jeu de données en générant aléatoirement des fruits.

Pour récupérer ces normes, une simple recherche sur Internet suffit, ce qui nous a permis de constituer les tableaux suivants :

(source : <http://www.crenoexpert.fr/flipbooks/expproduit/TABLEAUX-CALIBRES-FRUITES-2.pdf>).

Les cerises :

Diamètre minimal (mm)	Diamètre maximal (mm)	Poids minimal (g)	Poids maximal (g)
17	19	1	5
20	21	5	6
22	23	6	7
24	25	7	8,5
26	27	8,5	10
28	29	10	11,5

Les abricots :

Diamètre minimal (mm)	Diamètre maximal (mm)	Poids moyen (g)
35	39	27
40	44	41
45	49	54
50	54	74
55	59	100

Un petit script Python pour nous aider

Nous allons créer un script Python qui va nous permettre de générer un nombre de cerises et d'abricots en fonction des caractéristiques qui leur sont propres.

Après avoir créé un nouveau projet et installé les modules Pandas, Matplotlib et Scikit-learn, nous vous invitons à saisir ces quelques lignes de code dans un nouveau fichier de script que nous nommerons `generationFruits` :

```
#---- IMPORT DES MODULES --
import random
import pandas as pnd

#---- CARACTERISTIQUES-----

#CERISES
caracteristiquesCerises =
[[17,19,1,5],[20,21,5,6],[22,23,6,7],[24,25,7,8.5],[26,27,8.5,10],
[28,29,10,11.5]]

#ABRICOTS
caracteristiquesAbricots =
[[40,44,41],[45,49,54],[50,54,74],[55,59,100]]
```

Comme vous pouvez le constater, nous avons créé deux tableaux. L'un contenant les caractéristiques des cerises, l'autre les caractéristiques des abricots. Chaque tableau contient des sous-tableaux correspondant à chaque catégorie de fruit.

Ainsi, le tableau [40,44,41] présent dans le tableau caractéristiqueAbricot correspond à une caractéristique (calibre) ayant pour diamètre minimal 35 mm, pour diamètre maximal 39 mm et un poids moyen de 27 g.

Nous pouvons ensuite définir le nombre de fruits à générer :

```
■ nombreObservations = 2000
```

Nous générerons 2000 cerises et 2000 abricots.

```
#Génération des cerises
cerises = []
random.seed()
for iteration in range(nombreObservations):
    #choix au hasard d'une caractéristique
    cerise = random.choice(caracteristiquesCerises)
    #Génération d'un diamètre
    diametre = round(random.uniform(cerise[0], cerise[1]),2)
    #Génération d'un poids
    poids = round(random.uniform(cerise[2], cerise[3]),2)
    print ("Cerise "+str(iteration)+" "+str(cerise)+" :
"+str(diametre)+" - "+str(poids))
    cerises.append([diametre,poids])

#Génération des abricots
abricots = []
random.seed()
for iteration in range(nombreObservations):
    #choix au hasard d'une caractéristique
    abricot = random.choice(caracteristiquesAbricots)
    #Génération d'un diamètre
    diametre = round(random.uniform(abricot[0], abricot[1]),2)
    #Génération d'un poids
    borneMinPoids = abricot[2] / 1.10
    borneMaxPoids = abricot[2] * 1.10
    poids = round(random.uniform(borneMinPoids, borneMaxPoids),2)
    print ("Abricot "+str(iteration)+" "+str(abricot)+" :
"+str(diametre)+" - "+str(poids))
    abricots.append([diametre,poids])
```

Pour chaque cerise et chaque abricot, une caractéristique est choisie au hasard. Les diamètres minimal et maximal de cette caractéristique sont ensuite récupérés et le diamètre est généré aléatoirement dans cette plage.

Nous procédons de la même façon pour le poids de la cerise. Comme nous ne disposons que d'un poids moyen pour l'abricot par caractéristique, nous choisissons de créer aléatoirement un poids compris entre le poids moyen - 10 % et le poids moyen + 10 %. Le diamètre et le poids sont ensuite stockés dans un tableau.

Enfin, nous concaténons les tableaux de cerises et d'abricots et nous mélangeons le résultat.

Notre jeu d'observation est à présent constitué. Nous pouvons le sauvegarder au format CSV à l'aide du module Panda et de la fonction `to_csv` du Dataframe.

```
#Constitution des observations
fruits = cerises+abricots
print(fruits)

#Mélange des observations
random.shuffle(fruits)

dataFrame = pnd.DataFrame(fruits)
dataFrame.to_csv("datas/fruits.csv", index=False, header=False)
```

■ Remarque

Afin d'éviter toute erreur d'exécution, pensez à créer le répertoire `datas` dans votre projet avant d'exécuter votre code.

3. Algorithme des K-Means (K-Moyennes)

Il existe plusieurs algorithmes propres à l'apprentissage non supervisé. Nous allons cependant nous attarder sur celui appelé K-Mean (K-Moyennes) permettant de réaliser des classifications sur un nombre de groupes de prédiction connu et sur un petit nombre de données (inférieur à 10000).

Le clustering (partitionnement de données)

Le partitionnement de données ou encore appelé clustering est l'action de découper l'ensemble des observations en petits groupes ayant des caractéristiques communes.

Ce partitionnement des données est le résultat attendu de chaque algorithme d'apprentissage non supervisé. Ainsi, lorsqu'une nouvelle observation à classer sera proposée à l'algorithme, charge à lui de la positionner dans l'un des groupes qu'il aura déterminés. La définition du libellé du groupe donnant du sens à la classification (abricot, cerise...) est quant à elle notre charge, car bien entendu la machine est incapable de le faire étant donné qu'elle n'a pas conscience de la signification des données qu'elle utilise lors de son apprentissage (pour elle, ce ne sont que des chiffres).

4. Visualiser les données

La machine ayant pour fonction de déterminer par elle-même si les données que nous lui donnerons en paramètre sont issues d'une cerise ou d'un abricot, il est tout de même intéressant pour nous de contrôler sa prédiction.

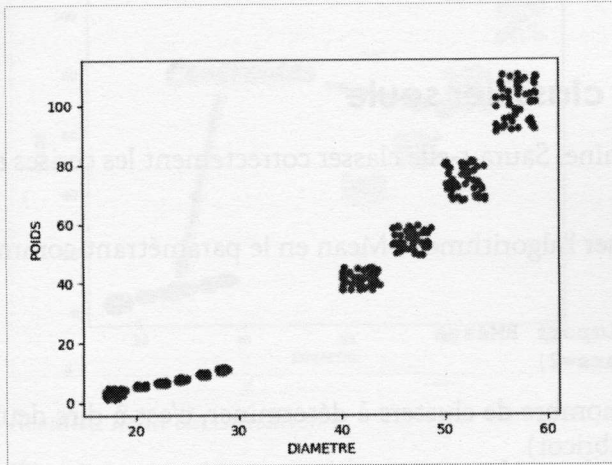
Pour cela, nous devons essayer de déterminer par nous-mêmes les deux clusters avant de demander à la machine de les trouver seule, mais comment y parvenir facilement? Eh bien à l'aide d'un graphique. Créons un nouveau fichier de script que nous appelons clustering et saisissons les lignes de code suivantes :

```
import pandas as pnd
import matplotlib.pyplot as plt

#Chargement des données
fruits = pnd.read_csv("datas/fruits.csv",
names=['DIAMETRE', 'POIDS'], header=None)

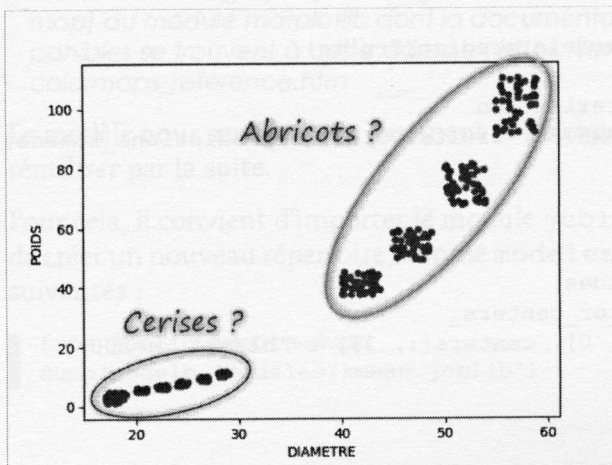
#Visualisation graphique des données
fruits.plot.scatter(x="DIAMETRE", y="POIDS")
plt.show()
```

Ce code a pour but de créer un nuage de points de l'ensemble des couples Diamètre/Poids.



Visualisation des données

À partir de ce graphique, nous sommes en mesure de déterminer par nous même les deux clusters correspondant potentiellement aux cerises et aux abricots :



Détermination manuelle des clusters

En effet, le diamètre d'une cerise est compris entre 1 et 29 selon notre jeu d'observation, ce qui tend à créer un premier cluster comme indiqué sur la figure précédente.

5. Laisser la machine classifier seule

Place maintenant à la machine. Saura-t-elle classer correctement les cerises et les abricots ?

Pour cela, nous allons utiliser l'algorithme K-Mean en le paramétrant comme suit :

```
from sklearn.cluster import KMeans
modele=KMeans(n_clusters=2)
```

Nous lui avons indiqué le nombre de clusters à déterminer, c'est-à-dire deux dans notre cas (Cerise ou Abricot).

Une fois ce paramétrage réalisé, passons à la phase d'apprentissage :

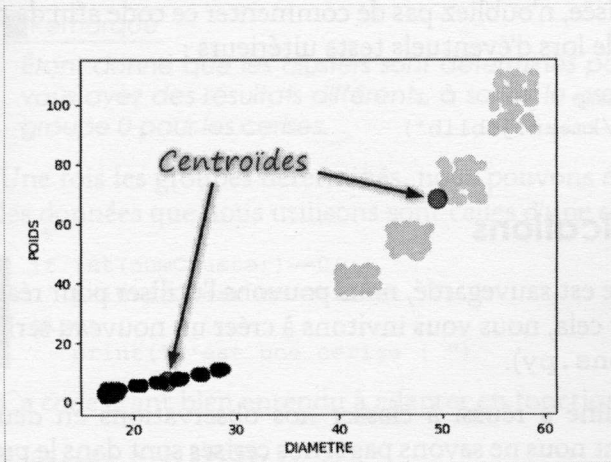
```
modele.fit(fruits)
```

Puis à la phase de prédictions avec un affichage graphique de celles-ci et les centroïdes des deux clusters :

```
#Predictions
predictions_kmeans = modele.predict(fruits)

#Affichage de la clusterisation
plt.scatter(fruits.DIAMETRE, fruits.POIDS, c=predictions_kmeans,
s=50, cmap='viridis')
plt.xlabel("DIAMETRE")
plt.ylabel("POIDS")

#Affichage des centroïdes
centers = modele.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200,
alpha=0.5)
plt.show()
```



Résultat de la classification

D'après ce graphique, on constate que la machine a réussi à classer correctement nos observations. En bas du graphique se trouvent comme nous l'avions prévu les cerises et dans la partie de droite les abricots.

Remarque

Les couleurs du graphique sont générées à l'aide de l'instruction `cmap` (color map) du module `matplotlib` dont la documentation et la liste de couleurs disponibles se trouvent à cette adresse : https://matplotlib.org/examples/color/colormaps_reference.htm

Le modèle nous semble donc pertinent et nous pouvons le sauvegarder pour le réutiliser par la suite.

Pour cela, il convient d'importer le module `joblib` dans notre projet Python, de créer un nouveau répertoire nommé `modeles` et d'utiliser les lignes de code suivantes :

```
from joblib import dump
dump(modele, 'modeles/kmean.joblib')
```

Une fois la sauvegarde réalisée, n'oubliez pas de commenter ce code afin de ne pas écraser votre sauvegarde lors d'éventuels tests ultérieurs :

```
#from joblib import dump  
#dump(modele, 'modeles/kmean.joblib')
```

6. Réaliser des classifications

À présent que notre modèle est sauvegardé, nous pouvons l'utiliser pour réaliser des classifications. Pour cela, nous vous invitons à créer un nouveau script Python (`classifications.py`).

Nous savons que la machine a réussi à classer nos observations en deux groupes distincts, cependant nous ne savons pas si nos cerises sont dans le premier ou dans le second groupe, car rien ne nous dit que le premier groupe choisi par la machine est celui situé en bas à gauche du graphique.

Pour connaître le groupe correspondant à nos fruits, nous allons choisir des valeurs pour chacun d'entre eux issues des données d'apprentissage et réaliser les prédictions.

```
#Chargement du modèle  
from joblib import load  
modele = load('modeles/kmean.joblib')  
  
#CERISE: 26.98 mm de diamètre ,8.75 grammes  
#ABRICOT: 55.7 mm de diamètre , 102.16 grammes  
  
cerise = [[26.98,8.75]]  
numCluster = modele.predict(cerise)  
print("Numéro de cluster des cerises: " + str(numCluster))  
  
abricot = [[55.7,102.16]]  
numCluster = modele.predict(abricot)  
print("Numéro de cluster des abricots: " + str(numCluster))
```

Ce qui nous donne comme résultat :

```
Numéro de cluster des cerises : [1]  
Numéro de cluster des abricots : [0]
```

Remarque

Étant donné que les clusters sont déterminés par la machine, il se peut que vous ayez des résultats différents, à savoir le groupe 1 pour les abricots et le groupe 0 pour les cerises.

Une fois les groupes déterminés, nous pouvons écrire un code indiquant que les données que nous utilisons sont celles d'une cerise ou d'un abricot.

```
if int(numCluster)==0:
    print("C'est un abricot !")
else:
    print("C'est une cerise ! ")
```

Ce code étant bien entendu à adapter en fonction des numéros de clusters.

```
cerise = [[26.98,8.75]]
numCluster = modele.predict(cerise)
if int(numCluster)==0:
    print("C'est un abricot !")
else:
    print("C'est une cerise ! ")

abricot = [[55.7,102.16]]
numCluster = modele.predict(abricot)
if int(numCluster)==0:
    print("C'est un abricot !")
else:
    print("C'est une cerise ! ")
```

7. Des erreurs de classifications

Certains et certaines d'entre vous ont sans doute remarqué que nous n'avons pas utilisé les premières caractéristiques des abricots dans notre génération de données :

Diamètre minimal (mm)	Diamètre maximal (mm)	Poids moyen (g)
35	39	27

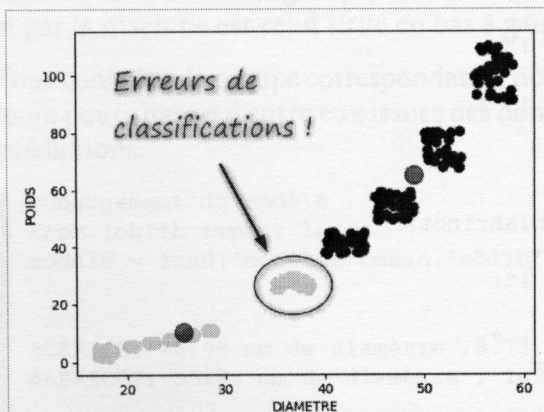
Nous vous invitons à modifier le script de génération de fruits (`generationFruits.py`) en incluant à présent la première caractéristique pour l'abricot :

```
caracteristiquesAbricots =  
[ [35, 39, 27], [40, 44, 41], [45, 49, 54], [50, 54, 74], [55, 59, 100] ]
```

Sans oublier de supprimer le fichier `fruits.csv` présent dans le répertoire `datas`.

Une fois ces opérations réalisées, nous pouvons générer de nouvelles données en exécutant le script.

Maintenant que nous disposons de nouvelles observations, nous allons vérifier que la machine a bien réussi à créer deux groupes distincts et à classer correctement nos fruits, tout ceci à l'aide d'un graphique.



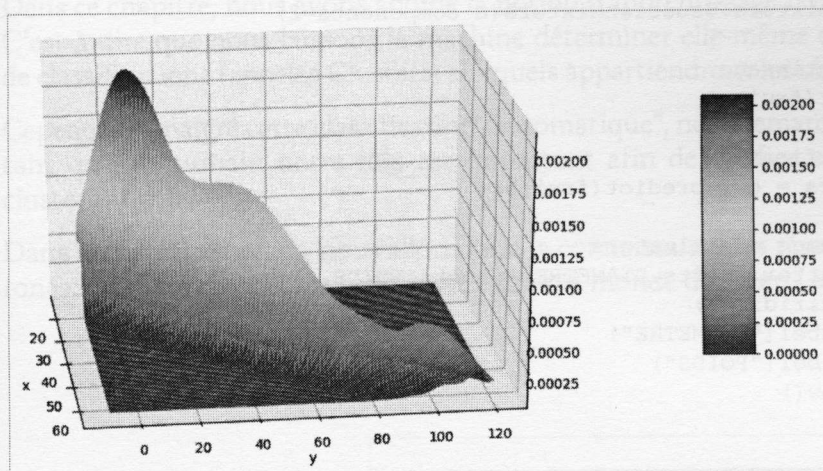
Erreurs de classifications

Sur la figure précédente, on constate que la machine a bien réussi à créer deux groupes distincts, mais qu'elle a commis des erreurs de classifications. En effet, on peut s'apercevoir que des abricots sont devenus des cerises !

Cela est dû au fait que les distances des observations calculées à l'aide de l'algorithme K-Mean par rapport aux deux centroïdes ont fait que ces données étaient plus proches du centroïde des cerises que celui des abricots. Il convient donc de tester un autre algorithme dédié au clustering afin de voir s'il est possible de corriger ces erreurs.

8. Algorithme de mélanges gaussiens ou Gaussian Mixture Model (GMM)

Dans la plupart des cas, les données suivent ce que l'on appelle une distribution normale ou gaussienne, pouvant se représenter sous forme d'une cloche symétrique en son milieu. Dans le cas de classification, le jeu d'observation peut se décomposer en plusieurs cloches correspondant aux différents clusters contenant chacune des observations.



Courbes gaussiennes

Nous avons représenté dans la figure ci-dessous les différentes courbes gaussiennes en 3D de notre jeu d'observation. On constate une première courbe représentant nos cerises et une seconde un peu moins marquée pour nos abricots.

Remarque

Le code de cette représentation est disponible en téléchargement sur le site de l'éditeur.

L'algorithme GMM (Mélange gaussien) permet de déterminer les différents clusters en séparant les données contenues dans les différentes courbes en forme cloche. Trop complexe pour cet ouvrage, nous ne nous attarderons pas

sur son fonctionnement détaillé. Sachez néanmoins que pour chaque cluster, l'algorithme détermine une moyenne et une variance, puis pour chaque observation il détermine une probabilité d'appartenir ou non à ce cluster.

Pour utiliser cet algorithme et visualiser la clusterisation effectuée (figure suivante), nous vous invitons à saisir ces quelques lignes de code :

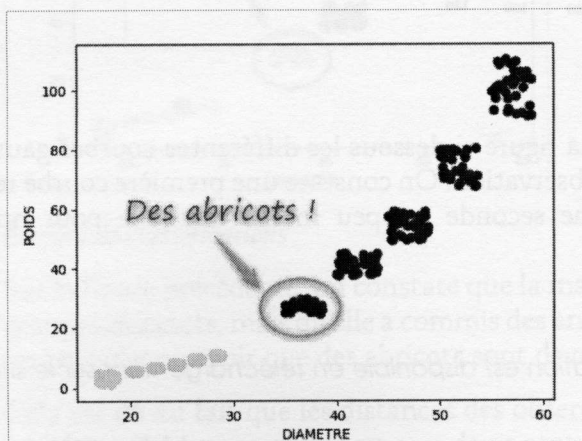
```
from sklearn import mixture

#Détermination des clusters (2 à trouver)
gmm = mixture.GaussianMixture(n_components=2)

#Apprentissage
gmm.fit(fruits)

#Classification
clusters = gmm.predict(fruits)

#Affichage des clusters
plt.scatter(fruits.DIAMETRE, fruits.POIDS, c=clusters, s=40,
            cmap='viridis');
plt.xlabel("DIAMETRE")
plt.ylabel("POIDS")
plt.show()
```



Correction des erreurs de classifications

On peut donc constater que la clusterisation est plus pertinente à l'aide de cet algorithme, car les erreurs de classifications ont été corrigées.

Nous pouvons à présent sauvegarder le modèle et l'utiliser à des fins de prédiction comme nous l'avons fait pour l'algorithme des K-Mean.

9. Pour conclure

Dans ce chapitre, nous avons abordé la notion d'apprentissage non supervisé. C'est-à-dire que nous laissons la machine déterminer elle-même des groupes de classifications (appelés Clusters) auxquels appartiendront les observations.

Cependant, malgré cette classification "automatique", nous remarquons qu'en tant qu'être humain notre rôle est important afin de vérifier et valider la clusterisation!

Dans le chapitre suivant, nous allons faire la connaissance des réseaux de neurones qui nous amèneront petit à petit dans le monde du Deep Learning!

Chapitre 10

Un neurone pour prédire

1. Ce que nous allons découvrir et les prérequis

Jusqu'à présent, nous avons découvert et mis en pratique les principes du Machine Learning à travers l'utilisation d'algorithmes issus des statistiques. Nous allons à présent voir comment une machine peut apprendre à partir d'algorithmes imaginés et conçus à partir des sciences cognitives à savoir les réseaux de neurones.

■ Remarque

Prérequis nécessaires pour bien aborder ce chapitre : avoir lu les chapitres Les fondamentaux du langage Python, Des statistiques pour comprendre les données, Principaux algorithmes du Machine Learning et Abricots, cerises et clustering.

2. 1957 - Le perceptron

Le perceptron, réalisé en 1957 par Frank Rosenbalt, est un algorithme d'apprentissage supervisé. Cependant, ce qui le différencie de ceux que nous avons découverts et utilisés dans le chapitre précédent c'est sa conception issue des sciences cognitives. Le perceptron sous sa forme la plus simple, avec lequel nous aurons l'occasion de faire connaissance d'ici quelques instants, est également nommé neurone formel (issu des travaux de Mc Culloch et Pitts en

1943) et a pour but de séparer des observations en deux classes (ou groupes) distinctes à condition que ces données soient linéairement séparables. Son rôle est donc de classer.

2.1 Un peu de biologie

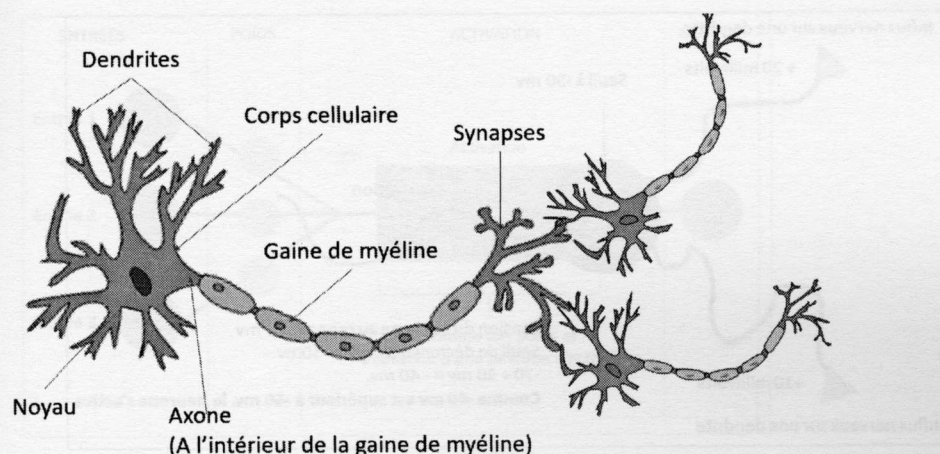
Rappelons-nous quelques instants nos cours de biologie du collège ou du lycée lorsque notre professeur nous expliquait le fonctionnement de notre cerveau.

Notre cerveau est composé de 86 à 100 milliards de neurones dont le rôle est d'acheminer et de traiter des messages dans notre organisme. Certains neurones ont un rôle dédié aux perceptions des sensations et aux mouvements alors que d'autres sont responsables des fonctions automatiques de notre corps (digestion, respiration...).

Biologiquement, un neurone est une cellule, composée :

- D'un corps cellulaire appelé également péricaryon
- D'un noyau
- De plusieurs ramifications appelées dendrites ayant pour fonction d'être les points d'entrée de l'information dans le neurone
- D'un chemin de sortie de l'information appelé axone, pouvant atteindre une longueur d'un mètre
- D'une gaine de myéline protégeant l'axone
- Des terminaisons axonales également appelées synapses connectées aux autres neurones

La communication entre neurones s'opère par l'échange de messages sous forme de variation de tension électrique. Un neurone peut recevoir plusieurs messages de la part d'autres neurones auxquels il est connecté.

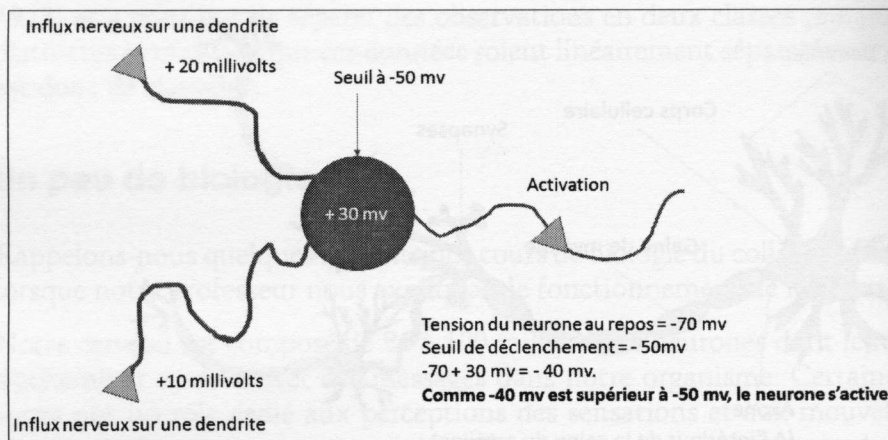


Représentation d'un neurone

Lorsqu'un neurone reçoit plusieurs messages, il effectue la somme des influx nerveux reçus puis, si cette somme dépasse un certain seuil, il s'active et transmet à son tour un message via son axone aux neurones connectés à celui-ci. Cette suite d'activation constitue notre mémoire. Car pour une action donnée, un ensemble de neurones s'active alors que d'autres restent inactifs, car un chemin s'est créé entre l'action et l'activation neuronale.

La figure suivante illustre le principe d'activation. Un neurone a une tension au repos de -70 millivolts. Lorsqu'il reçoit un message par ses dendrites, il effectue la somme de ces tensions. Si cette somme dépasse le seuil fixé (-50 mv), le neurone s'active.

	Neurone biologique	Neurone artificiel
Activation		
Axone		
Synapses		
Dendrites		



Principe d'activation d'un neurone

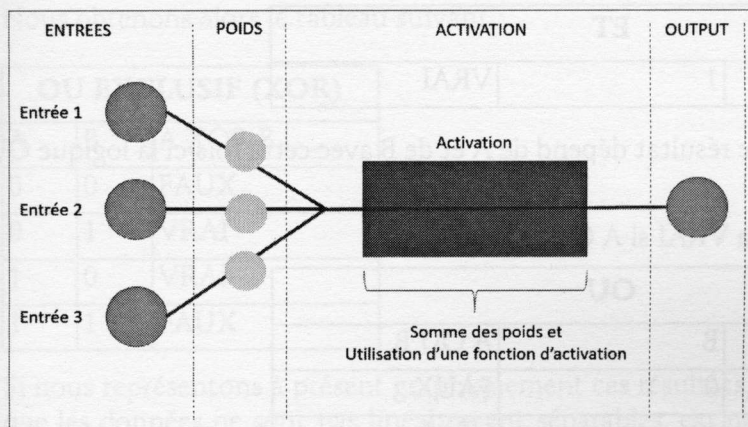
2.2 La biologie appliquée au machine learning

C'est sur le principe de fonctionnement d'un neurone évoqué précédemment que le perceptron a été conçu. Il s'agit en fait d'une transcription mathématique du fonctionnement d'un neurone.

Le perceptron va se caractériser par une couche de neurones en entrée et un neurone en sortie. Comme nous l'avons déjà évoqué auparavant, sous cette forme, le perceptron est également appelé neurone formel et ne permet que de réaliser des classifications sur des observations linéairement séparables.

Le tableau ci-dessous dresse le comparatif entre un neurone biologique et un neurone formel.

Neurone biologique	Neurone artificiel
Dendrites	Entrées (input)
Synapses	Poids
Axone	Sortie (output)
Activation	Fonction d'activation



Description du neurone formel

3. Des données linéairement séparables

Le perceptron simple couche ou neurone formel n'est en mesure de classer que des données linéairement séparables. Lorsque celles-ci ne le seront pas, nous utiliserons un perceptron multi-couche dont nous aurons l'occasion de découvrir les principes et le fonctionnement dans le chapitre suivant.

Comme vous le savez à présent, des données linéairement séparables sont celles pouvant être séparées par une droite. Pour modéliser simplement ce concept nous allons utiliser les fonctions logiques ET, OU et le OU Exclusif.

Si l'on considère deux éléments A et B. Si un résultat dépendant de A et de B est calculé en appliquant la logique ET (AND en anglais) pour A et B, alors :

- Le résultat sera VRAI si A ET B sont VRAI (leur valeur égale à 1).

ET		
A	B	A ET B
0	0	FAUX
0	1	FAUX
1	0	FAUX

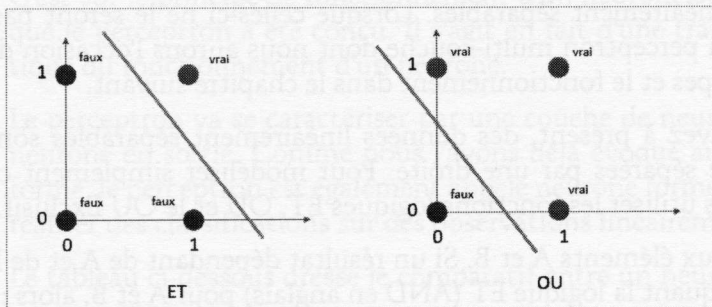
ET		
1	1	VRAI

Maintenant, si le résultat dépend de A et de B avec cette fois-ci la logique OU (OR en anglais).

– Le résultat sera VRAI si A OU B sont VRAI.

OU		
A	B	A OU B
0	0	FAUX
0	1	VRAI
1	0	VRAI
1	1	VRAI

Si l'on représente graphiquement ces deux cas, on constate qu'ils sont linéairement séparables.



Représentation graphique des fonctions logiques ET et OU

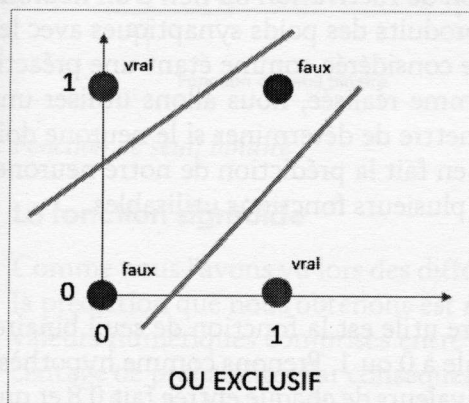
Considérons à présent la règle suivante appelée Ou Exclusif ou XOR :

- Le résultat est VRAI si une des valeurs A ou B est VRAI.
- Ou si les valeurs A et B sont distinctes.

Nous obtenons alors le tableau suivant :

OU EXCLUSIF (XOR)		
A	B	A XOR B
0	0	FAUX
0	1	VRAI
1	0	VRAI
1	1	FAUX

Si nous représentons à présent graphiquement ces résultats, nous constatons que les données ne sont pas linéairement séparables, car nous devons tracer deux droites pour y parvenir. Ce type de classification n'est donc pas possible avec le perceptron simple couche ou neurone formel.



Représentation graphique de la fonction logique Ou Exclusif

Ces informations sont importantes, car il nous est arrivé en parcourant le Web de découvrir des tutoriels montrant comment coder un perceptron en cherchant à classer des données régies sous la logique du XOR, ce qui n'est pas possible avec le perceptron simple couche que nous allons découvrir tout au long de ce chapitre.

4. Fonctions d'activation, rétropropagation et descente de gradient

Nous allons à présent entrer dans le détail du fonctionnement du perceptron en introduisant des notions que vous serez amené à rencontrer lorsque vous utiliserez des réseaux de neurones pour mener à bien vos projets de classification.

4.1 La fonction d'activation

Comme nous l'avons vu lors de la présentation du neurone biologique, celui-ci s'active si un seuil de tension électrique a été dépassé. Pour notre neurone artificiel, son activation sera également déclenchée en fonction d'un seuil.

La première étape dans la détermination de l'activation ou non d'un neurone artificiel est de réaliser la somme des produits des poids synaptiques avec les valeurs d'entrées. Cette étape peut être considérée comme étant une préactivation du neurone. Une fois cette somme réalisée, nous allons utiliser une fonction d'activation qui va nous permettre de déterminer si le neurone doit s'activer ou non. Cette activation sera en fait la prédiction de notre neurone. Pour calculer cette prédiction, il existe plusieurs fonctions utilisables.

4.1.1 La fonction de seuil binaire

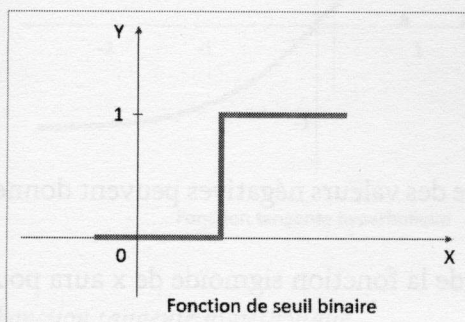
La première fonction qui peut nous être utile est la fonction de seuil binaire. Cette fonction retourne une valeur égale à 0 ou 1. Prenons comme hypothèse que la somme pondérée des différentes valeurs de chaque entrée fait 0,8 et que nous avons défini un seuil à 0,5. Comme 0,8 est supérieur à 0,5, la fonction renvoie la valeur 1 (le neurone s'active). Si la valeur de cette somme pondérée était inférieure à 0,5, la fonction d'activation renverrait 0.

Voici l'exemple chiffré :

Valeur en entrée	Poids lié à l'entrée	Valeur de l'entrée * valeur du poids
2	0,2	0,4

Valeur en entrée	Poids lié à l'entrée	Valeur de l'entrée * valeur du poids
1	0,1	0,1
3	0,1	0,3

La somme pondérée est donc de 0,8 (0,4 + 0,1 + 0,3) provoquant une activation du neurone.



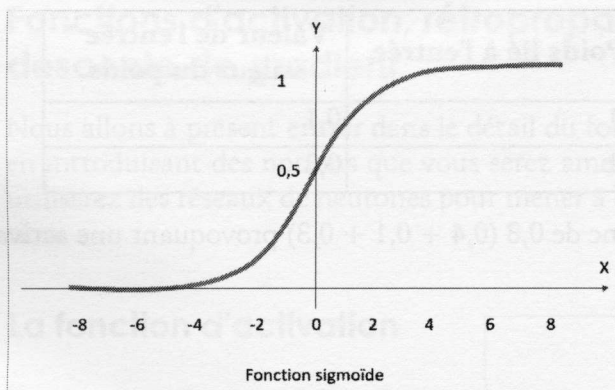
Fonction de seuil binaire

4.1.2 La fonction sigmoïde

Comme nous l'avons vu lors des différents exemples des chapitres précédents, la prédiction que nous obtenons est rarement égale à 0 ou 1, mais plutôt à des valeurs numériques comprises entre 0 et 1 (0,50, 0,99...) exprimant un pourcentage de probabilité. Par conséquent, la fonction de seuil binaire ne peut répondre à notre besoin. C'est pourquoi nous allons utiliser une autre fonction appelée Sigmoides dont les changements de valeurs entre 0 et 1 sont plus progressifs.

La formule mathématique de cette fonction est celle décrite ci-dessous, donnant naissance à la courbe en S illustrée par la figure qui suit.

$$\sigma(x) = \frac{1}{1 - e^{-x}}$$



Fonction sigmoïde ou courbe en S

L'inconvénient de cette fonction est que des valeurs négatives peuvent donner lieu à des valeurs positives.

Si par exemple $x = -0,2$, alors le calcul de la fonction sigmoïde de x aura pour valeur 0,45.

$$\sigma(-0,2) = \frac{1}{1 + e^{-(-0,2)}} = 0,45$$

La fonction sigmoïde est avant tout très utilisée dans des cas de prédictions de probabilités.

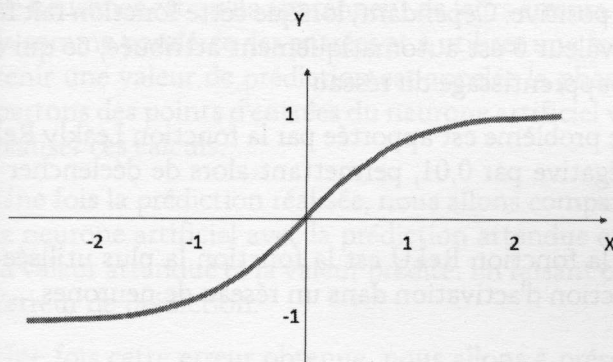
4.1.3 La fonction tangente hyperbolique (tanH)

La fonction de tangente hyperbolique ressemble quelque peu à la fonction sigmoïde. Elle présente cependant l'avantage que pour toute valeur négative celle-ci reste ou devient fortement négative :

$$\tan(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$

$$\tan(-0,2) = \frac{1 - e^{-2(-0,2)}}{1 + e^{-2(-0,2)}} = -0,19$$

La fonction tangente hyperbolique est très utilisée dans ce cas de classification entre deux classes.



Fonction tangente hyperbolique

Fonction tangente hyperbolique

4.1.4 La fonction ReLU (Rectified Linear Unit, unité de rectification linéaire)

Même si les fonctions sigmoïde et tangente hyperbolique sont utilisées en tant que fonction d'activation, celles-ci comportent tout de même un problème lié à ce que l'on appelle la saturation. En effet, les grandes valeurs sont cantonnées à l'intervalle $[0, 1]$ et les plus petites à l'intervalle $[-1 \text{ et } 0]$ et une fois que la saturation est atteinte, l'apprentissage du réseau devient complexe. Comme nous le verrons par la suite, l'apprentissage se fait en ajustant les différents poids du réseau de neurones en fonction de l'erreur qu'il a commise lors de sa classification. Dans le cas d'une saturation, la mise à jour de ces poids devient difficile et l'apprentissage s'en trouve impacté.

La fonction ReLU pallie ce problème via un fonctionnement assez simple. Si la valeur issue de la somme pondérée est inférieure à 0, celle-ci prend la valeur 0, sinon elle prend la valeur de la somme calculée.

L'un de ses avantages est son faible coût en termes de calcul, car il s'agit de prendre le maximum entre la valeur 0 et la valeur de la somme pondérée. Son second avantage est qu'elle ne souffre pas de la saturation, car elle n'a pas de valeur limite dans la zone positive. Cependant, lorsque cette fonction fait face à une valeur négative, la valeur 0 est automatiquement attribuée, ce qui entraîne de surcroît un non-apprentissage du réseau.

La solution pour pallier ce problème est apportée par la fonction Leaky ReLU qui multiplie la valeur négative par 0,01, permettant alors de déclencher un apprentissage du réseau.

Malgré son désavantage, la fonction ReLU est la fonction la plus utilisée de nos jours en tant que fonction d'activation dans un réseau de neurones.

4.1.5 La fonction softmax

La fonction sigmoïde (aussi appelée régression logistique) donne pour résultat une probabilité d'appartenance à un groupe donné. Dans le cas où nous devons classer une observation selon deux groupes distincts A et B et que nous obtenons la valeur 0.7 pour le groupe A par le biais de la fonction sigmoïde, cela signifie que l'observation appartient au groupe A avec une probabilité de 70 % et au groupe B avec une probabilité de 30 %.

Dans cet exemple, nous avons défini deux groupes A et B (aussi appelés classes) dont la somme des probabilités est égale à 1 ($0.7 + 0.3$), mais il se peut que nous devions classer l'observation parmi plusieurs classes. C'est alors qu'intervient l'algorithme de SoftMax qui attribue une probabilité à chacune de ces différentes classes tout en veillant à ce que la somme de ces probabilités soit égale à 1 :

Classe	Probabilité
Animal	0,01
Fruit	0,95
Véhicule	0,04

Ce type de fonction d'activation est généralement utilisé dans un réseau de neurones multicouche et dans le cas de classifications multiclassées.

5. La rétropropagation de l'erreur

Passons à présent à la notion de rétropropagation. La particularité des réseaux de neurones est qu'ils apprennent de leurs erreurs. L'étape consistant à réaliser la somme pondérée des entrées et à utiliser une fonction d'activation pour obtenir une valeur de prédiction est appelée la phase de propagation. Car nous partons des points d'entrées du neurone artificiel vers son point de sortie pour réaliser ces calculs.

Une fois la prédiction réalisée, nous allons comparer la prédiction réalisée par le neurone artificiel avec la prédiction attendue en faisant la différence entre la valeur attendue et la valeur prédite. En faisant cela, nous venons de calculer l'erreur de prédiction.

Une fois cette erreur obtenue, nous allons à présent parcourir le neurone en sens inverse (de la sortie vers les entrées) afin de prendre en compte l'erreur commise lors de la prédiction dans l'apprentissage en ajustant les valeurs des différents poids (nous verrons comment un peu plus loin dans ce chapitre). Cette phase est appelée la rétropropagation de l'erreur.

6. Les fonctions de perte (Loss function)

Là encore, ce terme vous deviendra vite familier au fur et à mesure de vos diverses expérimentations avec les réseaux de neurones.

Une fonction de perte, ou Loss function, est une fonction qui évalue l'écart entre les prédictions réalisées par le réseau de neurones et les valeurs réelles des observations utilisées pendant l'apprentissage. Plus le résultat de cette fonction est minimisé, plus le réseau de neurones est performant. Sa minimisation, c'est-à-dire réduire au minimum l'écart entre la valeur prédite et la valeur réelle pour une observation donnée, se fait en ajustant les différents poids du réseau de neurones.

6.1 L'erreur linéaire ou erreur locale

Comme nous venons de l'indiquer, l'erreur d'apprentissage, appelée aussi erreur locale, se calcule en réalisant la différence entre la valeur réelle à prédire et la valeur prédite par le neurone artificiel.

■ `Erreur = Prediction_reelle - Prediction_realisee`

C'est cette erreur que nous chercherons à minimiser au fur et à mesure des apprentissages.

Cette erreur peut être qualifiée d'erreur locale, car elle se focalise sur une observation donnée en comparant la valeur réelle et sa valeur prédite.

6.2 Erreur moyenne quadratique MSE ou erreur globale

L'erreur quadratique moyenne est une fonction d'erreur globale de l'apprentissage. C'est-à-dire que cette fonction va nous permettre de connaître globalement le pourcentage d'erreur commis par notre neurone artificiel sur l'ensemble de ses apprentissages.

Chaque observation est prédite par le neurone artificielle, une erreur locale en est déduite. Cette erreur locale va venir alimenter la fonction d'erreur quadratique afin de calculer la moyenne globale des erreurs de l'apprentissage. Si l'apprentissage se réalise normalement le résultat de la fonction doit diminuer au fur et à mesure de celui-ci.

Voici la formule Python permettant de calculer l'erreur moyenne quadratique :

$$MSE = \frac{1}{n} \sum_{i=1}^n E^2$$

E étant l'erreur locale élevée au carré (d'où le nom quadratique). Cela permet de supprimer les erreurs négatives, car en les élevant au carré, celles-ci deviennent positives.

```
predictions_attendues = [1,0,1]
predictions_realisees = [0.25,0.55,0.75]
somme = 0
i=0
for prediction in predictions_attendues:

    difference = predictions_attendues[i]-
predictions_realisees[i]
    carreDifference = difference * difference
    print("Difference = " + str(predictions_attendues[i]) + "-" +
str(predictions_realisees[i]) + "=" + str(difference))
    print("Difference * Difference = " + str(difference) + "*" +
str(difference) + str() + "=" + str(carreDifference))
    print("Somme = " + str(somme) + "+" + str(carreDifference))
    somme = somme + carreDifference

    print("")

moyenne_quadratique = 1/(len(predictions_attendues)) * somme

print("Erreur moyenne quadratique =" + str(moyenne_quadratique))
```

■ Remarque

Il existe une multitude d'autres erreurs globales utilisables. Cependant, nous avons fait le choix de vous expliquer celle étant la plus communément rencontrée afin que vous ne soyez pas perdu lorsque vous rencontrerez le terme MSE dans vos lectures annexes à cet ouvrage.

7. La descente de gradient

La descente de gradient a déjà été abordée dans le chapitre Principaux algorithmes du Machine Learning consacré aux statistiques, mais il est sans doute important d'y revenir à présent, car il s'agit d'un concept important dans la compréhension du fonctionnement des réseaux de neurones.

L'objectif de la descente de gradient est de minimiser la fonction d'erreur en ajustant petit à petit les paramètres d'apprentissage représentés par les différents poids.

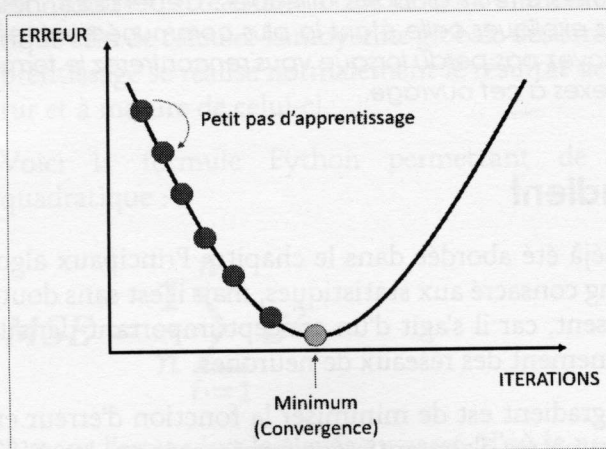
Reprenons l'image de la descente de la montagne. Vous vous situez au point le plus haut de la montagne et vous souhaitez atteindre la plaine en contrebas. Cependant, il fait nuit noire et vous n'êtes pas en mesure de voir où vous allez. Vous allez donc progresser doucement par petits pas jusqu'à atteindre le bas de la vallée.

La descente de gradient correspond à cette métaphore et se réalise par l'ajustement des différents poids du réseau de neurones jusqu'à obtenir une convergence, c'est-à-dire un minimum d'erreurs. Cet ajustement se fait par petit pas à l'aide d'un hyper paramètre appelé taux d'apprentissage (learning rate).

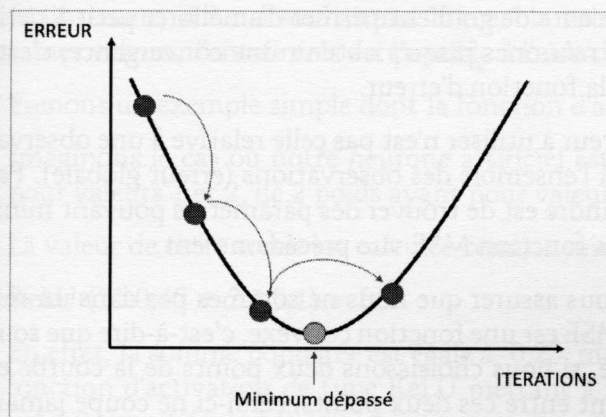
■ Remarque

La différence entre un paramètre et un hyper paramètre est que les premiers sont définis au niveau du modèle d'apprentissage (valeur des poids) et que les seconds sont définis au niveau de l'algorithme (taux d'apprentissage, nombre de couches de neurones par exemple).

Attention toutefois à ne pas choisir un taux d'apprentissage trop petit sous peine de devoir attendre longtemps avant la convergence ni un taux d'apprentissage trop grand qui aurait pour conséquence de dépasser ce minimum d'erreur et entraîner une augmentation de celle-ci. Cette augmentation empêcherait donc de trouver la bonne solution.



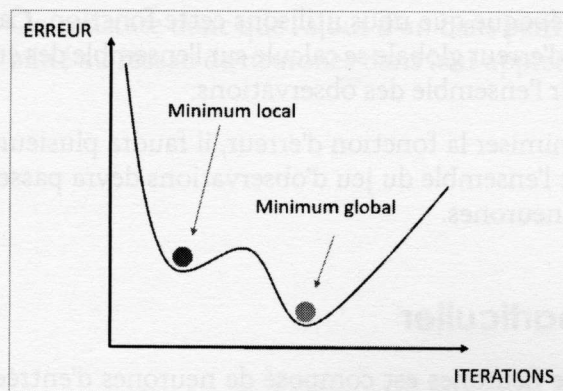
Descente de gradient - Convergence



Descente de gradient - Minimum dépassé

Parfois, pour atteindre la vallée en contrebas, il faut gravir une montagne se trouvant sur notre chemin, car le point le plus bas que nous cherchons à atteindre se trouve derrière cette montagne.

Si nous nous arrêtons avant de gravir la petite montagne, nous avons atteint certes un point plus bas que nous appellerons minimum local, mais ce point ne correspond pas à celui que nous souhaitons atteindre, autrement dit le minimum global.



Descente de gradient - Minimum local et minimum global

Vous l'aurez compris, la descente de gradient permet d'améliorer petit à petit les paramètres du réseau de neurones jusqu'à obtenir une convergence, c'est-à-dire une minimisation de la fonction d'erreur.

Cependant, la fonction d'erreur à utiliser n'est pas celle relative à une observation (erreur linéaire), mais à l'ensemble des observations (erreur globale). Par conséquent, l'objectif à atteindre est de trouver des paramètres pouvant minimiser le coût de l'erreur de la fonction MSE vue précédemment.

Comment pouvons-nous nous assurer que nous ne sommes pas dans un minimum local? La fonction MSE est une fonction convexe, c'est-à-dire que sous sa représentation graphique, si nous choisissons deux points de la courbe et que nous traçons un segment entre ces deux points, celui-ci ne coupe jamais la courbe. Par conséquent, il n'y existe pas de minimum local, il n'existe qu'un minimum global. En d'autres termes, si nous cherchons à minimiser cette fonction d'erreur nous sommes certains de trouver un minimum global sous réserve d'avoir choisi un taux d'apprentissage pas trop élevé ET d'attendre suffisamment longtemps, c'est-à-dire réaliser un certain nombre d'époques (epoch).

Une époque est définie comme étant le moment où l'ensemble des observations d'un jeu de données est passé dans le réseau de neurones et a subi les phases de propagation et de rétropropagation. Les époques servent à déterminer si nous sommes sur la bonne voie de minimisation de la fonction d'erreur globale, car c'est à l'issue d'une époque que nous utilisons cette fonction. Car rappelons-nous que la fonction d'erreur globale se calcule sur l'ensemble des erreurs de prédictions réalisées sur l'ensemble des observations.

Cela signifie donc que pour minimiser la fonction d'erreur, il faudra plusieurs époques, et que par conséquent l'ensemble du jeu d'observations devra passer plusieurs fois dans le réseau de neurones.

8. Le biais, un neurone particulier

Nous avons vu que le réseau de neurones est composé de neurones d'entrée, correspondant aux données d'une observation et qu'on attribue un poids à chaque entrée. Ce couple entrée/poids permet de réaliser la phase de propagation à l'aide d'une fonction d'activation.

Maintenant, si nous souhaitons "forcer" la valeur de la prédiction pour certaines valeurs d'entrée cela est possible à l'aide de ce que l'on appelle un biais.

Prenons un exemple simple dont la fonction d'activation est de type ReLU.

Imaginons le cas où notre neurone artificiel est composé de 2 entrées ayant pour valeurs 1 et 2, de 2 poids ayant pour valeurs -0,45 et 0,1 et d'une sortie.

La valeur de cette sortie est calculée comme suit :

$$\text{ReLU}(1 * -0,45 + 2 * 0,1) = 0$$

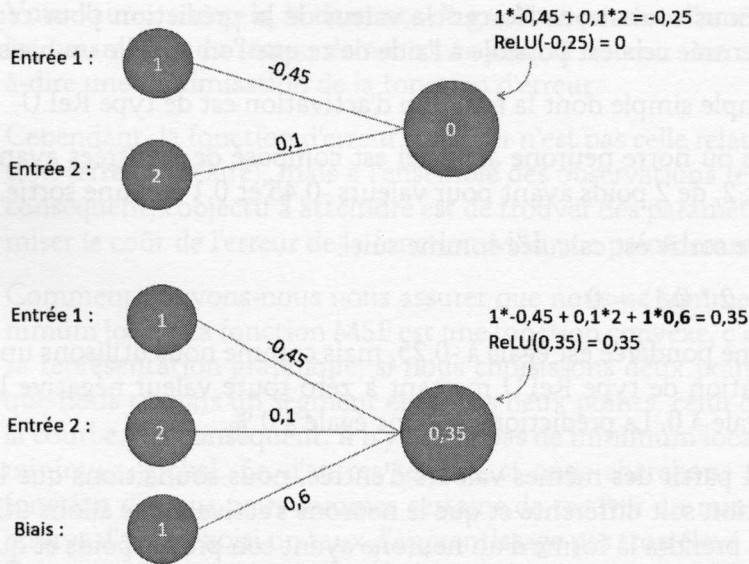
En effet, la somme pondérée est égale à -0,25, mais comme nous utilisons une fonction d'activation de type ReLU mettant à zéro toute valeur négative la prédiction est égale à 0. La prédiction est donc égale à 0 %.

Maintenant, si à partir des mêmes valeurs d'entrée, nous souhaitons que la valeur de prédiction soit différente et que le neurone s'active, nous allons utiliser un biais qui prendra la forme d'un neurone ayant son propre poids et qui va en quelque sorte forcer la prédiction.

Ajoutons donc un neurone qui prendra pour valeur 1 (c'est la valeur que prendra toujours le biais) avec un poids de 0,6. Ce qui nous donne le calcul suivant :

$$\text{ReLU}(1 * -0,45 + 2 * 0,1 + 1 * 0,6) = 0,35$$

On constate donc que l'ajout d'un biais permet de donner une meilleure flexibilité au réseau de neurones dans leur apprentissage.



Rôle du biais

Remarque

En ce qui concerne la valeur du poids du biais à son initialisation, il est commun d'utiliser la valeur 0.

9. Un cas pratique pour comprendre le perceptron

Nous allons à présent détailler de façon chiffrée un exemple qui nous permettra de bien comprendre les concepts vus jusqu'à présent. Mais pas de panique, les calculs ne seront pas compliqués et resteront accessibles en termes de compréhension.

Comme nous l'avons vu au début de ce chapitre, l'objectif du perceptron est de classifier les observations. Nous vous proposons de créer un modèle capable de déterminer si un étudiant, selon des critères précis, peut être reçu ou non dans une prestigieuse université : l'IA Academy.

L'admission dans cette université dépend de la réussite ou non de certains examens d'entrée. Le tableau ci-dessous regroupe différents cas d'admissions et de refus en fonction de la réussite ou non aux examens en mathématiques, informatique.

Réussite à l'examen de mathématiques	Réussite à l'examen d'informatique	Admis
OUI	NON	NON
OUI	OUI	OUI
NON	OUI	NON
NON	NON	NON

Vous l'aurez sans doute remarqué, l'admission à l'université répond à la fonction logique du ET.

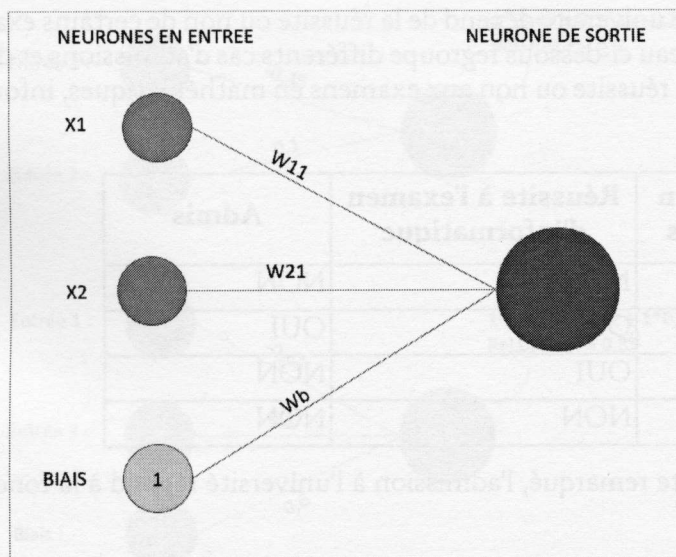
9.1 Initialisation du perceptron

Nous allons initialiser notre perceptron avec une couche de deux neurones en entrée correspondant chacun à la réussite aux examens et un neurone de sortie qui permettra de classifier l'étudiant en tant qu'admis ou refusé dans l'université. En complément des deux neurones d'entrée, nous allons en ajouter un autre appelé biais (seuil) qui a pour but de contrôler la prédisposition du neurone à s'activer ou non et qui prendra toujours la valeur 1.

Notre perceptron est à présent prêt à apprendre.

■ Remarque

Notons la notation particulière de chaque poids. W_{11} , W_{21} et W_b . W ayant pour signification Weight (Poids) suivi du numéro du neurone et du numéro de la couche. Ainsi, W_{11} se lit poids du premier neurone de la première couche, W_{21} poids du deuxième neurone de la première couche, ainsi de suite. W_b signifiant alors poids (Weight) du biais (bias en anglais).

*Neurone formel*

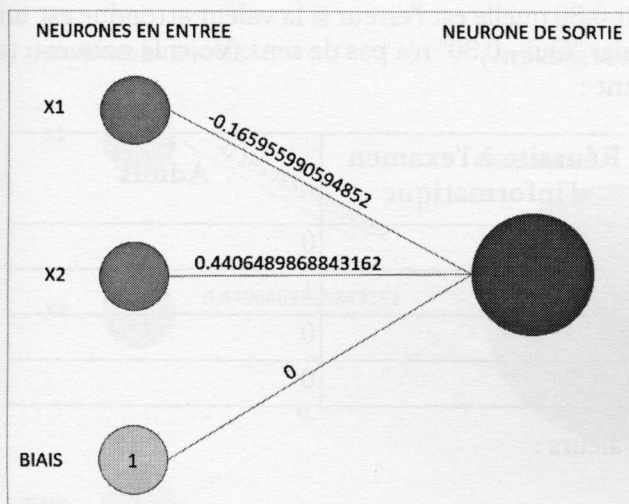
9.2 Les étapes d'apprentissage

9.2.1 Étape 1 : initialisation des poids

La première étape consiste à initialiser les poids. Cette initialisation se fait de façon aléatoire dans un intervalle compris entre -1 et 1 sauf pour le biais qui prendra la valeur 0.

Remarque

Il existe diverses méthodes d'initialisation des poids (Nguyen-Window, Xavier). Nous avons choisi d'utiliser la plus commune. L'intervalle -1 et 1 étant justifié par le fait que lorsque nous utilisons une méthode de génération aléatoire de nombres dans cet intervalle, nous avons approximativement autant de générations effectuées au-dessus et en dessous de zéro. Nous avons donc une distribution régulière.



Génération des poids

$W11 = -0.165955990594852$

$W21 = 0.4406489868843162$

$Wb = 0$

On constate dans notre cas que le poids $W21$ est plus important que les autres, cela signifie donc que la valeur contenue dans le neurone $X2$ a plus d'importance que les autres dans la prédiction. Ce qui est peut-être faux, car cette valeur est uniquement choisie au hasard. Cela ne signifie pas non plus que c'est cette valeur qui sera à la fin de l'apprentissage considérée comme importante, car comme nous l'avons vu, le neurone va apprendre de ses erreurs et ce poids va s'ajuster au fur et à mesure de l'apprentissage.

9.2.2 Étape 2 : chargement des données de la première observation

La seconde étape consiste à charger la première observation contenue dans notre jeu d'observations, venant alors alimenter les neurones $X1$, $X2$.

Bien entendu, les valeurs "Oui" et "Non" des différentes observations ont été remplacées par 1 pour Oui et 0 pour non, car vous le savez désormais, les chaînes de caractères ne sont pas exploitables en Machine Learning et nous verrons que travailler avec des valeurs numériques est important pour calculer l'erreur.

290 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

Car si notre neurone prédit 0,30 quelle est l'erreur si la valeur attendue est une chaîne de caractères, calculer "Oui - 0,30" n'a pas de sens. Voici le nouveau tableau d'observations suivant :

Réussite à l'examen de mathématiques	Réussite à l'examen d'informatique	Admis
1	0	0
1	1	1
0	1	0
0	0	0

Ce qui nous donne pour valeurs :

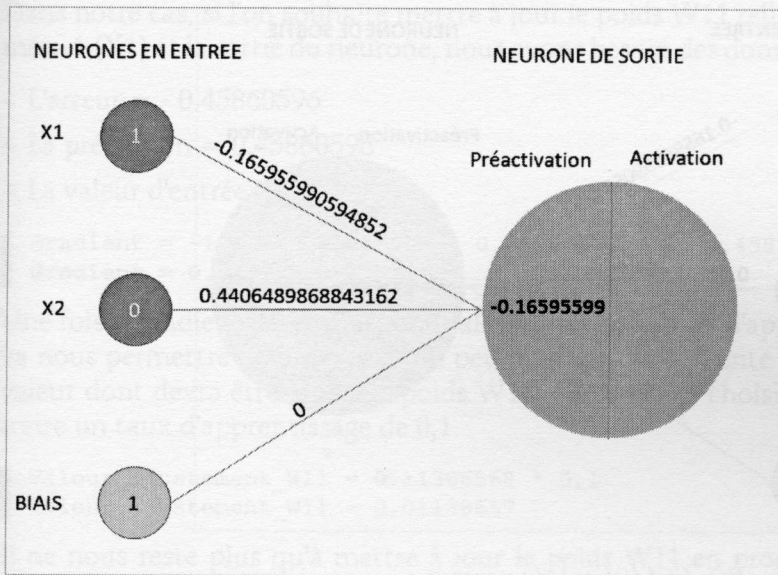
X1 =1

X2 =0

9.2.3 Étape 3 : préactivation

Calculons à présent la somme pondérée des différents poids, correspondant à la phase de préactivation en procédant comme suit :

```
sommePonderee = valeur du biais * wb + (w11*X1 + w21 * X2)
sommePonderee = 1*0 + (-0.165955990594852 * 1 +
0.4406489868843162 * 0)=
sommePonderee = -0.165955990594852
```



Calcul de la préactivation du neurone

9.2.4 Étape 4 : utilisation d'une fonction d'activation

La fonction d'activation que nous allons utiliser est la sigmoïde ou encore appelée courbe en S.

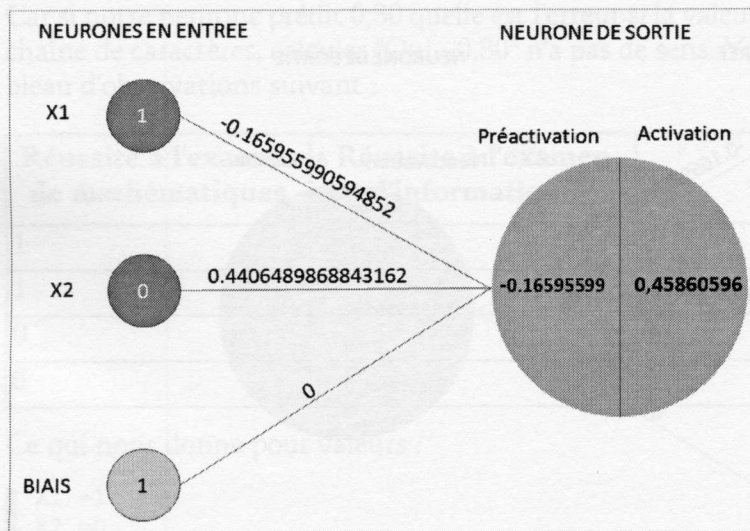
La fonction d'activation a pour rôle de réaliser la prédiction (nommée y) au niveau de notre neurone et a pour formule (exp étant l'exponentielle) :

$$y = 1 / (1 + \exp(-\text{somme_ponderee}))$$

Ce qui nous donne pour notre cas d'étude :

$$Y = 1 / (1 + \exp(-(-0.165955990594852)))$$

$$Y = 0,45860596$$



Calcul de l'activation (prédiction) du neurone

9.2.5 Étape 5 : calcul de l'erreur linéaire commise lors de l'apprentissage

La prédiction réalisée est de 0.363248, alors que nous attendions la valeur 0. Nous allons donc calculer la différence entre la valeur attendue et la prédiction réalisée pour déterminer l'erreur linéaire commise lors de l'apprentissage :

$$\text{Erreur} = 0 - 0.45860596$$

$$\text{Erreur} = - 0.45860596$$

9.2.6 Étape 6 : ajustement des poids synaptiques

Nous le savons à présent, le perceptron va apprendre de ses erreurs en ajustant les différents poids de chaque entrée jusqu'à atteindre une convergence. Cet ajustement se fait en fonction de l'erreur calculée précédemment et en réalisant la rétropropagation de l'erreur.

Cette rétropropagation s'effectue dans un premier en calculant le gradient se formulant comme suit :

$$\text{Gradient} = -1 * \text{ERREUR} * \text{PREDICTION} * (1 - \text{PREDICTION}) * \text{VALEUR_ENTREE}$$

Dans notre cas, si l'on souhaite mettre à jour le poids W11 reliant l'entrée numéro 1 (X1) et la sortie du neurone, nous avons besoin des données suivantes :

- L'erreur = - 0.45860596
- La prédiction = 0.45860596
- La valeur d'entrée X1 = 1

```
Gradient = -1 * - 0.45860596 * 0.45860596 * (1-0.45860596) * 1
Gradient = 0,11386568
```

Une fois le gradient déterminé, nous allons utiliser le taux d'apprentissage qui va nous permettre de progresser un peu plus dans la descente en calculant la valeur dont devra être ajusté le poids W11. Nous avons choisi de façon arbitraire un taux d'apprentissage de 0,1.

```
Valeur_ajustement_W11 = 0.11386568 * 0,1
Valeur_ajustement_W11 = 0,01138657
```

Il ne nous reste plus qu'à mettre à jour le poids W11 en procédant comme suit :

```
Nouveau_W11 = W11 - Valeur_ajustement_W11
Nouveau_W11 = -0.165955990594852 - 0,01138657
Nouveau_W11 = -0.15456942
```

Nous devons alors procéder de la même façon pour ajuster le poids W21 et le poids du biais.

	X1	X2	BIAIS
VALEUR	1	0	1
POIDS	-0,16595599	0,44064899	0
ERREUR	-0,45860596	-0,45860596	-0,45860596
PREDICTION	0,45860596	0,45860596	0,45860596
GRADIENT	-0,11386568	0	-0,11386568
TAUX APPRENTISSAGE	0,1	0,1	0,1
VALEUR_AJUSTEMENT	-0,01138657	0	-0,01138657
NOUVEAU POIDS	-0,15456942	0,44064899	0,01138657

Une fois les poids ajustés, nous devons charger les données de la seconde observation et recommencer le processus jusqu'à ce que la fonction d'erreur soit minimisée.

Au lieu de réaliser les différents calculs manuellement, nous allons créer un nouveau projet Python et coder notre premier neurone formel !

10. Codons notre premier neurone formel "From Scratch"

Les précédents paragraphes demandent sans doute plusieurs lectures avant d'être totalement compris. C'est pourquoi nous allons en complément de l'exemple chiffré réalisé précédemment coder notre premier neurone formel afin de bien comprendre toute la mécanique de l'apprentissage à travers un exemple concret.

Vous êtes maintenant habitué, nous vous laissons le soin de créer un nouveau projet dans l'éditeur PyCharm et d'y ajouter un nouveau fichier script que nous nommerons `perceptron.py`.

Pour mener à bien le projet, nous aurons besoin des librairies `matplotlib` et `numpy`.

10.1 Les données d'apprentissage

La première chose que nous allons faire est de créer nos données d'apprentissage et définir le taux d'apprentissage :

```
#-----  
#   OBSERVATIONS ET PREDICTIONS  
#-----  
  
observations_entrees = array([  
                                [1, 0],  
                                [1, 1],  
                                [0, 1],  
                                [1, 0]])  
  
predictions = array([[0], [1], [0], [0]])
```

10.2 Définition des poids

Passons maintenant à la génération des poids de façon aléatoire, compris dans l'intervalle de valeur $[-1,1]$:

```
#Génération des poids dans l'intervalle [-1;1]
random.seed(1)
borneMin = -1
borneMax = 1

w11 = (borneMax-borneMin) * random.random() + borneMin
w21 = (borneMax-borneMin) * random.random() + borneMin
w31 = (borneMax-borneMin) * random.random() + borneMin
```

N'oublions pas le biais qui aura pour valeur 1 et dont le poids sera égal à 0.

```
#Le biais
biais = 1
wb = 0
```

10.3 Gestion des hyperparamètres

Nous allons à présent définir les hyperparamètres représentant le nombre d'époques et le taux d'apprentissage.

```
#Taux d'apprentissage
txApprentissage = 0.1

#Nombre d'epochs
epochs = 300
```

10.4 Codage de fonctions utiles

Le code ci-dessous reprend les différentes fonctions dont nous aurons besoin à savoir :

- Le calcul de la somme pondérée
- Le calcul de la fonction d'activation de type sigmoïde
- Le calcul de l'erreur linéaire

- Le calcul du gradient
- Le calcul de la valeur d'ajustement du poids
- Le calcul de la nouvelle valeur du poids
- Le calcul de la fonction d'erreur moyenne quadratique (MSE)

```
#-----
#          FONCTIONS UTILES
#-----

def somme_ponderee(X1,W11,X2,W21,B,WB):
    return (B*WB+( X1*W11 + X2*W21))

def fonction_activation_sigmoide(valeur_somme_ponderee):
    return (1 / (1 + exp(-valeur_somme_ponderee)))

def fonction_activation_relu(valeur_somme_ponderee):
    return (max(0,valeur_somme_ponderee))

def erreur_lineaire(valeur_attendue, valeur_predite):
    return (valeur_attendue-valeur_predite)

def calcul_gradient(valeur_entree,prediction,erreur):
    return (-1 * erreur * prediction * (1-prediction) *
    valeur_entree)

def calcul_valeur_ajustement(valeur_gradient,
    taux_apprentissage):
    return (valeur_gradient*taux_apprentissage)

def calcul_nouveau_poids (valeur_poids, valeur_ajustement):
    return (valeur_poids - valeur_ajustement)

def calcul_MSE(predictions_realisees, predictions_attendues):
    i=0;
    for prediction in predictions_attendues:
        difference = predictions_attendues[i] -
```

```
predictions_realisees[i]
    carreDifference = difference * difference
    somme = somme + carreDifference
    moyenne_quadratique = 1 / (len(predictions_attendues)) *
somme
    return moyenne_quadratique
```

10.5 Passons à l'apprentissage!

Maintenant que nous disposons de tout ce dont nous avons besoin, nous pouvons passer à la phase d'apprentissage.

Pour mener à bien cette phase d'apprentissage, nous allons réaliser plusieurs époques (epoch), c'est-à-dire plusieurs passages complets de l'ensemble des observations contenues dans notre jeu de données à travers notre neurone formel. Pour chaque observation, nous allons réaliser une prédiction et calculer l'erreur pour ensuite procéder à la mise à jour des poids synaptiques.

Voici le code lié à l'apprentissage :

```
#-----
#   APPRENTISSAGE
#-----

for epoch in range(0, epochs):
    print("EPOCH (" + str(epoch) + "/" + str(epochs) + ")")
    predictions_realisees_durant_epoch = []
    predictions_attendues = []
    numObservation = 0
    for observation in observations_entrees:

        #Chargement de la couche d'entrée
        x1 = observation[0];
        x2 = observation[1];

        #Valeur de prédiction attendue
        valeur_attendue = predictions[numObservation][0]

        #Etape 1 : Calcul de la somme pondérée
        valeur_somme_ponderee =
somme_ponderee(x1, w11, x2, w21, biai, wb)
```

```

#Etape 2 : Application de la fonction d'activation
valeur_predite =
fonction_activation_sigmoide(valeur_somme_ponderee)

#Etape 3 : Calcul de l'erreur
valeur_erreur =
erreur_lineaire(valeur_attendue, valeur_predite)

#Mise à jour du poids 1
#Calcul du gradient de la valeur d'ajustement et du
nouveau poids
gradient_W11 =
calcul_gradient(x1, valeur_predite, valeur_erreur)
valeur_ajustement_W11 =
calcul_valeur_ajustement(gradient_W11, txApprentissage)
w11 = calcul_nouveau_poids(w11, valeur_ajustement_W11)

# Mise à jour du poids 2
gradient_W21 = calcul_gradient(x2, valeur_predite,
valeur_erreur)
valeur_ajustement_W21 =
calcul_valeur_ajustement(gradient_W21, txApprentissage)
w21 = calcul_nouveau_poids(w21, valeur_ajustement_W21)

# Mise à jour du poids du biais
gradient_Wb = calcul_gradient(biais, valeur_predite,
valeur_erreur)
valeur_ajustement_Wb =
calcul_valeur_ajustement(gradient_Wb, txApprentissage)
wb = calcul_nouveau_poids(wb, valeur_ajustement_Wb)

print("      EPOCH (" + str(epoch) + "/" + str(epochs) + ") -
Observation: " + str(numObservation+1) + "/" +
str(len(observations_entrees)))

#Stockage de la prédiction réalisée:
predictions_realisees_durant_epoch.append(valeur_predite)

predictions_attendues.append(predictions[numObservation][0])

```

```
#Passage à l'observation suivante
numObservation = numObservation+1

MSE = calcul_MSE(predictions_realisees_durant_epoch,
predictions)
Graphique_MSE.append(MSE[0])
print("MSE : "+str(MSE))
```

Rien de bien compliqué dans la compréhension pour ces quelques lignes. Si l'on exécute ce code, nous pouvons voir que la fonction d'erreur diminue bien au fil du temps, ce qui montre que notre neurone formel est bien en train d'apprendre.

```
EPOCH (297/300)
  EPOCH (297/300) - Observation: 1/4
  EPOCH (297/300) - Observation: 2/4
  EPOCH (297/300) - Observation: 3/4
  EPOCH (297/300) - Observation: 4/4
MSE : [0.08400651]
EPOCH (298/300)
  EPOCH (298/300) - Observation: 1/4
  EPOCH (298/300) - Observation: 2/4
  EPOCH (298/300) - Observation: 3/4
  EPOCH (298/300) - Observation: 4/4
MSE : [0.08390084]
EPOCH (299/300)
  EPOCH (299/300) - Observation: 1/4
  EPOCH (299/300) - Observation: 2/4
  EPOCH (299/300) - Observation: 3/4
  EPOCH (299/300) - Observation: 4/4
MSE : [0.08379514]
```

Mais avons-nous atteint le point de convergence?

10.6 À la recherche du point de convergence

Pour voir si nous avons atteint le point de convergence, nous allons tout d'abord afficher la courbe de la fonction d'erreur.

Pour cela, ajoutons avant la fonction d'apprentissage une liste qui nous permettra de stocker les différentes valeurs de la fonction d'erreurs MSE tout au long de la phase d'apprentissage :

```
#-----
#          GRAPHIQUE
#-----

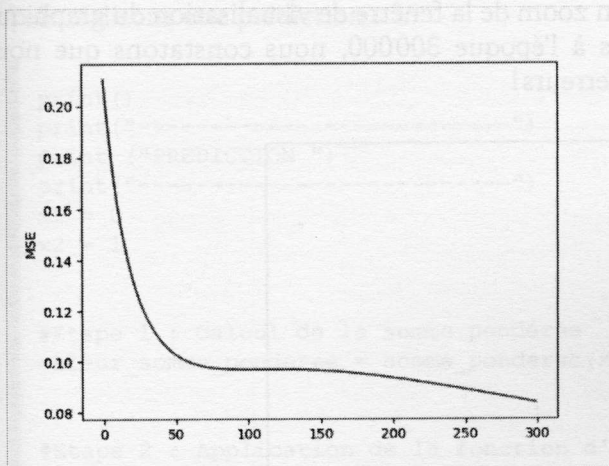
Graphique_MSE=[]

#-----
#          APPRENTISSAGE
#-----
```

Puis en fin d'apprentissage (après la boucle For epoch...), créons et affichons le graphique :

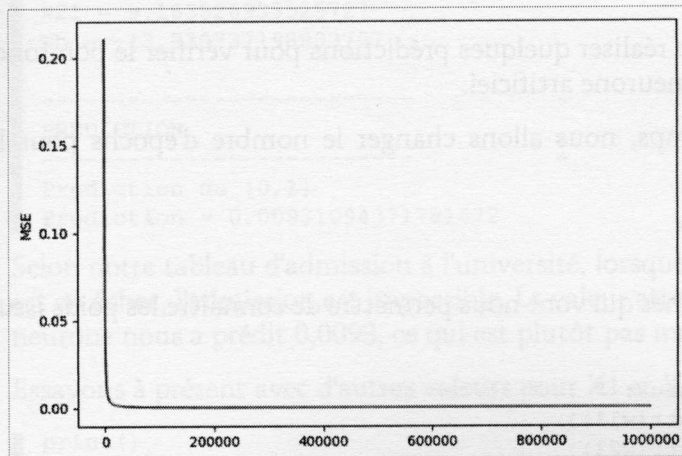
```
import matplotlib.pyplot as plt
plt.plot(Graphique_MSE)
plt.ylabel('MSE')
plt.show()
```

Ce qui nous donne la courbe représentée par la figure suivante où l'on constate une diminution de l'erreur puis un léger palier pour ensuite reprendre la descente, mais cela ne nous indique pas si la convergence a été atteinte, on peut juste en déduire qu'avec une fonction d'erreur montrant un taux de 8 % et vu l'allure de la courbe, nous pouvons encore espérer de meilleurs résultats.



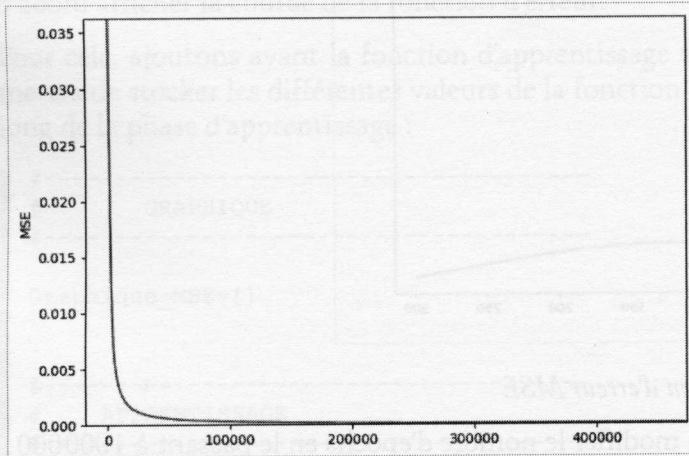
Graphique de la fonction d'erreur MSE

Nous vous invitons à modifier le nombre d'époques en le passant à 1000000 et à relancer l'apprentissage (attention, cela peut prendre du temps). Cette énorme valeur a volontairement été choisie pour vous montrer que notre algorithme continue toujours à améliorer son apprentissage, mais de façon très minime puisque la courbe de la fonction d'erreur semble se stabiliser.



Graphique de la fonction d'erreur avec 1000000 epochs

Si nous utilisons la fonction zoom de la fenêtre de visualisation du graphique et que nous nous rendons à l'époque 300000, nous constatons que nous sommes proches des 0 % d'erreurs!



Graphique de la fonction d'erreur proche de zéro avec 300000 époques

10.7 Tests de prédictions

Essayons à présent de réaliser quelques prédictions pour vérifier le bon fonctionnement de notre neurone artificiel.

Dans un premier temps, nous allons changer le nombre d'epochs pour le mettre à 300000.

```
#Nombre d'époques
epochs = 300000
```

Ajoutons quelques lignes qui vont nous permettre de connaître les poids issus de l'apprentissage :

```
print ("Poids finaux: ")
print ("W11 = "+str(w11))
print ("W21 = "+str(w21))
print ("Wb = "+str(wb))
```

Puis réalisons une prédiction :

```
print()
print("-----")
print ("PREDICTION ")
print("-----")
x1 = 0
x2 = 1

#Etape 1 : Calcul de la somme pondérée
valeur_somme_ponderee = somme_ponderee(x1,w11,x2,w21,biais,wb)

#Etape 2 : Application de la fonction d'activation
valeur_predite =
fonction_activation_sigmoide(valeur_somme_ponderee)

print("Prédiction du [" + str(x1) + "," + str(x2) + "]")
print("Prédiction = " + str(valeur_predite))

Poid finaux:
W11 = 9.163546255436527
W21 = 9.163526933525727
Wb = -13.830737198803797

-----
PREDICTION
-----
Prediction du [0,1]
Prediction = 0.00931094371781672
```

Selon notre tableau d'admission à l'université, lorsque l'un des deux examens est en échec, l'admission est impossible. La valeur attendue est donc de 0 et le neurone nous a prédit 0,0093, ce qui est plutôt pas mal!

Essayons à présent avec d'autres valeurs pour X1 et X2.

```
print()
print("-----")
print ("PREDICTION ")
print("-----")
```

```
x1 = 1
x2 = 1
```

```
Poid finaux:
W11 = 9.163546255436527
W21 = 9.163526933525727
Wb = -13.830737198803797
```

```
-----
PREDICTION
-----
```

```
Prediction du [1,1]
Prediction = 0.9889731719767871
```

Cette fois-ci, la prédiction est très proche de 1 comme attendu puisque nous lui avons indiqué une réussite aux deux examens.

Notre neurone formel est donc fonctionnel!

Comme nous pouvons le constater, en quelques lignes de code, nous avons réussi à programmer un neurone formel capable de réaliser des prédictions sur des données linéairement séparables (nous insistons sur ce point).

■ Remarque

Attention au surapprentissage! Si nous augmentons le nombre d'epochs, le neurone fera moins d'erreurs, mais sa généralisation ne pourra se faire. Nous vous invitons à consulter le chapitre Machine Learning et Pokémon : seconde partie pour plus d'informations sur ce point.

11. Un neurone artificiel avec TensorFlow

Après avoir codé notre neurone artificiel de façon "artisanale", nous allons à présent voir comment utiliser le module TensorFlow pour créer ce même neurone artificiel.

11.1 Un petit mot sur TensorFlow

De nos jours, lorsque nous demandons à des développeurs de nous conseiller un outil de Machine Learning et de Deep Learning, leur réponse est souvent le framework "TensorFlow" développé par Google.

En 2011, le projet Google Brain voit le jour en ayant pour objectif d'améliorer les performances commerciales et les expériences utilisateurs de Google, à l'aide du Deep Learning. En effet, au vu de la quantité de données accumulées par Google à l'aide de ses produits Gmail, Google Photo et son moteur de recherche, le Deep Learning semblait tout indiqué pour améliorer les performances commerciales de Google à partir de ces données, sans oublier l'axe recherche scientifique. De cette activité est né le produit interne à Google nommé DistBielef qui fut remanié afin de le rendre plus rapide et plus solide tout en prenant le nom de TensorFlow. En 2015, Google décide de rendre publique la bibliothèque TensorFlow, avec une version reconnue comme stable en 2017.

Tensorflow s'articule autour d'une architecture composée de trois briques qui sont :

- Le prétraitement des données
- La construction des modèles de prédictions
- L'évaluation des modèles

TensorFlow tire son nom du terme mathématique Tensor (Tenseur) exprimant la faculté de traiter des tableaux en multidimension et de Flow (Processus) permettant de représenter toutes les étapes nécessaires à la création d'un modèle.

Maintenant que nous en savons un peu plus sur le module TensorFlow, nous pouvons l'ajouter à notre projet pour découvrir comment l'utiliser sur notre cas pratique.

■ Remarque

L'installation du module TensorFlow dans l'outil de développement PyCharm est identique à l'installation des autres modules que vous avez maintenant l'habitude d'installer.

11.2 Données d'apprentissage et de tests

Une fois le module TensorFlow installé, nous vous invitons à créer un nouveau fichier de script Python dans votre projet et à l'intérieur de celui-ci y saisir ces quelques lignes :

```
import tensorflow as tf

#-----
#      DONNEES D'APPRENTISSAGE
#-----

valeurs_entrees_X = [[1., 0.], [1., 1.], [0., 1.], [0., 0.]]
valeurs_a_predire_Y = [[0.], [1.], [0.], [0.]]
```

Après avoir importé le module TensorFlow, nous créons des tableaux chargés de contenir les observations. Chaque donnée est spécifiée comme étant au format décimal pour un meilleur traitement.

11.3 Paramétrage du neurone

Nous allons à présent paramétrer le neurone comme suit :

```
#-----
#      PARAMETRES DU RESEAU
#-----
#Variable TensorFlow correspondant aux valeurs neurones d'entrée
tf_neurones_entrees_X = tf.placeholder(tf.float32, [None, 2])

#Variable TensorFlow correspondant au neurone de sortie
#(prédiction réelle)
tf_valeurs_reelles_Y = tf.placeholder(tf.float32, [None, 1])

#-- Poids --
#Création d'une variable TensorFlow de type tableau
#contenant 3 lignes et dont les valeurs sont de type décimal
#Ces valeurs sont initialisées au hasard
poids = tf.Variable(tf.random_normal([2, 1]), tf.float32)
```

```
-- Biais initialisé à 0 --
biais = tf.Variable(tf.zeros([1, 1]), tf.float32)

#La somme pondérée est en fait une multiplication de matrice
#entre les valeurs en entrée X et les différents poids
#la fonction matmul se charge de faire cette multiplication
sommeponderee = tf.matmul(tf_neurones_entrees_X, poids)

#Ajout du biais à la somme pondérée
sommeponderee = tf.add(sommeponderee, biais)

#Fonction d'activation de type sigmoïde permettant de calculer la
prédiction
prediction = tf.sigmoid(sommeponderee)

#Fonction d'erreur de moyenne quadratique MSE
fonction_erreur = tf.reduce_sum(tf.pow(tf_valeurs_reelles_Y-
prediction, 2))

#Descente de gradient avec un taux d'apprentissage fixé à 0.1
optimiseur =
tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize
(fonction_erreur)
```

Dans le paramétrage TensorFlow, il existe des variables pouvant être initialisées immédiatement et d'autres ultérieurement c'est-à-dire au cours des différentes phases de l'apprentissage. C'est le cas des données en entrée de chaque neurone et celles des prédictions attendues, car elles sont chargées au fur et à mesure de l'apprentissage, dans le but d'alimenter (feed) le modèle d'apprentissage. Pour ce type de données, nous utiliserons des variables de type `tf.Placeholder`.

Quant aux poids et au biais, ceux-ci sont initialisés dès le début et de façon aléatoire (random) et seront amenés à être calculés et modifiés tout au long de l'apprentissage d'où l'utilisation de variables de type `tf.Variable`.

On trouve ensuite dans cette succession de lignes de code la création de la fonction de calcul de la somme pondérée se faisant en deux temps.

Le premier temps en faisant la somme pondérée des poids et de valeurs d'entrée via une multiplication des matrices des valeurs d'entrées et des poids (`tf.matmul`).

Puis dans un second temps en y ajoutant (`add`) le biais (`tf.add`).

Viennent ensuite la définition de la fonction d'activation de type sigmoïde (`tf.sigmoid`), puis la fonction de calcul d'erreur établie en calculant la moyenne (`tf.reduce_mean`) des erreurs quadratiques. La fonction `tf.pow` signifiant "élever à la puissance", nous pouvons donc déduire que l'instruction :

```
■ tf.reduce_sum(tf.pow(tf_valeurs_reelles_Y-prediction,2))
```

permet de calculer la différence entre la valeur prédite et la valeur réelle attendue. Cette différence étant élevée au carré (notion de quadratique vue un peu plus haut dans ce chapitre).

On ajoute enfin une ligne permettant de définir l'optimiseur, c'est-à-dire dans notre cas la recherche d'une convergence en minimisant la fonction d'erreur à l'aide de la descente de gradient tout en utilisant un taux d'apprentissage de 0,1.*.

Notre neurone formel est à présent configuré. Nous allons pouvoir maintenant réaliser l'apprentissage.

11.4 L'apprentissage

Voici à présent le code permettant de réaliser l'apprentissage à l'aide de TensorFlow :

```
#-----
#   APPRENTISSAGE
#-----

#Nombre d'epochs
epochs = 10000

#Initialisation des variables
init = tf.global_variables_initializer()

#Démarrage d'une session d'apprentissage
```

```
session = tf.Session()
session.run(init)

#Pour la réalisation du graphique pour la MSE
Graphique_MSE=[]

#Pour chaque epoch
for i in range(epochs):

    #Réalisation de l'apprentissage avec mise à jour des poids
    session.run(optimiseur, feed_dict = {tf_neurones_entrees_X:
valeurs_entrees_X, tf_valeurs_reelles_Y:valeurs_a_predire_Y})

    #Calculer l'erreur
    MSE = session.run(fonction_erreur, feed_dict =
{tf_neurones_entrees_X: valeurs_entrees_X,
tf_valeurs_reelles_Y:valeurs_a_predire_Y})

    #Affichage des informations
    Graphique_MSE.append(MSE)
    print("EPOCH (" + str(i) + "/" + str(epochs) + ") - MSE: " +
str(MSE))

#Affichage graphique
import matplotlib.pyplot as plt
plt.plot(Graphique_MSE)
plt.ylabel('MSE')
plt.show()
```

Comme nous pouvons le voir, les premières étapes consistent à initialiser les variables et à créer une session. S'en suit l'apprentissage à proprement parler en réalisant plusieurs epochs (10000 dans notre cas).

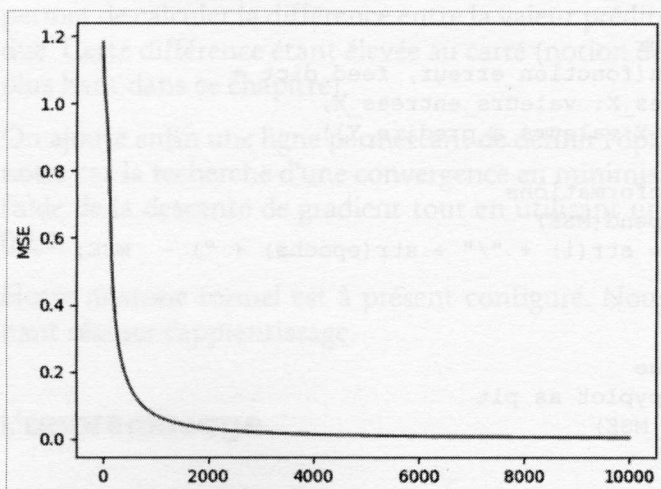
Là où nous avons écrit plusieurs méthodes pour la mise à jour des poids, celles-ci se réalisent un peu sous forme d'une boîte noire dans la fonction :

```
session.run(optimiseur, feed_dict = {tf_neurones_entrees_X:
valeurs_entrees_X, tf_valeurs_reelles_Y:valeurs_a_predire_Y})
```

Cette ligne de code indique à TensorFlow d'utiliser l'ensemble des données en entrée, de les stocker dans sa variable `tf_neurones_entrees_X` et de faire de même avec les valeurs à prédire en les stockant dans sa variable `tf_va-`

leurs_reelles. Puis d'utiliser l'optimiseur qui réalise la prédiction et cherche à minimiser la fonction d'erreur. On est donc en droit de se demander comment TensorFlow procède pour la mise jour de poids. Et bien en cherchant à minimiser la fonction d'erreur qui est calculée en fonction de la somme pondérée, qui elle-même est calculée en fonction des poids, TensorFlow réalise des opérations d'optimisation et de mise à jour des poids.

Si nous exécutons à présent le code, nous pouvons voir que l'apprentissage se réalise correctement et que la fonction d'erreur est minimisée plus rapidement. Cela est dû à la fonction d'optimisation qui est sans doute plus robuste que notre code écrit précédemment.



Graphique de la minimisation de la fonction d'erreur MSE avec TensorFlow

11.5 Tests de prédictions

Nous allons à présent tester le bon apprentissage de notre neurone formel.

Les différents poids étant mis à jour suite à l'apprentissage et gardés en mémoire, pour tester le bon apprentissage de notre neurone formel, nous allons utiliser le neurone de sortie chargé de réaliser la prédiction en lui passant chaque observation une à une via le paramètre `feed_dict`.

Le code ci-dessous montre que nous passons en paramètre l'observation dont nous souhaitons la prédiction en la stockant dans la variable TensorFlow `tf_neurones_entrees_X`, puis nous réalisons la prédiction à l'aide du neurone de sortie nommé `prediction` et en appelant la fonction `run` de la session TensorFlow.

```
print("--- VERIFICATIONS ----")

for i in range(0,4):
    print("Observation:"+str(valeurs_entrees_X[i])+ " - Attendu: "+str(valeurs_a_predire_Y[i])+ " - Prediction: "+str(session.run(prediction, feed_dict={tf_neurones_entrees_X:[valeurs_entrees_X[i]]})))
```

Voici les résultats obtenus correspondant aux classifications attendues :

```
--- VERIFICATIONS ---
Observation:[1.0, 0.0] - Attendu: [0.0] - Prediction:
[[0.038227]]
Observation:[1.0, 1.0] - Attendu: [1.0] - Prediction:
[[0.9545917]]
Observation:[0.0, 1.0] - Attendu: [0.0] - Prediction:
[[0.038227]]
Observation:[0.0, 0.0] - Attendu: [0.0] - Prediction:
[[0.00007514158]]
```

12. Un premier pas vers le Deep Learning

Nous voici au terme de ce chapitre qui nous a permis de comprendre le fonctionnement d'un neurone formel qui, rappelons-le, n'est **capable de ne classifier que des données linéairement séparables**. Cependant, nous l'avons vu au cours des différents exemples rencontrés jusqu'à présent, cette notion de séparation linéaire n'est pas adaptée à tous les problèmes de classification.

Par conséquent, dans le chapitre suivant nous allons voir qu'en ajoutant des couches de neurones supplémentaires à notre neurone formel, nous allons pouvoir classifier des données non linéairement séparables.

Chapitre 11

Utilisation de plusieurs couches de neurones

1. Ce que nous allons découvrir et les prérequis

Après avoir découvert dans le chapitre précédent le fonctionnement d'un neurone formel, nous allons à présent nous attarder sur les réseaux de neurones multicouches.

■ Remarque

Prérequis nécessaires pour bien aborder ce chapitre : avoir lu les chapitres Les fondamentaux du langage Python, Des statistiques pour comprendre les données, Principaux algorithmes du Machine Learning et Un neurone pour prédire.

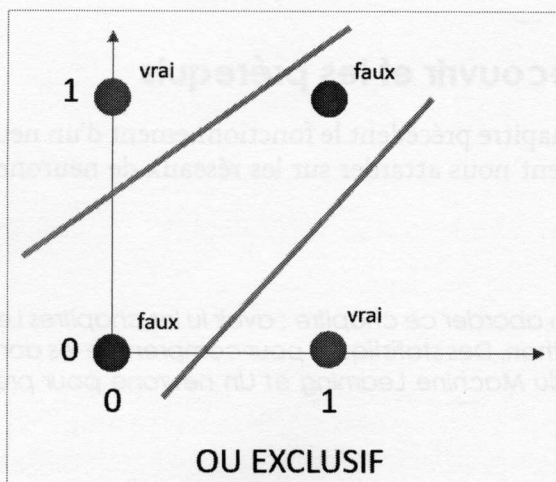
2. Fonctionnement des réseaux de neurones multicouches

Le fonctionnement d'un neurone formel à l'aide de neurones d'entrée, de poids, d'un biais et une fonction d'activation est assez simple à comprendre. L'objectif de ce neurone étant de minimiser sa fonction de coût en apprenant de ses erreurs. Nous avons beaucoup insisté dans le chapitre précédent sur le fait qu'un neurone formel est capable uniquement de classer des données linéairement séparables, or dans la pratique, il est bien rare que celles-ci le soient. C'est alors que sont apparus les réseaux de neurones multicouches.

Leur fonctionnement est identique à celui du neurone formel. À ce titre, nous retrouverons les actions de propagation et de rétropropagation s'appliquant à chaque couche du réseau. La fonction d'activation ne s'appliquant quant à elle qu'aux neurones chargés de réaliser la classification.

3. Le cas du Ou exclusif (XOR)

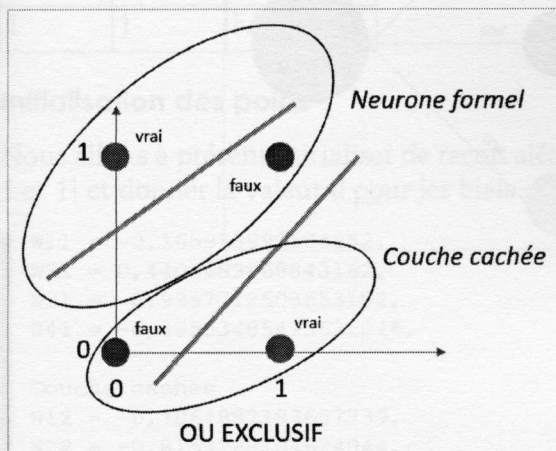
Souvenez-vous, nous avons vu que lorsque nous sommes face à l'utilisation de la fonction logique Ou Exclusif encore appelée XOR, les données ne sont pas linéairement séparables. En effet, il existe deux droites de séparations et non une seule. Par conséquent, un seul neurone ne peut réussir à classer les données.



Représentation graphique de la fonction logique OU Exclusif

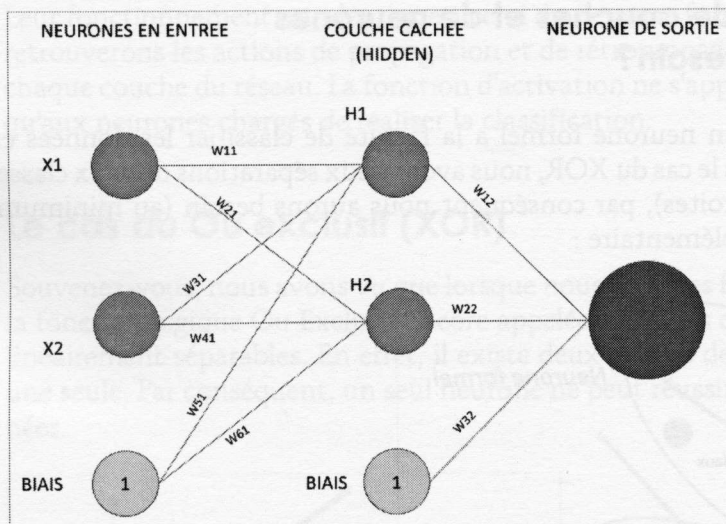
3.1 De combien de couches et de neurones avons-nous besoin?

Nous savons qu'un neurone formel a la faculté de classer les données en deux classes. Dans le cas du XOR, nous avons deux séparations de deux classes à réaliser (deux droites), par conséquent nous aurons besoin (au minimum) d'une couche supplémentaire :



Nombre de couches de neurones nécessaires

Nous pouvons alors modéliser notre réseau de neurones comme présenté sur la figure suivante, montrant les neurones d'entrée, la couche cachée, le neurone de sortie sans oublier les biais. Les neurones étant tous connectés les uns aux autres, nous disons que notre réseau est "complètement connecté" (*Full connected*).



Réseaux de neurones multicouches

3.2 Un exemple chiffré

Il n'y a pas de grande complexité au codage de ces réseaux, les étapes à suivre sont celles correspondant au neurone formel :

- Initialisation des poids
- Calcul de la préactivation (sommes pondérées)
- Activation
- Calcul de l'erreur
- Rétropropagation (mise à jour des poids de chaque couche)

3.2.1 Les données d'apprentissage

Le tableau ci-dessous reprend les différents résultats liés à la logique du Ou Exclusif respectant les règles suivantes :

- Le résultat est VRAI si un et un seul des opérandes A et B est VRAI.
- Le résultat est VRAI si les deux opérandes A et B ont des valeurs distinctes.

– Le résultat est VRAI si un nombre impair d'entrées est vrai.

OU EXCLUSIF (XOR)		
A	B	A XOR B
0	0	FAUX
0	1	VRAI
1	0	VRAI
1	1	FAUX

3.2.2 Initialisation des poids

Nous allons à présent initialiser de façon aléatoire les poids dans l'intervalle $[-1 \text{ et } 1]$ et donner la valeur 0 pour les biais :

```
W11 = -0,165955990594852,
W21 = 0,4406489868843162,
W31 = -0,9997712503653102,
W41 = -0,39533485473632046,
```

Couche cachée

```
W12 = -0,7064882183657739,
W22 = -0,8153228104624044,
```

Biais

```
W51 = 0,
W61 = 0,
W32 = 0,
```

3.2.3 Chargement des données d'entrée

Chargeons maintenant les données d'entrée correspondant au premier cas d'apprentissage :

```
X1 = 0
X2 = 0
ATTENDU = 0
```

3.2.4 Calcul de la préactivation du neurone de sortie

Calculons à présent la valeur du neurone H1 en réalisant une préactivation et une activation du neurone à l'aide d'une fonction d'activation de type sigmoïde :

```
Pre_activation_H1 = (X1*W11 + X2*W31) + (1*W51)
H1 = sigmoïde(Pre_activation_H1)
H1 = sigmoïde(0)
H1 = 1 / (1+EXP(0))
H1= 0,5
```

Puis procédons de la même façon pour la valeur du neurone H2.

```
Pre_activation_H2 = (X1*W21 + X2*W41) + (1*W51)
H2 = sigmoïde(Pre_activation_H2)
H2 = sigmoïde(0)
H2 = 1 / (1+EXP(0))
H2 = 0,5
```

Passons ensuite à la préactivation du neurone de sortie :

```
Pre_activation = (H1*W12 + H2*W22) + (1*W31)
Pre_activation = -0,760905514
```

3.2.5 Calcul de l'activation

Comme nous en avons l'habitude, nous allons à présent calculer l'activation du neurone de sortie à l'aide d'une fonction d'activation de type sigmoïde.

```
Y = sigmoïde(-0,760905514)
Y = 1 / (1+EXP(-0,760905514))
Y = 0,318449702
```

3.2.6 Calcul de l'erreur

L'erreur se calcule en faisant la différence entre la valeur attendue et la prédiction réalisée :

```
Erreur = 0 - 0,318449702
Erreur = -0,318449702
```

3.2.7 Mise à jour des poids

La mise à jour des poids s'effectue comme nous l'avons fait pour le réseau formel en calculant le gradient et la valeur d'ajustement et en partant du neurone de prédiction vers les neurones d'entrée. À noter que nous avons choisi un taux d'apprentissage de 0,1.

Voici le calcul pour la mise à jour du poids W12 :

```
Gradient = -1 * erreur * Y * (1-Y) * H1
```

```
Gradient = 0,03455808
```

```
Valeur_Ajustement = Taux d'apprentissage * Gradient
```

```
Valeur_Ajustement = 0,1 * Gradient
```

```
Valeur_Ajustement = 0,00345581
```

```
Nouveau W12 = W12 - Valeur_Ajustement
```

```
Nouveau W12 = -0,70994403
```

Nous avons procédé de la même façon pour la mise à jour des différents poids et obtenons les résultats suivants :

W12		W22	
H1	0,5	H2	0,5
Y	0,3184497	Y	0,3184497
Erreur	-0,3184497	Erreur	-0,3184497
Gradient	0,03455808	Gradient	0,03455808
taux apprentissage	0,1	taux apprentissage	0,1
valeur ajustement	0,00345581	valeur ajustement	0,00345581
W12	-0,70648822	W22	-0,81532281
Nouveau W12	-0,70994403	Nouveau W22	-0,81877862

W32		W61	
BIAIS	1	BIAIS	1

W32		W61	
Y	0,3184497	Y	0,3184497
Erreur	-0,3184497	Erreur	-0,3184497
Gradient	0,06911616	Gradient	0,06911616
taux apprentissage	0,1	taux apprentissage	0,1
valeur ajustement	0,00691162	valeur ajustement	0,00691162
W32	0	W61	0
Nouveau W32	-0,00691162	Nouveau W61	-0,00691162

W51		W41	
BIAIS	1	X2	0
Y	0,3184497	Y	0,3184497
Erreur	-0,3184497	Erreur	-0,3184497
Gradient	0,06911616	Gradient	0
taux apprentissage	0,1	taux apprentissage	0,1
valeur ajustement	0,00691162	valeur ajustement	0
W51	0	W41	-0,39533485
Nouveau W51	-0,00691162	Nouveau W41	-0,39533485

W31		W21	
X2	0	X1	0
Y	0,3184497	Y	0,3184497
Erreur	-0,3184497	Erreur	-0,3184497
Gradient	0	Gradient	0
taux apprentissage	0,1	taux apprentissage	0,1
valeur ajustement	0	valeur ajustement	0
W31	-0,99977125	W21	0,44064899
Nouveau W31	-0,99977125	Nouveau W21	0,44064899

W11	
X1	0
Y	0,3184497
Erreur	-0,3184497
Gradient	0
taux apprentissage	0,1
valeur ajustement	0
W11	-0,16595599
Nouveau W11	-0,16595599

Une fois que les poids sont mis à jour, nous pouvons alors charger le cas d'apprentissage suivant.

Comme vous l'avez constaté, le fait d'ajouter une couche complémentaire n'a pas ajouté de complexité en termes de calcul. Cependant, il faut bien penser à activer chaque neurone à l'aide d'une fonction d'activation.

3.3 Place au code avec TensorFlow!

Nous allons à présent coder ce réseau de neurones avec TensorFlow.

Pour ce faire, nous vous invitons à utiliser le projet réalisé dans le chapitre précédent et y ajouter un nouveau fichier de script Python que nous appellerons `Perceptron_Multicouches.py`.

Voici le code à insérer :

```
import tensorflow as tf
import numpy as np

#-----
#      DONNEES D'APPRENTISSAGE
#-----

#On transforme les données en décimales
```

322 — Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

```
valeurs_entrees_X = [[0., 0.], [0., 1.], [1., 0.], [1., 1.]]
valeurs_a_predire_Y = [[0.], [1.], [1.], [0.]]

#-----
#   PARAMETRES DU RESEAU
#-----
#Variable TensorFlow correspondant aux valeurs neurones d'entrée
tf_neurones_entrees_X = tf.placeholder(tf.float32, [None, 2])

#Variable TensorFlow correspondant au neurone de sortie
(prédiction reele)
tf_valeurs_reelles_Y = tf.placeholder(tf.float32, [None, 1])

#Nombre de neurones dans la couche cachée
nbr_neurones_couche_cachee = 2

#POIDS
#Les premiers sont au nombres de 4 : 2 en entrée (X1 et X2) et 2
poids par entrée
poids = tf.Variable(tf.random_normal([2, 2]), tf.float32)

#les poids de la couche cachée sont au nombre de 2 : 2 en entrée
(H1 et H2) et 1 poids par entrée
poids_couche_cachee = tf.Variable(tf.random_normal([2, 1]),
tf.float32)

#Le premier biais comporte 2 poids
biais = tf.Variable(tf.zeros([2]))

#Le second biais comporte 1 poids
biais_couche_cachee = tf.Variable(tf.zeros([1]))

#Calcul de l'activation de la première couche
#calcul de la somme pondérée (tf.matmul) à l'aide des données X1,
X2, W11,W12,W31,W41 et du bais
#puis application de la fonction sigmoïde (tf.sigmoid)
activation = tf.sigmoid(tf.matmul(tf_neurones_entrees_X, poids) +
biais)

#Calcul de l'activation de la couche cachée
```

```
#calcul de la somme pondérée (tf.matmul) à l'aide des données H1,
H2, W12,W21 et du biais
#puis application de la fonction sigmoïde (tf.sigmoid)
activation_couche_cachee = tf.sigmoid(tf.matmul(activation,
poids_couche_cachee) + biais_couche_cachee)

#Fonction d'erreur de moyenne quadratique MSE
fonction_erreur = tf.reduce_sum(tf.pow(tf_valeurs_reelles_Y-
activation_couche_cachee,2))

#Descente de gradient avec un taux d'apprentissage fixé à 0.1
optimiseur =
tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize
(fonction_erreur)

#Nombre d'epochs
epochs = 100000

#Initialisation des variables
init = tf.global_variables_initializer()

#Démarrage d'une session d'apprentissage
session = tf.Session()
session.run(init)

#Pour la réalisation du graphique pour la MSE
Graphique_MSE=[]

#Pour chaque epoch
for i in range(epochs):

    #Réalisation de l'apprentissage avec mise à jour des poids
    session.run(optimiseur, feed_dict = {tf_neurones_entrees_X:
valeurs_entrees_X, tf_valeurs_reelles_Y:valeurs_a_predire_Y})

    #Calculer l'erreur
    MSE = session.run(fonction_erreur, feed_dict =
{tf_neurones_entrees_X: valeurs_entrees_X,
tf_valeurs_reelles_Y:valeurs_a_predire_Y})

    #Affichage des informations
    Graphique_MSE.append(MSE)
    print("EPOCH (" + str(i) + "/" + str(epochs) + ") - MSE: "+
```

```
str(MSE))
```

```
#Affichage graphique
import matplotlib.pyplot as plt
plt.plot(Graphique_MSE)
plt.ylabel('MSE')
plt.show()
```

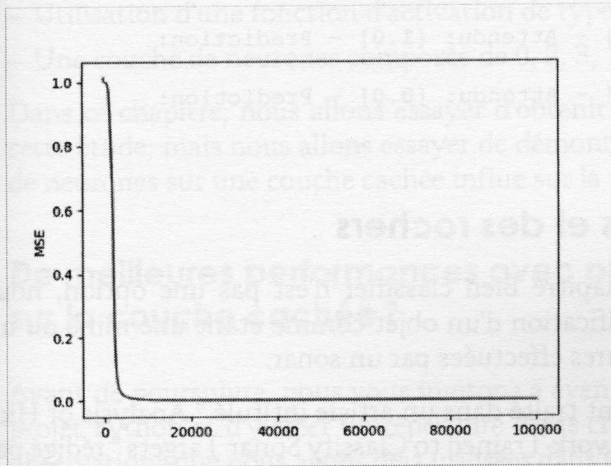
```
session.close()
```

Nous avons mis en exergue les parties correspondant à la mise en place de la couche cachée dans notre réseau de neurones. Ces parties concernent notamment :

- La création des poids de la couche cachée
- La création du biais relié à la couche cachée
- La prise en compte de la couche cachée dans la fonction d'erreur

Mis à part ces quelques ajouts et modifications, le code est identique à celui que nous avons utilisé pour le codage du neurone formel. De par ces quelques lignes, nous pouvons également apprécier la facilité de création d'un réseau de neurones à l'aide de TensorFlow.

La figure ci-après montre l'évolution de l'optimisation de l'erreur MSE diminuant au fil des apprentissages.



Graphique de la courbe d'erreur obtenue à l'aide de TensorFlow

Nous pouvons à présent réaliser quelques prédictions pour vérifier l'apprentissage de notre algorithme en ajoutant ces quelques lignes de code :

```
print("--- VERIFICATIONS ----")

for i in range(0,4):
    print("Observation:"+str(valeurs_entrees_X[i])+ " - Attendu: "+str(valeurs_a_predire_Y[i])+ " - Prediction: "+str(session.run(activation_couche_cachee, feed_dict={tf_neurones_entrees_X: [valeurs_entrees_X[i]]})))
```

Pour les quatre cas de notre jeu de données, nous réalisons une prédiction, en faisant appel à la dernière couche de notre réseau et en lui passant en paramètre le cas de test :

```
session.run(activation_couche_cachee, feed_dict={tf_neurones_entrees_X: [valeurs_entrees_X[i]]})
```

Nous obtenons alors les résultats suivants, montrant un bon apprentissage de notre réseau de neurones :

```
--- VERIFICATIONS ----
Observation:[0.0, 0.0] - Attendu: [0.0] - Prediction: [[0.02842014]]
Observation:[0.0, 1.0] - Attendu: [1.0] - Prediction:
```

```
[[0.9734256]]
```

```
Observation:[1.0, 0.0] - Attendu: [1.0] - Prediction:
```

```
[[0.9678324]]
```

```
Observation:[1.0, 1.0] - Attendu: [0.0] - Prediction:
```

```
[[0.02508838]]
```

4. Le retour des mines et des rochers

Souvenez-vous, dans le chapitre Bien classifier n'est pas une option, nous avons traité le cas de classification d'un objet comme étant une mine ou un rocher en fonction de mesures effectuées par un sonar.

Ce cas d'étude fut également traité dans un article intitulé " Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets " rédigé par Terry Sejnowski et R. Paul Gorman en 1988 dont le but était de **démontrer l'importance du rôle du nombre de neurones dans la couche cachée sur la précision de classification.**

Remarque

Leur étude est bien entendu téléchargeable sur le Web, mais nous avons fait le choix de vous en proposer une copie que vous pourrez retrouver dans le répertoire dédié au code de ce chapitre téléchargeable sur le site de l'éditeur.

Voici à présent les différents paramètres utilisés par ces deux scientifiques pour réaliser leur démonstration :

- 192 données d'apprentissage
- 16 données de tests
- 60 neurones d'entrée (correspondant à chacune des fréquences)
- 2 neurones de sortie (un pour la probabilité correspondant à une mine et l'autre pour la probabilité correspondant à un rocher)
- 300 epochs
- Un taux d'apprentissage de 2.0
- Des poids initialisés dans l'intervalle de valeurs compris entre - 0.3 et 0,3
- Une mise à jour des poids uniquement dans le cas où la valeur prédite diffère de plus ou moins 0.2 avec celle attendue

- Utilisation d'une fonction d'activation de type sigmoïde
- Une couche de neurones composée de 0, 2, 3, 12 ou 24 neurones

Dans ce chapitre, nous allons essayer d'obtenir les résultats obtenus lors de cette étude, mais nous allons essayer de démontrer également que le nombre de neurones sur une couche cachée influe sur la précision de la classification.

4.1 De meilleures performances avec plus de neurones sur la couche cachée?

Avant de poursuivre, nous vous invitons à éventuellement créer un nouveau projet Python et d'y créer un répertoire `datas` chargé de contenir les données observations que nous avons déjà utilisées dans le chapitre Bien classifier n'est pas une option.

4.1.1 Chargement des données d'apprentissage

Une fois cela réalisé, nous pouvons à présent charger les observations comme suit :

```
#-----
# CHARGEMENT DES OBSERVATIONS
#-----

import pandas as pnd
observations = pnd.read_csv("datas/sonar.all-data.csv")
```

On extrait et on transforme ensuite les données nécessaires à l'apprentissage et aux tests. Notons la création de classes qui correspondront aux valeurs attendues par les deux neurones de sortie. Si c'est une mine nous aurons pour valeur attendue 1 dans le neurone dédié à la prédiction de l'objet Mine et 0 dans celui dédié à l'objet Rocher.

```
#On ne prend que les données issues du sonar pour l'apprentissage
X = observations[observations.columns[0:60]].values

#On ne prend que les libellés
y = observations[observations.columns[60]]

#On encode : Les mines sont égales à 0 et les rochers égaux à 1
```

```

from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(y)
y = encoder.transform(y)

#On ajoute un encodage pour créer des classes :
# Si c'est une mine [1,0]
# Si c'est un rocher [0,1]
import numpy as np
n_labels = len(y)
n_unique_labels = len(np.unique(y))
one_hot_encode = np.zeros((n_labels,n_unique_labels))
one_hot_encode[np.arange(n_labels),y] = 1
Y=one_hot_encode

#Vérification en prenant les enregistrements 0 et 97
print("Classe Rocher:",int(Y[0][1]))
print("Classe Mine :",int(Y[97][1]))

```

4.1.2 Création des jeux d'apprentissage et de tests

Nous créons ensuite les jeux d'apprentissage et de tests conformes à ceux préconisés par Terry Sejnowski et R. Paul Godman, c'est-à-dire 192 observations d'apprentissage et 16 observations de tests (soit 0,07 %).

```

#On mélange les observations
from sklearn.utils import shuffle
X, Y = shuffle(X, Y, random_state=1)

#Création des jeux d'apprentissage et de tests
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split
(X, Y, test_size=0.07, random_state=42)

```

4.1.3 Paramétrage du réseau de neurones avec une couche cachée de 24 neurones

Vient ensuite la phase de paramétrage du réseau de neurones, avec la création de tuples poids et poids_biais pour le paramétrage des différents poids et biais des couches d'entrée, de la couche cachée et de la couche de sortie.

Voici le nombre de neurones utilisés pour chaque couche :

- Nombre de neurones présents dans la couche d'entrée : 60
- Nombre de neurones présents dans la couche cachée : 24
- Nombre de neurones en sortie : 2

À noter que les valeurs des poids sont générées dans l'intervalle $[-0.3, 0.3]$ comme préconise l'étude. Cette génération est réalisée grâce à la fonction `tf.random.uniform`. Cette fonction prend également en paramètre une liste définissant le nombre de neurones de la couche d'entrée et le nombre de neurones de la couche de sortie. Ainsi `[60,24]` signifie qu'il faut générer des poids pour 60 neurones connectés à 24 neurones, soit un total de 1440 poids (60×24)!

```
import tensorflow as tf

epochs = 300
nombre_neurones_entree = 60
nombre_neurones_sortie = 2
nombre_neurones_couche_cachee = 24
taux_apprentissage = 0.01

#Variable TensorFlow correspondant aux 60 valeurs des neurones
d'entrée
tf_neurones_entrees_X = tf.placeholder(tf.float32,[None, 60])

#Variable TensorFlow correspondant aux 2 neurones de sortie
tf_valeurs_reelles_Y = tf.placeholder(tf.float32,[None, 2])

poids = {
    # 60 neurones d'entrées vers 24 Neurones de la couche cachée
    'couche_entree_vers_cachee':
    tf.Variable(tf.random_uniform([60, 24], minval=-0.3,
maxval=0.3),
    tf.float32),

    # 24 neurones de la couche cachée vers 2 de la couche de
    sortie
    'couche_cachee_vers_sortie':
    tf.Variable(tf.random_uniform([24, 2], minval=-0.3, maxval=0.3),
    tf.float32),
```

```

}
poids_biais = {
    #1 biais de la couche d'entrée vers les 24 neurones de la
    couche cachée
    'poids_biais_couche_entree_vers_cachee':
    tf.Variable(tf.zeros([24]), tf.float32),

    #1 biais de la couche cachée vers les 2 neurones de la couche
    de sortie
    'poids_biais_couche_cachee_vers_sortie':
    tf.Variable(tf.zeros([2]), tf.float32),
}

```

Nous définissons ensuite une fonction chargée de créer le réseau de neurones. On y retrouve la définition des différentes fonctions d'activation de type sigmoïde de chaque couche en veillant à bien utiliser les poids adéquats en relation avec la couche concernée. Cette fonction nous retourne le modèle que l'on pourra utiliser par la suite dans la phase de test afin de vérifier le bon apprentissage du réseau de neurones.

```

def reseau_neurones_multicouches(observations_en_entrees, poids,
poids_biais):

    #Calcul de l'activation de la première couche
    premiere_activation =
    tf.sigmoid(tf.matmul(tf_neurones_entrees_X,
poids['couche_entree_vers_cachee']) +
poids_biais['poids_biais_couche_entree_vers_cachee'])

    #Calcul de l'activation de la seconde couche
    activation_couche_cachee =
    tf.sigmoid(tf.matmul(premiere_activation,
poids['couche_cachee_vers_sortie']) +
poids_biais['poids_biais_couche_cachee_vers_sortie'])

    return activation_couche_cachee

```

S'en suit la création du réseau de neurones proprement dite :

```

reseau = reseau_neurones_multicouches(tf_neurones_entrees_X,
poids, poids_biais)

```

Puis la spécification de la fonction d'erreur et de la fonction d'optimisation :

```
#Fonction d'erreur de moyenne quadratique MSE
fonction_erreur = tf.reduce_sum(tf.pow(tf.valeurs_reelles_Y-
reseau,2))

#Descente de gradient avec un taux d'apprentissage fixé à 0.01
optimiseur =
tf.train.GradientDescentOptimizer(learning_rate=taux_apprentissage
e)

.minimize(fonction_erreur)
```

4.1.4 Réalisation de l'apprentissage

```
#Initialisation des variables
init = tf.global_variables_initializer()

#Démarrage d'une session d'apprentissage
session = tf.Session()
session.run(init)

#Pour la réalisation du graphique pour la MSE
Graphique_MSE=[]

#Pour chaque epoch
for i in range(epochs):

    #Réalisation de l'apprentissage avec mise à jour des poids
    session.run(optimiseur, feed_dict = {tf_neurones_entrees_X:
train_x, tf.valeurs_reelles_Y:train_y})

    #Calculer l'erreur
    MSE = session.run(fonction_erreur, feed_dict =
{tf_neurones_entrees_X: train_x, tf.valeurs_reelles_Y:train_y})

    #Affichage des informations
    Graphique_MSE.append(MSE)
    print("EPOCH (" + str(i) + "/" + str(epochs) + ") - MSE: "+
str(MSE))

#Affichage graphique de la MSE
import matplotlib.pyplot as plt
```

```
plt.plot(Graphique_MSE)
plt.ylabel('MSE')
plt.show()
```

4.1.5 Calcul de la précision de l'apprentissage

Avant d'entrer dans le détail du code, voici comment sont évaluées les différentes classifications réalisées par le réseau de neurones :

Une fois l'apprentissage réalisé, les probabilités de chacune des classifications effectuées sont stockées dans le modèle :

Classification 1 [0.56, 0.89]

Classification 2 [0.90, 0.34]

À l'aide de la fonction `tf.argmax`, nous allons récupérer pour chaque classification l'index de la probabilité ayant la plus grande valeur.

Classification 1 [0.56, **0.89**] l'index est égal à 1

Classification 2 [**0.90**, 0.34] l'index est égal à 0

Pour rappel, les mines sont définies selon les valeurs de probabilités [1,0] et les rochers, [0,1] selon les valeurs, ce qui se traduit en termes d'index avec les valeurs suivantes :

Mine [**1**,0] l'index est égal à 0

Rocher [0,**1**] l'index est égal à 1

Dans le cadre d'une classification, nous nous attendions à ce que le réseau classe les valeurs des fréquences fournies en entrée comme étant une mine :

- Index attendu 0
- Probabilités de classifications réalisées : [0.56, **0.89**]
- Index de la probabilité la plus élevée est 1

La valeur 1 (index de la classification réalisée) étant différente de 0 (index de la classification attendue), il s'agit donc d'une mauvaise classification.

Imaginons à présent une nouvelle classification :

- Index attendu 0 (cas d'une mine)
- Probabilités de classifications réalisées : [0.90, 0.34]
- Index de la probabilité la plus élevée est 0

La valeur 0 (index de la classification réalisée) étant égale à 0 (index de la classification attendue), il s'agit donc d'une bonne classification.

Pour calculer la précision de l'algorithme sur un jeu de données précis (apprentissage, tests ou l'ensemble des données), nous allons pour chaque classification contenue dans ce jeu de données vérifier si celle-ci correspond à celles attendues pour pouvoir en déduire le nombre d'erreurs et en calculer la précision globale.

Voici à présent ce principe transformé à l'aide de lignes de code :

```
#Récupération des index des classifications réalisées

classifications = tf.argmax(reseau, 1)

#On compare les index issus des classifications avec ceux
attendus pour connaître le nombre de bonnes classifications

formule_calcul_bonnes_classifications = tf.equal(classifications,
tf.argmax(tf_valeurs_reelles_Y,1))

#On calcule ensuite La précision en faisant la moyenne (tf.mean)
# des bonnes classifications (après les avoir converties en
décimales tf.cast, tf.float32)
formule_precision =
tf.reduce_mean(tf.cast(formule_calcul_bonnes_classifications,
tf.float32))
```

```
#-----
# PRECISION SUR LES DONNEES DE TESTS
#-----
```

```
nb_classifications = 0;
```

```

nb_bonnes_classifications = 0

#On parcourt l'ensemble des données de tests (text_x)
for i in range(0, test_x.shape[0]):

    #On récupère les informations des données d'apprentissage
    # test_x que l'on reformate en une ligne composée de 60
    colonnes pour les données issues du sonar et en une ligne et deux
    colonnes pour la classification attendue
    donneesSonar = test_x[i].reshape(1,60)
    classificationAttendue = test_y[i].reshape(1,2)

    # On réalise la classification
    prediction_run = session.run(classifications,
    feed_dict={tf_neurones_entrees_X:donneesSonar})

    #On calcule la précision de la classification à l'aide de la
    formule établie auparavant
    accuracy_run = session.run(formule_precision,
    feed_dict={tf_neurones_entrees_X:donneesSonar,
    tf_valeurs_reelles_Y:classificationAttendue})

    #On affiche pour observation la classe originale et la
    classification réalisée
    print(i, "Classe attendue: ",
    int(session.run(tf_valeurs_reelles_Y
    [i][1], feed_dict={tf_valeurs_
    reelles_Y:test_y})), " Classification: ", prediction_run[0] )

    nb_classifications = nb_classifications+1
    if(accuracy_run*100 ==100):
        nb_bonnes_classifications = nb_bonnes_classifications+1

print("-----")
print("Précision sur les donnees de tests =
"+str((nb_bonnes_classifications/nb_classifications)*100)+"%")

```

Si nous exécutons à présent le script, nous pouvons voir que la précision de l'algorithme est de 86 % sur le jeu de test !

```

Classe attendue: 1 Classification: 1
Classe attendue: 0 Classification: 0
Classe attendue: 1 Classification: 1
Classe attendue: 1 Classification: 0 - Erreur
Classe attendue: 0 Classification: 0
Classe attendue: 1 Classification: 1
Classe attendue: 0 Classification: 0
Classe attendue: 1 Classification: 0 - Erreur
Classe attendue: 1 Classification: 0 - Erreur
Classe attendue: 0 Classification: 0
Classe attendue: 0 Classification: 0
Classe attendue: 0 Classification: 0
Classe attendue: 0 Classification: 0
Classe attendue: 0 Classification: 0
Classe attendue: 0 Classification: 0

```

Précision sur les donnees de tests = 80.0%

Dans le code dédié à ce chapitre que vous pouvez récupérer sur le site de l'éditeur, nous avons également codé les calculs de précision sur les données d'apprentissage et l'ensemble des données afin de vérifier que nous ne sommes pas dans un cas de surapprentissage.

```

Précision sur les donnees de tests = 80.0%
Précision sur les donnees d'apprentissage = 79.6875%
Précision sur l'ensemble des données = 79.71014492753623%

```

Comme il n'y a pas d'écart important entre la précision obtenue lors de l'apprentissage et la validation, nous ne sommes pas dans un cas de surapprentissage.

4.1.6 De meilleurs résultats avec une couche cachée composée de 24 neurones?

Voyons si nous obtenons de meilleurs résultats avec 12 neurones au lieu de 24. Pour cela, modifions quelque peu notre code :

```

poids = {
    # 60 neurones d'entrée vers les 12 neurones de la couche
    # cachée
    'couche_entree_vers_cachee':
    tf.Variable(tf.random_uniform([60, 12], minval=-0.3,
    maxval=0.3),

```

```

tf.float32),

    # 12 neurones de la couche cachée vers les 2 neurones de la
    couche de sortie
    'couche_cachee_vers_sortie':
    tf.Variable(tf.random_uniform([12, 2], minval=-0.3, maxval=0.3),
    tf.float32),

}

poids_biais = {
    #1 biais de la couche d'entrée vers les 12 neurones de la
    couche cachée
    'poids_biais_couche_entree_vers_cachee':
    tf.Variable(tf.zeros([12]), tf.float32),

    #1 biais de la couche cachée vers les 2 neurones de la couche
    de sortie
    'poids_biais_couche_cachee_vers_sortie':
    tf.Variable(tf.zeros([2]), tf.float32),
}

```

Et cette fois-ci, après avoir exécuté notre code, nous obtenons une précision de classification de 81 % sur l'ensemble des données, ce qui est supérieur à celui obtenu précédemment !

```

Précision sur les donnees de tests = 93.33333333333333%
Précision sur les donnees d'apprentissage = 80.72916666666666%
Précision sur l'ensemble des données = 81.64251207729468%

```

■ Remarque

Pour chacun des deux cas (utilisation de 12 ou 24 neurones), nous vous invitons à réaliser plusieurs apprentissages qui, vous le constaterez, vous offriront des valeurs différentes. Cela est dû aux différents facteurs d'initialisation et d'apprentissage choisis de façon aléatoire (poids, données d'apprentissage...).

Au vu de ces résultats, nous pouvons affirmer que nous sommes en concordance avec la conclusion de l'étude à savoir que le nombre de neurones utilisés sur une ou plusieurs couches cachées influence les résultats.

The results of the network classification experiments, as well as the analysis of variance, **demonstrate that the hidden layer contributed significantly** to the performance of the network classifier

4.1.7 Pouvons-nous obtenir de meilleurs résultats ?

Nous allons à présent voir si nous pouvons obtenir de meilleurs résultats en modifiant la configuration de notre réseau de neurones.

La question souvent posée est celle-ci : de combien de couches de neurones avons-nous besoin et combien celles-ci doivent-elles comporter de neurones ?

À cela, la réponse la plus communément apportée et que dans bien des cas, une seule couche cachée suffit à résoudre la majorité des problèmes de classification des données non linéaires. Pour le nombre de neurones contenus dans cette couche, celui-ci correspond à la moyenne du nombre de neurones en entrée et le nombre de neurones en sortie. Cependant, nous pouvons affirmer qu'il n'existe pas de méthode de calcul permettant d'identifier clairement le nombre de couches cachées nécessaires et leur nombre de neurones. Tout dépend du problème de classification à résoudre.

Pour notre cas, nous allons tenter plusieurs améliorations.

La première va consister à modifier le nombre d'epochs en le passant de 300 à 600.

```
■ epochs = 600
```

Puis à utiliser 26 neurones dans la couche cachée et à modifier la fonction de génération des poids ne prenant cette fois pas d'intervalle de valeur, mais se générant sous une forme de distribution de la loi normale (la fameuse courbe en forme de cloche) :

```
poids =
    #60 neurones d'entrée vers 24 neurones de la couche cachée
    'couche_entree_vers_cachee':
    tf.Variable(tf.random_normal([60, 26]), tf.float32),

    #26 neurones de la couche cachée vers 2 de la couche de
    sortie
    'couche_cachee_vers_sortie' :
    tf.Variable(tf.random_normal([26, 2]), tf.float32),
```

```

    }

    poids_biais = {
        #1 biais de la couche d'entrée vers les 24 neurones de la
        couche cachée
        'poids_biais_couche_entree_vers_cachee':
        tf.Variable(tf.zeros([26]), tf.float32),

        #1 biais de la couche cachée vers les 2 neurones de la couche
        de sortie
        'poids_biais_couche_cachee_vers_sortie':
        tf.Variable(tf.zeros([2]), tf.float32),
    }

```

Attention, augmenter le nombre d'epochs entraîne également l'augmentation du risque de surapprentissage!

Ce qui nous permet d'obtenir une précision de 94 % sur l'ensemble des données. Précision supérieure à celle que nous avons obtenue jusqu'à présent.

```

Précision sur les donnees de tests = 80.95238095238095%
Précision sur les donnees d'apprentissage = 97.57575757575758%
Précision sur l'ensemble des données = 94.20289855072464%

```

Il est difficile de savoir exactement le nombre de neurones devant figurer sur chacune des couches cachées. Cependant, une "règle" semble faire son apparition indiquant que le nombre de neurones de la couche cachée doit être égal à la moyenne entre le nombre de neurones d'entrée et le nombre de neurones en sortie. Soit dans notre cas $31 ((60+2)/2)$.

Si nous appliquons cette règle, nous obtenons un résultat de précision sur l'ensemble des données légèrement inférieur. Nous attirons votre attention sur le fait que la précision des données d'apprentissage est supérieure d'environ 12 % à celle obtenue sur les données de tests. Le modèle rencontre peut-être des difficultés à se généraliser (sur apprentissage).

```

Précision sur les données de tests = 83.33333333333334%
Précision sur les données d'apprentissage = 95.15151515151516%
Précision sur l'ensemble des données = 92.7536231884058%

```

5. Conclusion

Dans ce chapitre, nous avons vu le principe de fonctionnement des réseaux de neurones multicouches qui nous permettent de réaliser des classifications sur des données non linéairement séparables.

Nous avons également pu valider l'affirmation de Terry Sejnowski et R. Paul Gorman selon laquelle le nombre de neurones présents sur la couche cachée influe sur le résultat de la classification. Nous pouvons également constater que nous sommes parvenus à un meilleur résultat de classification que celui obtenu dans le chapitre Bien classifier n'est pas une option, où à l'aide de l'algorithme des Machines à vecteurs de supports, le score de précision ne dépassait pas les 89 %. Mais il convient toutefois de bien veiller à ce que le modèle puisse se généraliser en évaluant l'écart entre la précision obtenue sur les données d'apprentissage et celle obtenue sur les données de tests.

Dans le chapitre suivant, nous allons à présent aborder le domaine de la classification d'image à l'aide d'un autre type de réseau de neurones, appelé réseau de neurones convolutifs.

Chapitre 12

La classification d'images

1. Ce que nous allons découvrir et les prérequis

Dans le chapitre précédent, nous avons découvert l'utilisation d'un réseau de neurones pour réaliser des prédictions et des classifications. Nous allons à présent voir comment il nous est possible d'apprendre à notre machine à classifier des images à l'aide de réseaux de neurones un peu particuliers que l'on nomme "Réseaux de neurones convolutifs".

■ Remarque

Prérequis nécessaires pour bien aborder ce chapitre : avoir lu les chapitres Les fondamentaux du langage Python, Des statistiques pour comprendre les données, Principaux algorithmes du Machine Learning, Un neurone pour prédire et Utilisation de plusieurs couches de neurones.

2. Différence entre détection et classification d'images

Il est important de faire la distinction entre une détection et une classification d'images. Savoir détecter un cercle, un triangle, une forme, voire un visage, ne relève pas forcément du domaine de l'intelligence artificielle (bien que certains algorithmes en soient capables). En effet, ces tâches sont confiées à des algorithmes de traitement d'images que nous aurons l'occasion de manipuler dans le chapitre suivant.

Cependant, être en mesure de reconnaître une voiture d'un vélo, de réaliser de la reconnaissance faciale relève de l'intelligence artificielle où des algorithmes d'apprentissage ont été entraînés pour reconnaître (classifier) les éléments. Néanmoins, la détection de formes dans une image peut aider l'algorithme de classification, en ce sens où l'on peut facilement extraire un visage d'une photo ou d'une vidéo et le proposer à un algorithme de classification capable alors de réaliser de la reconnaissance faciale.

3. Des réseaux de neurones convolutifs pour classifier des images

Pour classifier des images, nous allons utiliser un réseau de neurones un peu particulier : le réseau de neurones convolutifs ou encore appelé CNN (Convolutional Neural Network ou encore ConvNet).

Cet algorithme d'apprentissage peut être assimilé à un millefeuille composé de plusieurs couches. Les dernières couches (hautes du millefeuille) sont constituées de réseaux de neurones tels que nous les avons découverts dans les chapitres précédents, dont les données en entrée sont issues des couches précédentes appelées couches de convolutions dont nous allons à présent découvrir le principe.

3.1 De nombreuses données d'apprentissage nécessaires

Pour que le réseau de neurones puisse réaliser une bonne classification d'images, il faut bien entendu lui donner une base d'apprentissage. Mais lorsqu'il s'agit d'images, leur nombre peut vite devenir impressionnant.

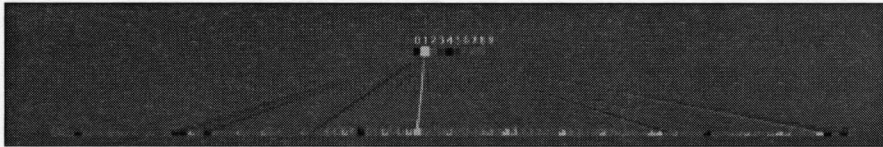
En effet, en tant qu'humain, notre cerveau est capable de réaliser la classification entre un chat et un chien, qu'il soit noir, blanc, de face, de profil, de dos, en plein jour, dans la pénombre... c'est-à-dire dans différents environnements et positions possibles ! Ce qui vous laisse imaginer le nombre d'images nécessaires.

Si vous souhaitez vous lancer dans un projet personnel de classification d'images, cela vous permet également d'apprécier le travail de photographie à réaliser. Heureusement pour nous, des jeux d'observations sont disponibles sur Internet pour nous entraîner.

Pour pouvoir réaliser son apprentissage, la machine va devoir manipuler l'image. Nous entrons donc dans le domaine du traitement d'image qui comme vous le savez sans doute est gourmand en ressources matérielles. Par conséquent, réaliser la classification d'images nécessite un temps important pour la création du jeu d'apprentissage et un investissement dans le matériel pour réaliser l'apprentissage.

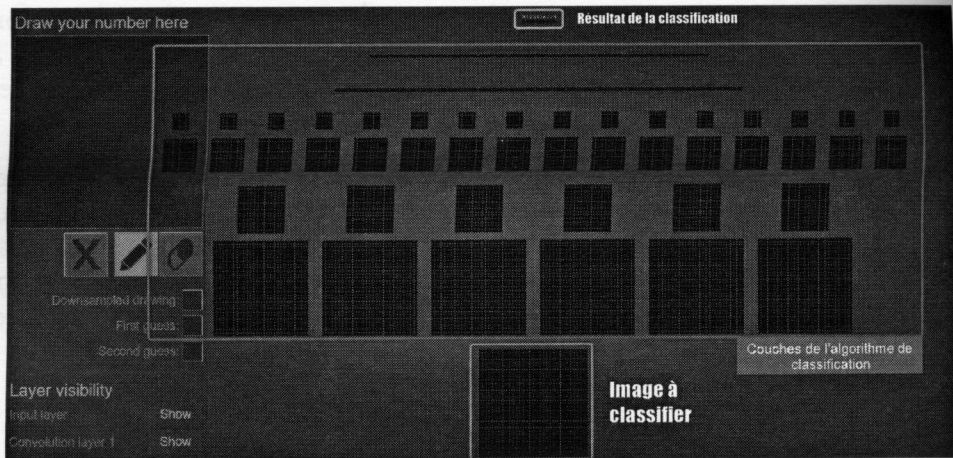
3.2 Un outil pour illustrer nos propos

Les réseaux de neurones convolutifs sont composés de plusieurs couches de traitements que l'on peut facilement schématiser. Grâce à Adam W Harley, nous pouvons aller plus loin en visualiser l'intérieur du réseau de neurones convolutifs à l'aide d'un outil interactif disponible à cette adresse : (<http://scs.ryerson.ca/~aharley/vis/conv/flat.html>).



Un outil de visualisation pour comprendre les réseaux de neurones convolutifs

Le principe de cet outil est assez simple. En haut de la partie de gauche, nous allons dessiner un chiffre de 0 à 9 et charge à la machine de classier ce dessin en tant que chiffre. Dans la partie de droite, nous pourrions visualiser toutes les étapes successives réalisées par la machine pour y parvenir. Chaque étape devant être considérée comme étant une couche de l'algorithme de classification.



Description de l'interface de l'outil permettant de comprendre le fonctionnement des réseaux de neurones convolutifs

■ Remarque

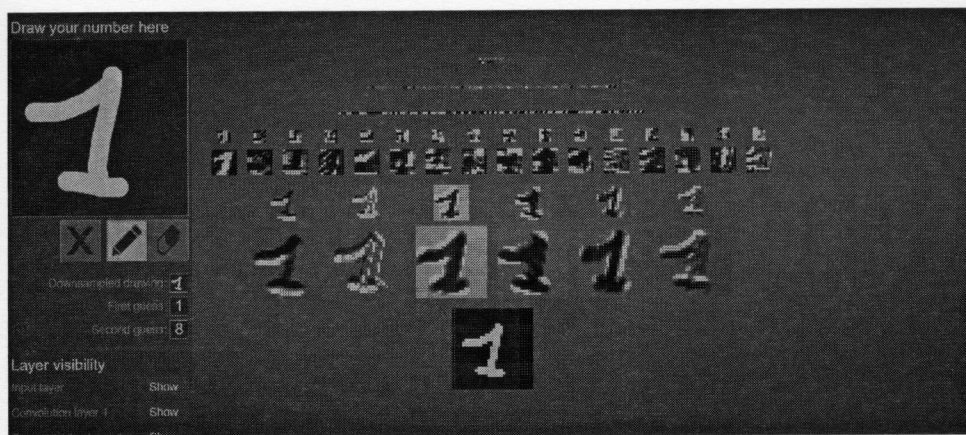
Sur certaines résolutions d'écran, la partie de gauche n'est pas entièrement visible. Dans ce cas, il suffit de dézoomer grâce à l'option de zoom des navigateurs web.

3.3 L'image d'entrée

Les réseaux de neurones à convolution servent à classer les images. Tout part donc d'une image source. Pour la créer, dessinons un chiffre entre 0 et 9 en haut de la partie de gauche et observons ce qui se passe dans la partie de droite.

La première chose que l'on observe est que notre image d'entrée (input layer) se trouve tout en bas de l'écran et se voit divisée en petits carrés. Ces carrés étant bien entendu les pixels de l'image.

Si l'on remonte petit à petit, on voit également que d'autres images sont apparues jusqu'à obtenir la prédiction du chiffre dessiné (1 pour notre cas).

*Classification du chiffre 1*

3.4 Les caractéristiques

Voyons à présent la notion de caractéristique. Imaginons que nous cherchions à réaliser une classification d'une image comportant un lapin. L'objectif étant que lorsque nous présentons une image de lapin à notre machine, celle-ci nous indique qu'il s'agit d'un lapin ou non.

L'une des solutions nous venant intuitivement à l'esprit pour résoudre ce problème est de comparer pixel par pixel deux images, une image d'apprentissage et une image à classifier. Si les deux images correspondent, on peut alors affirmer qu'il s'agit d'un lapin.

Sauf que cela comporte un inconvénient, car il faut que l'image à classifier soit strictement identique à l'image de référence, ce qui est tout à fait impossible dans les cas réels d'application.

La solution pour résoudre ce souci va consister à capturer des caractéristiques dans l'image source (des morceaux d'images) et à les rechercher dans l'image à classifier. C'est ce que l'on appelle le principe de convolution que nous allons à présent découvrir.

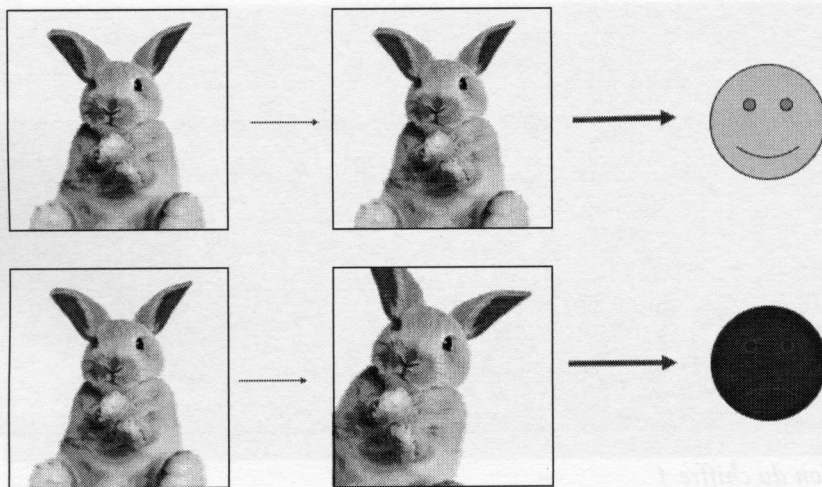
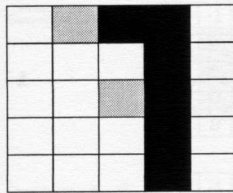


Illustration de la nécessité d'utiliser des caractéristiques pour la classification des images

3.5 La convolution

Comme nous venons de le voir, nous allons utiliser des caractéristiques d'une image pour essayer de les retrouver dans une autre image à classer. Si l'image à classer comporte un grand nombre de caractéristiques communes avec l'image source de classification, il y a de fortes probabilités que les images soient fortement similaires. La phase d'apprentissage va consister à extraire ces caractéristiques pour ensuite alimenter un réseau de neurones afin qu'il apprenne à faire le lien entre les caractéristiques de l'image source couplées à une labellisation (nom de l'objet à reconnaître). À l'issue de son apprentissage, le réseau de neurones sera en mesure de dire "J'ai appris que si je suis majoritairement en présence de ces caractéristiques alors, l'image que l'on me présente est un lapin".

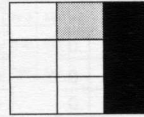
Pour éclaircir ce concept, considérons une image ayant une dimension de 5 pixels sur 5 pixels et une caractéristique de 3 pixels sur 3 pixels. Cette caractéristique est également appelée filtre ou kernel. Les pixels noirs de l'image porteront la valeur 1, les blancs 0 et les gris -1.



Image

0	-1	1	1	0
0	0	0	1	0
0	0	-1	1	0
0	0	0	1	0
0	0	0	1	0

Images (pixels)



Caractéristique

0	-1	1
0	0	1
0	0	1

Caractéristique (pixels)

Extraction d'une caractéristique d'une image

La convolution va consister à appliquer le filtre sur l'ensemble de l'image afin d'en déduire une nouvelle image.

Les quatre figures suivantes expliquent le procédé :

- On place le filtre en haut et à gauche de l'image.
- Pour chaque valeur de pixel de l'image se situant dans le filtre, on la multiplie par la valeur du pixel du filtre.
- On réalise la somme de ces produits.
- On se décale d'un pixel et on recommence le produit et la somme. Ce déplacement porte le nom de Stride et dans notre cas, a la valeur 1.

Ce processus étant à réaliser jusqu'à ce que l'ensemble de l'image soit filtré, donnant ainsi naissance à une nouvelle image plus petite que l'image d'origine et ayant de nouvelles valeurs comme le montre le tableau ci-dessous. Ces valeurs ayant pour but de faire ressortir certaines caractéristiques de l'image.

1	2	-1
-1	3	-1
-1	4	-1

Remarque

Nous verrons lors de la mise en pratique que dans la phase de convolution nous allons spécifier une méthode d'activation. Dans la plupart des cas, nous choisirons la méthode ReLU, permettant de supprimer les valeurs négatives de l'image de convolution en les remplaçant par la valeur 0. Sans cette activation, les résultats d'apprentissage ne seraient pas ceux attendus.

348 Intelligence Artificielle Vulgarisée

Le Machine Learning et le Deep Learning par la pratique

	Image	Kernel	Produit	Somme																																																					
Convolution 1	<table><tr><td>0</td><td>-1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>-1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	-1	1	1	0	0	0	0	1	0	0	0	-1	1	0	0	0	0	1	0	0	0	0	1	0	<table><tr><td>0</td><td>-1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr></table>	0	-1	1	0	0	1	0	0	1	<table><tr><td>0x0</td><td>-1x1</td><td>1x1</td></tr><tr><td>0x0</td><td>0x0</td><td>0x1</td></tr><tr><td>0x0</td><td>0x0</td><td>-1x1</td></tr></table>	0x0	-1x1	1x1	0x0	0x0	0x1	0x0	0x0	-1x1	<table><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>-1</td></tr></table>	0	1	1	0	0	0	0	0	-1	1
0	-1	1	1	0																																																					
0	0	0	1	0																																																					
0	0	-1	1	0																																																					
0	0	0	1	0																																																					
0	0	0	1	0																																																					
0	-1	1																																																							
0	0	1																																																							
0	0	1																																																							
0x0	-1x1	1x1																																																							
0x0	0x0	0x1																																																							
0x0	0x0	-1x1																																																							
0	1	1																																																							
0	0	0																																																							
0	0	-1																																																							
Convolution 2	<table><tr><td>0</td><td>-1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>-1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	-1	1	1	0	0	0	0	1	0	0	0	-1	1	0	0	0	0	1	0	0	0	0	1	0	<table><tr><td>0</td><td>-1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr></table>	0	-1	1	0	0	1	0	0	1	<table><tr><td>-1x0</td><td>1x-1</td><td>1x1</td></tr><tr><td>0x0</td><td>0x0</td><td>1x1</td></tr><tr><td>0x0</td><td>-1x0</td><td>1x1</td></tr></table>	-1x0	1x-1	1x1	0x0	0x0	1x1	0x0	-1x0	1x1	<table><tr><td>0</td><td>-1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr></table>	0	-1	1	0	0	1	0	0	1	2
0	-1	1	1	0																																																					
0	0	0	1	0																																																					
0	0	-1	1	0																																																					
0	0	0	1	0																																																					
0	0	0	1	0																																																					
0	-1	1																																																							
0	0	1																																																							
0	0	1																																																							
-1x0	1x-1	1x1																																																							
0x0	0x0	1x1																																																							
0x0	-1x0	1x1																																																							
0	-1	1																																																							
0	0	1																																																							
0	0	1																																																							

Étapes de convolutions

Convolution 3

0	-1	1	1	0
0	0	0	1	0
0	0	-1	1	0
0	0	0	1	0
0	0	0	1	0

0	-1	1
0	0	1
0	0	1

1x0	1x-1	0x1
0x0	1x0	0x1
-1x0	1x0	0x1

0	-1	0
0	0	0
0	0	0

-1

Convolution 4

0	-1	1	1	0
0	0	0	1	0
0	0	-1	1	0
0	0	0	1	0
0	0	0	1	0

0	-1	1
0	0	1
0	0	1

0x0	0x-1	0x1
0x0	0x0	-1x1
0x0	0x0	0x1

0	0	0
0	0	-1
0	0	0

-1

Étapes de convolutions

Convolution 5	0	-1	1	1	0
	0	0	0	1	0
	0	0	-1	1	0
	0	0	0	1	0
	0	0	0	1	0
	0	0	0	1	0

0	-1	1
0	0	1
0	0	1

0x0	0x-1	1x1
0x0	-1x0	1x1
0x0	0x0	1x1

0	0	1
0	0	1
0	0	1

3

Convolution 6

0	-1	1	1	0
0	0	0	1	0
0	0	-1	1	0
0	0	0	1	0
0	0	0	1	0

0	-1	1
0	0	1
0	0	1

0x0	1x-1	0x1
-1x0	1x0	0x1
0x0	1x0	0x1

0	-1	0
0	0	0
0	0	0

-1

Convolution 7

0	-1	1	1	0
0	0	0	1	0
0	0	-1	1	0
0	0	0	1	0
0	0	0	1	0

0	-1	1
0	0	1
0	0	1

0x0	0x-1	-1x1
0x0	0x0	0x1
0x0	0x0	0x1

0	0	-1
0	0	0
0	0	0

-1

Étapes de convolutions

Convolution 8

0	-1	1	1	0
0	0	0	1	0
0	0	-1	1	0
0	0	0	1	0
0	0	0	1	0

0	-1	1
0	0	1
0	0	1

0x0	-1x-1	1x1
0x0	0x0	1x1
0x0	0x0	1x1

0	1	1
0	0	1
0	0	1

4

Convolution 9

0	-1	1	1	0
0	0	0	1	0
0	0	-1	1	0
0	0	0	1	0
0	0	0	1	0

0	-1	1
0	0	1
0	0	1

-1x0	1x-1	0x1
0x0	1x0	0x1
0x0	1x0	0x1

0	-1	0
0	0	0
0	0	0

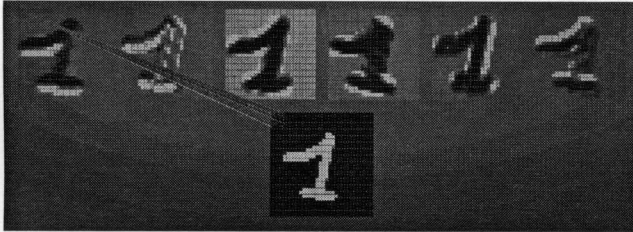
-1

Étapes de convolutions

On note qu'à la convolution n°8, la somme est plus importante que celle des autres convolutions. Cela est dû au fait que le filtre appliqué correspond parfaitement à la partie de l'image qui est en train d'être analysée, il y a donc un point de concordance.

Revenons à présent à notre outil de reconnaissance de chiffres. On peut observer qu'au-dessus de l'image d'entrée, il y a 6 autres images. Ces images sont issues de la convolution de 6 filtres. À droite, on trouve l'image issue du filtre n°1 et à gauche celle du filtre n°6. Les filtres appliqués sont d'une taille de 5 pixels * 5 pixels et sont bien entendu différents.

Si vous passez la souris sur un pixel de l'image issue de la convolution, vous pourrez voir à partir de quelle partie de l'image source celui-ci a été conçu.



Principe de convolutions

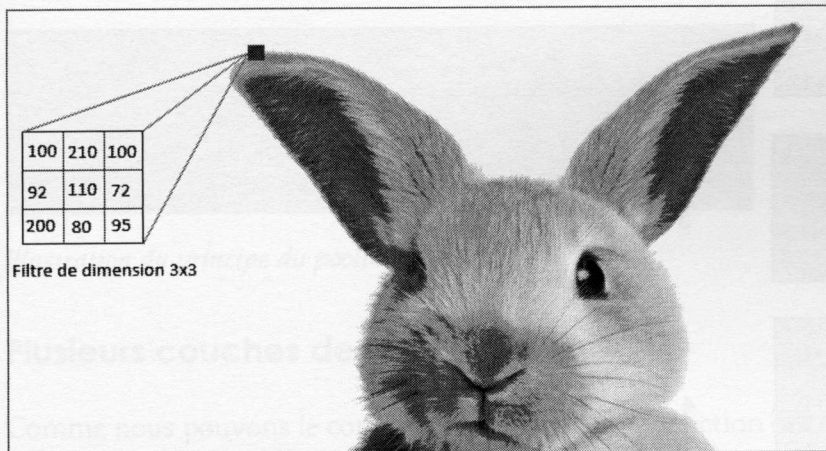
Comme nous venons de le voir, le nombre de filtres utilisés correspond aux caractéristiques à apprendre. Nous aurions donc tendance à dire que plus de filtres différents sont utilisés dans la phase d'apprentissage, plus cette phase d'apprentissage sera efficace ; cependant, cela peut entraîner un surapprentissage de l'algorithme et avoir l'effet inverse de celui escompté.

Malheureusement, il n'existe pas de méthode miracle permettant de déterminer la quantité de filtres à utiliser ainsi que leur dimension, il faut bien souvent définir ces valeurs de façon empirique.

Si on applique ce traitement de convolution sur notre image de lapin (cf. ci-dessous), on peut se rendre compte que le nombre de calculs peut vite devenir important si la taille de l'image est grande et si l'on multiplie le nombre de filtres. Ce calcul demandant alors beaucoup de ressources au niveau de processeur et de la carte graphique. Nous verrons que dans notre cas pratique, nous allons configurer des couches de convolutions à 128 filtres ayant une taille de 3 pixels par 3 pixels ! C'est pourquoi dans la plupart des cas, de petites images sont utilisées (28x28 pixels ou 32x32 pixels). Notons que certains fabricants de cartes graphiques (Nvidia) se sont spécialisés dans la réalisation de matériel dédié au Deep Learning avec des produits dépassant les 2000 € pour les cartes les plus performantes.

Remarque

Petit aparté technique : en termes de traitement d'image, il est préférable de travailler avec des images en niveau de gris. En effet, une image en couleur contient 3 couches de couleurs (rouge, vert et bleu), alors qu'une image en niveau de gris n'en contient qu'une. Cela améliore les temps de traitement, car au lieu de traiter 3 couches, nous ne devons en traiter qu'une seule.



Convolution appliquée sur une grande image

3.6 Pooling

Maintenant que nous disposons d'une image filtrée, nous allons lui appliquer un nouveau traitement appelé **Pooling** permettant d'extraire les caractéristiques importantes issues de la convolution.

Ce traitement consiste à déplacer une fenêtre dans l'image issue de la convolution. Dans le cadre d'un **MaxPooling**, on recherche la valeur maximale contenue à l'intérieur de cette fenêtre, dans le cadre d'un **Average Pooling**, on calcule la moyenne des valeurs contenues dans la fenêtre et enfin dans le cadre d'un **Pooling Stochastique**, on retient une valeur en fonction d'estimations probabilistiques.

Reprenons l'image issue de notre convolution et appliquons-lui une fenêtre de pooling de 2 pixels sur 2 pixels avec un stride (déplacement) de 1 tout comme nous l'avons fait dans l'étape de convolution :

Calcul du
Maximum

1	2	-1
-1	3	-1
-1	4	-1

3

1	2	-1
-1	3	-1
-1	4	-1

3

1	2	-1
-1	3	-1
-1	4	-1

4

1	2	-1
-1	3	-1
-1	4	-1

4

Ce qui nous donne une nouvelle image contenant uniquement les valeurs remarquables suivantes :

3	3
4	4

Comme nous pouvons le constater, à l'issue d'une convolution et d'un pooling, notre image source faisant 5x5 pixels se trouve réduite en une image de 2x2 tout en conservant les caractéristiques de l'image issue de la convolution.

Cette réduction de taille est notamment bénéfique aux différents calculs à venir.

Si l'on observe la couche de pooling dans l'outil de classification des chiffres dessinés manuellement, on constate que chaque couche de convolution donne lieu à un pooling. Le pooling utilisé pour chaque filtre est un MaxPooling d'une dimension de 2×2 avec un stride (déplacement) de deux.

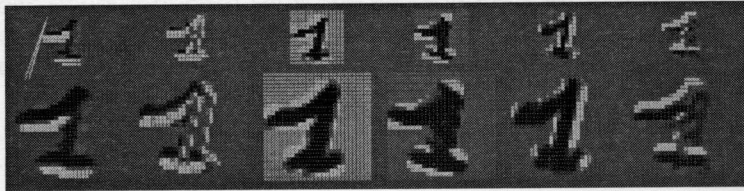


Illustration du principe du pooling

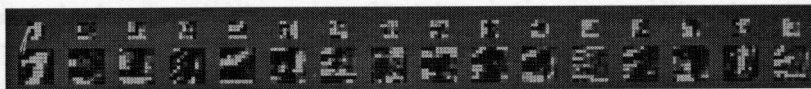
3.7 Plusieurs couches de convolutions

Comme nous pouvons le constater dans l'outil de prédiction des chiffres dessinés manuellement, la couche de pooling vue précédemment est elle-même connectée à une couche de convolution, qui est aussi connectée à une couche de pooling.

Ce qui nous donne les couches de convolution suivantes :

- Une première couche de convolution composée de 6 filtres de dimension 5×5 avec un stride de 1
- Une couche de pooling de taille 2×2
- Une seconde couche de convolution composée de 16 filtres de dimension 5×5 avec un stride de 1
- Une couche de pooling de taille 2×2

L'objectif étant de réduire l'image tout en conservant les informations importantes.



Les différentes couches de convolutions et de pooling

Remarque

Les paramètres des différentes couches utilisées dans l'outil de visualisation sont disponibles dans le document PDF "An Interactive Node-Link Visualization of Convolutional Neural Networks" rédigé par Adam W. Harley et téléchargeable à cette adresse :

http://scs.ryerson.ca/~aharley/vis/harley_vis_jsvc15.pdf

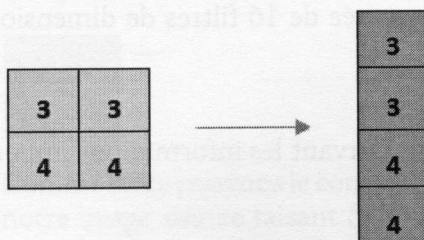
3.8 Mise à plat (Flatten)

À l'issue des différentes opérations de convolutions et de pooling, nous allons préparer les données pour qu'elles soient intégrables dans un réseau de neurones.

Cette étape de préparation consiste à "mettre à plat" les différentes images issues des opérations de pooling sous forme d'une seule colonne (ou ligne dans l'outil). Nous créons alors un vecteur :



"Mise à plat" des différentes images (Flatten)



Étape de mise à plat (Flatten)

3.9 L'apprentissage

Enfin, la dernière phase du réseau de neurones à convolution est l'apprentissage proprement dit à l'aide de couches de neurones totalement connectés ayant pour entrée la couche de mise à plat réalisée précédemment et pour sortie une prédiction.

Le fonctionnement de l'apprentissage est identique à celui que nous avons découvert dans le chapitre précédent : définition des poids, prédiction, rétropropagation...

En regardant les dernières couches de l'outil de prédiction, on peut voir le réseau de neurones permettant de prédire une probabilité par catégorie de chiffre (de 0 à 9).

■ Remarque

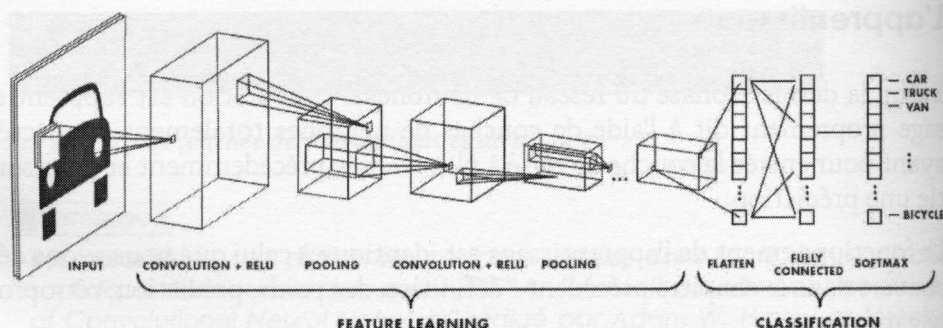
Selon l'auteur de l'outil, le réseau de neurones est composé de 784 neurones en entrée, 300 neurones dans la première couche cachée, de 100 dans la seconde et 10 en sortie correspondant à chacun des différents chiffres.



Prédiction issue du réseau de neurones

3.10 Un schéma global qui résume tout

Pour résumer tout ce que nous venons de voir, nous pouvons nous appuyer sur un schéma très populaire représenté par la figure suivante (<https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html>) où l'on retrouve toutes les étapes d'un réseau de neurones à convolution.



Fonctionnement d'un réseau convolutif (sources mathworks)

4. Un cas pratique autour de la mode

4.1 Présentation de Kaggle

Kaggle (www.kaggle.com) est le site de référence en termes de propositions de défis liés au Machine Learning. Ce site regorge de cas d'études et de challenges proposés par de grands groupes tels que McDonalds, Netflix... Chacun d'entre nous peut tenter de résoudre ces différents défis en proposant ses propres modèles de prédiction. Ce site permet également d'apprendre énormément sur les techniques liées au Machine Learning, car chaque proposition de solution soumise par les participants aux différents défis est visible, commentée et peut ainsi être étudiée. Ce site est entièrement gratuit, alors pourquoi s'en priver ?

C'est à partir de ce site que nous allons réaliser notre cas pratique qui permettra d'illustrer les principes de codage d'un réseau de neurones à convolution.

4.2 Parlons un peu de Keras

Dans le chapitre précédent, nous avons utilisé la librairie TensorFlow. Il existe un module complémentaire à cette librairie appelé Keras se voulant avant tout facile d'utilisation. Cette facilité se trouve notamment dans la phase de création et de paramétrage des différents modèles d'apprentissage, sans oublier qu'il se prête très bien à l'utilisation de réseaux de neurones à convolution, car il possède nativement les fonctions pour y parvenir.

4.3 Classifier des robes, pulls et chaussures?

L'un des cas d'étude les plus utilisés dans la mise en pratique des réseaux de neurones convolutifs est la classification de lettres manuscrites à l'aide de la collection d'observations MNIST (*Mixed National Institute of Standards and Technology*). Cependant, en 2018, Zalando a publié sa propre collection d'images nommée Zalando-MNIST. Cette collection d'images ayant pour but de permettre à des algorithmes de s'entraîner en la classification d'objets de mode : pull, robe, sac... C'est cette collection que nous allons utiliser pour notre cas pratique.

Au-delà de l'aspect très concret du projet, la phase d'apprentissage de ce cas d'étude peut être exécutée par la plupart des ordinateurs. En effet, comme nous l'avons évoqué en début de chapitre, la classification d'images requiert des performances accrues en termes de mémoire, de puissance processeur et de performance de la carte graphique afin d'obtenir de bons résultats et rapidement.

■ Remarque

Il est également possible d'utiliser le Cloud en louant des machines virtuelles situées sur des serveurs dédiés à l'intelligence artificielle, comme le propose Amazon. Ces machines peuvent alors être configurées en termes de puissance en fonction du projet à réaliser.

Nous vous invitons à présent à télécharger les données disponibles sur le site de l'éditeur puis à créer un nouveau projet Python dans lequel il sera nécessaire de créer un dossier datas et d'y déposer l'ensemble des données précédemment téléchargées.



Copie des données d'apprentissage dans le répertoire datas du projet

4.4 De quelles données disposons-nous?

Avant de nous lancer dans toute phase d'apprentissage, nous devons connaître les données dont nous disposons.

Comme nous cherchons à classifier des images, nous devrions donc être en possession de quelques-unes pour réaliser l'apprentissage. Mais visiblement, nous ne disposons pas de fichier image tel que nous avons l'habitude d'en rencontrer. En effet, pas de fichier JPEG ni de PNG.

Cependant, si nous ouvrons le fichier `fashion-mnist_train.csv`, nous pouvons nous apercevoir que celui-ci contient des observations ayant pour features :

- Un label
- Des pixels numérotés de 1 à 784 ayant des valeurs différentes

Voyons à présent ce que nous dit la documentation de ce jeu d'observations disponible sur le site de Kaggle (<https://www.kaggle.com/plarmuseau/zalando-image-classifier/data>) :

- Le jeu d'observations comporte 60000 images d'apprentissage et 10000 images de test.
- Chaque image a une hauteur de 28 pixels et une largeur de 28 pixels, pour un total de 784 pixels.

- Chaque pixel est associé à une seule valeur de pixel, indiquant la luminosité de ce pixel. Cette valeur de pixel est un entier compris entre 0 et 255.
- Les ensembles de données d'entraînement et de tests comportent 785 colonnes.
- La première colonne est constituée des labels de classe et représente l'article vestimentaire. Le reste des colonnes contient les valeurs des pixels de l'image associée.

Toujours à partir de la documentation, nous pouvons déterminer les différents labels :

- 0 - T-shirt/haut
- 1 - Pantalon
- 2 - Pull
- 3 - Robe
- 4 - Manteau
- 5 - Sandales
- 6 - Chemise
- 7 - Baskets
- 8 - Sac
- 9 - Bottes de cheville

À partir de ces informations, nous pouvons donc affirmer que nos images tant attendues sont présentes dans ce fichier sous forme de valeurs par pixel.

Afin de nous en assurer, nous allons essayer d'afficher la première image.

```
import pandas as pnd
import numpy as np

#Définition de la longueur et de la largeur de l'image
LONGUEUR_IMAGE = 28
LARGEUR_IMAGE = 28

#Chargement des images
observations_entrainement = pnd.read_csv('datas/zalando/fashion-
mnist_train.csv')
```

```
#On exclut la première colonne (les labels) pour constituer un  
tableau de pixels
```

```
X = np.array(observations_entrainement.iloc[:, 1:])
```

```
from matplotlib import pyplot as plt
```

```
premiereImage = X[0]
```

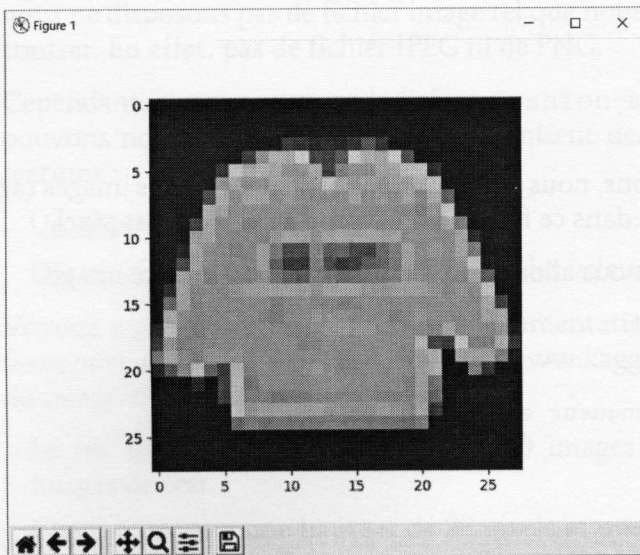
```
premiereImage = premiereImage.reshape([LONGUEUR_IMAGE,  
LARGEUR_IMAGE]);
```

```
plt.imshow(premiereImage)
```

```
plt.show()
```

Après avoir importé les modules Pandas et Numpy, nous avons spécifié les dimensions de l'image afin de les utiliser ultérieurement à divers endroits de notre code.

Nous avons ensuite chargé les images en mémoire dans une variable à l'aide du module Pandas et avons exclu la première colonne pour constituer un tableau ne contenant que les données des images (les pixels). La première image a ensuite été recherchée dans le tableau puis formatée (reshape) pour obtenir une taille de 28 px sur 28 px pour enfin être affichée à l'aide du module matplotlib.



Visualisation d'une image contenue dans les données d'apprentissage

4.5 Préparation des données d'apprentissage

Maintenant que nous avons une petite idée des données que nous allons utiliser, il nous faut les préparer pour que celles-ci soient exploitables par notre réseau de neurones.

La première étape va consister à créer nos jeux d'apprentissage et jeux de tests comme suit :

```
import pandas as pnd
import numpy as np
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split

#Définition de la longueur et de la largeur de l'image
LONGUEUR_IMAGE = 28
LARGEUR_IMAGE = 28

#Chargement des données d'entraînement
observations_entrainement = pnd.read_csv('datas/zalando/fashion-
mnist_train.csv')

#On ne garde que les features "pixels"
X = np.array(observations_entrainement.iloc[:, 1:])

#On crée des catégories à l'aide du module Keras
y = to_categorical(np.array(observations_entrainement.iloc[:,
0]))

#Répartition des données d'entraînement en données
d'apprentissage et données de validation
#80% de donnée d'apprentissage et 20% de donnée de validation
X_apprentissage, X_validation, y_apprentissage, y_validation =
train_test_split(X, y, test_size=0.2, random_state=13)
```

Par ces quelques lignes de code, nous venons donc de créer notre jeu d'apprentissage composé de données d'apprentissage et de données de validation. À noter l'utilisation du module Keras et de la fonction `to_categorical` qui permet de créer un tableau binaire des différentes catégories.

Dans notre exemple, nous disposons de 10 catégories soit un tableau de 10 valeurs possibles : [0,0,0,0,0,0,0,0,0,0]

La première image de notre jeu de données porte le numéro 2 dans sa feature label, nous allons donc mettre un 1 à la troisième position (la première correspondant à la catégorie portant le numéro 0) :

[0,0,1,0,0,0,0,0,0,0]

La seconde image a pour label le numéro 9 ce qui nous donne le tableau suivant :

[0,0,0,0,0,0,0,0,0,1]

Ce processus est ainsi réalisé pour chaque observation afin de constituer un tableau global :

[[0,0,1,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0,1]...]

L'étape suivante va consister à reformater toutes les données des images au format 28 * 28 et à transformer les valeurs de chaque pixel dans une plage de valeurs comprise entre 0 et 1. Cette transformation est une mise à l'échelle de ces valeurs des pixels (scaling) en divisant chaque valeur de pixel par 255 (valeur maximale d'un pixel).

```
# On redimensionne les images au format 28*28 et on réalise un
scaling sur les données des pixels
X_apprentissage =
X_apprentissage.reshape(X_apprentissage.shape[0], LARGEUR_IMAGE,
LONGUEUR_IMAGE, 1)
X_apprentissage = X_apprentissage.astype('float32')
X_apprentissage /= 255
```

On réalise ensuite les mêmes opérations sur les données de validation :

```
X_validation = X_validation.reshape(X_validation.shape[0],
LARGEUR_IMAGE, LONGUEUR_IMAGE, 1)
X_validation = X_validation.astype('float32')
X_validation /= 255
```

4.6 Préparation des données de tests

Les données de tests vont nous permettre de vérifier l'efficacité de notre modèle d'apprentissage. Ces données doivent donc être préparées de la même façon que les données d'apprentissage :

```
#Préparation des données de tests

observations_test = pnd.read_csv('datas/zalando/fashion-
mnist_test.csv')

X_test = np.array(observations_test.iloc[:, 1:])
y_test = to_categorical(np.array(observations_test.iloc[:, 0]))

X_test = X_test.reshape(X_test.shape[0], LARGEUR_IMAGE,
LONGUEUR_IMAGE, 1)
X_test = X_test.astype('float32')
X_test /= 255
```

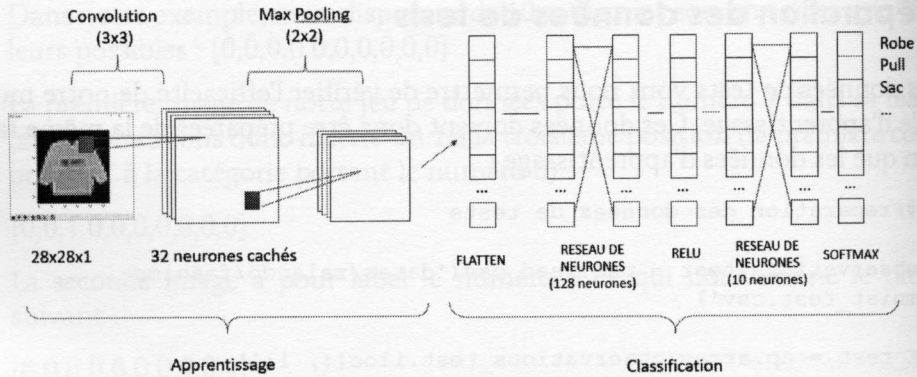
4.7 Un réseau avec une seule couche de convolution

Nous allons effectuer un premier apprentissage avec un réseau de neurones ne comportant qu'une seule couche de convolution.

4.7.1 Configuration

La configuration de notre premier réseau de neurones à convolution va comporter les éléments suivants (cf. figure suivante) :

- Une couche de convolution de 32 filtres de taille 3x3 avec une fonction d'activation de type ReLU
- Un pooling composé d'un filtre de 2x2
- Une phase de flatten (mise en colonne des résultats du pooling)
- Un réseau de neurones complètement connecté de 128 neurones avec une fonction d'activation de type ReLU
- Un dernier réseau de neurones complètement connecté comportant 10 neurones et une fonction d'activation de type Softmax correspondant aux 10 catégories d'images à prédire



Réseau convolutif comportant une seule couche de convolutions

Voici à présent le code associé à cette configuration :

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

#On spécifie les dimensions de l'image d'entrée
dimentionImage = (LARGEUR_IMAGE, LONGUEUR_IMAGE, 1)

#On crée le réseau de neurones couche par couche
reseauNeurone1Convolution = Sequential()

#1- Ajout de la couche de convolution comportant
# 32 filtres de de taille 3x3 (Kernel) parcourant l'image
# Une fonction d'activation de type ReLU (Rectified Linear
Activation)
# Une image d'entrée de 28px * 28 px
reseauNeurone1Convolution.add(Conv2D(32, kernel_size=(3, 3),
activation='relu', input_shape=dimentionImage))

#2- Définition de la fonction de pooling avec une fenêtre de 2px
sur 2 px
reseauNeurone1Convolution.add(MaxPooling2D(pool_size=(2, 2)))

#3- Ajout d'une fonction d'ignorance
reseauNeurone1Convolution.add(Dropout(0.2))

#5 - On transforme en une seule ligne
```

```
reseauNeurone1Convolution.add(Flatten())
```

#6 - Ajout d'un réseau de neurones composé de 128 neurones avec une fonction d'activation de type ReLU

```
reseauNeurone1Convolution.add(Dense(128, activation='relu'))
```

#7 - Ajout d'un réseau de neurones composé de 10 neurones avec une fonction d'activation de type softmax

```
reseauNeurone1Convolution.add(Dense(10, activation='softmax'))
```

■ Remarque

La notion de Drop Out précisée dans le code consiste à ignorer certains neurones dans les phases d'apprentissage et de rétropropagation afin d'éviter le surapprentissage.

4.7.2 Compilation, apprentissage et test

Maintenant que notre réseau est configuré, nous pouvons le compiler et réaliser l'apprentissage :

```
import keras
reseauNeurone1Convolution.compile(loss=keras.losses.categorical_
crossentropy,

optimizer=keras.optimizers.Adam(),
metrics=['accuracy'])
```

La compilation se fait en précisant les paramètres suivants :

- La fonction de minimisation de l'erreur
- L'optimiseur utilisant l'algorithme Adam (méthode de calcul de descente de gradient)
- Et la valeur mesurée, dans notre cas la précision

Avant d'exécuter notre code, nous allons paramétrer l'apprentissage comme suit :

- 10 passages complets du jeu de données à travers le réseau de neurones (epoch)
- 256 images traitées à la fois (batch_size)
- Un affichage des logs d'apprentissage dans la console (verbose=1)

- La validation du modèle est basée sur les images et labels de validation définis plus en amont de ce chapitre (80 % de données d'apprentissage et 20 % de données de validation)

```
historique_apprentissage =
reseauNeurone1Convolution.fit(X_apprentissage, y_apprentissage,
                               batch_size=256,
                               epochs=10,
                               verbose=1,
                               validation_data=(X_validation, y_validation))
```

En d'autres termes, sachant que notre jeu d'apprentissage comporte 60000 images et que nous n'en avons retenu que 80 % pour la phase d'apprentissage, cette dernière se réalisera donc sur 48000 images.

Comme il nous est impossible de passer en une seule fois les 48000 images à travers le réseau de neurones, nous devons les envoyer par paquets à l'aide d'un batch. Selon notre paramétrage, le batch enverra 256 images à travers le réseau de neurones. Par conséquent, pour réaliser un passage complet de l'ensemble des images dans le réseau de neurones (epoch), il nous faudra donc 187 itérations ($48000 / 256$). Mettons ce chiffre de côté, nous verrons qu'il nous sera utile un peu plus loin dans ce chapitre.

Nous ajoutons également l'évaluation du modèle d'apprentissage sur les données de tests afin de déterminer la précision d'apprentissage :

```
evaluation = reseauNeurone1Convolution.evaluate(X_test, y_test,
                                                  verbose=0)
print('Erreur:', evaluation[0])
print('Précision:', evaluation[1])
```

Et terminons par l'affichage d'un graphique où figureront les courbes de précision et d'erreur après chaque passage complet du jeu de données à travers le réseau de neurones (epoch).

```
#Données de précision (accuracy)
plt.plot(historique_apprentissage.history['acc'])
plt.plot(historique_apprentissage.history['val_acc'])
plt.title('Précision du modèle')
plt.ylabel('Précision')
plt.xlabel('Epoch')
plt.legend(['Apprentissage', 'Test'], loc='upper left')
plt.show()
```

```
# Plot training & validation loss values
plt.plot(historique_apprentissage.history['loss'])
plt.plot(historique_apprentissage.history['val_loss'])
plt.title('Erreur')
plt.ylabel('Erreur')
plt.xlabel('Epoch')
plt.legend(['Apprentissage', 'Test'], loc='upper left')
plt.show()
```

Nous pouvons à présent lancer l'apprentissage!

À noter que la phase d'apprentissage prend environ 5 minutes dans notre cas avec un PC équipé d'un processeur Intel Core i5 2,6G GHz avec 8 Go de mémoire et une carte graphique Intel HD Graphics 620 équipée de 64 Mo de RAM.

Ce temps peut donc varier en fonction de votre matériel.

4.7.3 Conclusion sur l'apprentissage

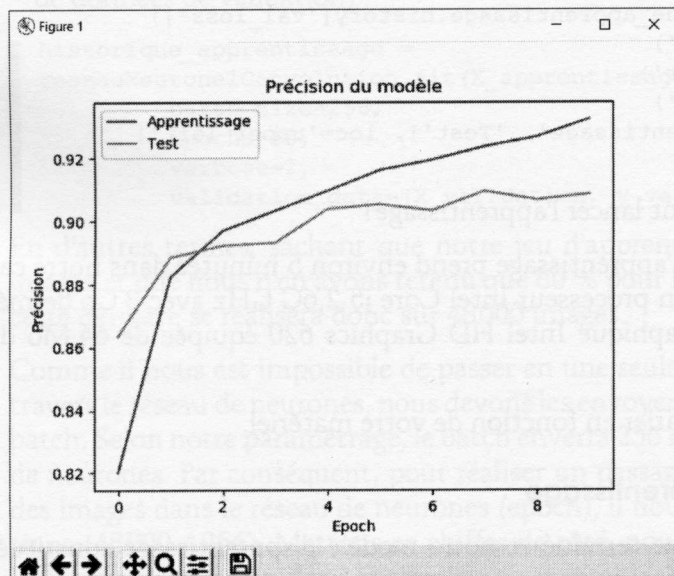
Une fois l'apprentissage terminé et notre modèle d'apprentissage constitué, nous obtenons les chiffres suivants sur notre jeu de tests (ceux-ci pouvant varier).

```
Erreur : 0.23872388958334922
Précision: 0.9154
```

Signifiant que nous avons une précision de classification de 91,54 % avec un taux d'erreur de 23,87 %.

Ces informations sont reprises dans les deux graphiques qui suivent où l'on constate une augmentation de la précision dans la phase d'apprentissage jusqu'à atteindre 93 % ainsi qu'une diminution de l'erreur. Ces deux informations étant tout à fait logiques.

Lorsque l'on applique notre modèle sur les données de tests, on constate les graphiques suivants :

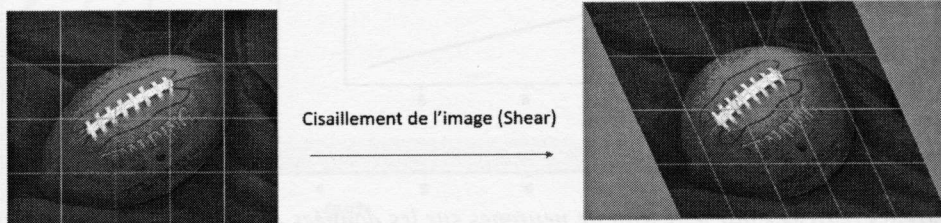


Précision du réseau de neurones sur les données d'apprentissage et de tests

Le principe de cette fonction est le suivant :

À partir d'une image contenue dans le jeu d'apprentissage, la fonction va procéder à des actions de rotation, de zoom, de translation de façon aléatoire afin de créer de nouvelles images. Voici les paramètres de cette fonction :

- `rotation_range` est l'angle de rotation de l'image compris entre 0 et 180.
- `width_shift` et `height_shift` sont les plages de translation verticale ou horizontale de l'image.
- `shear_range` est le niveau de cisaillement de l'image (figure ci-dessous).
- `zoom_range` est le niveau de zoom de l'image.



Cisaillement de l'image

Nous allons à présent générer de nouvelles images et procéder à un nouvel apprentissage.

```
nouvelles_images_apprentissage =
generateur_images.flow(X_apprentissage, y_apprentissage,
batch_size=256)
nouvelles_images_validation =
generateur_images.flow(X_validation, y_validation,
batch_size=256)

historique_apprentissage =
reseauNeurone1Convolution.fit_generator(nouvelles_images_apprent
issage,

steps_per_epoch=48000//256,

epochs=50,

validation_data=nouvelles_images_validation,
```

```
validation_steps=12000//256,  
  
use_multiprocessing=False,  
  
verbose=1 )  
  
evaluation = reseauNeurone1Convolution.evaluate(X_test, y_test,  
verbose=0)  
print('Erreur :', evaluation[0])  
print('Précision:', evaluation[1])
```

Pour notre nouvel apprentissage, nous utilisons la fonction `fit_generator` du module Keras au lieu de la fonction `fit` utilisée dans le précédent apprentissage. Cela est dû au fait que pour chaque sous-tâche d'apprentissage (batch) les images que nous utilisons sont différentes, car générées de façon aléatoire avec notre générateur d'image. La fonction `fit` n'est pas en mesure de gérer ce type de changement à la volée, contrairement à la fonction `fit_generator` prévue à cet effet.

On note également une augmentation du nombre de passages complets du jeu de données à travers le réseau de neurones passant de 10 à 50 car nous disposons d'un nombre d'images plus important.

La fonction `fit_generator` peut être vue comme une boucle d'apprentissage qui n'a pas de fin. Afin que cette boucle s'arrête, nous devons lui indiquer le nombre d'itérations à effectuer par epoch, c'est le rôle du paramètre `steps_per_epoch` prenant la valeur 187 comme nous l'avions calculé lors du premier apprentissage. Ainsi lorsque le nombre d'itérations sera atteint pour une epoch, l'epoch suivante sera déclenchée jusqu'à atteindre le nombre total d'epoch spécifiées, soit 50 dans notre cas.

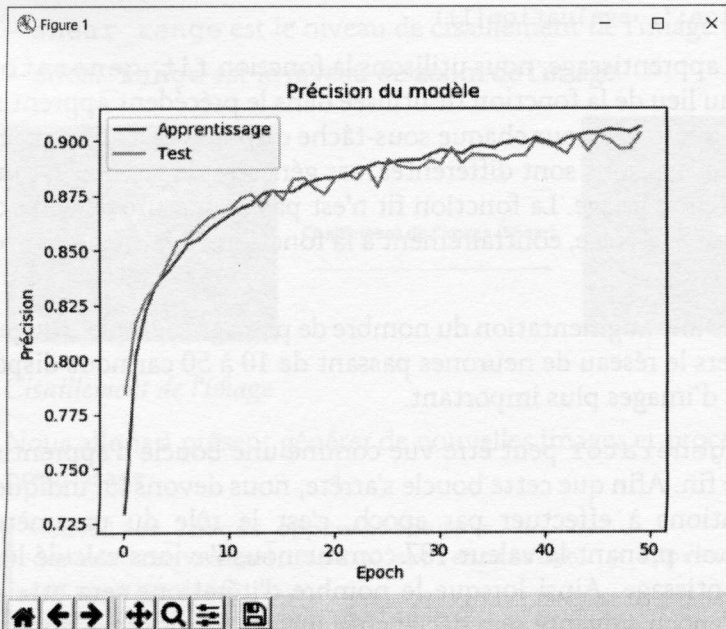
Nous pouvons à présent exécuter notre nouvel apprentissage et constater dans un premier temps que celui-ci prend beaucoup plus de temps! (Environ 40 minutes contre les 5 minutes lors du précédent apprentissage). Cela vient du fait qu'en complément de l'apprentissage, de nouvelles images sont générées à chaque batch, ce qui nécessite des ressources matérielles plus importantes et ralentit donc l'ensemble du processus.

Mais en dépit de ce temps d'apprentissage, on peut s'apercevoir une petite amélioration de l'apprentissage frôlant les 92 %.

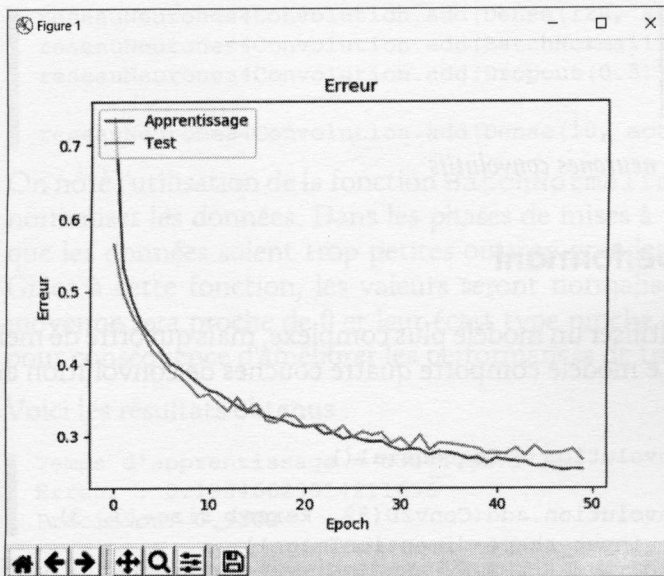
Erreur : 0.2299670210123062

Précision: 0.9175

Et une diminution du taux d'erreur entre la phase d'apprentissage et la phase de test. On peut donc considérer que ce modèle est plus performant que le précédent.



Amélioration de la précision du réseau de neurones



Diminution du nombre d'erreurs du réseau de neurones

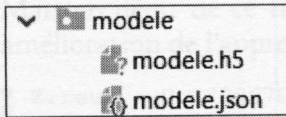
4.7.5 Sauvegarde du modèle

Maintenant que nous avons un modèle un peu plus performant, nous pouvons le sauvegarder afin de l'utiliser par la suite.

Pour cela, il convient de créer un nouveau répertoire dans le projet que nous nommerons "modele" qui sera chargé de contenir la définition modèle au format JSON et les différents poids issus de l'apprentissage seront quant à eux sauvegardés dans un fichier ayant pour extension H5. Ces deux fichiers pouvant être utilisés par la suite dans une application annexe.

```
#Sauvegarde du modèle
# Serialisation du modèle model_json =
reseauNeurone1Convolution.to_json() with open("modele/mode-
le.json", "w") as json_file:
    json_file.write(model_json)

# Serialisation des poids
reseauNeurone1Convolution.save_weights("modele/modele.h5")
print("Modèle sauvegardé !")
```



Sauvegarde du réseau de neurones convolutifs

4.8 Un modèle plus performant

Nous allons à présent utiliser un modèle plus complexe, mais qui offre de meilleures performances. Ce modèle comporte quatre couches de convolution définies comme suit :

```
reseauNeurones4Convolution = Sequential()

reseauNeurones4Convolution.add(Conv2D(32, kernel_size=(3, 3),
activation='relu', input_shape=dimensionImage))
reseauNeurones4Convolution.add(BatchNormalization())

reseauNeurones4Convolution.add(Conv2D(32, kernel_size=(3, 3),
activation='relu'))
reseauNeurones4Convolution.add(BatchNormalization())
reseauNeurones4Convolution.add(MaxPooling2D(pool_size=(2, 2)))
reseauNeurones4Convolution.add(Dropout(0.25))

reseauNeurones4Convolution.add(Conv2D(64, kernel_size=(3, 3),
activation='relu'))
reseauNeurones4Convolution.add(BatchNormalization())
reseauNeurones4Convolution.add(Dropout(0.25))

reseauNeurones4Convolution.add(Conv2D(128, kernel_size=(3, 3),
activation='relu'))
reseauNeurones4Convolution.add(BatchNormalization())
reseauNeurones4Convolution.add(MaxPooling2D(pool_size=(2, 2)))
reseauNeurones4Convolution.add(Dropout(0.25))

reseauNeurones4Convolution.add(Flatten())

reseauNeurones4Convolution.add(Dense(512, activation='relu'))
reseauNeurones4Convolution.add(BatchNormalization())
reseauNeurones4Convolution.add(Dropout(0.5))
```

```
reseauNeurones4Convolution.add(Dense(128, activation='relu'))
reseauNeurones4Convolution.add(BatchNormalization())
reseauNeurones4Convolution.add(Dropout(0.5))

reseauNeurones4Convolution.add(Dense(10, activation='softmax'))
```

On note l'utilisation de la fonction `BatchNormalisation()` qui permet de normaliser les données. Dans les phases de mises à jour des poids, il se peut que les données soient trop petites ou trop grandes par rapport aux autres. Grâce à cette fonction, les valeurs seront normalisées, c'est-à-dire que leur moyenne sera proche de 0 et leur écart type proche de 1. Cette phase a aussi pour conséquence d'améliorer les performances de traitement.

Voici les résultats obtenus :

```
Temps d'apprentissage = 53044.6571731
Erreur : 0.19346623014211656
Précision: 0.9302
```

Soit 93 % de bonne classification. Mais cela a un coût, car avec la configuration de notre PC que nous vous avons présenté dans ce chapitre, la phase d'apprentissage a mis environ 8H00!!

5. Utilisation du modèle avec de nouvelles images

Nous allons à présent utiliser le dernier modèle avec de nouvelles images.

Fort heureusement pour vous, nous avons réalisé une sauvegarde du modèle précédemment généré vous évitant ainsi d'attendre le temps de sa création. Libre à vous de le télécharger et de le placer dans le répertoire modèle du projet.

Nous vous invitons à créer un nouveau fichier Python que l'on nommera `classification.py`.

La première étape va être de charger le modèle comme suit :

```
#-----
# CHARGEMENT DU MODELE
#-----
```

```
#Chargement de la description du modèle
```

```
fichier_json = open('modele/modele_4convolutions.json', 'r')  
modele_json = fichier_json.read()  
fichier_json.close()
```

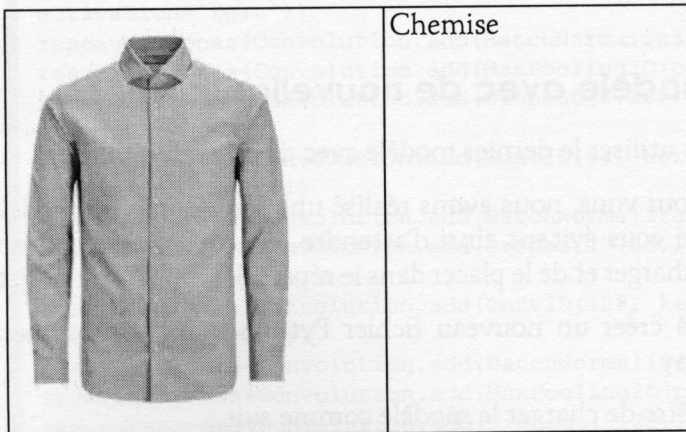
```
#Chargement de la description des poids du modèle  
from keras.models import model_from_json  
modele = model_from_json(modele_json)  
#Chargement des poids  
modele.load_weights("modele/modele_4convolutions.h5")
```





On définit ensuite les différentes catégories de classifications :

```
#Définition des catégories de classifications  
classes = ["Un T-shirt/haut", "Un pantalon", "Un pull",  
"Une robe", "Un manteau", "Des sandales", "Une chemise", "Des  
baskets", "Un sac", "Des bottes de cheville"]
```

Ensuite, libre à vous de choisir des images de vêtements sur Internet et de les sauvegarder dans un répertoire dans votre projet que vous nommerez images.

Pour notre part, nous avons choisi les images suivantes :



	<p>Pull</p>
	<p>Baskets</p>
	<p>Robe</p>
	<p>Pantalon</p>

L'image doit être transformée pour être utilisée dans notre modèle : Conversion en niveaux de gris, et redimensionnement. Pour cela, nous allons utiliser le module Pilow comme suit :

```
#-----  
# CHARGEMENT ET TRANSFORMATION D'UNE IMAGE  
#-----  
  
from PIL import Image, ImageFilter  
  
#Chargement de l'image et conversion de celle-ci en nuances de  
gris (L = greyScale)  
image = Image.open("images/pantalon.jpg").convert('L')  
  
#Dimension de l'image  
longueur = float(image.size[0])  
hauteur = float(image.size[1])  
  
#Création d'une nouvelle image  
nouvelleImage = Image.new('L', (28, 28), (255))  
  
#Redimensionnement de l'image  
#L'image est plus longue que haute, on la met à 20 pixels  
if longueur > hauteur:  
    #On calcule le ratio d'agrandissement entre la hauteur et  
    la longueur  
    ratioHauteur = int(round((20.0 / longueur * hauteur), 0))  
    if (ratioHauteur == 0):  
        nHauteur = 1  
  
    #Redimensionnement  
    img = image.resize((20, ratioHauteur),  
Image.ANTIALIAS).filter(ImageFilter.SHARPEN)  
    #Position horizontale  
    position_haut = int(round(((28 - ratioHauteur) / 2), 0))  
  
    nouvelleImage.paste(img, (4, position_haut)) # paste  
    resized image on white canvas  
else:  
  
    ratioHauteur = int(round((20.0 / hauteur * longueur), 0)) #  
    resize width according to ratio height  
    if (ratioHauteur == 0): # rare case but minimum is 1 pixel  
        ratioHauteur = 1
```

```
#Redimensionnement
img = image.resize((ratioHauteur, 20),
Image.ANTIALIAS).filter(ImageFilter.SHARPEN)

#Calcul de la position verticale
hauteur_gauche = int(round(((28 - ratioHauteur) / 2), 0))
nouvelleImage.paste(img, (hauteur_gauche, 4))

#Récupération des pixels
pixels = list(nouvelleImage.getdata())

#Normalisation des pixels
tableau = [(255 - x) * 1.0 / 255.0 for x in pixels]
```

Enfin, nous allons soumettre l'image à notre modèle et afficher les résultats :

```
import numpy as np
#Transformation du tableau en tableau numpy
img = np.array(tableau)

#On transforme le tableau linéaire en image 28x20
image_test = img.reshape(1, 28, 28, 1)

prediction = modele.predict_classes(image_test)
print()
print("Selon moi l'image est : "+classes[prediction[0]])
print()

#Extraction des probabilités
probabilites = modele.predict_proba(image_test)

i=0
for classe in classes:
    print(classe + " : "+str((probabilites[0][i]*100))+"%")
    i=i+1
```

Voici les résultats obtenus pour l'image concernant le pull :

```
Selon moi l'image est : Un pull
Un T-shirt/haut: 3.5228632390499115%
Un pantalon: 0.09517147555015981%
Un pull: 74.61184859275818%
Une robe: 1.9539179280400276%
Un manteau: 13.278758525848389%
```

Des sandales: 0.007388717494904995%
Une chemise: 6.171694025397301%
Des baskets: 0.004739782161777839%
Un sac: 0.35167725291103125%
Des bottes de cheville: 0.0019266608433099464%

Pour l'image concernant le pantalon :

Selon moi l'image est : **Un pantalon**
Un T-shirt/haut: 0.0033752712624846026%
Un pantalon: 99.9247670173645%
Un pull: 0.004438478936208412%
Une robe: 0.04969787551090121%
Un manteau: 0.0058175690355710685%
Des sandales: 0.0003859092430502642%
Une chemise: 0.006958609446883202%
Des baskets: 0.00030909859560779296%
Un sac: 0.004130171146243811%
Des bottes de cheville: 0.00013043996887063258%

Maintenant concernant la robe :

Selon moi l'image est : **Une robe**
Un T-shirt/haut: 0.023915201018098742%
Un pantalon: 0.024725537514314055%
Un pull: 0.000664757271806593%
Une robe: 99.91241097450256%
Un manteau: 0.006916730490047485%
Des sandales: 2.0109922616029507e-06%
Une chemise: 0.028603975079022348%
Des baskets: 4.448996548944706e-06%
Un sac: 0.002755552668531891%
Des bottes de cheville: 5.146377901610322e-06%

Cependant, le modèle se trompe parfois dans ses prédictions!

Dans le cas de la chemise :

Selon moi l'image est : **Un manteau**
Un T-shirt/haut: 0.06315075443126261%
Un pantalon: 0.014026503777131438%
Un pull: 2.549430914223194%
Une robe: 0.4956232849508524%
Un manteau: 95.14198303222656%
Des sandales: 0.0012443994819477666%

Une chemise: 1.7253907397389412%

Des baskets: 0.0017185098840855062%

Un sac: 0.0069410343712661415%

Des bottes de cheville: 0.00047413413994945586%

Dans le cas des baskets :

Selon moi l'image est : **Des sandales**

Un T-shirt/haut: 0.0063549421611242%

Un pantalon: 0.003332917913212441%

Un pull: 0.004925281973555684%

Une robe: 0.018478820857126266%

Un manteau: 0.03851952496916056%

Des sandales: 95.15430927276611%

Une chemise: 0.009490534284850582%

Des baskets: 4.69491071999073%

Un sac: 0.04609071183949709%

Des bottes de cheville: 0.02358692290727049%

Cela montre le fait que l'algorithme n'est pas parfait et qu'il peut sans doute être encore amélioré.

6. Pour conclure ce chapitre

Dans ce chapitre, nous avons abordé les notions de classification d'images à l'aide de réseaux de neurones convolutifs. Son nom peut paraître de prime abord un peu barbare, mais si l'on procède à son analyse, son fonctionnement est assez simple à comprendre.

Cependant, nous avons vu que l'apprentissage de ce type d'algorithme nécessite de nombreuses images, du temps et de bonnes ressources matérielles.

Enfin, nous avons également vu comment sauvegarder un modèle et l'utiliser à travers un nouveau script dans le but de réaliser des prédictions sur de nouvelles images inconnues du modèle. Cette approche étant similaire à l'utilisation du modèle dans une nouvelle application.

Dans le chapitre suivant, nous allons construire un projet complet alliant détection et reconnaissance d'image.

Chapitre 13

Votre ordinateur sait lire!

1. Ce que nous allons découvrir et les prérequis

Dans ce chapitre, nous allons réaliser un projet dont l'objectif est de faire lire à votre ordinateur et à voix haute les lettres de l'alphabet qui lui seront présentées grâce à une webcam. Nous allons donc aborder dans ce chapitre la détection et la reconnaissance basées sur un flux vidéo.

■ Remarque

Prérequis : avoir lu l'ensemble de l'ouvrage.

Étant donné que nous sommes à présent vers la fin de l'ouvrage, nous avons souhaité orienter ce chapitre sous la forme d'un cas pratique que vous réaliserez en autonomie. En effet, l'ensemble des concepts nécessaires à la réalisation du choix de l'algorithme d'apprentissage ont été abordés. Mais pas de panique, nous vous guiderons tout au long du processus. Enfin, nous terminerons par une explication du fonctionnement de la capture vidéo et de son utilisation dans notre projet.

2. Votre mission

Vous avez pour objectif de faire en sorte que, lorsqu'une lettre manuscrite est présentée face à la webcam de votre ordinateur, celui-ci la détecte et en fait la reconnaissance. Avec un petit plus non négligeable, il devra être capable de la lire à voix haute.

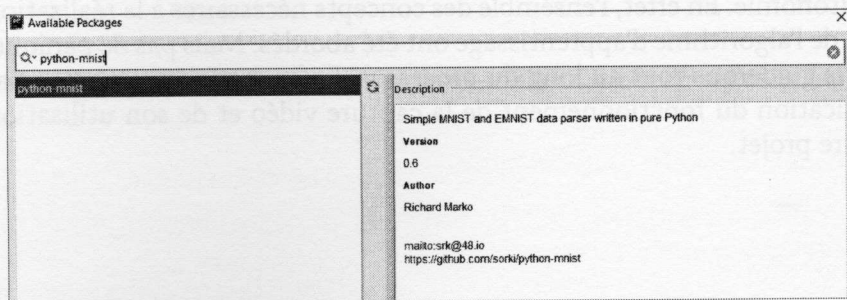
Ne vous souciez pas pour le moment des phases de détection et de lecture à haute voix, nous vous les expliquerons en détail en fin de chapitre.

Le périmètre de votre mission se borne à l'apprentissage de la reconnaissance d'une lettre manuscrite. Pour y parvenir, nous allons vous guider par le biais de questions auxquelles nous vous invitons à répondre au fur et à mesure.

2.1 Question n°1 : de quelles données avez-vous besoin ?

Réponse : *Vous avez besoin d'un jeu d'observations contenant les différentes lettres de l'alphabet.*

Nous avons vu dans le chapitre précédent qu'il existait une base de données nommée MNIST (*Modified ou Mixed National Institute of Standards and Technology*) permettant d'obtenir un jeu d'observations pour la reconnaissance de chiffres. Cependant, il existe également un jeu d'observations pour la reconnaissance de lettres. Pour l'obtenir, nous allons utiliser le module `python-mnist` comportant à la fois le jeu d'observations de chiffres et le jeu d'observations de lettres. Le module et sa documentation se trouvent à cet endroit : <https://github.com/sorki/python-mnist>.



Installation du module Python-mnist

2.2 Question n°2 : comment utiliser le module Python-Mnist ?

Réponse : la première chose à faire est de regarder la documentation qui, avouons-le, est un peu complexe à comprendre au premier regard. C'est pourquoi nous allons vous aider.

Dans la documentation, il est fait mention de ceci :

To use EMNIST datasets you need to call:

```
mndata.select_emnist('digits')
```

Where digits is one of the available EMNIST datasets. You can choose from

balanced

byclass

bymerge

digits

letters

mnist

Ce qui signifie que le module propose plusieurs types de jeux de données, à savoir digits pour les chiffres et letters pour les lettres (celui qui nous intéresse). Pour les utiliser, nous devons utiliser la méthode `select_emnist`.

■ Remarque

EMnist est un jeu d'observations étendu du Mnist standard :

<https://www.nist.gov/node/1298471/emnist-dataset>

Mndata étant une variable initialisée comme suit :

```
from mnist import MNIST
mndata = MNIST('./dir_with_mnist_data_files')
images, labels = mndata.load_training()
```

Library tries to load files named t10k-images-idx3-ubyte train-labels-idx1-ubyte train-images-idx3-ubyte and t10k-labels-idx1-ubyte. If loading throws an exception check if these names match.

On constate que nous avons besoin de spécifier un répertoire pour utiliser le module. Comme l'indique la ligne en gras mentionnée ci-dessus, le répertoire spécifié doit contenir les fichiers **t10k-images-idx3-ubyte train-labels-idx1-ubyte train-images-idx3-ubyte et t10k-labels-idx1-ubyte** pour que le module fonctionne.


Comment faire, car nous ne les avons pas ?

Regardons dans le fichier README.rst de la documentation, nous donnant cette information :

MNIST is a database of handwritten digits available on <http://yann.lecun.com/exdb/mnist/>. EMNIST is an extended MNIST database <https://www.nist.gov/itl/iad/image-group/emnist-dataset>.

Nous n'avons donc plus qu'à nous rendre à l'adresse indiquée et télécharger le fichier :

Where to download ?

- [Readme.txt file](#)
- Binary format as the original MNIST dataset 
- [Matlab format dataset](#)
- EMNIST paper, available at: <https://arxiv.org/abs/1702.05373v1>

Téléchargement des données d'apprentissage

Une fois le fichier téléchargé (plus de 500 Mo), nous devons le dézipper afin d'obtenir les fichiers suivants :

- emnist-**letters**-test-images-idx3-ubyte.gz
- emnist-**letters**-test-labels-idx1-ubyte.gz
- emnist-**letters**-train-images-idx3-ubyte.gz
- emnist-**letters**-train-labels-idx1-ubyte.gz

■ Remarque

Le logiciel WinRar fera très bien ce travail de dézippage.

Nous disposons alors des fichiers nécessaires que nous pouvons mettre dans le répertoire `datas` de notre projet que nous aurons préalablement créé.

Essayons à présent ces quelques lignes de code :

```
#Chargement des images
emnist_data = MNIST(path='datas\\', return_type='numpy')
emnist_data.select_emnist('letters')
Images, Libelles = emnist_data.load_training()
```

Nous pouvons constater que nous n'avons pas d'erreur, ce qui est plutôt bon signe.

Comme vous pouvez le constater, obtenir des données, ce n'est pas forcément aisé et cela nécessite de bien lire la documentation.

2.3 Question n°3 : de quelles données disposez-vous à présent?

Réponse : *il faut que vous sachiez de combien de données vous disposez, s'il y a des données manquantes et avoir éventuellement un aperçu de celles-ci.*

Pour cela, nous allons tout simplement calculer le nombre d'images de labels :

```
print("Nombre d'images =" + str(len(Images)))
print("Nombre de libellés =" + str(len(Libelles)))
```

Puis constater que ceux-ci sont égaux, signifiant qu'il ne manque donc pas de données :

```
Nombre d'images =124800
Nombre de libellés =124800
```

Si nous voulons afficher des graphiques ou des images, nous devons faire appel au module `matplotlib` et au module `numpy` pour pouvoir manipuler plus facilement les données. Notons au passage le redimensionnement des images (28x28 pixels) :

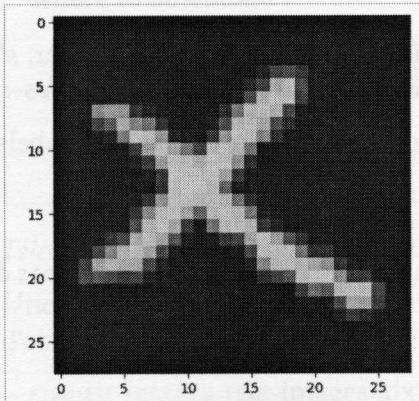
```
#Dimension des images de travail et d'apprentissage
longueurImage = 28
largeurImage = 28
```

```
#Les images sont sous forme d'un tableau de 124800 lignes et 784
colonnes
#On les transforme en un tableau comportant 124800 lignes
contenant un tableau de 28*28 colonnes pour pouvoir les manipuler
et les afficher.
# On transforme également les libellés en un tableau ne comportant
qu'une seule colonne

print("Transformation des tableaux d'images et de libelles...")
Images = Images.reshape(124800, largeurImage, longueurImage)
Libelles= Libelles.reshape(124800, 1)

print("Affichage de l'image N°70000...")
from matplotlib import pyplot as plt
plt.imshow(Images[70000])
plt.show()
```

La figure suivante montre que l'image N°70000 est un X :



L'image 70000 est un X

Regardons à présent le libellé correspondant à l'image n°70000 :

```
■ print(Libelles[70000])
```

Cette instruction nous renvoyant la valeur 24, nous pouvons en déduire que chaque libellé correspond au numéro de la lettre dans l'alphabet, car X en est la 24^e.

Comme en informatique on aime bien que les valeurs de listes commencent à zéro, nous allons retirer la valeur 1 à chaque libellé. Ainsi la lettre X aura la valeur 23.

```
print("En informatique, les index des listes doivent commencer à  
zéro...")  
Libelles = Libelles-1  
  
print("Libellé de l'image N°70000...")  
print(Libelles[70000])
```

Maintenant que nous en savons un peu plus sur nos données, nous pouvons réfléchir sur l'algorithme que nous allons utiliser.

2.4 Question n°4 : est-ce un problème de régression ou de classification?

Réponse : nous sommes face à un problème de classification.

Les problèmes de régression sont pour objectif de prédire une valeur. Dans notre cas, nous cherchons à classer une image parmi les 26 lettres de l'alphabet, il s'agit donc un problème de classification.

2.5 Question n°5 : quel algorithme allez-vous utiliser?

Réponse : vous allez utiliser un réseau de neurones convolutifs.

En effet, comme il s'agit de réaliser la classification d'une image, les réseaux de neurones convolutifs semblent tout indiqués.

2.6 Question n°6 : comment allez-vous créer vos jeux d'apprentissage et de tests ?

Réponse : *les jeux d'apprentissage et de tests sont créés à l'aide du module Sklean.*

```
print("Création des jeux d'apprentissage et de tests...")

from sklearn.model_selection import train_test_split
images_apprentissage, images_validation, libelles_apprentissage,
libelles_validation = train_test_split(Images, Libelles,
test_size=0.25, random_state=111)
```

2.7 Question n°7 : les images sont elles au bon format ?

Réponse : *les images ont des valeurs de pixels comprises entre 0 et 255. L'idéal est de travailler sur une échelle de valeurs comprise entre 0 et 1, par conséquent vous allez réaliser une opération de Scaling (mise à l'échelle).*

Pour ce faire, nous allons prendre chaque valeur de pixel des images d'apprentissage et de tests pour les diviser par 255.

Mais avant cela, nous allons préparer les images pour qu'elles soient utilisables par le réseau de neurones convolutifs, notamment par la couche de convolution.

Le paramètre `input_shape` de la fonction `Conv2D` du module `keras` correspondant à l'image sur laquelle doit s'appliquer la couche de convolution doit être sous la forme (`longueur_image`, `largeur_image`, nombre de dimensions de l'image). L'image dispose de trois dimensions : une pour le rouge, une pour le vert et une pour le bleu. Dans notre cas, nous utilisons des images en niveau de gris n'ayant qu'une seule dimension (channel), par conséquent le dernier paramètre doit être égal à 1.

Nous allons donc à présent transformer les tableaux des images d'apprentissage et de tests pour qu'elles prennent en compte ce nouveau paramètre :

```
#Ajout d'une troisième valeur à nos tableaux d'images pour
qu'ils puissent être utilisés par le réseau de neurones, notamment
le paramètre input_shape de la fonction Conv2D

images_apprentissage =
images_apprentissage.reshape(images_apprentissage.shape[0],
largeurImage, longueurImage, 1)
print(images_apprentissage.shape)

images_validation =
images_validation.reshape(images_validation.shape[0],
largeurImage, longueurImage, 1)
imageTravail = (largeurImage, longueurImage, 1)
```

Chaque image du jeu d'apprentissage sera traitée par le réseau de neurones convolutifs. Nous créons alors une nouvelle variable qui contiendra l'ensemble des données de l'image qui sera ensuite utilisée comme donnée d'entrée à la couche de convolution :

```
#Création d'une variable servant d'image de travail au réseau de
neurones
imageTravail = (largeurImage, longueurImage, 1)
```

Passons à présent à la phase de mise à l'échelle des valeurs de chaque pixel :

```
print("Valeur des pixels avant la mise à l'échelle :")
print(Images_apprentissage[40000])

#Mise à l'échelle
images_apprentissage = images_apprentissage.astype('float32')/255
images_validation = images_validation.astype('float32')/255
```

Exemple de valeur des pixels avant mise à l'échelle :

0 0 0 0 0 0 0 0 0 32 77 124 113 77 7 0 0 0 0

Exemple de valeur des pixels après mise à l'échelle :

0. 0. 0. 0.00784314 0.01568628 0.01568628

2.8 Question n°8 : qu'est-ce que la catégorisation des libellés en One-Hot et comment procéder pour la réaliser?

Réponse : imaginons que vous avez quatre catégories possibles. Ceci se traduit par un tableau de quatre zéros [0,0,0,0]. Si le libellé de la première observation correspond à la première catégorie, vous aurez alors pour ce libellé une valeur de type [1,0,0,0], et s'il correspond à la troisième catégorie vous aurez la valeur de type [0,0,1,0].

Pour réaliser cela facilement, il faut utiliser la fonction `to_categorical` du module Keras :

```
# Création des catégories en One-Hot encoding
nombre_de_classes = 26
libelles_apprentissage =
keras.utils.to_categorical(libelles_apprentissage,
nombre_de_classes)
libelles_validation =
keras.utils.to_categorical(libelles_validation,
nombre_de_classes)
```

2.9 Question n°9 : avez-vous une petite idée des paramètres à utiliser pour créer le réseau de neurones?

Réponse : ce n'est pas une question évidente. Ce cas d'apprentissage étant maintes et maintes fois traité, une réponse peut sans doute être trouvée sur Internet pour ensuite être testée.

De notre côté, nous nous sommes documentés et avons trouvé sur le Web le modèle suivant :

- Une première couche de convolutions comportant 32 filtres de dimension 3x3
- Une seconde couche de convolutions comportant 64 filtres de dimension 3x3
- Une couche de pooling
- Une couche de "mise à plat"

- Une couche d'apprentissage comportant 128 neurones en entrée et dont la fonction d'activation est de type ReLU
- Enfin, une couche de type SoftMax utilisée pour définir les pourcentages de probabilité pour les 26 lettres de l'alphabet

Voici donc ce modèle créé avec le module Keras, où l'on retrouve en entrée de la première couche de convolution l'image à analyser, dont les données sont stockées dans la variable `imageTravail` alimentant le paramètre `input_shape` de la fonction `Conv2D`.

```
reseauCNN = Sequential()
reseauCNN.add(Conv2D(32, kernel_size=(3, 3),
                    activation='relu',
                    input_shape=imageTravail))

#Une seconde couche de 64 filtres de dimension 3x3
reseauCNN.add(Conv2D(64, (3, 3), activation='relu'))

#Une fonction de pooling
reseauCNN.add(MaxPooling2D(pool_size=(2, 2)))
reseauCNN.add(Dropout(0.25))

#Une mise à plat
reseauCNN.add(Flatten())

#Le réseau de neurones avec en entrée 128 neurones
#une fonction d'activation de type ReLU
reseauCNN.add(Dense(128, activation='relu'))
reseauCNN.add(Dropout(0.5))

#Une dernière couche de type softmax
reseauCNN.add(Dense(nombre_de_classes, activation='softmax'))
```

Nous compilons ensuite ce réseau de cette façon :

```
reseauCNN.compile(loss=keras.losses.categorical_crossentropy,
                  optimizer=keras.optimizers.Adadelta(),
                  metrics=['accuracy'])
```

Avec une fonction d'erreur de type `categorical_crossentropy`, un optimiseur de cette fonction de type `Adadelta()` tout en prenant la précision (`accuracy`) comme valeur d'indicateur de performance.

■ Remarque

Adadelta est une méthode de descente de gradient qui adapte son taux d'apprentissage au fur et à mesure des apprentissages effectués.

Nous ne nous étendrons pas sur les différents paramètres. Sachez cependant que la fonction d'erreur `categorical_crossentropy` est une fonction optimisée pour la réalisation de classifications multiclassées, à condition que celles-ci soient passées par la case de l'encodage One-Hot (ce qui est notre cas).

On réalise ensuite l'apprentissage à l'aide de 10 epochs et un `batch_size` de 128.

Après 1 h d'apprentissage, le modèle est sauvegardé dans le répertoire modèle sous le nom `modele_cas_pratique.h5`.

```
# Apprentissage avec une phase de validation
# sur les jeux de tests
# le paramètre verbose=1, permet d'afficher les logs lors de
l'apprentissage
batch_size = 128
epochs = 10

reseauCNN.fit(images_apprentissage, libelles_apprentissage,
              batch_size=batch_size,
              epochs=epochs,
              verbose=1,
              validation_data=(images_validation,
                              libelles_validation))

# Sauvegarde du modèle
reseauCNN.save('modele/modele_cas_pratique.h5')
```

Enfin, on vérifie la précision du réseau de neurones convolutifs sur les images de tests :

```
# Evaluation de la précision
score = reseauCNN.evaluate(images_validation,
                           libelles_validation, verbose=0)
print('Erreur sur les données de validation:', score[0])
print('Précision sur les données de validation:', score[1])
```

2.10 Question n°10 : trouvez-vous le résultat satisfaisant?

Réponse : *vous pouvez à présent lancer l'apprentissage et, après 5 minutes, vous constatez que la précision est de 93 %, ce qui est plutôt bien.*

■ Précision sur les données de validation: 0.9316105769230769

■ Remarque

À l'issue de l'apprentissage, il est important de vérifier dans le répertoire `modele` que le modèle a bien été sauvegardé sous le nom `modele_cas_pratique.h5`.

2.11 Mission accomplie!

Félicitations, la première partie de votre mission est à présent accomplie. Voici le code complet permettant à votre machine d'apprendre à reconnaître les lettres manuscrites :

```
from __future__ import print_function
import keras
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from sklearn.model_selection import train_test_split

#Chargement des images
print("Chargement des images")
from mnist import MNIST
mndata = MNIST('datas')
Images, Libelles = mndata.load_training()

#Conversion des images et libellés en tableau numpy
import numpy as np
Images = np.asarray(Images)
Libelles = np.asarray(Libelles)

#Dimension des images de travail et d'apprentissage
longueurImage = 28
```

```
largeurImage = 28

#Les images sont sous forme d'un tableau de 124800 lignes et 784
colonnes
#On les transforme en un tableau comportant 124800 lignes
contenant un tableau de 28*28 colonnes
print("Transformation des tableaux d'images...")
Images = Images.reshape(124800, largeurImage, longueurImage)
Libelles= Libelles.reshape(124800, 1)

#En informatique, les index des listes doivent commencer à
zéro...)
Libelles = Libelles-1

#Création des jeux d'apprentissage et de tests
images_apprentissage, images_validation, libelles_apprentissage,
libelles_validation = train_test_split(Images, Libelles,
test_size=0.25,random_state=42)

#Ajout d'une troisième valeur à nos tableaux d'images pour qu'ils
puissent être utilisés par le réseau de neurones, notamment le
paramètre input_shape de la fonction Conv2D
images_apprentissage =
images_apprentissage.reshape(images_apprentissage.shape[0],
largeurImage, longueurImage, 1)
print(images_apprentissage.shape)

images_validation =
images_validation.reshape(images_validation.shape[0],
largeurImage, longueurImage, 1)

#Création d'une variable servant d'image de travail au réseau
de neurones
imageTravail = (largeurImage, longueurImage, 1)

#Mise à l'échelle
images_apprentissage = images_apprentissage.astype('float32')/
255
images_validation = images_validation.astype('float32')/255

# Création des catégories en One-Hot encoding
nombre_de_classes = 26
```

```
libelles_apprentissage =
keras.utils.to_categorical(libelles_apprentissage,
nombre_de_classes)
libelles_validation =
keras.utils.to_categorical(libelles_validation,
nombre_de_classes)

# Réseau de neurones convolutifs
# 32 filtres de dimension 3x3 avec une fonction d'activation de
type ReLU
# Le filtre a en entrée l'image de travail
reseauCNN = Sequential()
reseauCNN.add(Conv2D(32, kernel_size=(3, 3),
                    activation='relu',
                    input_shape=imageTravail))

#Une seconde couche de 64 filtres de dimension 3x3
reseauCNN.add(Conv2D(64, (3, 3), activation='relu'))

#Une fonction de pooling
reseauCNN.add(MaxPooling2D(pool_size=(2, 2)))
reseauCNN.add(Dropout(0.25))

#Une mise à plat
reseauCNN.add(Flatten())

#Le réseau de neurones avec en entrée 128 neurones
#une fonction d'activation de type ReLU
reseauCNN.add(Dense(128, activation='relu'))
reseauCNN.add(Dropout(0.5))

#Une dernière couche de type softmax
reseauCNN.add(Dense(nombre_de_classes, activation='softmax'))

#Compilation du modèle
reseauCNN.compile(loss=keras.losses.categorical_crossentropy,
                  optimizer=keras.optimizers.Adadelta(),
                  metrics=['accuracy'])

# Apprentissage avec une phase de validation
# sur les jeux de tests
batch_size = 128
epochs = 10
```

```
reseauCNN.fit(images_apprentissage, libelles_apprentissage,
              batch_size=batch_size,
              epochs=epochs,
              verbose=1,
              validation_data=(images_validation,
                              libelles_validation))

# Sauvegarde du modèle
reseauCNN.save('modele/modele_cas_pratique.h5')

# Evaluation de la précision du modèle
score = reseauCNN.evaluate(images_validation,
                           libelles_validation, verbose=0)
print('Précision sur les données de validation:', score[1])
```

L'important à retenir dans cet apprentissage est la démarche utilisée s'appuyant sur la recherche et la compréhension des données jusqu'à la création d'un modèle d'apprentissage. N'oublions pas qu'Internet est une source non négligeable de modèles et de sources de données permettant de nous aider à résoudre les différents problèmes de régression ou de classification.

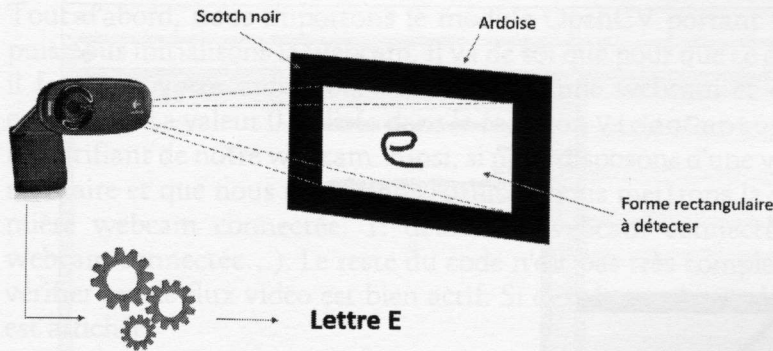
3. La reconnaissance de lettres sur une vidéo

Nous allons à présent découvrir comment nous allons détecter la lettre écrite par un utilisateur puis comment nous allons réaliser sa classification.

3.1 Une ardoise en guise de support

Nous allons utiliser une ardoise au centre de laquelle nous écrirons la lettre. L'ardoise sera ensuite présentée à la caméra. Un script Python sera chargé de rechercher une forme rectangulaire figurant sur cette vidéo et ayant une aire précise. Cette forme sera la zone d'écriture où se trouve la lettre écrite de façon manuscrite et à reconnaître.

Une fois cette forme rectangulaire détectée, une photo sera faite de celle-ci puis analysée afin de prédire la lettre présente à l'intérieur.



Processus de reconnaissance d'image à l'aide d'une webcam

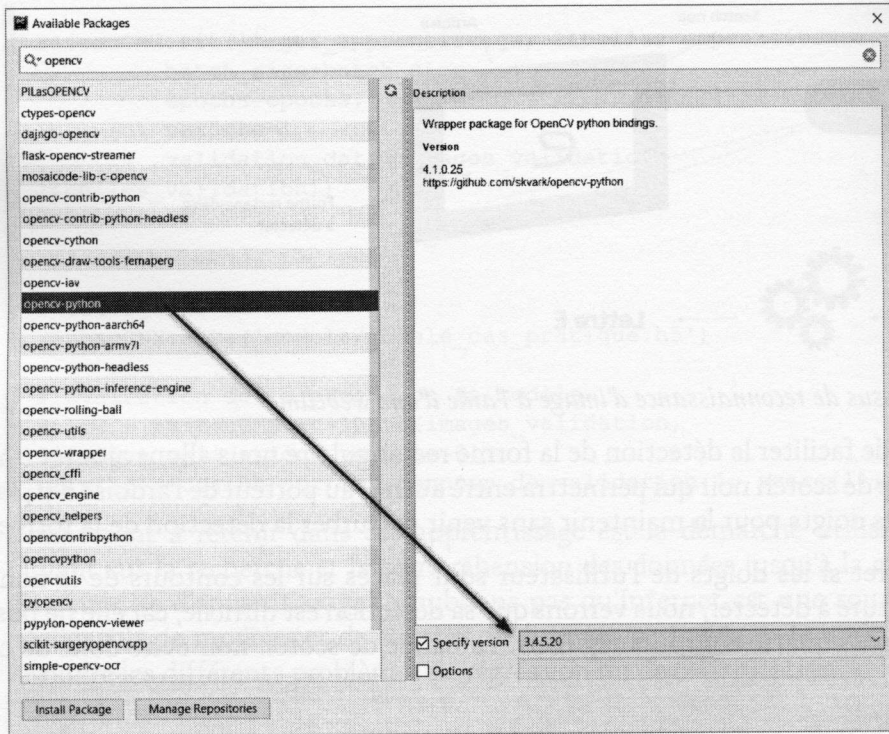
Afin de faciliter la détection de la forme rectangulaire nous allons ajouter une bande de scotch noir qui permettra entre autres, au porteur de l'ardoise d'y placer ses doigts pour la maintenir sans venir perturber la détection de la forme.

En effet si les doigts de l'utilisateur sont placés sur les contours de la zone d'écriture à détecter, nous verrons que sa détection est difficile, car elle ne possède plus quatre contours réguliers. La bande de scotch noir évite donc de ce souci.

3.2 OpenCV, un module de traitement d'images

La détection de forme se fait aisément à l'aide d'un module prévu à cet effet et portant le nom d'OpenCV.

Nous vous invitons donc à ajouter ce module dans sa **version 3.4.5.20** à votre projet.



Installation du module OpenCV dans sa version 3.4.5.20

3.2.1 Utiliser la webcam

Une fois le module OpenCV ajouté, nous allons créer un nouveau fichier nommé `lecture.py` afin d'y ajouter ces quelques lignes :

```
import cv2

print('Initialisation de la webcam')
webcam = cv2.VideoCapture(0)
if webcam.isOpened():
    longueurWebcam = webcam.get(3)
    largeurWebcam = webcam.get(4)
    print('Résolution:' + str(longueurWebcam) + " X " +
          str(largeurWebcam))
else:
    print('ERREUR')
```

Tout d'abord, nous importons le module OpenCV portant le nom de `cv2`, puis nous initialisons la webcam. Il va de soi que pour que ce code fonctionne, il faut que votre ordinateur soit équipé d'une webcam et que celle-ci soit connectée. La valeur 0 utilisée dans la fonction `VideoCapture` correspond à l'identifiant de notre webcam. Ainsi, si nous disposons d'une webcam complémentaire et que nous souhaitons l'utiliser, nous mettrons la valeur 1 (0: première webcam connectée, 1: deuxième webcam connectée, 2: troisième webcam connectée...). Le reste du code n'est pas très complexe, il permet de vérifier que le flux vidéo est bien actif. Si c'est le cas, la résolution de l'image est affichée.

On récupère ensuite les paramètres d'affichage de la webcam pour les afficher à titre d'information.

Nous allons à présent afficher à l'écran l'image capturée :

```
while True:

    # Capture de l'image dans la variable Frame
    # La variable lectureOK est égale à True si la fonction
    read() est opérationnelle
    (lectureOK, frame) = webCam.read()

    # Affichage de l'image capturée par la webcam
    cv2.imshow("IMAGE", frame)

    # Condition de sortie de la boucle While
    # > Touche Escape pour quitter
    key = cv2.waitKey(1)
    if key == 27:
        break

#On libère la webcam et on détruit toutes les fenêtres
webCam.release()
cv2.destroyAllWindows()
```

Lorsque nous exécutons ce script, l'image de la webcam est affichée à l'écran. Pour quitter cet affichage, nous devons appuyer sur la touche **Escape** de notre clavier.

Remarque

Lors du codage des fonctions qui vont suivre, nous vous invitons à faire attention aux indentations (tabulations), car si l'instruction se trouve en dehors de la boucle While (c'est-à-dire au même niveau que le texte "While True"), elle ne sera pas exécutée.

3.2.2 Détecter les formes rectangulaires

Pour détecter les formes rectangulaires, nous allons devoir travailler sur l'image pour qu'elle soit optimisée pour la reconnaissance de formes. Voici les étapes à suivre :

- Utiliser l'espace de couleur HSV (TSV en français pour teinte, saturation et couleur) au lieu de l'espace de couleur RGB (Rouge Vert Bleu) par défaut pour permettre une meilleure détection des objets, car cet espace de couleur est moins sensible à la variation de luminosité.
- Transformer l'image en niveau de gris.
- Activer la mise en évidence des contours à l'aide de l'algorithme de Canny.

```
tsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
contours_canny = cv2.Canny(gris, 30, 200)
```

Toutes ces transformations peuvent être également rendues visibles en créant de nouvelles fenêtres graphiques de visualisation comme le propose le code ci-dessous :

```
while True:

    # Capture de l'image dans la variable Frame
    # La variable lectureOK est égale à True si la fonction
    read() est opérationnelle
    (lectureOK, frame) = camera.read()

    (grabbed, frame) = camera.read()
    tsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    contours_canny = cv2.Canny(gris, 30, 200)
```

```
# Affichage de l'image capturée par la webcam
cv2.imshow("IMAGE", frame)
cv2.imshow("HSV", tsv)
cv2.imshow("GRIS", gris)
cv2.imshow("CANNY", contours_canny)
```

Nous allons ensuite détecter les contours à l'aide de la fonction `findContours` prenant en paramètre une copie de l'image filtrée à l'aide de l'algorithme de canny. Le paramètre `cv2.RETR_EXTERNAL` permet quant à lui d'indiquer à la fonction que nous souhaitons uniquement les contours parents. C'est-à-dire que si un contour "enfant" est à l'intérieur d'un autre contour (le contour "parent"), la fonction ne nous indiquera que le contour parent. Enfin, le paramètre `cv2.CHAIN_APPROX_SIMPLE` permet de ne pas stocker en mémoire tous les points du contour, mais uniquement ses points de départ et de fin.

```
contours = cv2.findContours(contours_canny.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) [1]
```

Nous allons ensuite parcourir ces contours afin de réaliser les étapes suivantes pour chaque contour détecté :

- En calculer le périmètre.
- Évaluer le type de forme (rectangle, triangle...).
- Récupérer les points `x`, `y` ainsi que la longueur `w` et la largeur `h` du contour dans le cas d'un contour de type rectangle.

```
for contour in contours:
    perimetre = cv2.arcLength(contour, True)
    approx = cv2.approxPolyDP(contour, 0.012 *
perimetre, True)
    x, y, w, h = cv2.boundingRect(approx)
```

La méthode `approxPolyDP` permet de déterminer par approximation le type de forme et renvoie le nombre de mouvements (points). Cette approximation est réalisée à l'aide de l'algorithme Douglas-Peucker prenant un paramètre nommé `epsilon`, calculé dans notre cas par la formule `0.012 * perimetre`.

Nous ne nous étendrons pas sur l'explication de cet algorithme, car le traitement de l'image et la détection de contours ne sont pas le sujet de l'ouvrage.

Sachez néanmoins que vous pouvez trouver la documentation complète sur le site de Wikipédia :

https://fr.wikipedia.org/wiki/Algorithme_de_Douglas-Peucker.

Enfin, via la méthode `boundingRect`, nous récupérons les coordonnées de la forme rectangulaire détectée ainsi que sa longueur (w) et sa largeur (h).

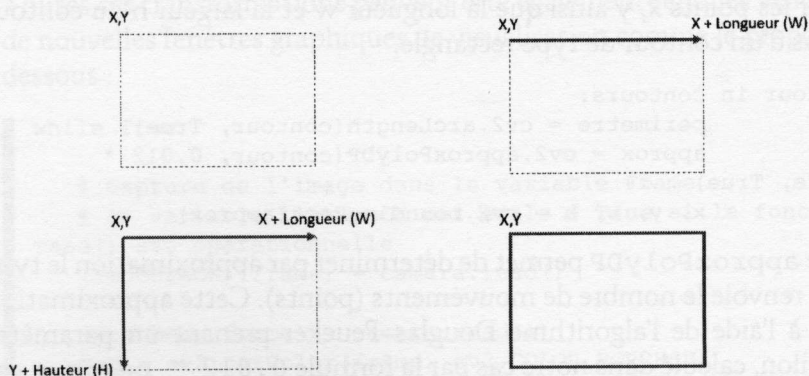
Nous allons à présent encadrer les contours détectés grâce à la fonction `rectangle` du module `OpenCV` :

```
if len(approx) == 4:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)
```

Si le nombre de points renvoyés par l'approximation de forme est égal à 4, nous sommes donc face à un parallélogramme. Nous pouvons donc dessiner un contour rectangulaire sur la vidéo (`frame`), avec pour point de départ les coordonnées X, Y du contour détecté, puis pour point final la position X plus la largeur du contour et le point Y plus la hauteur.

Remarque

Attention, le point ayant pour coordonnées $X=0$ et $Y=0$ se trouve en haut à gauche de la vidéo. Par conséquent, si la valeur Y augmente le point descend sur l'image.



Méthode d'encadrement réalisée par la fonction `rectangle` du module `OpenCV`

3.2.3 Détecter la zone d'écriture

Comme nous l'avons évoqué un peu plus en amont dans ce chapitre, la lettre devant être reconnue se trouve dans une zone d'écriture entourée d'un bord noir.

Nous venons de voir comment notre application était capable de détecter les formes rectangulaires présentes dans le flux vidéo reçu par la webcam, nous allons maintenant lui donner la faculté de détecter uniquement la zone d'écriture. Pour cela, nous allons définir la longueur et la largeur de cette zone et demander à notre script de ne détecter que les formes rectangulaires ayant ces dimensions.

Ajoutons quelques variables en dessous de la ligne d'import d'OpenCV :

```
import cv2

#dimensions de l'ardoise
zoneEcritureLongueurMin = 540
zoneEcritureLongueurMax = 590
zoneEcritureLargeurMin = 300
zoneEcritureLargeurMax = 340
```

À cause des différents mouvements, détecter une forme ayant une dimension précise est très difficile sur une vidéo. C'est pourquoi nous avons préféré utiliser une plage de dimensions pour la longueur et la largeur, caractérisant la zone d'écriture. De ce fait, si une forme rectangulaire ayant une longueur comprise entre 540 et 590 pixels et une largeur comprise en 300 et 340 est détectée, alors nous considérons qu'il s'agit de la zone d'écriture.

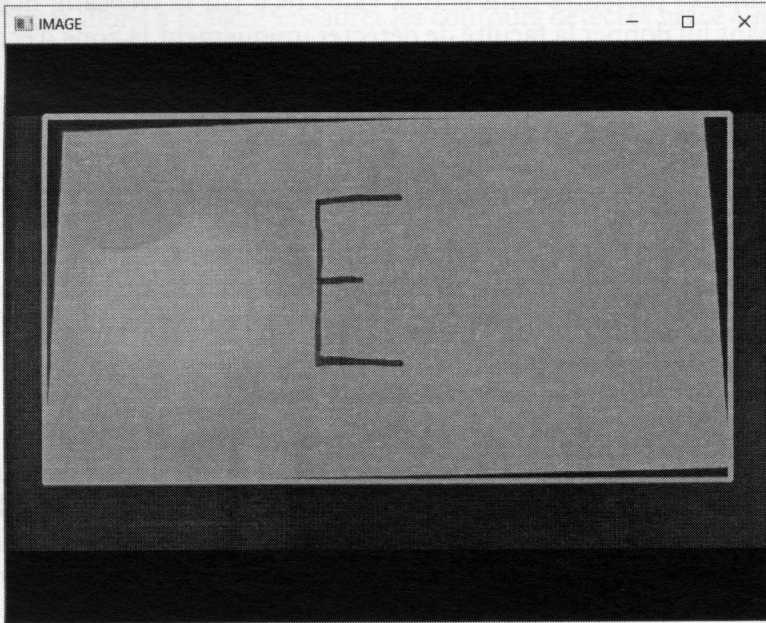
Modifions à présent notre code en y ajoutant ces paramètres :

```
if len(approx) == 4 and h>zoneEcritureLargeurMin and
w>zoneEcritureLongueurMin and h<zoneEcritureLargeurMax and
w<zoneEcritureLongueurMax:
```

Nous sommes à présent en mesure de détecter la zone d'écriture présente sur notre ardoise. Afin de pouvoir facilement tester ce point sur votre ordinateur, nous vous invitons à utiliser le fichier `lecture_lettre_etape_1.py` téléchargeable sur le site de l'éditeur et reprenant le code écrit jusqu'à présent.

Remarque

Les paramètres de longueur et de largeur de la zone d'écriture doivent être adaptés afin de correspondre au mieux à votre zone d'écriture. Ces paramètres varient en fonction de la distance entre votre webcam et l'ardoise. Pour faciliter ce paramétrage, nous avons utilisé un pupitre sur lequel nous avons posé l'ardoise, permettant alors d'avoir une distance constante.



Détection de la zone d'écriture

3.2.4 Détecter et extraire la lettre écrite

Pour détecter la lettre écrite, nous allons rechercher l'existence de contours dans la zone d'écriture. Si des contours existent, alors nous procédons aux étapes suivantes :

- Extraction de la lettre
- Mise en évidence des contours de la lettre pour une meilleure reconnaissance par le réseau de neurones

- Redimensionnement de l'image afin que sa longueur et sa largeur soient de 28 pixels

Voici le code associé que vous pourrez retrouver dans le fichier `lecture_lettre_etape_2.py`.

```
#On encadre la zone d'écriture en fonction des paramètres de
longueur et largeur de l'ardoise
if len(approx) == 4 and h>zoneEcritureLargeurMin and
w>zoneEcritureLongueurMin and h<zoneEcritureLargeurMax and
w<zoneEcritureLongueurMax:

    #Encadrement de la zone d'écriture
    area = cv2.contourArea(contour)
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3);

    # Capture de l'image à partir de la zone d'écriture avec une
    marge intérieure (padding) de 10
    # pixels afin d'isoler uniquement la lettre
    lettre = gris[y + 10:y + h - 10, x + 10:x + w - 10]

    # On détecte les contours de la lettre à l'aide de
    l'algorithme de Canny
    cannyLettre = cv2.Canny(lettre, 30, 200)
    contoursLettre = cv2.findContours(cannyLettre.copy(),
    cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)[1]

    # S'il existe une lettre de dessinée
    if len(contoursLettre) > 5:

        # Création d'un tableau pour le stockage de l'image de la
        lettre (Ne pas oublier d'importer le module numpy)
        captureAlphabetTMP = np.zeros((400, 400), dtype=np.uint8)

        # On détecte le plus grand contour à l'aide du paramètre
        Reverse = True classant les contours du plus grand au plus petit
        et en sélectionnant le premier [0] contour
        cnt = sorted(contoursLettre, key=cv2.contourArea,
        reverse=True)[0]

        # On stocke les coordonnées du rectangle de délimitation de
        la lettre
        xc, yc, wc, hc = cv2.boundingRect(cnt)

        for contourLettre in contoursLettre:
```

```
area = cv2.contourArea(contour)
if area > 1000:

    # On dessine les contours de la lettre pour une
    meilleure lecture (Trait de 10 px)
    cv2.drawContours(captureAlphabetTMP,
contourLettre, -1, (255, 255, 255), 10)

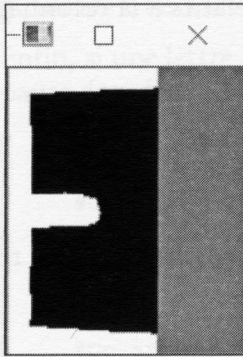
    # On capture la lettre et on stocke les valeurs
    des pixels de la zone capturée dans un tableau
    captureLettre = np.zeros((400, 400),
dtype=np.uint8)
    captureLettre = captureAlphabetTMP[yc:yc + hc,
xc:xc + wc]

    #Des ombres peuvent être capturées dans la zone
    d'écriture provoquant alors des erreurs de
    #reconnaissance. Si une ombre est détectée, une
    des dimensions du tableau de capture est
    #égale à zéro, car aucun contour de lettre n'est
    détecté

    affichageLettreCapturee = True
    if (captureLettre.shape[0] == 0 or
captureLettre.shape[1] == 0):
        print("ERREUR A CAUSE DES OMBRES ! : ")
        affichageLettreCapturee = False

    #Si ce n'est pas une ombre, on affiche la lettre
    capturée à l'écran
    if affichageLettreCapturee:
        cv2.destroyWindow("ContoursLettre");
        cv2.imshow("ContoursLettre", captureLettre)

    # Redimensionnement de l'image
    newImage = cv2.resize(captureLettre, (28, 28))
    newImage = np.array(newImage)
    newImage = newImage.astype('float32') / 255
    newImage.reshape(1, 28, 28, 1)
```



Lettre détectée

3.2.5 Reconnaître la lettre écrite et la faire lire à votre ordinateur

Vient ensuite la reconnaissance de la lettre en faisant appel à notre réseau de neurones que nous avons entraîné au début de ce chapitre et la lecture à voix haute de la lettre reconnue.

Pour cela, nous devons tout d'abord importer les modules nécessaires à la parole et à l'utilisation du réseau de neurones :

```
#Module de parole
import pyttsx3 as pyttsx

#Module Keras permettant l'utilisation de notre réseau de
neurones
from keras.models import load_model
```

Et ajoutons ensuite un module permettant de gérer les processus.

```
#Module de gestion des processus
import threading
```

Ce module va nous permettre de faire une pause de quelques secondes entre chaque prédiction et la lecture de la lettre, évitant ainsi une éventuelle saturation de l'application.

Le code complet est disponible dans le fichier `lecture_lettre_etape_3.py` où de nombreux commentaires ont été insérés afin de vous faciliter la compréhension du code.

Nous allons néanmoins vous présenter les points clés relatifs à la reconnaissance et à la lecture à voix haute de la lettre reconnue.

Tout d'abord, nous chargeons le modèle d'apprentissage :

```
#Chargement du modèle entraîné
cnn_model = load_model('modele_cas_pratique.h5')
kernel = np.ones((5, 5), np.uint8)
```

Puis nous créons un tableau permettant de faire la correspondance entre le numéro prédit et la lettre à lire.

```
#Tableau de lettres avec leur numéro
lettres = {1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'E', 6: 'F', 7:
'G', 8: 'H', 9: 'I', 10: 'J',
          11: 'K', 12: 'L', 13: 'M', 14: 'N', 15: 'O', 16: 'P',
17: 'Q', 18: 'R', 19: 'S', 20: 'T',
          21: 'U', 22: 'V', 23: 'W', 24: 'X', 25: 'Y', 26: 'Z',
27: '-'}

```

Nous avons également créé une fonction permettant de réactiver la lecture à voix haute. Notons l'usage de l'instruction `global` qui permet à une variable d'être modifiée à n'importe quel endroit dans le code.

```
#Par défaut on active la lecture de lettre à voix haute
lectureActivee = True

#Temps d'attente en secondes entre chaque lecture de lettre à
voix haute
dureeDesactivationLectureDeLettre = 5

#fonction de réactivation de la lecture de lettre à voix haute
def activationLecture():
    print('Activation de la lecture de lettres')
    global lectureActivee
    lectureActivee=True
```

La prédiction de la lettre se fait ensuite en indiquant à notre application qu'une lettre a été prédite (`lettrePredite = True`) :

```
# Réalisation de la prédiction
prediction = cnn_model.predict(newImage.reshape(1, 28, 28, 1))[0]
prediction = np.argmax(prediction)

# On indique qu'une lettre a été prédite
```

```
■ lettrePredite = True
```

Enfin, si une lettre a été prédite, on peut alors la lire à voix haute et faire une pause dans le processus de lecture.

```
if lettrePredite:

    #On désactive la lecture de lettre à voix haute
    print('Désactivation de la lecture de lettre ' +
          str(dureeDesactivationLectureDeLettre) + " secondes")
    lectureActivee = False

    #On affiche le numéro de la lettre prédite
    #On ajoute +1, car la première lettre de l'alphabet a pour
    #valeur 0 dans notre modèle de prédiction
    #Alors qu'elle a la valeur 1 dans notre tableau de
    #correspondance
    print("Détection:" + str(lettrePredite))
    print("Prédiction = " + str(prediction))

    #Lecture à voix haute de la lettre prédite
    if (lettrePredite and prediction != 26):
        engine.say('Je lis la lettre ' +
                   str(lettres[int(prediction) + 1]))
        engine.runAndWait()
        lettrePredite = False

    if (lettrePredite and prediction == 26):
        engine.say('Je ne comprends pas la lettre écrite ')
        engine.runAndWait()
        lettrePredite = False

    #Pause du processus de lecture de la lettre puis appel à la
    #fonction activationLecture pour la réactivation de la
    #lecture
    timer = threading.Timer(dureeDesactivationLectureDeLettre,
                             activationLecture)
    timer.start()
```

Le code de ce projet est quelque peu complexe à comprendre. C'est pourquoi nous vous invitons à le parcourir plusieurs fois et à modifier quelques paramètres afin de comprendre leur utilité.

Néanmoins, cet exemple permet de voir comment il est possible à partir d'un modèle d'apprentissage de créer une application pouvant utiliser ce modèle. Cette application offre également un moyen d'échange et de communication entre l'humain et la machine offrant un effet "Whaou" où la machine semble prendre vie et ressembler quelque peu à un humain.

La suite logique de ce projet est de lui donner la faculté de lire les phrases, puis d'en comprendre le sens afin de réaliser une conversation avec nous. Mais nous vous laissons le soin d'écrire à présent cette histoire.

4. Et voilà!

Vous venez de créer votre premier projet permettant d'établir une relation entre l'Homme et la machine!

En écrivant sur une ardoise, l'utilisateur reste distant de la machine et n'interagit pas directement avec elle. La machine, grâce à la vision et à la parole couplées à un algorithme de classification, peut entrer en communication avec lui et donne ainsi l'impression de comprendre l'utilisateur. La machine semble "intelligente"!

Nous vous invitons à proposer à des adultes de tester votre application, puis de la faire tester à des enfants. Lesquels, d'après vous, seront les plus étonnés?

Nous aurions pu conclure cet ouvrage sur ce cas pratique, mais nous avons souhaité ajouter un dernier chapitre vous expliquant comment le premier ChatBot a été conçu. C'est un peu un retour dans le passé que nous avons souhaité réaliser afin de vous faire prendre définitivement conscience que l'intelligence artificielle n'est pas une science nouvelle et que parfois, sous sa forme la plus simple, elle peut encore nous étonner!

Chapitre 14

Hommage au premier ChatBot

1. Introduction

Dans le chapitre précédent, nous avons vu comment il nous était possible de créer une interaction entre l'Homme et la machine à l'aide de moyens sophistiqués que sont la détection de formes et la reconnaissance de lettres à l'aide de réseaux de neurones convolutifs.

Cependant, nous ne pouvions nous quitter sans avoir rendu hommage au premier ChatBot qui rendit possible également cette rencontre entre l'Homme et la machine avec des moyens techniques beaucoup plus simples. C'est également ce ChatBot qui, il y a dix ans environ, et par un hasard de lecture, nous a personnellement donné l'envie de découvrir ce qu'était l'intelligence artificielle. Nous avons donc souhaité partager avec vous, ce qui pour nous, fut le point de départ d'une belle aventure.

2. Eliza

Écrit par Joseph Weizenbaum (Allemagne) entre 1964 et 1966, le programme Eliza est le premier ChatBot jouant le rôle d'un psychiatre où l'utilisateur du programme n'est autre que le patient.

Comparé aux ChatBots actuels, Eliza fait pâle figure. Malgré son fonctionnement simple basé sur de la substitution de mot-clé, de nombreux utilisateurs ont cru qu'ils conversaient avec un vrai docteur !

2.1 Comment fonctionne Eliza ?

Lorsqu'une phrase est proposée à Eliza, l'algorithme recherche dans cette phrase un mot-clé dont il a connaissance et propose alors une réponse en fonction de ce mot-clé. Cette réponse étant dans la majeure partie des cas une question ouverte pour susciter l'utilisateur à poursuivre la conversation.

Par exemple :

– Soit l'affirmation donnée par l'utilisateur : *"Je voudrais une voiture."*

L'algorithme a pour information que lorsqu'il reconnaît le mot-clé "Je voudrais", il doit extraire le texte situé après ce mot-clé et l'utiliser dans une phrase de réponse :

```
(
  r'Je voudrais (.*)',
  (
    "Peux tu m'expliquer pourquoi tu as choisi %1 ?",
    "Pourquoi veux-tu %1 ?",
    "Pourquoi %1 ?",
    "Qui d'autre sait que tu voudrais 1 % ?",
  ),
),
```

La réponse devant être choisie au hasard parmi les quatre possible, les réponses émises par Eliza pouvant alors être :

- Peux-tu m'expliquer pourquoi tu as choisi **une voiture** ?
- Pourquoi veux-tu **une voiture** ?

- Pourquoi **une voiture** ?
- Qui d'autre sait que tu voudrais **une voiture**

Nous vous invitons à remplacer le mot "voiture" par la phrase "savoir comment tu fonctionnes" pour vous rendre compte de la limite de ce fonctionnement :

- Peux-tu m'expliquer pourquoi tu as choisi **savoir comment tu fonctionnes**

La complexité de ce ChatBot ne réside pas dans son codage ou bien encore sa compréhension de fonctionnement, mais dans la création des différents mots-clés et réponses associées, car ceux-ci doivent être en concordance avec le cas d'usage du ChatBot (médecine, agence de voyages...) tout en veillant à une cohérence dans les réponses émises (tournure de phrases...).

2.2 Le code d'Eliza

Voici à présent le code complet de ce ChatBot qui ne manquera pas de vous étonner au vu de sa simplicité! Libre à vous de modifier les clés et les valeurs afin de créer un ChatBot qui pourrait vous servir dans l'un de vos projets.

■ Remarque

Ce projet utilise le module NLTK (Natural Language Tool Kit) devant préalablement être ajouté à votre projet. La fonction Chat permet de faire le lien entre les clés et les valeurs.

```
from __future__ import print_function

#Le sous module reflection permet de faire les liens entre les
clés et les réponses à apporter
from nltk.chat.util import Chat, reflections

Cles_valeurs = (
    (
        r'Bonjour(.*)',
        (
            "Bonjour... je suis contente de discuter avec toi
aujourd'hui",
```

```

        "Salut !! Quoi de neuf aujourd'hui ?",
    ),
),
(
    r'J\'ai besoin (.*)',
    (
        "Pourquoi as-tu besoin %1 ?",
        "Est-ce que ça t'aiderait vraiment %1 ?",
        "Es tu sûr d'avoir besoin %1 ?",
    ),
),
(
    r'Pourquoi ne pas (.*)',
    (
        "Tu crois vraiment que je n'ai pas 1 % ?",
        "Peut-être qu'un jour, je finirai par %1.",
        "Tu veux vraiment que je fasse 1 % ?",
    ),
),
(
    r'Pourquoi je ne peux pas (.*)',
    (
        "Penses tu que tu devrais être capable de %1 ?",
        "Si tu pouvais %1, que ferais-tu ?",
        "Je ne sais pas... pourquoi tu ne peux pas %1 ?",
        "Tu as vraiment essayé ?",
    ),
),
(
    r'Je ne peux pas (.*)',
    (
        "Comment sais tu que tu ne peux pas %1 ?",
        "Tu pourrais peut-être faire 1 % si tu essaies.",
        "Qu'est-ce qu'il te faudrait pour avoir 1 % ?",
    ),
),
(
    r'Je suis (.*)',
    (
        "Es-tu venu me voir parce que tu es %1 ?",
        "Depuis combien de temps êtes-vous %1 ?",
        "Que penses-tu d'être %1 ?",
        "Qu'est-ce que ça te fait d'être %1 ?",
        "Aimes tu être %1 ?",
    ),
),

```

```

        "Pourquoi me dis-tu que tu es à 1 % ?",
        "Pourquoi penses-tu que tu es à 1 % ?",
    ),
),
(
    r'Es-tu (.*)',
    (
        "Pourquoi est-ce important que je sois %1 ?",
        "Tu préférerais que je ne sois pas %1 ?",
        "Tu crois peut-être que je suis %1.",
        "Je suis peut-être %1 -- qu'en penses-tu ?",
    ),
),
(
    r'Quoi (.*)',
    (
        "Pourquoi cette question ?",
        "En quoi une réponse à ça t'aiderait ?",
        "Qu'en penses-tu ?",
    ),
),
(
    r'Comment (.*)',
    (
        "Comment tu crois ?",
        "Tu peux peut-être répondre à ta propre
question."
        "Qu'est-ce que tu demandes vraiment ?",
    ),
),
(
    r'Parce que (.*)',
    (
        "C'est la vraie raison ?",
        "Quelles autres raisons me viennent à l'esprit ?",
        "Cette raison s'applique-t-elle à autre chose ?",
        "Si %1, quoi d'autre doit être vrai ?",
    ),
),
(
    r'(.*) désolé (.*)',
    (
        "Il y a de nombreuses fois où il n'est pas

```

nécessaire de s'excuser."

```

    "Qu'est-ce que tu ressens quand tu t'excuses ?",
    ),
),
(
    r'Je pense que (.*)',
    ("Doute tu de %1 ?",
     "Tu le penses vraiment ?",
     "Mais tu n'es pas sûr de %1 ?"),
),
(
    r'Oui',
    ('Tu me sembles bien sûr.',
     "OK, mais peux-tu développer un peu ?")
),
(
    r'(.*) ordinateur(.*)',
    (
        "Tu parles vraiment de moi ?",
        "Ça te paraît étrange de parler à un ordinateur ?",
        "Comment te sens tu avec les ordinateurs ?",
        "Te sens tu menacé par les ordinateurs ?",
    ),
),
(
    r'Est-ce (.*)',
    (
        "Penses-tu que c'est %1 ?",
        "Peut-être que c'est %1 -- qu'en penses-tu ?",
        "Si c'était %1, que ferais-tu ?",
        "Ça pourrait bien être ce %1.",
    ),
),
(
    r'C'est (.*)',
    (
        "Tu me sembles très certain.",
        "Si je te disais que ce n'est probablement pas %1,
que ressentirais-tu ?",
    ),
),
(

```

```

r'Peux-tu (.*)',
(
    "Qu'est-ce qui te fait croire que je ne peux pas
faire 1 % ?",
    "Si je pouvais %1, alors quoi ?",
    "Pourquoi me demandes-tu si je peux %1 ?",
),
),
(
r'Je peux (.*)',
(
    "Peut-être que tu ne voulais pas de %1.",
    "Veux-tu être capable de %1 ?",
    "Si tu pouvais %1, tu le ferais ?",
),
),
(
r'Vous êtes (.*)',
(
    "Pourquoi penses tu que je suis %1 ?",
    "Est-ce que ça te fait plaisir de penser que je
suis %1 ?",
    "Peut-être voudrais-tu que je sois %1.",
    "Tu parles peut-être vraiment de toi ?",
    "Pourquoi dis tu que je suis %1 ?",
    "Pourquoi penses tu que je suis %1 ?",
    "On parle de toi ou de moi ?",
),
),
(
r'Au revoir',
(
    "Merci de m'avoir parlé.",
    "Au revoir."
),
),
(
r'(.*)',
(
    "S'il te plaît, dis-m'en plus.",
    "Changeons un peu de sujet.... Parle-moi de toi.",
    "Peux tu m'en dire plus à ce sujet ?",
    "Pourquoi dis-tu ça ?",

```

```

        "Je vois.",
        "Très intéressant.",
        "Je vois.  dis m'en plus ?",
    ),
),
)

#Lancement du programme

eliza_chatbot = Chat(Cles_valeurs, reflections)

print("Programme Eliza\n-----")
print('=' * 72)
print("Bonjour.  Comment vas tu?")

eliza_chatbot.converse()

```

Voici à présent un exemple de conversation :

```

Bonjour.  Comment vas-tu?
>Bonjour
Salut !! Quoi de neuf aujourd'hui ?
>Je sais que vous êtes un ordinateur
Te sens-tu menacé par les ordinateurs ?
>non
Je vois.
>J'ai besoin de parler à un humain
Es-tu sûr d'avoir besoin de parler à un humain ?
>oui
OK, mais peux-tu développer un peu ?
>

```

Bien entendu, la discussion atteint vite ses limites, de par le nombre minimal de clés valeurs, mais aussi par les réponses apportées par la machine. Mais cela montre qu'en quelques lignes de code, il nous est possible de remonter dans le passé pour revenir aux sources du premier ChatBot Eliza avec un effet d'étonnement toujours aussi présent !

3. D'autres ChatBots!

Le sous-module Chat du module NLTK (*Natural Language Toolkit*) offre l'implémentation de différents ChatBots tels qu'Eliza, Rude... fonctionnant sur le principe de la réflexion, c'est-à-dire la recherche de mot-clé et la construction d'une réponse à partir de ce mot-clé.

Si vous le souhaitez, vous pouvez saisir ces quelques lignes de code dans un nouveau fichier Python afin de les tester. Attention, le ChatBot Rude risque de quelque peu vous agacer dans ses réponses.

■ Remarque

Les conversations réalisées avec ces ChatBots doivent nécessairement se faire en anglais.

```
from __future__ import print_function

from nltk.chat.util import Chat
from nltk.chat.eliza import eliza_chat
from nltk.chat.iesha import iesha_chat
from nltk.chat.rude import rude_chat
from nltk.chat.suntsu import suntsu_chat
from nltk.chat.zen import zen_chat

bots = [
    (eliza_chat, 'Eliza (Pyschiatre)'),
    (iesha_chat, 'Iesha (Adolescent junky)'),
    (rude_chat, 'Rude (ChatBot abusif)'),
    (suntsu_chat, 'Suntsu (Proverbes chinois)'),
    (zen_chat, 'Zen (Perles de sagesses)'),
]

def chatbots():
    import sys

    print('Quel ChatBot souhaitez vous tester ?')
    botcount = len(bots)
    for i in range(botcount):
        print(' %d: %s' % (i + 1, bots[i][1]))
    while True:
        print('\nChoisissez votre ChatBot 1-%d: ' % botcount,
```

```

end=' ')
    choice = sys.stdin.readline().strip()
    if choice.isdigit() and (int(choice) - 1) in
range(botcount):
        break
    else:
        print(' Erreur: ce ChatBot n\'existe pas')

    chatbot = bots[int(choice) - 1][0]
    chatbot()

chatbots()

```

4. C'est déjà la fin!

Ce cas pratique sonne également la fin de cet ouvrage. Au fur et à mesure de votre lecture et à travers des cas concrets, vous avez pu découvrir les aspects du Machine Learning et du Deep Learning. Nous avons essayé de rendre simples des concepts complexes et espérons y être parvenus.

Comme nous l'avons évoqué dans la présentation de cet ouvrage, celui-ci doit avant tout être **considéré comme un point d'entrée à la compréhension et la mise en œuvre de projets gravitant autour de l'intelligence artificielle**. N'en déplaise aux spécialistes, beaucoup de concepts ont été abordés de façon succincte, mais cette approche vous permet à présent de poursuivre votre aventure en ayant en tête les principes fondamentaux qui vous permettront entre autres de comprendre les différents articles et tutoriels que vous serez amené à lire sur Internet ou dans des ouvrages spécialisés.

Quelle suite à donner? Il faut à présent vous lancer dans la réalisation de projets de Machine Learning et de Deep learning. Pour cela, nous vous invitons à réaliser les challenges présents sur le site Kaggle. Ceux-ci vous permettront d'asseoir vos compétences et d'en acquérir de nouvelles. Sans oublier le fait d'échanger avec de nombreuses personnes à travers le monde œuvrant dans le domaine de l'intelligence artificielle, vous permettant d'obtenir un regard sur vos travaux. Bien entendu, nous sommes conscients qu'à l'issue de cette lecture, trouver les différents paramètres d'un réseau de neurones pour résoudre un problème donné est une tâche ardue. Mais cette tâche est également complexe pour les experts en la matière!

Ne vous découragez pas et apprenez de vos erreurs. Faire du Machine Learning ou du Deep Learning, c'est un peu comme une recette de cuisine que l'on améliore au fil du temps. On ajoute, supprime ou modifie des ingrédients là où vous ajouterez des neurones à votre réseau, augmenterez le nombre de couches cachées ou choisirez une plus grande valeur pour le paramètre C de l'algorithme SVM. N'ayez pas peur, lancez-vous!

Notez également que nous restons disponibles afin d'échanger avec vous sur cet ouvrage et répondrons aux questions que vous vous posez. Pour conclure, nous vous souhaitons de prendre plaisir à ouvrir les champs du possible de l'intelligence artificielle!

B

Bagging, 117

Biais, 244

Boîtes à moutaches, 24

Boosting, 117

C

Centrage de réduction, 111

Centroïde, 123

Chaîne de caractères, 42

ChatBot, 413, 415, 421

A

- accuracy, 211
- Algorithme K-Mean, 258
- Algorithme SVM, 214, 247
- Algorithmes, 185, 195
 - apprentissage supervisé, 110
 - DBSCAN (Density Based Spatial Clustering of Application with Noise), 125
 - de prédiction, 195
 - non supervisés, 123
 - régression linéaire, 181
- Apprentissage, 107, 176, 243, 245, 288, 297, 308
 - non supervisé, 110, 123
 - supervisé, 110, 133, 251
- Arbre de décision, 115, 184
 - régression, 184

B

- Bagging, 117
- Biais, 284
- Boîtes à moustaches, 94
- Boosting, 117

C

- Centrage de réduction, 111
- Centroïde, 123
- Chaîne de caractères, 42
- ChatBot, 413, 415, 421

Classification, 110, 193, 246, 260, 389

d'images, 341

de prédiction, 195

de texte, 223

erreurs, 261

Cluster, 123, 258

Clustering, 251

ConvNet, 342

Convolution, 346, 353

Convolutional Neural Network, 342

Courbe de Gauss, 100

D

DataFrame, 74

DBSCAN (Density Based Spatial Clustering of Application with Noise), 125

Déductions, 85

Deep Learning, 28, 311

Descente de gradient, 112, 274

Détection, 341

valeurs extrêmes, 94

Dispersion, 86

de données, 76

écart type, 87

Étendue, 86

interquartile, 90

quartiles, 90

Distribution gaussienne, 99, 127

Données, 99, 133, 204, 237, 239

acquisition, 239

analyser et explorer, 203

d'apprentissage, 140

- de tests, 140, 209, 306, 363
- de type numérique, 147
- extrêmes, 204, 216
- linéairement séparables, 271
- nombre, 164
- préparation, 141, 239
- préparer et nettoyer, 196
- qualitatives, 72
- quantitatives, 72
- visualiser, 256

Données d'apprentissage, 196, 294, 306, 342, 361

- acquisition, 252

E

Eliza, 414

EMNIST, 385

Erreur

- globale, 280

- linéaire, 280

- locale, 280

- moyenne quadratique MSE, 280

Excel, 74

F

Feature, 148, 150

- de catégorisation, 146

Fonction MSE, 284

Fonctions d'activation, 274

Fonctions de perte (Loss function), 279

Forêts aléatoires, 116

G

Gaussian Mixture Model (GMM), 263

Gradient boosting, 117

H

HeatMap, 173

Hyperparamètre, 295

I

Intelligence artificielle
peur, 30

J

Jeux d'apprentissage, 177, 243, 390

Jeux d'observations, 167, 194

Jeux de tests, 177, 243, 390

K

Kaggle, 356

Keras, 357

K-Means (K-Moyennes)
algorithme, 255

KNN (K-Nearest Neighbours), 120

L

Lemmatisation, 232

Listes, 45

 accès à un élément, 45

 connaître le nombre d'éléments, 46

 suppression d'un élément, 45

Loi normale, 99

M

Machine à vecteurs de support (SVM), 118

Machine Learning, 28, 131, 134, 163, 270

matmul, 308

Matplotlib, 137, 387

MaxPooling, 351

Mean-shift, 124

Mélange gaussien (Gaussian Mixture Models (GMM)), 127, 263

Minimum global, 283

Minimum local, 283

Mise à l'échelle, 111

Mise à plat (Flatten), 354

Modèle d'apprentissage, 186, 187

 sauvegarde, 186

Modèle de prédiction, 209

modulo, 83

N

Naive Bayes, 121, 224, 229, 249
Neurone, 267, 284, 306, 313, 327
 artificiel, 304
 biais, 284
 couches, 315
 formel, 294
 réseaux, 313
NLTK (Natural Language Tool Kit), 415
Normalisation, 111
 du texte, 230
Noyau, 119
numpy, 73

O

Observations, 107, 153, 154, 177
One-Hot, 392
OpenCV, 399
Ou exclusif (XOR), 314

P

Pandas, 73
Perceptron, 267, 286
 initialisation, 287
Pipeline d'apprentissage, 244
Placeholder, 307
Point de convergence, 300
Pooling, 351
Pooling Stochastique, 351

- Précision de l'apprentissage, 211
- Prédiction, 243
 - tests, 302, 310
- Probabilités, 104
- Problème à résoudre, 140
- Python, 35, 36, 39, 73, 105, 134, 238
 - boucles, 47
 - dictionnaires, 46
 - installation, 36
 - manipulation de chaînes de caractères, 42
 - opérations de base, 40
 - premier script, 55
 - structures conditionnelles, 47
 - utilisation des listes, 45
- Python-Mnist, 385

R

- Random forest, 185
 - algorithme, 185
- random.uniform, 329
- random_normal, 337
- Reconnaissance, 398
- Régression, 110, 184, 389
 - linéaire multiple, 111, 182
 - linéaire univariée, 110
 - logistique, 114
 - polynomiale, 113
- Réseau
 - avec une seule couche de convolution, 363
- Réseaux de neurones, 392
 - convolutifs, 342, 389
 - multicouches, 313

S

Rétropropagation, 274
de l'erreur, 279

Scaling, 111, 390

Scikit-Learn, 209

Seaborn, 137

sigmoid, 308

Sous-apprentissage, 187

Sous-apprentissage (underfitting), 186

Statistiques, 69, 164

écart type, 165

features, 72, 76

maximale, 165

médiane, 80

mesure de tendance centrale, 75

minimale, 165

moyenne (mean), 165

moyenne arithmétique, 78

observations, 72, 76

quartiles, 166

types de données, 72

valeurs maximales, 76

valeurs minimales, 76

Stops words, 231

Surapprentissage, 187

Surapprentissage (overfitting), 186

SVM (Machine à vecteurs de supports), 249

algorithme, 247

T

- Tendance centrale, 85
- TensorFlow, 304, 305, 321
- Tests
 - de prédictions, 302, 310
- TF-IDF, 244
- Théorème de Bayes, 121, 225
- Threading, 409
- to_categorical, 361, 392
- Traitement
 - valeurs extrêmes, 96
- Traitement automatique du langage naturel (TALN), 224
- Tuples, 46
- Types d'apprentissage, 29

V

- Valeurs
 - extrêmes, 204
 - maximales, 165
 - minimales, 165
- Variable, 40, 307
- Visualisation graphique, 97

Intelligence artificielle vulgarisée

Le Machine Learning et le Deep Learning par la pratique

L'**intelligence artificielle** est aujourd'hui incontournable. Ce livre a pour objectif de présenter de façon vulgarisée les concepts du **Machine Learning** et du **Deep Learning** pour les mettre en application dans des projets basés sur de l'intelligence artificielle, en mettant de côté autant que possible les formules mathématiques et statistiques. Il s'adresse avant tout aux développeurs mais intéressera également toute personne novice en la matière.

Avec une démarche progressive, chaque notion étudiée dans ce livre est illustrée par des **cas pratiques écrits en langage Python**. Des connaissances dans ce langage sont ainsi un plus.

Après une **introduction à l'intelligence artificielle** et l'**identification des craintes** qu'elle suscite, l'auteur propose quelques rappels sur les fondamentaux du **langage Python** ainsi qu'une révision de certaines **notions statistiques** pour appréhender au mieux les **algorithmes du Machine Learning**. Le lecteur peut ensuite mettre en pratique certains de ces algorithmes et découvrir comment donner la faculté à sa machine de **prédire des valeurs** et de **réaliser des classifications**.

Vient ensuite l'étude de l'**apprentissage non supervisé** et de l'usage des **réseaux de neurones** qui permet de surcroît au lecteur de découvrir comment les **neurosciences** ont eu un impact sur l'intelligence artificielle. Le livre se termine avec la **réalisation de cas pratiques** : un premier mêlant réseau de neurones et parole et un second relatif au premier chatbot.

Des éléments complémentaires sont en téléchargement sur le site www.editions-eni.fr.

Après un début de carrière en tant que développeur et expert technique, **Aurélien VANNIEUWENHUYZE** est aujourd'hui fondateur et dirigeant des entreprises QSTOM-IT et Junior Makers Place. C'est dans cette seconde structure qu'il propose des activités de codage et des activités scientifiques pour enfants et adolescents, et notamment des formations sur l'intelligence artificielle avec une pédagogie accessible à tous, reprise dans ce livre. Titulaire du certificat d'études supérieures de pilotage de la transformation digitale obtenu à HEC et auteur d'ouvrages sur la technologie Flex ou le Framework Agile Scrum, il aide également les entreprises à réussir leur transformation numérique, tant sur le plan de l'agilité que par la sensibilisation et la mise en place de l'intelligence artificielle.



sur www.editions-eni.fr :
→ Le code source des différents cas pratiques.



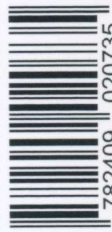
www.editions-eni.fr

Pour plus d'informations :



29,90 €

ISBN : 978-2-409-02073-5



9 782409 020735