



كلية العلوم والتكنولوجيات. طنجة



Machine Learning

Cycle d'Ingénieur LSI

Pr. Lotfi ELAACHAK
Département Génie Informatique

2023 — 2024

Table des matières

1	Introduction	4
1.1	C'est quoi l'apprentissage machine ?	4
1.1.1	Intelligence artificielle	4
1.1.2	Pourquoi l'intelligence artificielle ?	5
1.1.3	Histoire de IA	5
1.1.4	Turing Test	5
1.1.5	Chinese room	5
1.1.6	Fonction d'approximation	6
1.1.7	Stratégies d'IA Cas d'utilisation : Tic Tac Toe	6
1.1.8	Apprentissage machine	7
1.1.9	Applications	9
1.2	Science des données (data sciences)	9
1.2.1	Les données	10
1.3	L'algorithme d'apprentissage	11
1.3.1	C'est quoi l'apprentissage supervisé	11
1.3.2	C'est quoi l'apprentissage non supervisé	11
1.3.3	C'est quoi l'apprentissage par renforcement	12
2	Avant de commencer	14
2.1	Préparation et Analyse de données	14
2.1.1	Les valeurs manquantes	14
2.1.2	Les valeurs dupliquées	15
2.1.3	Les valeurs aberrantes "Outliers"	15
2.1.4	Encodage des valeurs catégoriques vers numérique	16
2.1.5	Analyse de données	16
2.2	La visualisation des données	17
2.3	Sélection des caractéristiques "Features Selection "	18
2.3.1	Sélection univariée «Univariate Selection, US»	19
2.3.2	Élimination récursive de caractéristiques "Recursive Feature Elimination, RFE "	19
2.3.3	Analyse des composants principaux "Principal Component Analysis, PCA"	20
2.3.4	Importance des caractéristiques "Feature Importance , FI"	21
2.4	Normaliser et Standardiser des caractéristiques	22
2.4.1	MinMaxScaler (Méthode de Scaling)	23
2.4.2	RobustScaler (Méthode de Scaling Statistique)	23
2.4.3	StandardScaler (méthode pour Standardiser)	23
2.4.4	Normalizer (méthode pour normaliser)	24
2.5	La mesure des performances	24

2.5.1	Mesures de performance d'un model de classification	24
2.5.2	ROC et AUC	29
2.5.3	Matrice de confusion	32
2.5.4	Rapport de classification	33
2.5.5	Mesures de performance d 'un model de régression	33
2.6	Underfitting and Overfitting	35
2.7	Données asymétries « Umbalanced Data »	36
2.7.1	Intuition pour les données déséquilibrées	36
2.7.2	Évaluer les modèles de classification asymétriques	37
2.7.3	Sous-échantillonnage de la classe majoritaire	37
2.7.4	Sur-échantillonnage de la classe minoritaire	38
2.7.5	Algorithmes sensibles aux coûts "Cost-Sensitive Algorithms"	39
3	Apprentissage supervisé	41
3.1	La régression linière simple	41
3.1.1	Comment fonctionne la régression linière simple	41
3.1.2	Implémentation avec python	44
3.1.3	Implémentation avec scikit-learn	45
3.2	La régression linière multiple	47
3.2.1	Le gradient descente	47
3.2.2	Comment fonctionne la régression linière multiple	49
3.2.3	Implémentation avec python	50
3.2.4	Implémentation avec scikit-learn	51
3.3	La régression linière polynomial	52
3.3.1	Comparaison de la régression polynomiale et linéaire simple	53
3.3.2	Jouer avec le degré polynomial	54
3.4	Régression logistique	55
3.4.1	La solution est une régression logistique	56
3.4.2	Algorithme d'apprentissage	57
3.4.3	Comment la régression logistique fonctionne	58
3.4.4	Implémentation avec scikit-learn	60
3.5	KNN	61
3.5.1	Comment algorithme KNN fonctionne ?	62
3.5.2	Implémentation avec scikit-learn classification binaire	63
3.5.3	Implémentation avec scikit-learn classification multi classes	64
3.5.4	Comment choisir la valeur optimale de K ?	65
3.6	Naive bayes	67
3.6.1	Modèle discriminatif vs Modèle génératif	67
3.6.2	Pourquoi le naive bays ??	68
3.6.3	Probabilités conditionnelles	68
3.6.4	Classification multi-classes avec Naive Bayes	70
3.6.5	Exemple Python	71
3.6.6	Exemple sickit-learn	72

3.7	Support Vector Machine	73
3.7.1	SVM linéaire	74
3.7.2	SVM Non-linéaire	84
3.7.3	Formulation du problème de programmation quadratique avec la cvxopt	87
3.8	Arbre de decision	92
3.9	Ensemble Learning	100
3.9.1	Techniques Simple	100
3.9.2	Techniques Avancées	101
3.9.3	Code source	106
4	Apprentissage non supervisé	111
4.1	Kmeans	111
4.1.1	Comment Kmeans marche	111
4.1.2	Choisir le bon K	113
4.1.3	Exemple sickit-learn	113
4.2	Expectation Maximization EM	115
4.2.1	1D - Mixture Model	115
4.2.2	Fonctionnement de l'algorithme EM	115
4.2.3	Exemple EM : 1D	117
4.2.4	Convergence de EM	118
4.2.5	Exemple Python	118
4.3	Self Organized Map / Kohennen Map	119
4.3.1	Fonctionnement des SOMs	120
4.3.2	SOM : L'algorithme d'apprentissage en détail	122
4.3.3	Code source Python	127
4.4	PCA	128
4.4.1	Exemple numérique de PCA : de $n = 2$ vers $n = 1$	128
4.4.2	Implémentation python	131
4.4.3	Implémentation sickit-learn	132
4.4.4	Exemple complet	132
4.5	TSNE	134
4.5.1	t-SNE vs PCA	134
4.5.2	Fonctionnement du t-SNE	135
4.5.3	Algorithme	135
4.5.4	Python code source	139
5	Apprentissage par renforcement	141
5.1	Q-Learning	142
5.1.1	Exemple Q-Learning pas a pas	143
5.1.2	Implémentation python	147

Introduction

1.1 C'est quoi l'apprentissage machine ?

1.1.1 Intelligence artificielle

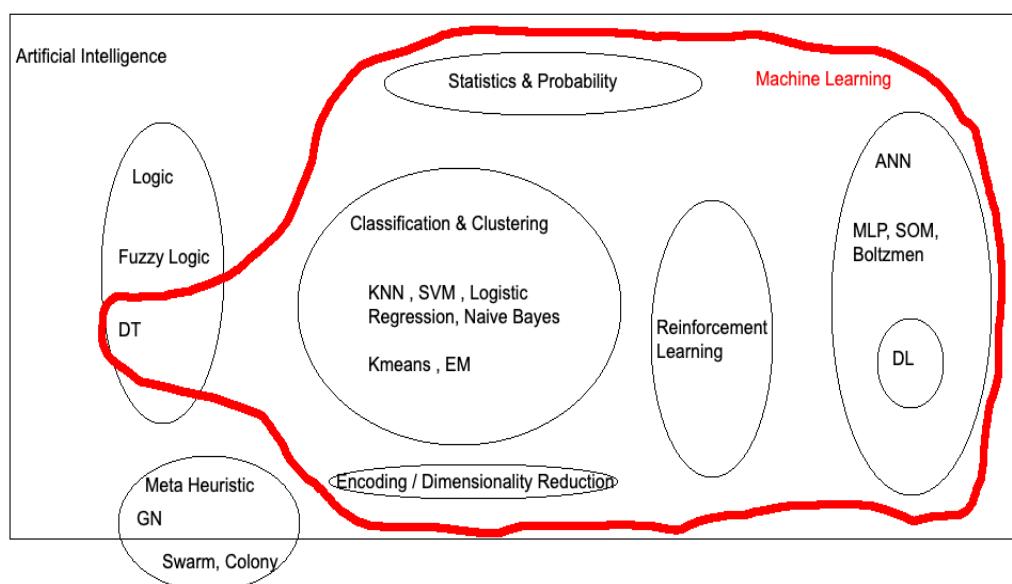
L'intelligence artificielle (IA, ou AI en anglais pour Artificial Intelligence) consiste à mettre en œuvre un certain nombre de techniques visant à permettre aux machines d'imiter une forme d'intelligence réelle. L'IA se retrouve implantée dans un nombre grandissant de domaines d'application.

La notion d'intelligence artificielle a vu le jour dans les années 1950 grâce au mathématicien Alan Turing. Dans son livre Computing Machinery and Intelligence, ce dernier soulève la question d'apporter aux machines une forme d'intelligence.

De Google à Microsoft en passant par Apple, IBM ou Facebook, toutes les firmes internationales dans le monde de l'informatique planchent aujourd'hui sur les problématiques de l'intelligence artificielle en tentant de l'appliquer à quelques domaines précis. Chacun a ainsi mis en place des réseaux de neurones artificiels constitués de serveurs et permettant de traiter de lourds calculs au sein de gigantesques bases de données.

L'IA, quelques exemples d'usage :

- La vision artificielle, par exemple, permet à la machine de déterminer précisément le contenu d'une image pour ensuite la classer automatiquement selon l'objet, la couleur ou le visage repéré.
- Les algorithmes IA sont en mesure d'optimiser leurs calculs au fur et à mesure qu'ils effectuent des traitements. C'est ainsi que les filtres antispam deviennent de plus en plus efficaces au fur et à mesure que l'utilisateur identifie un message indésirable ou au contraire traite les faux-positifs.
- La reconnaissance vocale par exemple les assistants virtuels capables de transcrire les propos formulés en langage naturel puis de traiter les requêtes soit en répondant directement via une synthèse vocale, soit avec une traduction instantanée ou encore en effectuant une requête relative à la commande.



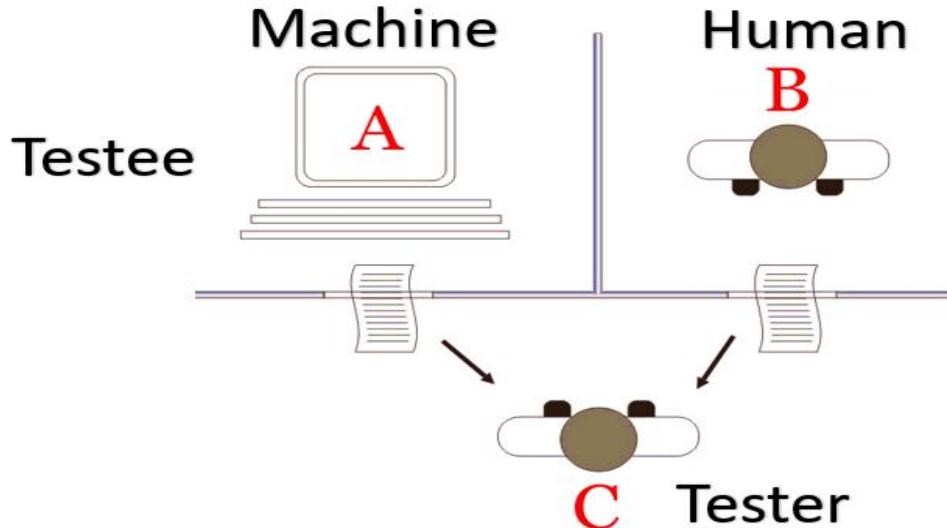
1.1.2 Pourquoi l'intelligence artificielle ?

- Progrès récents dans le domaine des algorithmes et de la structure des données
- Disponibilité des données
- Infrastructure et puissance de calcul (GPUs, TPUs, HPC, Cloud, etc)

1.1.3 Histoire de IA

- 1901-1933 : Principal component analysis (K. Pearson, H. Hotelling). — 1943 : McCulloch & Pitts : Boolean circuit model of brain.
- 1950 : Turing's “Computing Machinery and Intelligence”
- 1955 : Support Vector Machine (Vapnik).
- 1955-2001 : Random Forest (L. Breiman).
- 1958 : Perceptron (F. Rosenblatt).
- 1965 : Robinson's complete algorithm for logical reasoning — 1965 : Fuzzy Sets (L. Zadeh).
- 1970—90 : Knowledge-based approaches (Expert systems) — 1982 : Self Organized Map (T. kohonen).
- 1986 : back propagation (E. Rumelhart , G. Hinton).
- 1986-1993 : Decision Tree : ID3, C4.5 (J.R Quinlin).
- 1997 : Long short-term memory (LSTM) (Hochreiter).
- 2001 : Convolutionnal Neural Network (Lecunn & Benejio). — 2017 : Transformer (Ashish Vaswani et. al).
- 2022 : The Forward-Forward Algorithm (G. Hinton).

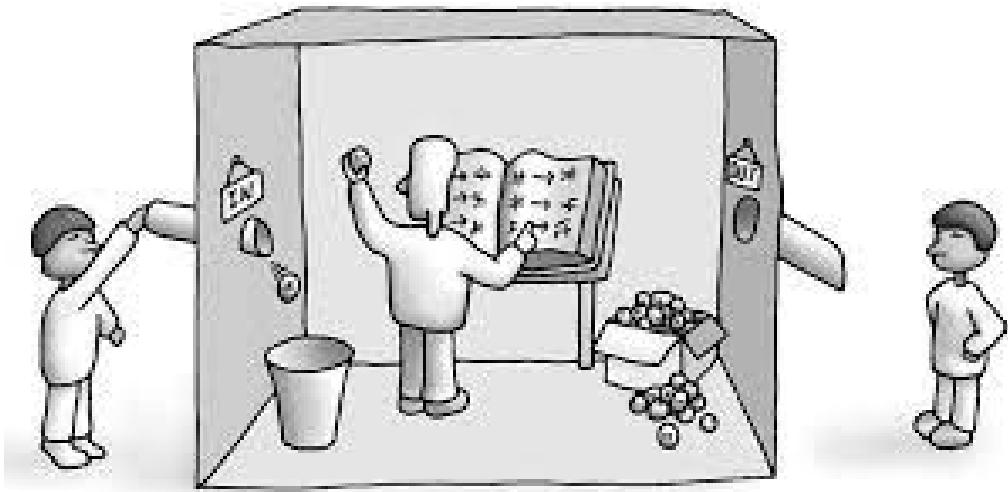
1.1.4 Turring Test



Le temps mis par le juge pour deviner que la pièce A est l'ordinateur ?

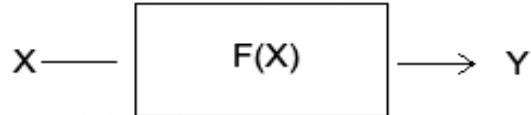
1.1.5 Chinese room

John Searle's 1980 paper.



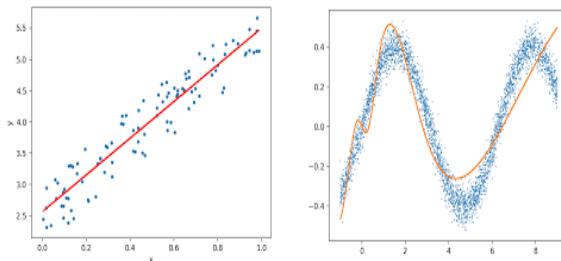
- Le robot peut-il comprendre le chinois ?
- La pièce peut-elle comprendre le chinois ?

1.1.6 Fonction d'approximation



avec $F(X)$ est une fonction inconnue.

- IA : opérer avec intelligence sur les données pour extraire les relations entre les entrées et les sorties.



1.1.7 Stratégies d'IA Cas d'utilisation : Tic Tac Toe

Version 1 : Table de recherche / Approche statique / Stupide

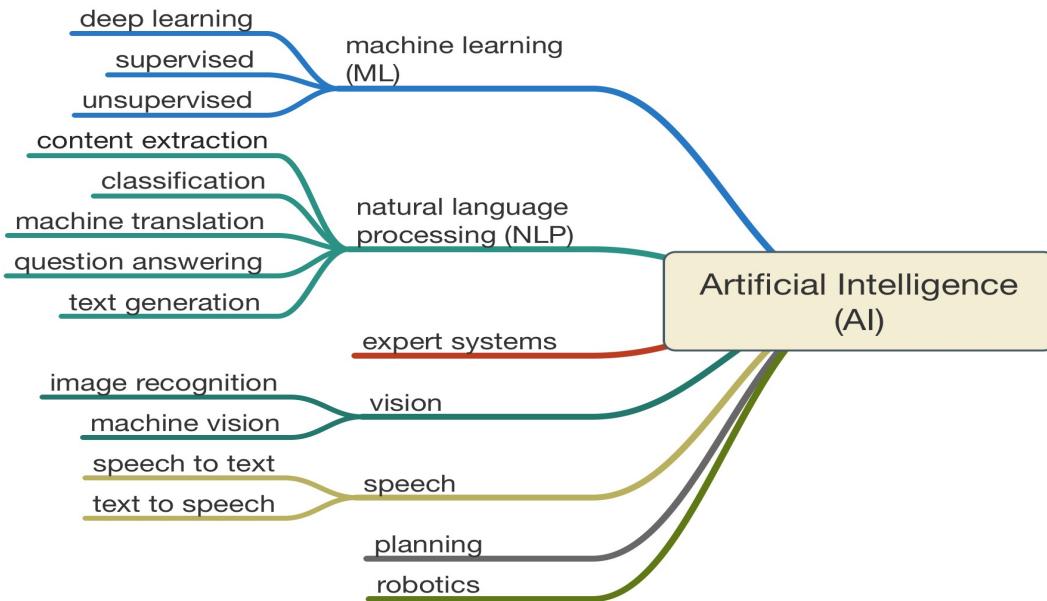
- Réponse pour chaque situation "codée en dur".

Version 2 : Stratégie

- A- Tenter de placer deux marques à la suite.
- B- si l'adversaire a deux marques à la suite, faire la 3ème place.

Version 3 : Approche dynamique / intelligente

- Représenter l'état du jeu (comprendre le jeu) : Position actuelle du plateau ? Prochains coups légaux.
- Utiliser une fonction d'évaluation :
 - Evaluer le prochain coup (quelle est la probabilité de gagner).
 - Regarder en avant si possible, pour estimer les mouvements de l'adversaire.

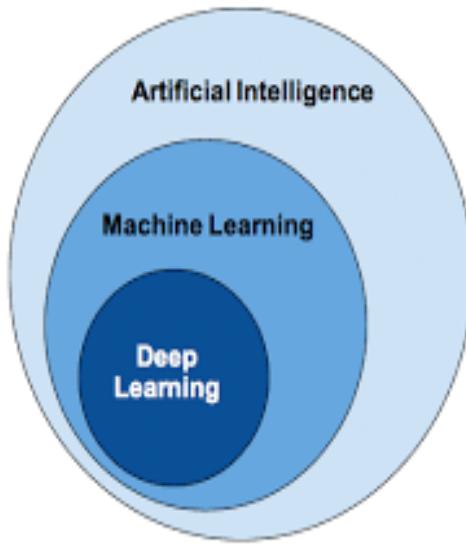


1.1.8 Apprentissage machine

L'apprentissage machine est un sous domaine de l'intelligence artificielle. D'une manière plus précise l'apprentissage machine implique la construction des modèles mathématiques pour bien comprendre les données. L'apprentissage entre dans la mêlée lorsque nous donnons à ces modèles des paramètres accordables qui peuvent être adaptés aux données observées ; De cette façon, le programme peut être considéré comme un outil d'apprentissage à partir des données.

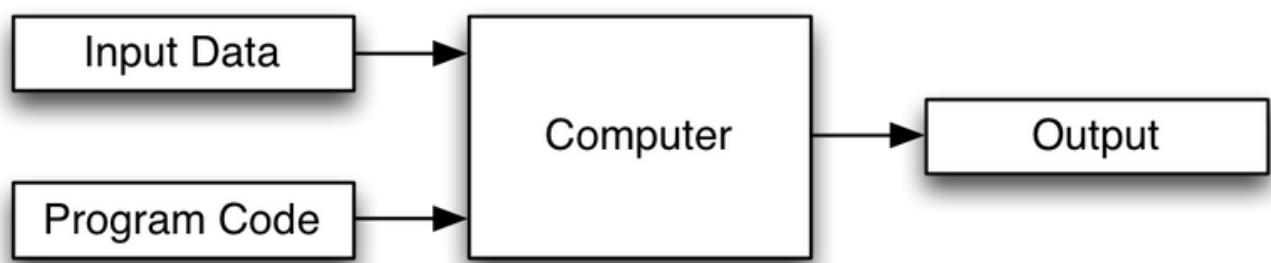
Une fois ces modèles sont implémentés par des données, ils peuvent être utilisés pour prédire et comprendre les aspects des nouvelles données.

Le machine Learning constitue une manière de modéliser des phénomènes, dans le but de prendre des décisions stratégiques.

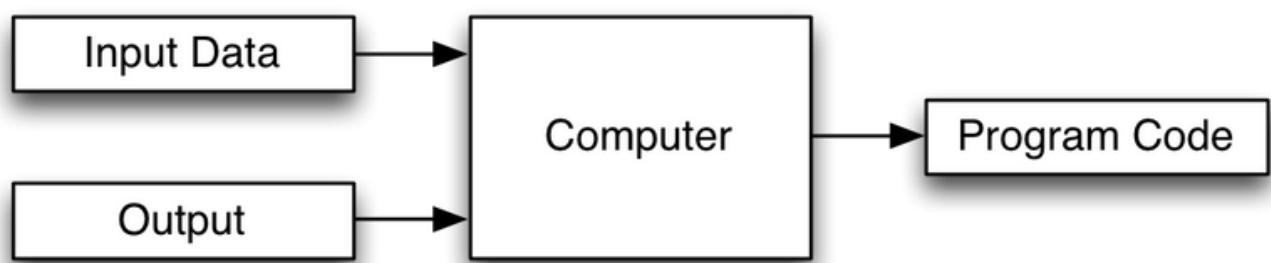


Développement logiciel vs programmation d'apprentissage machine :

Traditional Software Development



Machine Learning Programming

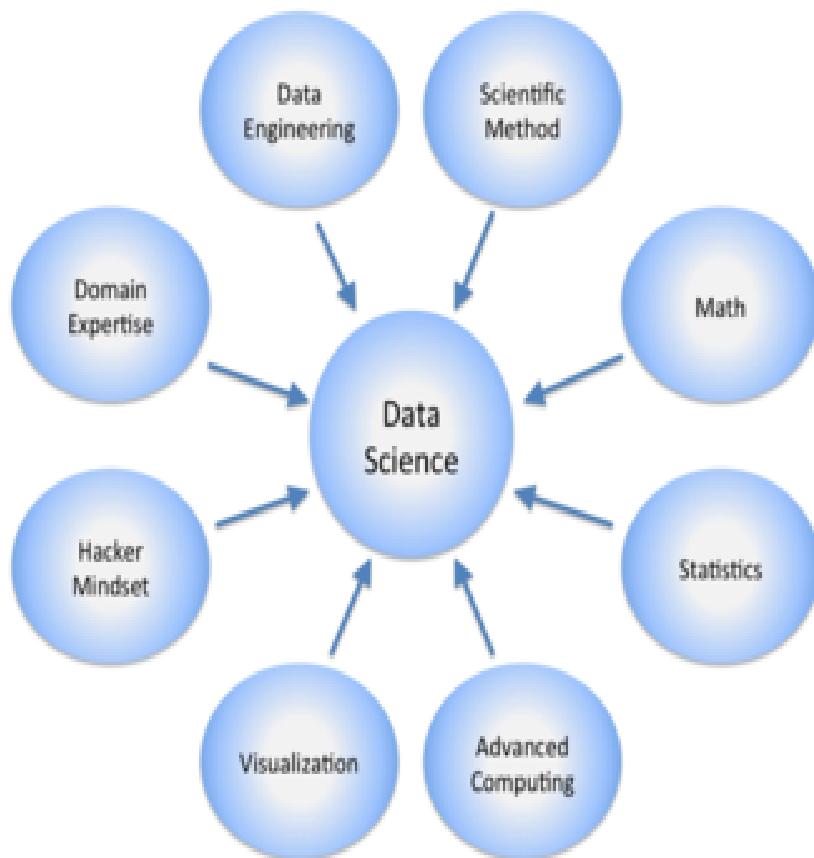


1.1.9 Applications

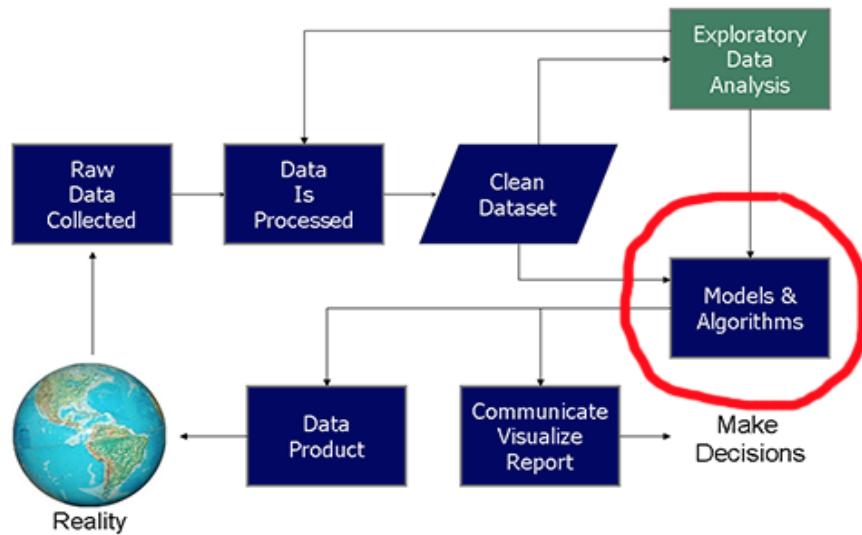


1.2 Science des données (data sciences)

La science des données est l'extraction de connaissance d'ensembles de données. Elle emploie des techniques et des théories tirées de plusieurs autres domaines plus larges des mathématiques, la statistique principalement, la théorie de l'information et la technologie de l'information, notamment le traitement de signal, des modèles probabilistes, l'apprentissage automatique, l'apprentissage statistique, la programmation informatique, l'ingénierie de données, la reconnaissance de formes et l'apprentissage, la visualisation, l'analytique prophétique, la modélisation d'incertitude, le stockage de données, la compression de données et le calcul à haute performance. Les méthodes qui s'adaptent aux données de masse sont particulièrement intéressantes dans la science des données, bien que la discipline ne soit généralement pas considérée comme limitée à ces données.



Les étapes de travail d'un data scientiste sont les suivantes :



1.2.1 Les données

Les données constituent littéralement le nerf de la guerre de machine Learning / data science, on peut citer trois type de source de donnée qui peuvent alimenter les différents algorithmes de machine Learning : base de données, les données brutes, IoT (internet des objets).

Les bases de données :

Les bases de données constituent la source principale de récupération de données par les algorithmes de la machine Learning. Ces bases de données peuvent comprendre différents types d'information, une bonne partie généralement spécifique à l'activité de l'entreprise.

Les données brutes :

Les données brutes, souvent plus complexes et nécessitant des prétraitements spécifiques pour les rendre manipulables par les algorithmes.

Le texte, rédigé en langage naturel (humain). Cela comporte tous les types de texte auxquels on peut penser naturellement (articles, livres, messages, etc.), mais aussi d'autres types de textes tels que du code HTML ou encore des séquences d'ADN.

Les images sont aussi une des sources de captation de l'environnement souvent utile sur des problématiques d'entreprise. Beaucoup d'entreprises ont des banques d'images à traiter pour les classer par type ou autre.

IoT :

Les objets connectés sont une autre source de données brutes, qui récupèrent un grand nombre de données grâce à leurs capteurs.



1.3 L'algorithme d'apprentissage

L'algorithme d'apprentissage constitue la méthode avec laquelle le modèle statistique va se paramétrier à partir des données d'exemple. Il existe de nombreux algorithmes différents !

Quelques exemples d'algorithmes de machine Learning :

- La régression linéaire
- K-nn
- Les Support Vector Machine (SVM)
- Les réseaux de neurones
- Les random forests
- etc.

1.3.1 C'est quoi l'apprentissage supervisé

Si les classes sont prédéterminées et les exemples connus, le système apprend à classer selon un modèle de classement ; on parle alors d'apprentissage supervisé.

Le processus se passe en deux phases. Lors de la première phase (dite d'apprentissage), il s'agit de déterminer un modèle des données étiquetées.

La seconde phase (dite de test) consiste à prédire l'étiquette d'une nouvelle donnée, connaissant le modèle préalablement appris. Parfois il est préférable d'associer une donnée non pas à une classe unique, mais une probabilité d'appartenance à chacune des classes prédéterminées.

Classification

Dans l'apprentissage automatique et dans les statistiques, la classification est une approche d'apprentissage supervisé dans laquelle le programme informatique apprend à partir des données saisies, puis utilise cet apprentissage pour classer une nouvelle observation. Cet ensemble de données peut simplement être bi-classe (comme identifier si la personne est un homme ou une femme ou que le courrier est du spam ou non) ou il peut aussi être multi-classe. Voici quelques exemples de problèmes de classification : reconnaissance de la parole, reconnaissance de l'écriture manuscrite, identification bio métrique, classification de documents, etc.

Régression

Les modèles de régression sont utilisés pour prédire une valeur continue. Prédire les prix d'une maison en fonction de ses caractéristiques telles que la taille, le prix, etc. est l'un des exemples les plus courants de la régression. C'est une technique supervisée.

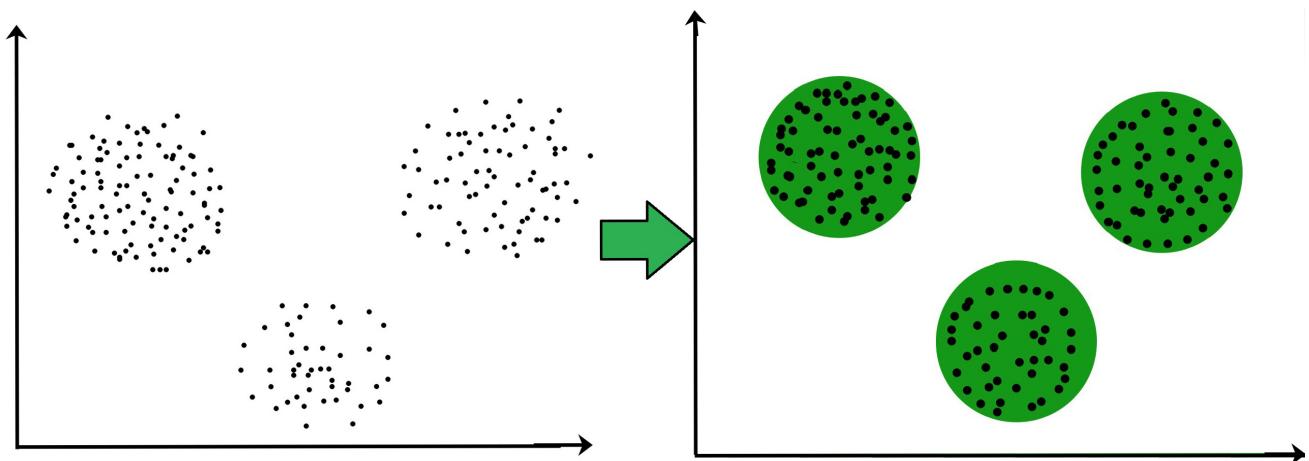
1.3.2 C'est quoi l'apprentissage non supervisé

Quand le système ou l'opérateur ne disposent que d'exemples, mais non d'étiquettes, et que le nombre de classes et leur nature n'ont pas été prédéterminés, on parle d'apprentissage non supervisé.

L'algorithme doit découvrir par lui-même la structure plus ou moins cachée des données. Le système doit cibler les données selon leurs attributs disponibles, pour les classer en groupe homogènes d'exemples. La similarité est généralement calculée selon une fonction de distance entre paires d'exemples. C'est ensuite à l'opérateur d'associer ou déduire du sens pour chaque groupe et pour les motifs (patterns en anglais) d'apparition de groupes, ou de groupes de groupes, dans leur « espace ».

Clustering

Il consiste à diviser la population ou les points de données en un certain nombre de groupes, de sorte que les points de données du même groupe soient plus similaires aux autres points de données du même groupe et différents des points de données des autres groupes. C'est fondamentalement une collection d'objets basée sur la similitude et la dissimilarité entre eux.



Réduction de la dimensionnalité

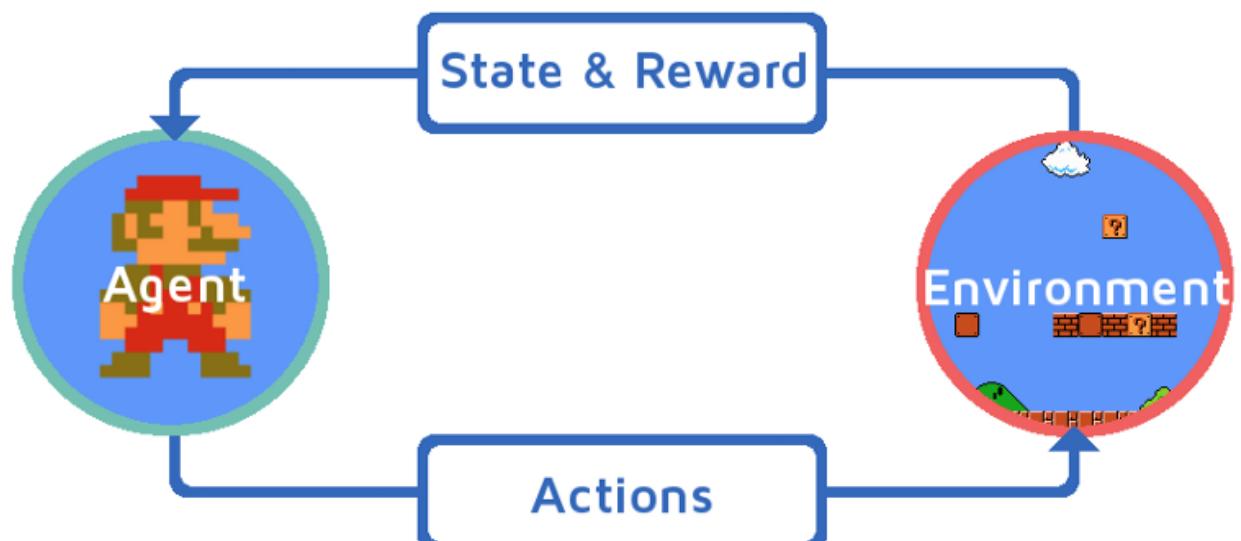
En statistique, apprentissage automatique et théorie de l'information, la réduction de dimensionnalité ou réduction de dimension est le processus de réduction du nombre de variables aléatoires considérées [1] en obtenant un ensemble de variables principales. Les approches peuvent être divisées en sélection et extraction de caractéristiques.

La réduction de la dimensionnalité ressemble beaucoup à la compression. Il s'agit d'essayer de réduire la complexité des données tout en conservant autant que possible la structure pertinente.

1.3.3 C'est quoi l'apprentissage par renforcement

L'apprentissage par renforcement est un domaine de l'apprentissage automatique. Il s'agit de prendre des mesures appropriées pour maximiser les bénéfices dans une situation donnée. Il est utilisé par divers logiciels et machines pour trouver le meilleur comportement possible ou le meilleur chemin à suivre dans une situation donnée. L'apprentissage par renforcement diffère de l'apprentissage supervisé en ce sens que, dans l'apprentissage supervisé, les données de formation ont la clé de réponse ; le modèle est donc formé avec la bonne réponse alors que dans l'apprentissage par renforcement, il n'y a pas de réponse, mais l'agent de renforcement décide quoi faire. pour effectuer la tâche donnée. En l'absence de jeu de données de formation, il est tenu d'apprendre de son expérience.

L'algorithme apprend un comportement étant donné une observation. L'action de l'algorithme sur l'environnement produit une valeur de retour qui guide l'algorithme d'apprentissage.



Avant de commencer

2.1 Préparation et Analyse de données

2.1.1 Les valeurs manquantes

Découverte des valeurs manquantes

Le concept de valeurs manquantes est important à comprendre afin de gérer avec succès les données. Si les valeurs manquantes ne sont pas gérées correctement par le chercheur, il peut finir par tirer une inférence inexacte sur les données. En raison d'une mauvaise manipulation, le résultat obtenu par le chercheur sera différent de ceux où les valeurs manquantes sont présentes.

Les valeurs manquantes

```
import numpy as np
import pandas as pd
url = "/Users/macbookair/Documents/cours/cours apprentissage machine/datasets/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']# les noms des colonnes
df = pd.read_csv(url, names=names) # charger le fichier csv
df.isnull().sum()
#Autre méthode
[features for features in df.columns if df[features].isnull().sum()>0]
```

Gérer les valeurs manquantes

Supprimer les valeurs nulles :

Cette méthode de prétraitement des données est couramment utilisée pour gérer les valeurs nulles. Ici, nous supprimons soit une ligne particulière si elle a une valeur nulle pour une fonctionnalité particulière et une colonne particulière si elle a plus de 75% de valeurs manquantes.

Supprimer les valeurs manquantes

```
df.dropna(inplace=True)
df.isnull().sum()
```

Remplacer la valeur manquante :

Cette stratégie peut être appliquée à une fonctionnalité qui a des données numériques comme la colonne année ou la colonne objectif de l'équipe à domicile. Nous pouvons calculer la moyenne, la médiane ou le mode de l'entité et la remplacer par les valeurs manquantes.

Remplacer les valeurs manquantes

```

df['year'] = df['year'].replace(np.NaN , df['year'].mean())

# ou bien par le mode

df['year'] = df['year'].fillna(df['year'].mode()[0])

```

2.1.2 Les valeurs dupliquées

Une partie importante de l'analyse des données consiste à analyser les valeurs en double et à les supprimer. La méthode Pandas drop_duplicates() aide à supprimer les doublons du bloc de données.

Remplacer les valeurs manquantes

```
df.drop_duplicates(keep=False,inplace=True)
```

2.1.3 Les valeurs abirantes "Outliers"

En statistique, une donnée aberrante (en anglais outlier) est une valeur ou une observation qui est « distante » des autres observations effectuées sur le même phénomène.

Exemple statisitque

Soit le DataSet suivant :

[173, 179, 180, 184, 187, 190, 195, 201, 204, 215, 272]

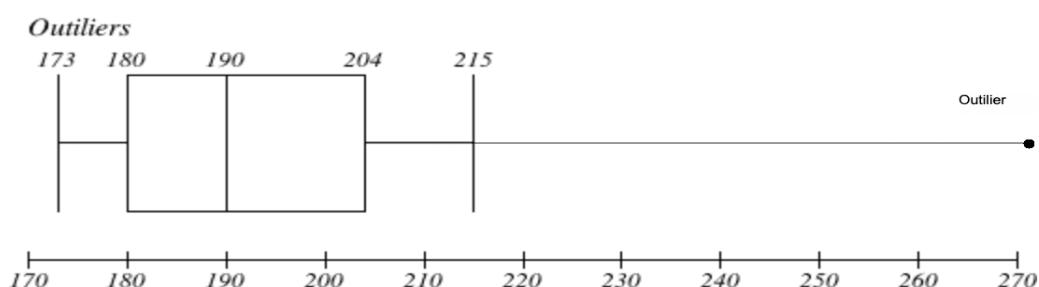
Median = 190.

$Q_1 = 180$.

$Q_3 = 204$.

IR : Inter Quartil Range : $Q_3 - Q_1 = 204 - 180 = 24$.

Clôture inférieure : $Q_1 - 1.5 * IR = 180 - 1.5 * 24 = 144$. Clôture superieur : $Q_3 + 1.5 * IR = 204 + 1.5 * 24 = 240$.



BoxPlot / Outliers

```

import numpy as np
import pandas as pd

```

```

url = "/Users/macbookair/Documents/cours/cours apprentissage machine/datasets/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']# les noms des colonnes
df = pd.read_csv(url, names=names) # charger le fichier csv
df[['sw', 'sh', 'pw', 'ph']].plot(kind='box', subplots=True, layout=(2,2), sharex=False,
sharey=False)

```

2.1.4 Encodage des valeurs catégoriques vers numérique

Encodage

```

names = ['CustomerID', 'Genre', 'Age', 'Annual_Income', 'Spending_Score']
df = pd.read_csv("/Volumes/D/cours/cours_apprentissage_machine/datasets/train.csv")
df.info()
df['Gender'].unique()
df['Gender'] = df['Gender'].map({'M':1 , 'F': 0})
df['Age'].unique()
df['Age'] = df['Age'].map({'0-17':1 , '18-25': 2, '26-35': 3 , '36-45': 4, '46-50': 5 , '51-55':6 , '55+':7})

df['Gender'] = pd.get_dummies(df['Gender'] , drop_first=1)

# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['City_Category']= label_encoder.fit_transform(df['City_Category'])

df['City_Category'].unique()

df['Age']= label_encoder.fit_transform(df['Age'])

df['Age'].unique()

```

2.1.5 Analyse de données

L'analyse des données consiste à évaluer des données à l'aide d'outils analytiques et statistiques permettant de découvrir des informations utiles et de faciliter la prise de décision. Il existe plusieurs méthodes d'analyse de données, notamment l'exploration de données « data mining », l'analyse de texte, l'informatique décisionnelle et la visualisation de données.

Pour analyser les données, il existe plusieurs outils et langages, dans notre cas on va utiliser python comme langage et pandas comme outil d ‘analyse de données vue le succès de ces deux. Il existe des dizaines de tutoriels intéressants sur l’outil pandas et l’analyse visuelle des données. D’une manière général, Pandas est une bibliothèque Python qui fournit des moyens étendus d’analyse de données. Les scientifiques travaillent souvent avec des données stockées dans des formats de table tels que .csv, .tsv ou .xlsx. Pandas est très pratique pour charger, traiter et analyser ces données tabulaires à l’aide de requêtes de type SQL. En conjonction avec Matplotlib et Seaborn, Pandas offre un large éventail d’opportunités d’analyse visuelle des données tabulaires. Les principales structures de données dans les pandas sont implémentées avec les classes Series et DataFrame. Le premier est un tableau unidimensionnel indexé d’un type de données fixe. Ce dernier est une structure de données bidimensionnelle - une table - dans laquelle chaque colonne contient des données du même type.

Voici quelques méthodes principales pour analyser les données sous Pandas :

Pandas Analyse de données

```

import numpy as np
import pandas as pd
url = "/Users/macbookair/Documents/cours/cours apprentissage machine/datasets/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']# les noms des colonnes
df = pd.read_csv(url, names=names) # charger le fichier csv
print(df.head())
print(df.shape) # nombres de lignes et colonnes
print(df.columns) # les noms des colonnes
print(df.info()) # information sur le DataFarmre
df.describe() # le resume statistique sur les donnees
df.sort_values(by='Total day charge', ascending=False).head() # trier les donnees selon le critere
df.loc[0:5, 'State':'Area code'] # afficher les lignes 0 - 5 des colonnes
df.iloc[0:5, 0:3] # afficher 5 lignes et 3 colonnes selon des index
print(df.groupby('class').size()) # la distrubution de la colonne class
df['class'].unique()

```

Pandas possède d’autres méthodes avancées comme le Groupby, le Table Pivot, etc.

2.2 La visualisation des données

La visualisation de données est la représentation graphique des données. Il s’agit de produire des images qui communiquent des relations entre les données représentées aux utilisateurs. Cette communication est réalisée grâce à l’utilisation d’un mappage systématique entre les différents graphes et les valeurs de données lors de la création de la visualisation.

Ce mappage établit comment les valeurs de données seront représentées visuellement, déterminant comment et dans quelle mesure une propriété d’une marque graphique, telle que la taille ou la couleur, changera pour refléter le changement de la valeur d’une variable.

Python possède plusieurs bibliothèques pour dessiner des graphes depuis des données, parmi ces bibliothèques on peut trouver : matplotlib, Seaborn.

On peut afficher pas mal de types de graphes/diagrammes :

- Étant donné que les variables d'entrée sont numériques, on peut créer des diagrammes à boîtes de chacun.
- Créer un histogramme de chaque variable d'entrée pour avoir une idée de la distribution, pour les données qui ont apparition gaussien, Ceci est utile à noter car nous pouvons utiliser des algorithmes pouvant exploiter cette hypothèse.
- Graphes multivariées, permet de voir les interactions entre les variables

Data Vizualisation

```
import pandas
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
# diagramme à boîtes
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
# histogrammes
df.hist()
plt.show()
# scatter plot matrix
scatter_matrix(dataset)
plt.show()
```

2.3 Sélection des caractéristiques "Features Selection "

Les caractéristiques de données qu'on utilise pour former des modèles d'apprentissage machine ont une énorme influence sur les performances. c'est-à-dire ses caractéristiques non pertinentes ou partiellement pertinentes peuvent avoir un impact négatif sur les performances des modèles déjà établis.

La sélection des caractéristiques est un processus dans lequel on sélectionne automatiquement les fonctionnalités des données qui contribuent le plus à la variable de prédiction ou à la sortie.

La présence des caractéristiques non pertinentes dans des données peut réduire la précision de nombreux modèles, en particulier les algorithmes linéaires tels que la régression linéaire et logistique.

La sélection des caractéristiques avant la modélisation des données présente trois avantages :

- Réduit les Overfitting : Moins de données redondantes signifie moins d'opportunités pour prendre des décisions basées sur le bruit.
- Améliore la précision : moins de données trompeuses signifie que la précision de la modélisation est améliorée.
- Réduit le temps de l'entraînement : moins de données signifie que les algorithmes s'entraînent plus rapidement.

Pour le langage Python la bibliothèque la plus réputée pour l'apprentissage machine est sklearn, cette bibliothèque plusieurs méthode pour la sélection les caractéristiques de données , Dans cette section on va répertorier 4 recettes de sélection de fonctionnalités pour l'apprentissage automatique en Python «Sélection univariée, Élimination récursive de caractéristiques, Analyse des composants principaux et Importance des caractéristiques ».

2.3.1 Sélection univariée «Univariate Selection, US»

Les tests statistiques peuvent être utilisés pour sélectionner les entités qui ont la relation la plus forte avec la variable de sortie.

La bibliothèque scikit-learn fournit la classe SelectKBest qui peut être utilisée avec une suite de différents tests statistiques pour sélectionner un nombre spécifique de fonctionnalités.

L'exemple ci-dessous utilise le test statistique du chi carré (chi 2) pour les caractéristiques non négatives afin de sélectionner 4 des meilleures caractéristiques à partir du jeu de données sur l'apparition du diabète chez les Indiens Pima.

Univariate Selection

```
# Feature Extraction with Univariate Statistical Tests (Chi-squared for classification)

import pandas
import numpy
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# load data
url = "/Users/macbookair/Documents/cours/cours apprentissage machine/datasets/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, Y)
# summarize scores
numpy.set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
# summarize selected features
print(features[0:5,:])
```

Résultat :

```
111.52 1411.887 17.605 53.108 2175.565 127.669 5.393 181.304
148. 0. 33.6 50. 85. 0. 26.6 31. 183. 0. 23.3 32.
89. 94. 28.1 21. 137. 168. 43.1 33.
```

On peut voir les scores pour chaque attribut et les 4 attributs choisis (ceux avec les scores les plus élevés) : plas, test, masse et âge.

2.3.2 Élimination récursive de caractéristiques "Recursive Feature Elimination, RFE "

L'élimination des caractéristiques récursives fonctionne en supprimant récursivement les attributs et en construisant un modèle sur les attributs restants.

Il utilise la précision du modèle pour identifier les attributs (et la combinaison d'attributs) qui contribuent le plus à la prédiction de l'attribut cible.

On peut en savoir plus sur la classe RFE dans la documentation scikit-learn.

L'exemple ci-dessous utilise RFE avec l'algorithme de régression logistique pour sélectionner les 3 principales caractéristiques. Le choix de l'algorithme importe peu tant qu'il est habile et cohérent.

RFE

```
# Feature Extraction with RFE
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load data
url = "/Users/macbookair/Documents/cours/cours apprentissage machine/datasets/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = LogisticRegression()
rfe = RFE(model, 3)
fit = rfe.fit(X, Y)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

Résultat :

```
Num Features : 3
Selected Features : [ True False False False False True True False]
Feature Ranking : [1 2 3 5 6 1 1 4]
```

On peut voir que RFE a choisi les 3 principales fonctionnalités telles que preg, mass et pedi. Celles-ci sont marquées true dans le tableau support_ et marquées d'un choix «1» dans le tableau ranking_.

2.3.3 Analyse des composants principaux "Principal Component Analysis, PCA"

: L'analyse en composantes principales (ou ACP) utilise l'algèbre linéaire pour transformer l'ensemble de données en une forme compressée. Généralement, cela s'appelle une technique de réduction des données. Une propriété de PCA est que vous pouvez choisir le nombre de dimensions ou de composant principal dans le résultat transformé.

Dans l'exemple ci-dessous, nous utilisons PCA et sélectionnons 3 composants principaux. En savoir plus sur la classe PCA dans scikit-learn en consultant l'API de PCA.

PCA

```
# Feature Extraction with PCA
```

```

import numpy
from pandas import read_csv
from sklearn.decomposition import PCA
# load data
url = "/Users/macbookair/Documents/cours/cours apprentissage machine/datasets/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(X)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)

```

Résultat :

```

Explained Variance : 0.88854663 0.06159078 0.02579012
-2.02176587e-03 9.78115765e-02 1.60930503e-02 6.07566861e-02
9.93110844e-01 1.40108085e-02 5.37167919e-04 -3.56474430e-03
-2.26488861e-02 -9.72210040e-01 -1.41909330e-01 5.78614699e-02
9.46266913e-02 -4.69729766e-02 -8.16804621e-04 -1.40168181e-01
-2.24649003e-02 1.43428710e-01 -9.22467192e-01 -3.07013055e-01
2.09773019e-02 -1.32444542e-01 -6.39983017e-04 -1.25454310e-01

```

On peut voir que l'ensemble de données transformé (3 composantes principales) présente peu de ressemblance avec les données source.

2.3.4 Importance des caractéristiques "Feature Importance , FI"

Des arbres de décision comme Random Forest et Extra Trees peuvent être utilisés pour estimer l'importance des entités.

Dans l'exemple ci-dessous, nous construisons un classifieur ExtraTreesClassifier pour l'ensemble de données sur l'apparition du diabète chez les Indiens Pima. On peut en savoir plus sur la classe ExtraTreesClassifier dans l'API scikit-learn.

Extra Trees Classifier

```

# Feature Importance with Extra Trees Classifier
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier
# load data
url = "/Users/macbookair/Documents/cours/cours apprentissage machine/datasets/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values

```

```

X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = ExtraTreesClassifier()
model.fit(X, Y)
print(model.feature_importances_)

```

Résultat :

0.10588253 0.2589818 0.10649364 0.06728506 0.07050147 0.13298477 0.11858309 0.13928766

Vous pouvez voir que nous obtenons un score d'importance pour chaque attribut, plus le score est grand, plus l'attribut est important. Les scores suggèrent l'importance des plas, de l'âge et de la masse.

2.4 Normaliser et Standardiser des caractéristiques

De nombreux algorithmes d'apprentissage machine fonctionnent mieux ou convergent plus rapidement lorsque les caractéristiques sont à une échelle relativement similaire et / ou presque distribuées normalement. Des exemples de telles familles d'algorithmes incluent :

- régression linéaire et logistique
- voisins les plus proches
- les réseaux de neurones
- machines à vecteurs de support avec fonctions de noyau à polarisation radiale
- analyse en composantes principales
- analyse discriminante linéaire

D'une façons général :

- Échelle/Scale signifie généralement changer la plage des valeurs. La forme de la distribution ne change pas. La plage est souvent définie entre 0 et 1.
- Standariser signifie généralement changer les valeurs pour que l'écart type de distribution par rapport à la moyenne soit égal à un. Il produit quelque chose de très proche d'une distribution normale. La mise à l'échelle est souvent implicite.
- Normaliser peut signifier l'une des choses ci-dessus (et plus!).

La normalisation est une technique souvent appliquée dans le cadre de la préparation de données pour l'apprentissage automatique. Le but de la normalisation est de changer les valeurs des colonnes numériques du jeu de données en une échelle commune.

Par exemple, considérons un ensemble de données contenant deux caractéristiques, âge (x_1) et revenu (x_2). Où l'âge varie de 0 à 100 ans et le revenu de 0 à 20 000 et plus. Le revenu est environ 1 000 fois supérieur à l'âge et se situe entre 0 et 20 000. Donc, ces deux caractéristiques sont dans des gammes très différentes. Lorsque nous effectuons une analyse plus poussée, comme une régression linéaire multivariée, par exemple, le revenu attribué influencera davantage le résultat intrinsèquement du fait de sa valeur plus grande. Mais cela ne signifie pas nécessairement qu'il est plus important en tant que prédicteur.

MinMaxScaler, RobustScaler, StandardScaler et Normalizer sont des méthodes scikit-learn permettant de pré-traiter des données pour un apprentissage automatique. La méthode dont on a besoin, le cas échéant, dépend du type de modèle et de valeurs de la caractéristique.

2.4.1 MinMaxScaler (Méthode de Scaling)

Pour chaque valeur d'une entité, MinMaxScaler soustrait la valeur minimale de l'entité, puis se divise par la plage. La plage est la différence entre le maximum original et le minimum original. La plage par défaut pour la fonctionnalité renvoyée par MinMaxScaler est comprise entre 0 et 1.

MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler = MinMaxScaler()
print(scaler.fit(data))
MinMaxScaler(copy=True, feature_range=(0, 1))
print(scaler.data_max_)
# result [ 1. 18.]
print(scaler.transform(data))
# result [[0. 0.] [0.25 0.25] [0.5 0.5] [1. 1.]]
print(scaler.transform([[2, 2]]))
# result [[1.5 0.]]
```

2.4.2 RobustScaler (Méthode de Scaling Statistique)

RobustScaler transforme le vecteur de caractéristiques en soustrayant la médiane, puis en le divisant par la plage interquartile (valeur de 75% - valeur de 25%). Il faut Noter que la plage de chaque fonctionnalité après l'application de RobustScaler est plus large que pour MinMaxScaler. Donc on utilise RobustScaler si on souhaite réduire les effets des valeurs aberrantes par rapport à MinMaxScaler.

RobustScaler

```
from sklearn.preprocessing import RobustScaler
X = [[ 1., -2., 2.],
      [-2., 1., 3.],
      [ 4., 1., -2.]]
transformer = RobustScaler().fit(X)
print(transformer)
RobustScaler(copy=True, quantile_range=(25.0, 75.0), with_centering=True,
with_scaling=True)
print(transformer.transform(X))
# result array([[ 0. , -2. , 0. ], [-1. , 0. , 0.4], [ 1. , 0. , -1.6]])
```

2.4.3 StandardScaler (méthode pour Standardiser)

StandardScaler normalise une caractéristique en soustrayant la moyenne puis en la mettant à l'échelle de la variance. La variance unitaire signifie la division de toutes les valeurs par l'écart type. StandardScaler ne répond pas à la définition stricte de l'échelle que j'ai introduite précédemment. StandardScaler fait la moyenne de la distribution 0. Environ 68% des valeurs se situeront entre -1 et 1.

StandardScaler

```

from sklearn.preprocessing import StandardScaler
data = [[0, 0], [0, 0], [1, 1], [1, 1]]
scaler = StandardScaler()
print(scaler.fit(data))
print(scaler.mean_)
# result [0.5 0.5]
print(scaler.transform(data))
# result [-1. -1.] [-1. -1.] [ 1.  1.] [ 1.  1.])
print(scaler.transform([[2, 2]]))
# result [[3. 3.]]

```

2.4.4 Normalizer (méthode pour normaliser)

Normalizer fonctionne sur les lignes et non sur les colonnes. Normalizer transforme toutes les entités en valeurs comprises entre -1 et 1 (ce texte a été mis à jour en juillet 2019).

Normalizer

```

from sklearn.preprocessing import Normalizer
X = [[4, 1, 2, 2],
[1, 3, 9, 3],
[5, 7, 5, 1]]
transformer = Normalizer().fit(X) # fit does nothing.
print(transformer)
Normalizer(copy=True, norm='l2')
print(transformer.transform(X))
#result array([[0.8, 0.2, 0.4, 0.4], [0.1, 0.3, 0.9, 0.3], [0.5, 0.7, 0.5, 0.1]])

```

2.5 La mesure des performances

Mesurer les performances fait partie intégrante du travail de modélisation. Il faut en général déterminer une mesure principale, souvent spécifique à la tâche à accomplir. Le choix de cette métrique est très important !

Donc après avoir suivi l'ingénierie des caractéristiques, la sélection et, bien sûr, l'implémentation d'un modèle et obtenu des résultats sous forme de probabilité ou de classe, l'étape suivante consiste à déterminer l'efficacité du modèle basé sur une métrique utilisant des jeux de données de test. Dans le cas d'un apprentissage supervisé vu qu'il y a deux grandes familles « classification et régression », donc il faut savoir mesurer la performance des deux.

2.5.1 Mesures de performance d'un model de classification

Differentes mesures de performance sont utilisées pour évaluer différents algorithmes d'apprentissage machine pour les problèmes de classification. Nous pouvons utiliser des métriques de performance de classification telles que Log-Loss, Accuracy, AUC (Area under Curve), Recall, etc.

la matrice de la confusion

La matrice Confusion est l'une des métriques les plus intuitives et les plus simples utilisées pour rechercher l'exactitude et la précision du modèle. Il est utilisé pour un problème de classification où la sortie peut être constituée de deux types de classes ou plus.

la matrice de confusion est un tableau à deux dimensions («Réel» et «Prédit») et des ensembles de «classes» dans les deux dimensions. Nos classifications réelles sont des colonnes et les prédites sont des lignes.

		Actual	
		Positives(1)	Negatives(0)
Predicted	Positives(1)	TP	FP
	Negatives(0)	FN	TN

1. True Positives (TP) : Les vrais positifs sont les cas où la classe réelle du point de données était 1 (True) et la prédite est également 1 (True).
2. True Negatives (TN) : Les vrais négatifs sont les cas où la classe réelle du point de données était 0 (False) et la prédiction est également 0 (False)
3. False Positives (FP) : Les faux positifs sont les cas où la classe réelle du point de données est 0 (False) et la prédiction est 1 (True). False est parce que le modèle a prédit de manière incorrecte et positif parce que la classe prédite était positive. (1)
4. False Negatives (FN) : Les faux négatifs sont les cas où la classe réelle du point de données était 1 (Vrai) et la prédiction est 0 (Faux). False est dû au fait que le modèle a prédit de manière incorrecte et négatif car la classe prédite était négative. (0)

Exemple d'une matrice de confusion dans un cas avec trois classes :

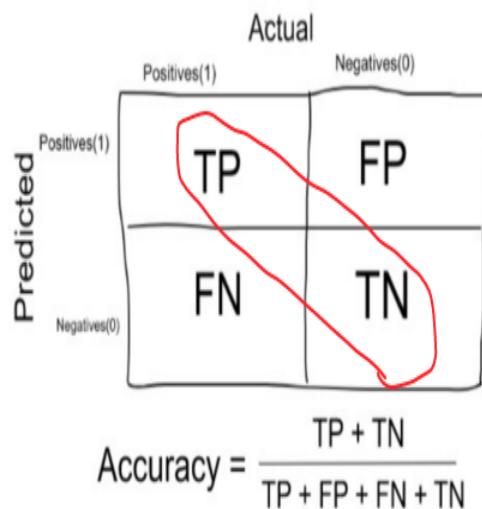
		Actual class		
		Cat	Dog	Rabbit
Predicted class	Cat	5	2	0
	Dog	3	3	2
	Rabbit	0	1	11

pour la cas de chat :

		Actual class	
		Cat	Non-cat
Predicted class	Cat	5 True Positives	2 False Positives
	Non-cat	3 False Negatives	17 True Negatives

Accuracy

La précision dans les problèmes de classification est le nombre de prédictions correctes faites par le modèle sur toutes les sortes de prédictions effectuées.



Quand utiliser l'accuracy : l'accuracy est une bonne mesure lorsque les classes de variables cibles dans les données sont presque équilibrées.

Ex : 60% des classes de nos images de fruits sont des pommes et 40% des oranges. Un modèle qui prédit si une nouvelle image est Pomme ou Orange, 97% de fois correctement est une très bonne mesure dans cet exemple.

Quand NE PAS utiliser l'accuracy : l'accuracy ne doit JAMAIS être utilisée comme mesure lorsque les classes de variables cibles dans les données constituent la majorité d'une classe.

Exemple : Dans un exemple de détection du cancer avec 100 personnes, seulement 5 personnes ont un cancer. Disons que notre modèle est très mauvais et prédit chaque cas comme étant sans cancer. Maintenant, même si le modèle est terrible pour prédire le cancer, l'accuracy d'un si mauvais modèle est également de 95%.

Accuracy

```
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
```

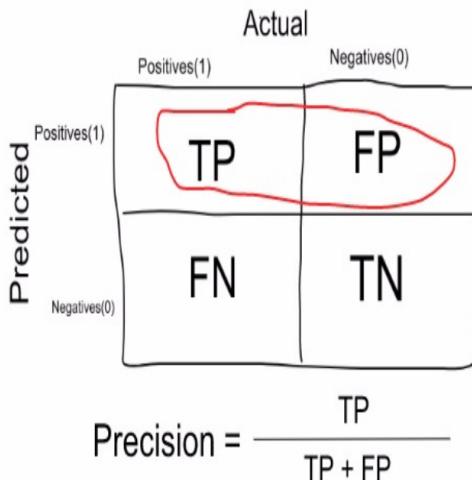
```

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LogisticRegression()
scoring = 'accuracy'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))

```

Précision

La précision est une mesure qui nous indique quelle proportion de patients pour lesquels nous avons diagnostiqué un cancer ont effectivement eu un cancer. Les personnes prédictes cancéreuses sont les TP et les FP et les personnes atteintes d'un cancer sont les TP.



Recall et Sensitivity

Le Recall est une mesure qui nous indique quelle proportion de patients ayant un cancer a été diagnostiquée par l'algorithme comme ayant un cancer.

		Actual	
		Positives(1)	Negatives(0)
Predicted	Positives(1)	TP	FP
	Negatives(0)	FN	TN

$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$

Specificity

Specificity : est une mesure qui nous indique quelle proportion de patients non atteints de cancer a été prédicté par le modèle comme non cancéreuse. Les négatifs actuels (les personnes sans cancer sont FP et TN) et les personnes diagnostiquées par nous qui n'ont pas le cancer sont TN. (Remarque : la FP est incluse car la personne n'a pas réellement de cancer alors que le modèle prédit.

		Actual	
		Positives(1)	Negatives(0)
Predicted	Positives(1)	TP	FP
	Negatives(0)	FN	TN

$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$

F1 Score

Le F1 Score est la moyenne pondérée de la précision et du Recall. Par conséquent, ce score prend en compte à la fois les faux positifs et les faux négatifs. Intuitivement, il n'est pas aussi facile à comprendre que l'accuracy, mais F1 est généralement plus utile que l'accuracy, en particulier si la répartition des classes est inégale. La précision fonctionne mieux si les faux positifs et les faux négatifs ont un coût similaire. Si le coût des faux positifs et des faux négatifs est très différent, il vaut mieux regarder à la fois Precision et Recall.

$$\text{F1 Score} = 2 * (\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$$

Evaluation Metrics

```
from sklearn.metrics import precision_recall_fscore_support as score
```

```

predicted = [1,2,3,4,5,1,2,1,1,4,5]
y_test = [1,2,3,4,5,1,2,1,1,4,1]

precision, recall, fscore, support = score(y_test, predicted)

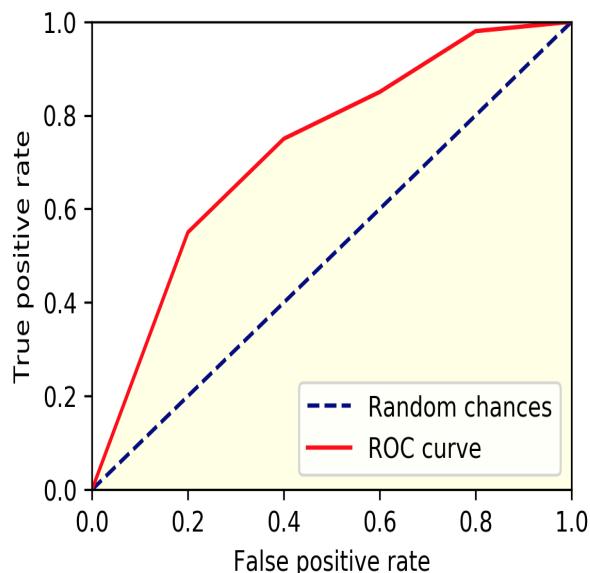
print('precision: {}'.format(precision))
print('recall: {}'.format(recall))
print('fscore: {}'.format(fscore))
print('support: {}'.format(support))

```

2.5.2 ROC et AUC

La courbe AUC - ROC est une mesure de performance pour un problème de classification à différents paramètres de seuils. ROC est une courbe de probabilité et l'AUC représente le degré ou la mesure de la séparabilité. Il indique dans quelle mesure le modèle est capable de distinguer les classes. Plus l'AUC est élevée, mieux le modèle est de prédire 0 comme 0 et 1 comme 1. Par analogie, plus l'AUC est élevée, mieux le modèle consiste à faire la distinction entre les patients atteints de maladie et ceux qui n'en ont pas.

La courbe ROC est tracée avec TPR contre le FPR où TPR est sur l'axe des y et FPR est sur l'axe des x.

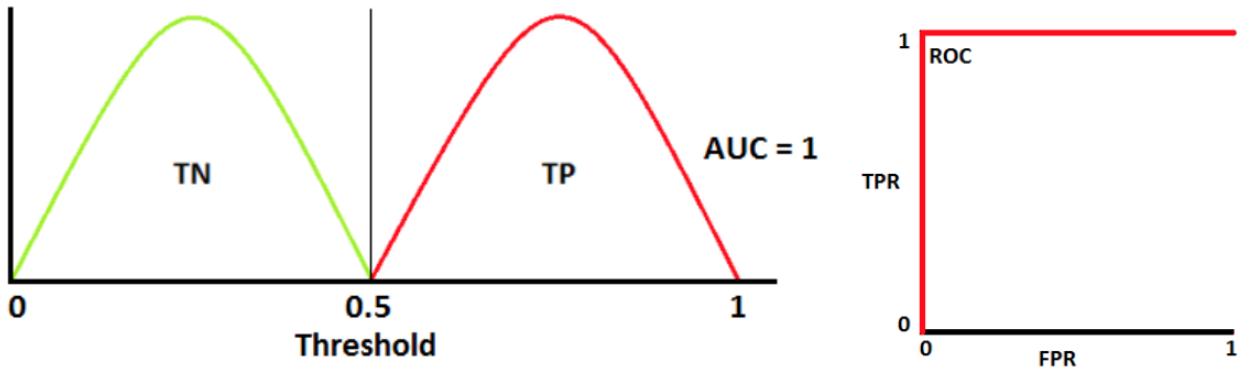


True Positive Rate = True Positives / (True Positives + False Negatives) = Sensitivity

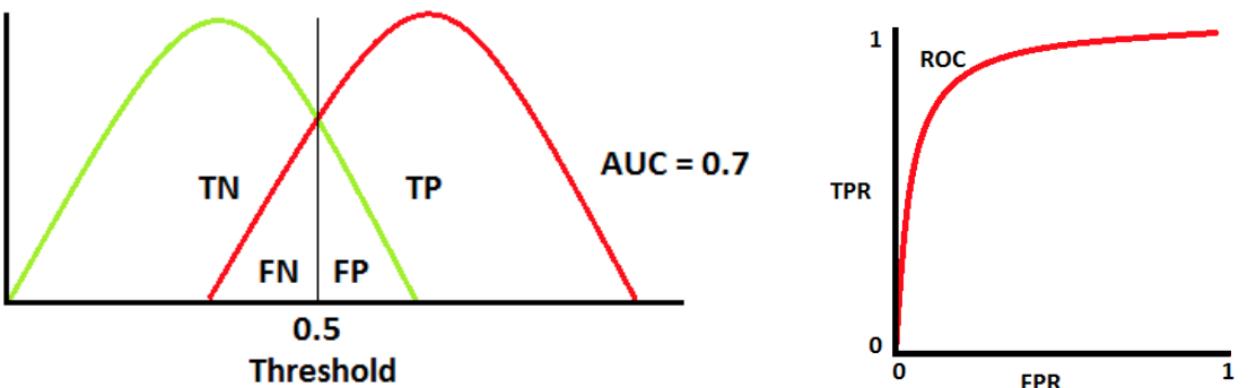
False Positive Rate = False Positives / (False Positives + True Negatives)= Specificity

La courbe ROC est un outil utile pour plusieurs raisons :

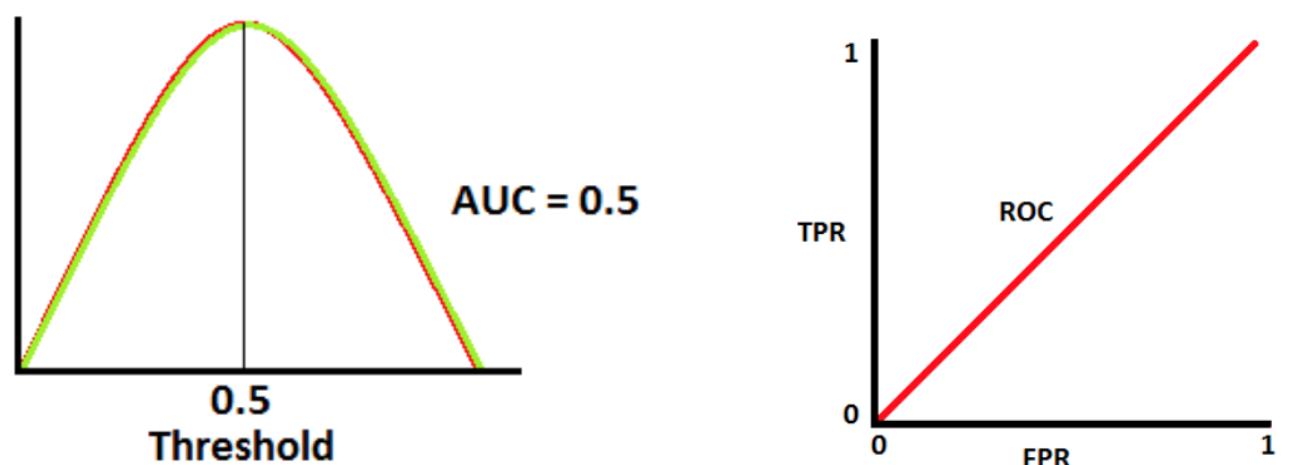
- Les courbes de différents modèles peuvent être comparées directement en général ou pour différents seuils.
- L'aire sous la courbe (AUC) peut être utilisée comme résumé de la compétence du modèle.



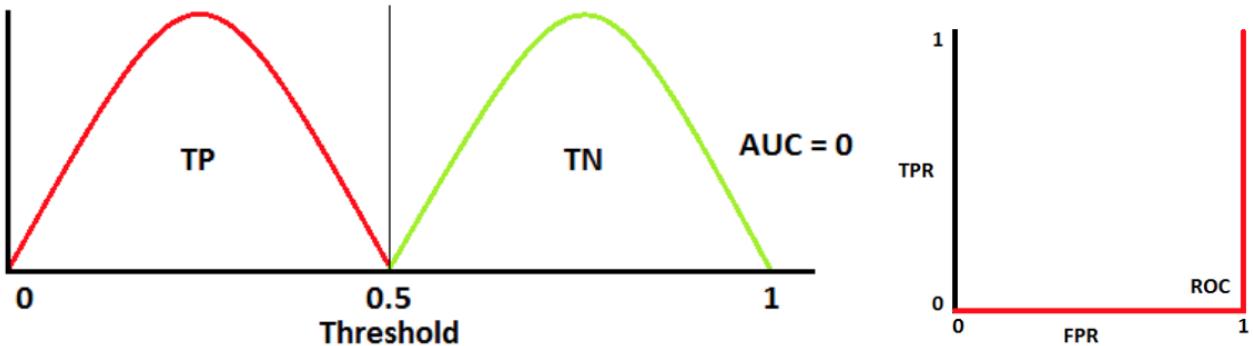
C'est une situation idéale. Lorsque deux courbes ne se chevauchent pas du tout, le modèle a une mesure idéale de séparabilité. Il est parfaitement capable de distinguer la classe positive de la classe négative.



Lorsque deux distributions se chevauchent, nous introduisons une erreur de type 1 et de type 2. Selon le seuil, nous pouvons les minimiser ou les maximiser. Lorsque l'AUC est de 0,7, cela signifie qu'il y a 70% de chances que le modèle soit capable de distinguer la classe positive de la classe négative.



pire situation. Lorsque l'AUC est d'environ 0,5, le modèle n'a aucune capacité de discrimination pour distinguer la classe positive de la classe négative.



Lorsque l'AUC est d'environ 0, le modèle effectue en fait un mouvement alternatif entre les classes. Cela signifie que le modèle prédit la classe négative comme une classe positive et vice versa.

Exemple python :

ROC

```
# roc curve and auc
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)
# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2
)
# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(testy))]
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainX, trainy)
# predict probabilities
lr_probs = model.predict_proba(testX)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(testy, ns_probs)
lr_auc = roc_auc_score(testy, lr_probs)
# summarize scores
print('No Skill: ROC AUC=% .3f' % (ns_auc))
print('Logistic: ROC AUC=% .3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(testy, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(testy, lr_probs)
# plot the roc curve for the model
```

```

pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()

```

2.5.3 Matrice de confusion

La matrice de confusion est une présentation pratique de la précision d'un modèle avec deux classes ou plus.

Le tableau présente les prévisions sur l'axe des x et les résultats de précision sur l'axe des y. Les cellules du tableau sont le nombre de prédictions faites par un algorithme d'apprentissage automatique.

Par exemple, un algorithme d'apprentissage automatique peut prédire 0 ou 1 et chaque prédiction peut en fait avoir été 0 ou 1. Les prédictions pour 0 qui étaient en fait 0 apparaissent dans la cellule pour prédiction = 0 et réelle = 0, tandis que les prédictions pour 0 qui étaient en fait, 1 apparaît dans la cellule pour prédiction = 0 et réel = 1. Etc.

Classification Report

```

import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-
      diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size
                      =test_size, random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)

```

2.5.4 Rapport de classification

Scikit-learn fournit un rapport de commodité lorsque on travaille sur des problèmes de classification pour donner une idée rapide de la précision d'un modèle à l'aide d'un certain nombre de mesures.

La fonction `classification_report()` affiche la précision, le rappel, le score F1 et le support pour chaque classe.

L'exemple ci-dessous illustre le rapport sur le problème de classification binaire.

Classification Report

```
# Cross Validation Classification Report
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size =test_size, random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print(report)
```

2.5.5 Mesures de performance d'un model de régression

Contrairement à un problème de classification, l'objectif d'un problème de régression n'est pas de faire des prédictions sur une variable discrète (s'agit-il d'un spam électronique ou non ?). Au lieu de cela, nous serions chargés de prévoir les salaires des employés par exemple. Donc Il serait difficile de prédire avec précision le salaire de quelqu'un.

Pour cette raison il faut évaluer la performance d'un tel model, en appliquent des formule pour bien juger cette performance, les formules souvent utilisées dans ce contexte.

MAE =

$$\left(\frac{1}{n} \right) \sum_{i=1}^n |\hat{y}_i - y_i|$$

RMSE =

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2}$$

Exemple MAE et RMSE :

On va prédire les salaires de 4 employés en se basant sur le modèle de régression linéaire que l'on a établi. Les valeurs prédictives : $980 + 1943 + 1239 + 2020 = 6182$. Les valeurs réelles : $1000 + 1500 + 2000 + 2500 = 7000$

Selon la méthode MAE :

$$\begin{aligned} |1000 - 980| &= 20. \\ |1500 - 1943| &= 443. \\ |2000 - 1239| &= 761. \\ |2500 - 2020| &= 480 \# \text{The average of the error summation.} \\ (20 + 443 + 761 + 480) / 4 &= 426. \end{aligned}$$

Selon le résultat obtenu on constate que la prédition soit décalée de 426 (en moyenne) par rapport à la vérité.

Un MAE de 426\$ indique que nous devrions nous attendre, en moyenne, à un écart de 426\$, ce qui est assez élevé compte tenu de la fourchette de notre variable dépendante.

RMSE :

$$\begin{aligned} (1000 - 980)^2 &= 400 \\ (1500 - 1943)^2 &= 196249 \\ (2000 - 1239)^2 &= 579121 \\ (2500 - 2020)^2 &= 230400 \# \text{The average of the square error summation} \\ (400 + 196249 + 579121 + 230400) / 4 &= 251543 \# \text{Square root of the square error summation} \\ \text{Square root of } \$251,543 &\text{ is } \tilde{\$}502 \end{aligned}$$

On note que le RMSE est plus grand que le MAE. Étant donné que la RMSE corrige la différence entre les prédictions et la valeur réelle, toute différence significative est d'autant plus importante qu'elle est équitablement calculée. RMSE est plus sensible aux valeurs aberrantes.

Sklearn regression metrics :

MAE RMSE

```
# Cross Validation Regression MAE
import pandas
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv"
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = pandas.read_csv(url, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LinearRegression()

scoring = 'neg_mean_absolute_error'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MAE: %.3f (%.3f)" % (results.mean(), results.std()))
```

```

scoring = 'neg_mean_squared_error'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MSE: %.3f (%.3f)") % (results.mean(), results.std())

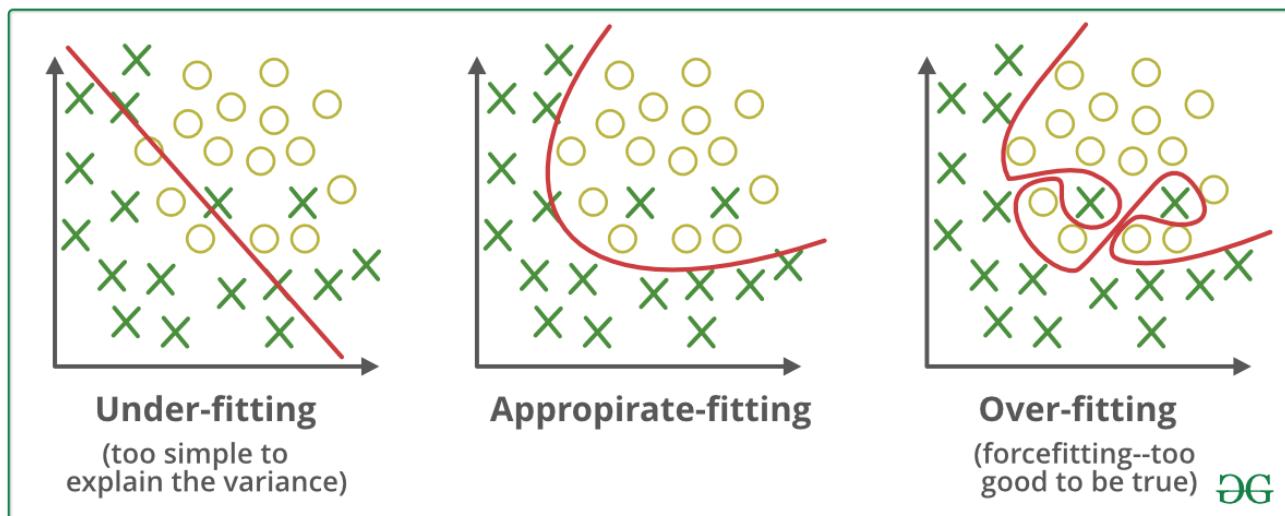
scoring = 'r2'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("R^2: %.3f (%.3f)") % (results.mean(), results.std())

```

2.6 Underfitting and Overfitting

Underfitting : Underfitting fait référence à un modèle qui ne peut ni modéliser les données d'apprentissage ni se généraliser à de nouvelles données. Un modèle d'apprentissage automatique sous-adapté « Underfit » n'est pas un modèle approprié et sera évident car il aura de mauvaises performances sur les données d'entraînement.

Overfitting : Overfitting fait référence à un modèle qui modélise trop bien les données d'apprentissage. Il survient lorsqu'un modèle apprend les détails et le bruit dans les données d'apprentissage dans la mesure où ils ont un impact négatif sur les performances du modèle avec de nouvelles données. Cela signifie que les fluctuations sonores ou aléatoires dans les données d'apprentissage sont captées et apprises en tant que concepts par le modèle. Le problème est que ces concepts ne s'appliquent pas aux nouvelles données et ont un impact négatif sur la capacité des modèles à généraliser.



Pour éviter le Overfitting :

- Cross- Validation : Un moyen standard de détecter les erreurs de prédiction hors échantillon consiste à utiliser le 5-fold cross validation.
- Early Stopping : ses règles nous fournissent des indications sur le nombre d'itérations pouvant être exécutées avant que l'apprenant ne commence à trop s'ajuster.
- Pruning : Le taille pruning est largement utilisée lors de la création de modèles associés. Cela supprime simplement les noeuds, ce qui ajoute peu de puissance prédictive au problème à résoudre.
- Regularization : Il introduit un terme de coût pour apporter plus de fonctionnalités avec la fonction objectif. Par conséquent, il essaie de remettre à zéro les coefficients de nombreuses variables et donc de réduire les coûts.

2.7 Données asymétries « Umbalanced Data »

Les problèmes de classification déséquilibrés posent un défi pour la modélisation prédictive car la plupart des algorithmes d'apprentissage automatique utilisés pour la classification ont été conçus autour de l'hypothèse d'un nombre égal d'exemples pour chaque classe.

Il en résulte des modèles dont les performances prédictives sont faibles, en particulier pour la classe minoritaire. Il s'agit d'un problème car, généralement, la classe minoritaire est plus importante et, par conséquent, le problème est plus sensible aux erreurs de classification de la classe minoritaire que de la classe majoritaire.

Classe de majorité : Plus de la moitié des exemples appartiennent à cette classe, souvent le cas négatif ou normal.

Classe minoritaire : moins de la moitié des exemples appartiennent à cette classe, souvent le cas positif ou anormal.

2.7.1 Intuition pour les données déséquilibrées

La fonction `make_classification()` scikit-learn peut être utilisée pour définir un ensemble de données synthétique avec une asymétrie de classe souhaité. L'argument «poids» spécifie le rapport d'exemples dans la classe négative, par ex.

0.99, 0.01

signifie que 99% des exemples appartiendront à la classe majoritaire et les 1% restants appartiendront à la classe minoritaire.

Imbalanced classification problem

```
# plot imbalanced classification problem
from collections import Counter
from sklearn.datasets import make_classification
from matplotlib import pyplot
from numpy import where
# define dataset
X, y = make_classification(n_samples=1000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99, 0.01], flip_y=0)
# summarize class distribution
counter = Counter(y)
print(counter)
# scatter plot of examples by class label
for label, _ in counter.items():
    row_ix = where(y == label)[0]
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1], label=str(label))
pyplot.legend()
pyplot.show()
```

2.7.2 Évaluer les modèles de classification asymétriques

La précision des prédictions « Accuracy » est la mesure la plus courante pour les tâches de classification, bien qu'elle soit inappropriée et potentiellement dangereusement trompeuse lorsqu'elle est utilisée sur des tâches de classification déséquilibrées.

La raison en est que si 98% des données appartiennent à la classe négative, vous pouvez atteindre une précision de 98% en moyenne en prédisant simplement la classe négative tout le temps, en obtenant un score qui semble naïvement bon, mais en pratique n'a aucune compétence .

Les alternatives populaires de l'accuracy dans ce cas sont les scores de précision et de rappel qui permettent d'envisager les performances du modèle en se concentrant sur la classe minoritaire, appelée classe positive.

Evaluate imbalanced classification

```
# evaluate imbalanced classification model with different metrics
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
# generate dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
                           n_clusters_per_class=1, weights=[0.99], flip_y=0)
# split into train/test sets with same class ratio
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, stratify=y)
# define model
model = LogisticRegression(solver='liblinear')
# fit model
model.fit(trainX, trainy)
# predict on test set
yhat = model.predict(testX)
# evaluate predictions
print('Accuracy: %.3f' % accuracy_score(testy, yhat))
print('Precision: %.3f' % precision_score(testy, yhat))
print('Recall: %.3f' % recall_score(testy, yhat))
print('F-measure: %.3f' % f1_score(testy, yhat))
```

Comment contourner le problème ?

Sous-échantillonnage de la classe majoritaire

Sur-échantillonnage de la classe minoritaire

2.7.3 Sous-échantillonnage de la classe majoritaire

Une approche simple pour utiliser des algorithmes d'apprentissage automatique standard sur un ensemble de données déséquilibré consiste à modifier l'ensemble de données d'apprentissage pour

avoir une distribution de classe plus équilibrée.

Ceci peut être réalisé en supprimant des exemples de la classe majoritaire, appelés «sous-échantillonnage». Un inconvénient possible est que les exemples de la classe majoritaire utiles lors de la modélisation peuvent être supprimés.

La bibliothèque d'apprentissage asymétrique fournit de nombreux exemples d'algorithmes de sous-échantillonnage. Cette bibliothèque peut être installée facilement en utilisant pip ; par exemple :

```
pip install imbalanced-learn
```

Une approche rapide et fiable consiste à supprimer au hasard des exemples de la classe majoritaire pour réduire le déséquilibre à un rapport moins sévère ou même pour que les classes soient égales.

Undersampling the majority class

```
# example of undersampling the majority class
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
# generate dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
                           n_clusters_per_class=1, weights=[0.99, 0.01], flip_y=0)
# summarize class distribution
print(Counter(y))
# define undersample strategy
undersample = RandomUnderSampler(sampling_strategy=0.5)
# fit and apply the transform
X_under, y_under = undersample.fit_resample(X, y)
# summarize class distribution
print(Counter(y_under))
```

2.7.4 Sur-échantillonnage de la classe minoritaire

Une alternative à la suppression d'exemples de la classe majoritaire consiste à ajouter de nouveaux exemples de la classe minoritaire.

Ceci peut être réalisé en dupliquant simplement des exemples dans la classe minoritaire, mais ces exemples n'ajoutent aucune nouvelle information. Au lieu de cela, de nouveaux exemples issus de la minorité peuvent être synthétisés à l'aide d'exemples existants dans l'ensemble de données de formation. Ces nouveaux exemples seront «proches» des exemples existants dans l'espace des fonctionnalités, mais différents de manière petite mais aléatoire.

L'algorithme SMOTE est une approche populaire pour sur-échantillonner la classe minoritaire. Cette technique peut être utilisée pour réduire le déséquilibre ou pour uniformiser la distribution des classes.

Oversampling the minority class

```
# example of oversampling the minority class
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE
# generate dataset
```

```

X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
                           n_clusters_per_class=1, weights=[0.99, 0.01], flip_y=0)
# summarize class distribution
print(Counter(y))
# define oversample strategy
oversample = SMOTE(sampling_strategy=0.5)
# fit and apply the transform
X_over, y_over = oversample.fit_resample(X, y)
# summarize class distribution
print(Counter(y_over))

```

On peut combiner les deux techniques précédentes en utilisant la fonction SMOTEENN qui combine à la fois le sur-échantillonnage SMOTE de la classe minoritaire et le sous-échantillonnage Edited Nearest Neighbors de la classe majoritaire.

Undersampling and Oversampling

```

example of both undersampling and oversampling
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.combine import SMOTEENN
# generate dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
                           n_clusters_per_class=1, weights=[0.99, 0.01], flip_y=0)
# summarize class distribution
print(Counter(y))
# define sampling strategy
sample = SMOTEENN(sampling_strategy=0.5)
# fit and apply the transform
X_over, y_over = sample.fit_resample(X, y)
# summarize class distribution
print(Counter(y_over))

```

2.7.5 Algorithmes sensibles aux coûts "Cost-Sensitive Algorithms"

L'apprentissage sensible aux coûts est un sous-domaine de l'apprentissage automatique qui prend en compte les coûts des erreurs de prédiction (et éventuellement d'autres coûts) lors de la formation d'un modèle d'apprentissage automatique. De nombreux algorithmes d'apprentissage automatique peuvent être mis à jour pour être sensibles aux coûts, où le modèle est pénalisé pour les erreurs de classification erronée d'une classe plus que l'autre, comme la classe minoritaire.

La bibliothèque scikit-learn offre cette capacité pour une gamme d'algorithmes via l'attribut `class_weight` spécifié lors de la définition du modèle. Une pondération peut être spécifiée qui est inversement proportionnelle à la distribution des classes.

Si la distribution des classes était de 0,99 à 0,01 pour les classes majoritaires et minoritaires, l'argument `class_weight` pourrait être défini comme un dictionnaire qui définit une pénalité de 0,01 pour les erreurs commises pour la classe majoritaire et une pénalité de 0,99 pour les erreurs commises avec la classe minoritaire , par exemple 0 : 0,01, 1 : 0,99.

Il s'agit d'une heuristique utile qui peut être configurée automatiquement en définissant l'argument `class_weight` sur la chaîne «balanced».

Undersampling and Oversampling

```
example of cost sensitive logistic regression for imbalanced classification
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
# generate dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
                           n_clusters_per_class=1, weights=[0.99], flip_y=0)
# split into train/test sets with same class ratio
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, stratify=y)
# define model
model = LogisticRegression(solver='liblinear', class_weight='balanced')
# fit model
model.fit(trainX, trainy)
# predict on test set
yhat = model.predict(testX)
# evaluate predictions
print('F-Measure: %.3f' % f1_score(testy, yhat))
```

Apprentissage supervisé

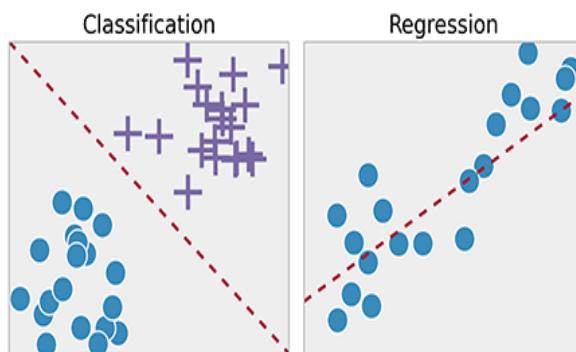
L'apprentissage supervisé (supervised learning en anglais) est une technique d'apprentissage automatique où l'on cherche à produire automatiquement des règles à partir d'une base de données d'apprentissage contenant des « exemples » (en général des cas déjà traités et validés).

Récupération des données dites annotées de leur sorties pour entraîner le modèle, à associé un label ou une classe cible et puis l'algorithme devienne capable de la prédire sur de nouvelles données non annotées une fois entraîné. Dans notre exemple, les données d'entrée seraient des images, et la cible (ou target en anglais) la catégorie de photos que vous voulez.

Régression vs Classification :

Le type de sortie que l'on attend de notre programme : est-ce une valeur continue (un nombre) ou bien une valeur discrète (une catégorie) ? Le premier cas est appelé une régression, le second une classification.

Par exemple si je veux déterminer le coût par clic d'une publicité web, j'effectue une régression. Si je veux déterminer si une photo est un chat ou un chimpanzé, j'effectue une classification.



3.1 La régression linéaire simple

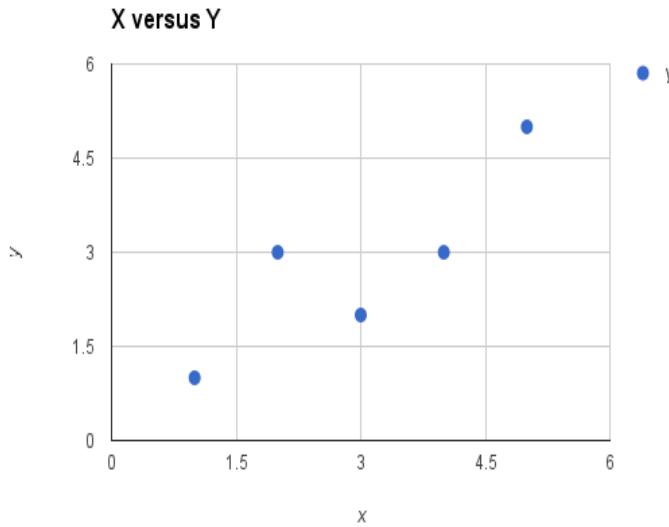
Un modèle de régression linéaire est un modèle de régression qui cherche à établir une relation linéaire entre une variable, dite expliquée, et une ou plusieurs variables, dites explicatives.

La régression linéaire est parmi les algorithmes les plus anciens de l'apprentissage supervisé.

On entraîne généralement le modèle sur un échantillon d'apprentissage et on teste ensuite les performances prédictives du modèle sur un échantillon de test.

3.1.1 Comment fonctionne la régression linéaire simple

X	Y
1	1
2	3
4	3
3	2
5	5



On peut constater que la relation entre X et Y est presque linéaire. On peut dessiner une ligne en diagonale en bas à gauche de l'intrigue vers le haut à droite pour décrire généralement la relation entre les données.

Avec une régression linéaire simple, on peut modeler les données comme suit :

$$y = B_0 + B_1 * x$$

Il s'agit d'une ligne où y est la variable de sortie que nous voulons prédire, x est la variable d'entrée que nous connaissons et B_0 et B_1 sont des coefficients dont nous avons besoin d'estimer qui déplace la ligne.

Techniquement, B_0 s'appelle l'interception car elle détermine où la ligne intercepte l'axe des y . Dans l'apprentissage par machine, on appelle cela le biais, car il est ajouté pour compenser toutes les prédictions que nous faisons. Le terme B_1 s'appelle la pente car il définit la pente de la ligne ou comment x se traduit par une valeur y avant d'ajouter notre biais.

L'objectif est de trouver les meilleures estimations pour les coefficients pour minimiser les erreurs dans la prédiction de y de x .

On peut commencer par estimer la valeur de B_1 comme suit (Méthode des moindres carrés) :

$$SS_{xy} = \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}$$

$$SS_{xx} = \sum_{i=1}^n (x_i - \bar{x})^2 = \sum_{i=1}^n x_i^2 - n(\bar{x})^2$$

$$B_1 = \frac{Cov(x, y)}{Var(x, y)} = \frac{SS_{xy}}{SS_{xx}}$$

$$B_0 = \bar{y} - B_1 \bar{x}$$

$$B_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Les x_i et y_i se réfèrent au fait qu'il faut répéter ces calculs sur toutes les valeurs de l'ensemble de données.

On peut calculer B_0 via l'utilisation de B_1 et quelques statistiques formes selon le data set :

$$B_0 = \bar{y} + B_1 \bar{x}$$

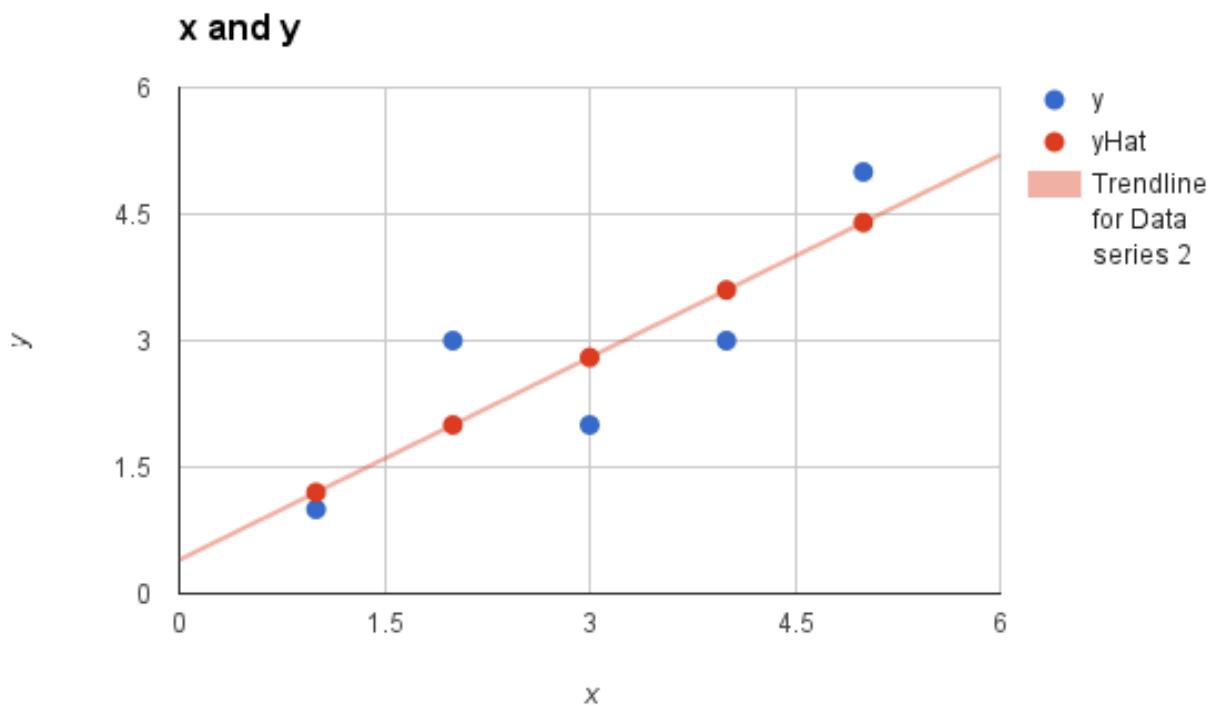
X	Y				
1	1				
2	3				
4	3				
3	2				
5	5				
mean = $(1/n) * \text{sum}(x)$		mean(X) = 3			
		mean(Y) = 2,8			
xi-mean (x)					
1	3	-2			
2	3	-1			
4	3	1			
3	3	0			
5	3	2			
			$((xi - \text{mean}(x)) * (yi - \text{mean}(y))$	$((xi - \text{mean}(x))^2$	
			3,6	4	
			-0,2	1	
			0,2	1	
			0	0	
			4,4	4	
y - mean(y)			somme= 8	10	
1	2,8	-1,8			
3	2,8	0,2			
3	2,8	0,2			
2	2,8	-0,8			
5	2,8	2,2			
			B1 = 0,8		

$$B_0 = 2,8 - 0,8 * 3 = 0,4$$

Pour faire la production en se basant sur la formule :

$$y = B_0 + B_1 * x \Rightarrow y = 0,4 + 0,8 * X$$

X	Y	Predicted Y
1	1	1,2
2	3	2
4	3	3,6
3	2	2,8
5	5	4,4



Estimation d'erreur : On peut estimer l'erreur via Root Mean Squared Error or RMSE.

Y _i	P _i	P _i - Y _i	(P _i - Y _i) ²
1	1.2	0.2	0.04
3	2	-1	1
3	3.6	0.6	0.36
2	2.8	0.8	0.64
5	4.4	-0.6	0.36

$$RMSE = \sqrt{\left(\frac{(0.4 + 1 + 0.36 + 0.64 + 0.36)}{5}\right)} = 0.69$$

Note : on peut calculer les coefficients b en utilisant le stochastique gradient descend.

3.1.2 Implémentation avec python

Regression Simple

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x, m_y = np.mean(x), np.mean(y)
    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x - n*m_y*m_x)
```

```

SS_xx = np.sum(x*x - n*m_x*m_x)
# calculating regression coefficients
b_1 = SS_xy / SS_xx
b_0 = m_y - b_1*m_x
return(b_0, b_1)

def plot_regression_line(x, y, b):
# plotting the actual points as scatter plot
plt.scatter(x, y, color = "m", marker = "o", s = 30)
# predicted response vector
y_pred = b[0] + b[1]*x
# plotting the regression line
plt.plot(x, y_pred, color = "g")
# putting labels
plt.xlabel('x')
plt.ylabel('y')
# function to show plot
plt.show()

def main():
# observations
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
# estimating coefficients
b = estimate_coef(x, y)
print("Estimated coefficients:\nb_0 = {} \nb_1 = {}".format(b[0], b[1]))
# plotting regression line
plot_regression_line(x, y, b)

if __name__ == "__main__":
main()

```

3.1.3 Implémentation avec scikit-learn

Regression Simple sKlearn

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

dataset = pd.read_csv('/Users/macbookair/Documents/cours/cours apprentissage
    machine/datasets/Weather.csv')

```

```

print(dataset.describe())

X = dataset['MinTemp'].values.reshape(-1,1)
y = dataset['MaxTemp'].values.reshape(-1,1)

#Data Plot
dataset.plot(x='MinTemp', y='MaxTemp', style='o')
plt.title('MinTemp vs MaxTemp')
plt.xlabel('MinTemp')
plt.ylabel('MaxTemp')
plt.show()

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=0)

# least square method
regressor = LinearRegression()
regressor.fit(X_train, y_train) #training the algorithm

#SGD Optimizer
#from sklearn.linear_model import SGDRegressor
#regressor = SGDRegressor(max_iter=10000, tol=0.1)

#To retrieve the intercept:
print(regressor.intercept_)
#For retrieving the slope:
print(regressor.coef_)

y_pred = regressor.predict(X_test)

df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
print(df)

#plot the line
plt.scatter(X_test, y_test, color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred
    )))


```

3.2 La régression linière multiple

$$x \in R^d, y \in R$$

pour n exemples

Mathématiquement, on peut écrire la régression linière sous forme :

$$h(\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

- $h(\theta)$: la valeur à prédire.
- θ : les coefficients du modèle. Ces valeurs sont « apprises » lors de l'étape de construction du modèle.
- θ_0 : est interception.
- θ_1 le coefficient pour X_1 .
- θ_n le coefficient pour X_n .

$$x_0 = 1 \rightarrow Intercept$$

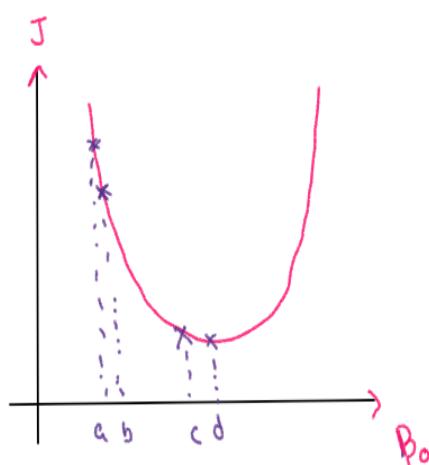
$$h(\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

$$h(\theta) = \sum_{i=0}^n \theta_i x_i = \theta^T X$$

Comment estimer les coefficients ?

3.2.1 Le gradient descente

Le gradient descente c'est une technique d'optimisation afin de minimiser une fonction réelle différentiable.

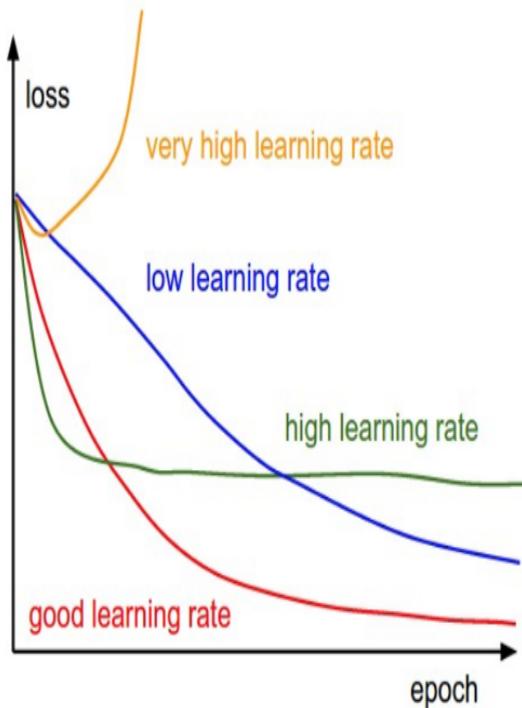


$$\text{gradient } (a-b) > \text{gradient } (c-d)$$

Il existe des techniques qui calcule cout de model « MAE, RSE, etc », le gradient descente « mise à jour de coef θ » :

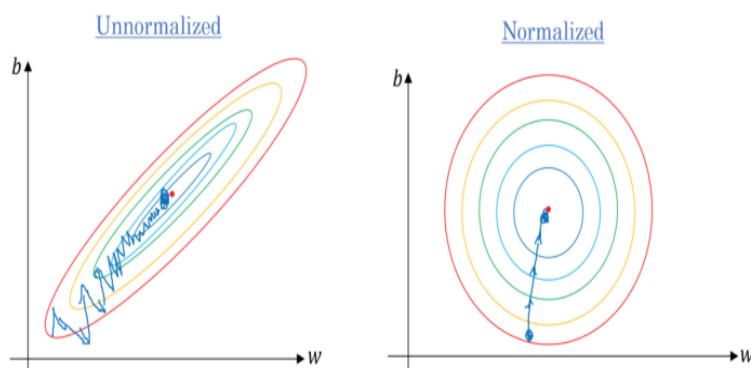
$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

α : Learning rate.



Les valeurs usuelles de Learning rate sont : 0.001, 0.003, 0.01, 0.03, 0.1, 0.3.

Note : il faut normaliser les données de chaque propriétés « Features » entre 0 – 1.



Apres dérivation partielle :

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Nous obtenons la formule finale pour mettre à jour notre paramètre θ_j et obtenir la valeur optimale de manière itérative.

Pour cette tâche, il existe un algorithme mathématique appelé descente de gradient Stochastique :

- Descente de gradient est le processus d'optimisation d'une fonction en suivant les gradients de la fonction de coût.

- Descente de gradient, est simplement un algorithme qui fait de petites étapes le long d'une fonction pour trouver un minimum local.
- Dans l'apprentissage par machine, nous pouvons utiliser une technique qui évalue et met à jour les coefficients chaque itération appelée descente de gradient stochastique pour minimiser l'erreur d'un modèle appliqué sur une source de donnée.

3) Stochastic G.D.

for i in range(M):

$$\theta_j := \theta_j - \alpha \cdot \text{only one example} \\ (\hat{y}^i - y^i) x_j^i$$

$$\theta = \theta - \text{learning_rate} * \text{error} * x$$

θ est le coefficient ou le poids optimisé, learning_rate est un taux d'apprentissage (par exemple 0.01), l'erreur est l'erreur de prédiction pour le modèle sur les données attribuées au poids et x est la valeur d'entrée.

3.2.2 Comment fonctionne la régression linéaire multiple

Prenons le DataSet suivant avec deux premières colonnes (X_1 et X_2) les variables d'entrées et la dernière colonne (Y) variable de sortie.

X1	X2	Y
1	2	1
2	4	6
3	10	3
4	-6	4
5	4	5
6	3	6
7	1	2

Puisque on a deux attributs d'entrés (X_1 et X_2) et que nous souhaitons utiliser la régression linéaire, on parle de régression linéaire multiple.

Avec une régression linéaire multiple, on va modéliser nos données comme suit :

$$y = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2$$

θ_0 , θ_1 et θ_2 sont des coefficients que nous devons estimer.

θ_0 est appelée interception car elle détermine où la ligne intercepte l'axe des y.

Pour calculer ces coefficients il existe plusieurs méthodes : moindre carrées, descente de gradient, descente de gradient stochastique, etc.

Si on prend la méthode descente de gradient stochastique qui minimise une fonction en suivant les gradients de la fonction de coût, on peut aboutir à la formule suivante capable de calculer les coefficients :

$$\theta = \theta - \text{learning_rate} * \text{error} * x$$

Plus qu'on a tous les outils nécessaire pour entraîner notre modèle (DataSet, Formule de modélisation, Méthode d'optimisation » on peut entraîner notre modèle à travers plusieurs itérations pour chaque ligne et une epoch pour les 7 lignes.

Descente de gradient itération n ° 1 :

Commençons par des valeurs de 0.0 pour les trois coefficients.

$$\theta_0 = 0.0$$

$$\theta_1 = 0.0$$

$$\theta_2 = 0.0$$

$$Y = 0,0 + 0,0 * x_1 + 0,0 * x_2$$

Nous pouvons calculer l'erreur pour une prédiction comme suit :

$$\text{erreur} = p(i) - y(i) \Rightarrow \text{erreur} = p(1) - y(1) \Rightarrow \text{erreur} = 0 - 1 = -1$$

Nous pouvons maintenant utiliser cette erreur dans notre équation pour la descente de gradient pour mettre à jour les poids. Nous allons commencer par mettre à jour l'interception θ_0 d'abord, parce que c'est plus facile.

$$\alpha = \text{learning_rate} = 0.01$$

$$\theta_0(t+1) = \theta_0(t) - \alpha * \text{error} = 0 - 0.01 * -1 = 0.01$$

Voyons maintenant comment mettre à jour la valeur de θ_1 et θ_2 . Nous utilisons la même équation avec un petit changement. L'erreur est filtrée par l'entrée qui l'a provoquée. Nous pouvons mettre à jour θ_1 et θ_2 en utilisant l'équation :

$$\theta_1(t+1) = \theta_1(t) - \alpha * \text{error} * x_1 = 0 - 0.01 * -1 * 1 = 0.01 \quad \theta_2(t+1) = \theta_2(t) - \alpha * \text{error} * x_2 = 0 - 0.01 * -1 * 2 = 0.02$$

Donc le modèle devient : $Y = 0,01 + 0,01 * x_1 + 0,02 * x_2$, on peut passer à la deuxième itération concernant ligne 2 de notre DataSet, ainsi de suite jusqu'à une condition d'arrêt : nombre d'epochs (dans notre cas chaque epoch est équivalente à 7 itérations).

3.2.3 Implémentation avec python

Regression Multiple

```
def predict(row, coefficients):
    yhat = coefficients[0]
    for i in range(len(row)-1):
        yhat += coefficients[i + 1] * row[i]
    return yhat

# Estimate linear regression coefficients using stochastic gradient descent
def coefficients_sgd(train, l_rate, n_epoch):
    coef = [0.0 for i in range(len(train[0]))]
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
```

```

yhat = predict(row, coef)
error = yhat - row[-1]
sum_error += error**2
coef[0] = coef[0] - l_rate * error
for i in range(len(row)-1):
    coef[i + 1] = coef[i + 1] - l_rate * error * row[i]
print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
return coef

# Calculate coefficients
dataset = [[1, 1], [2, 3], [4, 3], [3, 2], [5, 5]]
l_rate = 0.001
n_epoch = 50
coef = coefficients_sgd(dataset, l_rate, n_epoch)
print(coef)

```

Note : il existe d'autre méthode d'apprentissage pour la régression linéaire : « Gardiant descent , moindre carrée, etc. »

3.2.4 Implémentation avec scikit-learn

Regression Multiple sKlearn

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
dataset = pd.read_csv('/Users/macbookair/Documents/cours/cours apprentissage
    machine/datasets/winequality.csv')

print(dataset.describe())

X = dataset[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
    'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH',
    'sulphates', 'alcohol']].values

y = dataset['quality'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

#coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])

```

```

#print(coeff_df)
y_pred = regressor.predict(X_test)

df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

df1 = df.head(25)

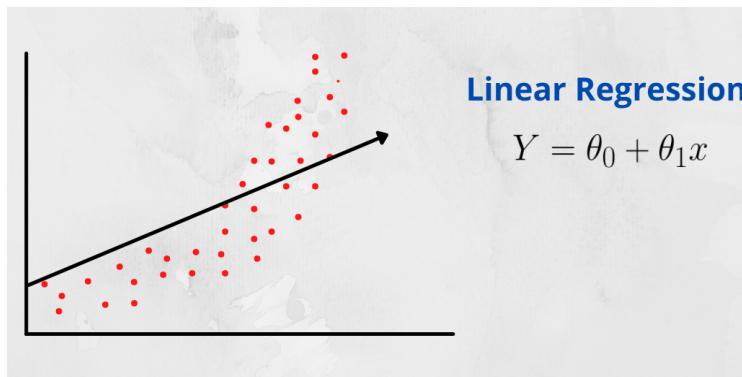
print(df1)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

```

3.3 La régression linière polynomial

Dans un algorithme de régression linéaire simple, cela ne fonctionne que lorsque la relation entre les données est linéaire. Mais supposons que si nous avons des données non linéaires, la régression linéaire ne sera pas capable de tracer une ligne de meilleur ajustement et elle échoue dans de telles conditions. considérez le diagramme ci-dessous qui a une relation non linéaire et vous pouvez y voir les résultats de la régression linéaire, ce qui ne fonctionne pas bien signifie qu'il ne se rapproche pas de la réalité.



La régression polynomiale est une forme de régression linéaire où, en raison de la relation non linéaire entre les variables dépendantes et indépendantes, nous ajoutons des termes polynomiaux à la régression linéaire pour la convertir en régression polynomiale.

Supposons que nous ayons X comme donnée indépendante et Y comme donnée dépendante. Avant d'introduire les données dans un mode de prétraitement, nous convertissons les variables d'entrée en termes polynomiaux en utilisant un certain degré.

Prenons un exemple : ma valeur d'entrée est 35 et le degré d'un polynôme est 2 ; je vais donc trouver 35 puissance 0, 35 puissance 1 et 35 puissance 2. Cela permet d'interpréter la relation non linéaire dans les données. L'équation du polynôme devient quelque chose comme ceci.

$$h(\theta) = \theta_0x_0 + \theta_1x_1 + \theta_2x_2^2 + \dots + \theta_nx_n^n$$

avec $x_0 = 1$.

Le degré d'ordre à utiliser est un Hyperparamètre, et nous devons le choisir judicieusement. Mais l'utilisation d'un degré élevé de polynôme tente de overfit les données et pour des valeurs plus petites de degré, le modèle tente de underfit. Nous devons donc trouver la valeur optimale d'un degré.

3.3.1 Comparaison de la régression polynomiale et linéaire simple

SLR VS PR

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score

X = 6 * np.random.rand(200, 1) - 3
y = 0.8 * X**2 + 0.9*X + 2 + np.random.randn(200, 1)
#equation used -> y = 0.8x^2 + 0.9x + 2
#visualize the data
plt.plot(X, y, 'b.')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=2)

lr = LinearRegression()
lr.fit(x_train, y_train)
y_pred = lr.predict(x_test)
print(r2_score(y_test, y_pred))

plt.plot(x_train, lr.predict(x_train), color="r")
plt.plot(X, y, "b.")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()

#applying polynomial regression degree 2
poly = PolynomialFeatures(degree=2, include_bias=True)
x_train_trans = poly.fit_transform(x_train)
x_test_trans = poly.transform(x_test)
#include bias parameter
```

```

lr = LinearRegression()
lr.fit(x_train_trans, y_train)
y_pred = lr.predict(x_test_trans)
print(r2_score(y_test, y_pred))

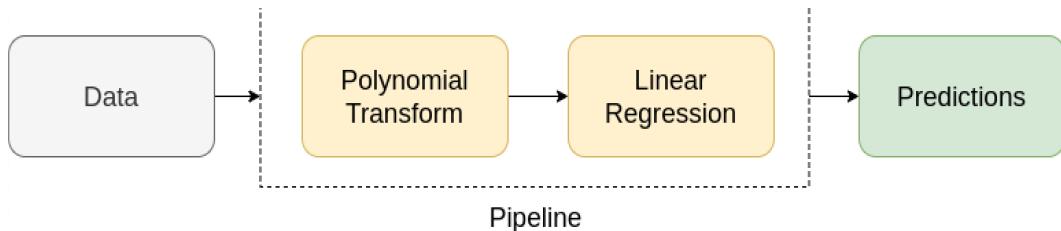
print(lr.coef_)
print(lr.intercept_)

X_new = np.linspace(-3, 3, 200).reshape(200, 1)
X_new_poly = poly.transform(X_new)
y_new = lr.predict(X_new_poly)
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.plot(x_train, y_train, "b.", label='Training points')
plt.plot(x_test, y_test, "g.", label='Testing points')
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()

```

3.3.2 Jouer avec le degré polynomial

Nous allons maintenant concevoir une fonction qui vous aidera à trouver la bonne valeur d'un degré. Ici, nous appliquons toutes les étapes de prétraitement que nous avons effectuées ci-dessus dans une fonction et nous y faisons correspondre le graphique de prédiction final. Il suffit de passer le degré pour que la fonction construise un modèle et trace un graphique d'un degré particulier. Nous allons créer un pipeline d'étapes de prétraitement qui simplifie le processus.



PR Pipeline

```

from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

def polynomial_regression(degree):
    X_new=np.linspace(-3, 3, 100).reshape(100, 1)
    X_new_poly = poly.transform(X_new)
    polybig_features = PolynomialFeatures(degree=degree, include_bias=False)
    std_scaler = StandardScaler()
    lin_reg = LinearRegression()
    polynomial_regression = Pipeline([
        ("poly_features", polybig_features),
        ("std_scaler", std_scaler),
    ])

```

```

        ("lin_reg", lin_reg),
    ])
polynomial_regression.fit(X, y)
y_newbig = polynomial_regression.predict(X_new)
#plotting prediction line
plt.plot(X_new, y_newbig, 'r', label="Degree " + str(degree), linewidth=2)
plt.plot(x_train, y_train, "b.", linewidth=3)
plt.plot(x_test, y_test, "g.", linewidth=3)
plt.legend(loc="upper left")
plt.xlabel("X")
plt.ylabel("y")
plt.axis([-3, 3, 0, 10])
plt.show()

polynomial_regression(10)

```

3.4 Régression logistique

La régression logistique est un algorithme de classification utilisé pour attribuer des observations à un ensemble discret de classes. Certains des exemples de problèmes de classification sont les spams par courrier électronique ou non, les transactions en ligne frauduleuses ou non, les tumeurs malignes ou bénignes.

$$x \in R^d, y \in \{0, 1\}$$

pour n exemples.

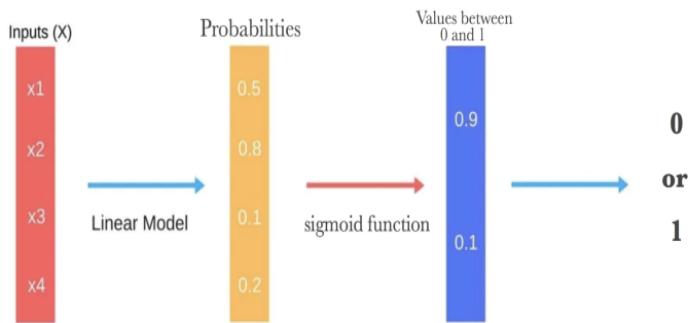
$y^{(i)}$ = 1 Positive example.

$y^{(i)}$ = 0 Negative example.

Two Class Classification		
$y \in \{0, 1\}$	1 or Positive Class	0 or Negative Class
Email	Spam	Not Spam
Tumor	Malignant	Benign
Transaction	Fraudulent	Not Fraudulent

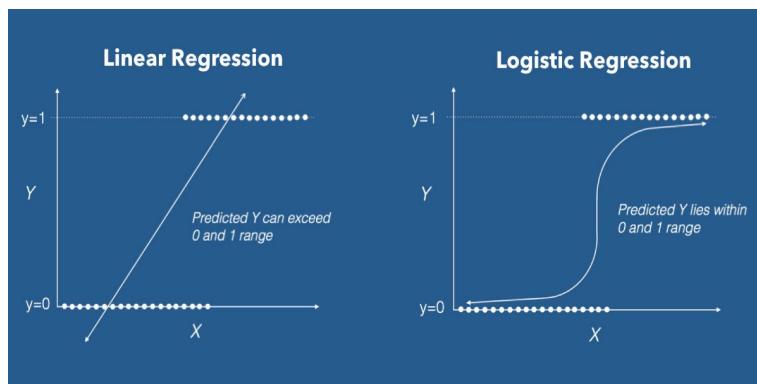
La régression logistique mesure la relation entre la variable dépendante (notre class, ce que nous voulons prédire) et la ou les variables indépendantes (nos caractéristiques), en estimant les probabilités à l'aide de sa fonction logistique sous-jacente.

Ces probabilités doivent ensuite être transformées en valeurs binaires pour pouvoir véritablement prédire. C'est la tâche de la fonction logistique, également appelée fonction sigmoïde. La fonction Sigmoid est capable de prendre n'importe quel nombre réel et la mapper en une valeur comprise entre 0 et 1, mais jamais exactement à ces limites. Ces valeurs comprises entre 0 et 1 seront ensuite transformées en 0 ou 1 à l'aide d'un classifieur de seuil.



Quels sont les types de régression logistique :

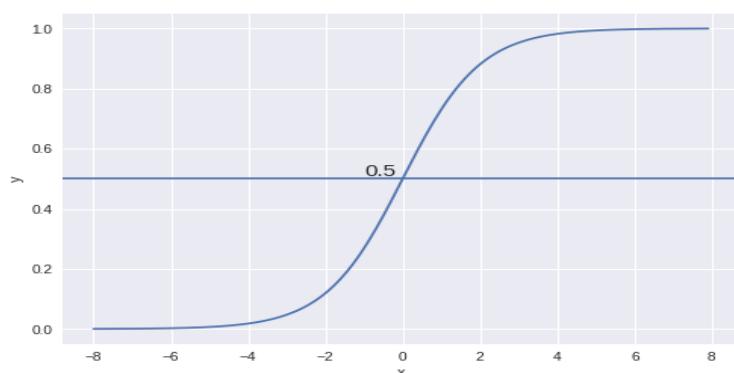
Binaire (p. Ex. Tumeur maligne ou bénigne) Fonctions multi-linéaires failClass (par exemple, chats, chiens ou moutons)



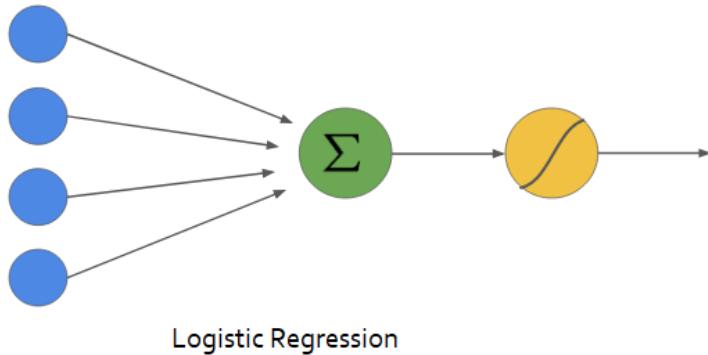
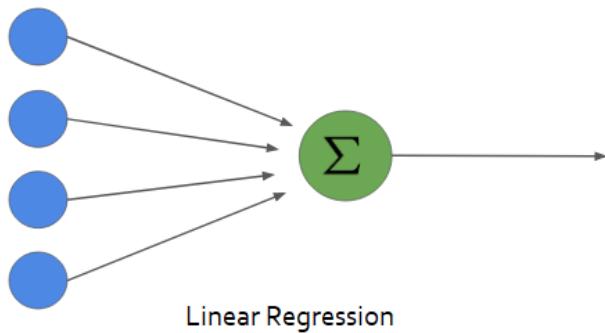
3.4.1 La solution est une régression logistique

On peut dire qu'une régression logistique est un modèle de régression linéaire, mais il utilise une fonction de coût plus complexe. Cette fonction de coût peut être définie comme étant la «fonction sigmoïde» ou encore appelée «fonction logistique» au lieu d'une fonction linéaire.

$$\gamma(t) = \frac{1}{1 + e^{-t}}$$



En régression linéaire, la sortie est la somme pondérée des entrées. La régression logistique est une régression linéaire généralisée en ce sens que nous ne produisons pas directement la somme pondérée des entrées, mais que nous la transmettons via une fonction qui peut mapper toute valeur réelle comprise entre 0 et 1.



Nous pouvons voir que la valeur de la fonction sigmoïde est toujours comprise entre 0 et 1. La valeur est exactement 0,5 à $X = 0$. Nous pouvons utiliser 0,5 comme seuil de probabilité pour déterminer les classes. Si la probabilité est supérieure à 0,5, on la classe dans la classe 1 ($Y = 1$) ou dans la classe 0 ($Y = 0$).

3.4.2 Algorithme d'apprentissage

Le Modèle

$$h_{\theta}(x) = \frac{1}{1 + e^{(-\theta^T x)}}$$

Cost function

$$\sum_{i=1}^n y^{(i)} \log(\theta^T x) + (1 - y^{(i)}) \log(1 - \theta^T x)$$

Gradient Descent

Pour un exemple "SGD".

$$\theta_j := \theta_j - \alpha [y^{(i)} - h_{\theta}(x)] \cdot x_j^{(i)}$$

Pour tous les exemples.

$$\theta_j := \theta_j - \alpha \sum_{i=1}^n [y^{(i)} - h_{\theta}(x)] \cdot x_j^{(i)}$$

On peut ajouter $1/n$ pour standardisation.

$$\theta_j := \theta_j - \frac{1}{n} \alpha \sum_{i=1}^n [y^{(i)} - h_{\theta}(x)] \cdot x_j^{(i)}$$

3.4.3 Comment la régression logistique fonctionne

Le processus de fonctionnement de régression logistique va suivre ces étapes :

- Calculer la fonction logistique.
- Calculer les coefficients d'un modèle de régression logistique utilisant une descente de gradient stochastique.
- Faire des prédictions en utilisant un modèle de régression logistique.

Ce DataSet a deux variables d'entrée (X_1 et X_2) et une variable de sortie (Y). Les variables d'entrées sont des nombres aléatoires à valeurs réelles. La variable de sortie a deux valeurs, ce qui fait du problème un problème de classification binaire.

X1	X2	Y
2.7810836	2.550537003	0
1.465489372	2.362125076	0
3.396561688	4.400293529	0
1.38807019	1.850220317	0
3.06407232	3.005305973	0
7.627531214	2.759262235	1
5.332441248	2.088626775	1
6.922596716	1.77106367	1
8.675418651	-0.2420686549	1
7.673756466	3.508563011	1

Le modèle de régression logistique prend des entrées à valeur réelle et permet de prédire la probabilité que l'entrée appartienne à la classe par défaut (classe 0).

Si la probabilité est $> 0,5$, nous pouvons considérer le résultat comme une prédiction pour la classe par défaut (classe 0), sinon la prédiction concerne l'autre classe (classe 1).

Pour cet ensemble de données, la régression logistique a trois coefficients, tout comme la régression linéaire, par exemple :

$$\text{output} = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2$$

L'algorithme d'apprentissage consistera à découvrir les meilleures valeurs pour les coefficients (θ_0, θ_1 et θ_2) sur la base des données d'apprentissage.

Contrairement à la régression linéaire, le résultat est transformé en probabilité en utilisant la fonction logistique :

$$p(\text{classe} = 0) = 1 / (1 + e^{\hat{(-\text{output})}})$$

Nous pouvons estimer les valeurs des coefficients en utilisant la descente de gradient stochastique. Il s'agit d'une procédure simple pouvant être utilisée par de nombreux algorithmes d'apprentissage automatique. Cela fonctionne en utilisant le modèle pour calculer une prédiction pour chaque instance de l'ensemble d'apprentissage et en calculant l'erreur pour chaque prédiction.

Nous pouvons appliquer la descente de gradient stochastique au problème de recherche des coefficients pour le modèle de régression logistique comme suit :

Étant donné chaque instance de formation :

- Calculez une prédiction en utilisant les valeurs actuelles des coefficients.
- Calculez les nouvelles valeurs de coefficient en fonction de l'erreur dans la prédiction.

Le processus est répété jusqu'à ce que le modèle soit suffisamment précis ou pour un nombre d'itérations fixe.

Donc commencent par la première itération en affectant 0,0 à chaque coefficient et en calculant la probabilité que la première instance de formation appartienne à la classe 0.

$$\theta_0 = 0.0 \quad \theta_1 = 0.0 \quad \theta_2 = 0.0$$

La première instance de formation est : $x_1 = 2.7810836$, $x_2 = 2.550537003$, $Y = 0$

En utilisant l'équation ci-dessus, nous pouvons intégrer tous ces chiffres et calculer une prédition :

$$prediction = 1/(1 + e^{-(\theta_0 + \theta_1 * x_1 + \theta_2 * x_2)})$$

$$prediction = 1/(1 + e^{-(0.0 + 0.0 * 2.7810836 + 0.0 * 2.550537003)})$$

$$prediction = 0.5$$

Apres Nous pouvons calculer les nouvelles valeurs de coefficient en utilisant une simple équation de mise à jour.

$$\theta = \theta + \alpha * (y - prediction) * prediction * (1 - prediction) * x$$

Alpha est un paramètre que on doir spécifier au début du parcours d'entraînement. Il s'agit du taux d'apprentissage et de la mesure dans laquelle les coefficients changent ou apprennent à chaque mise à jour, dans notre cas on va prendre alpha = 0.3.

$$\theta_0 = \theta_0 + 0.3 * (0 - 0.5) * 0.5 * (1 - 0.5) * 1.0$$

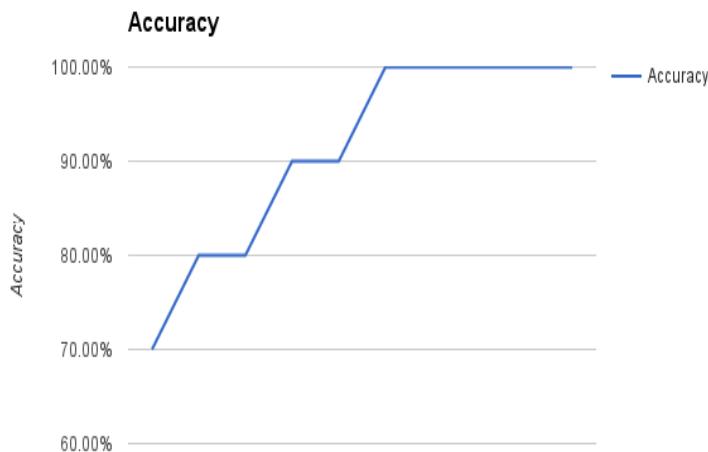
$$\theta_1 = \theta_1 + 0.3 * (0 - 0.5) * 0.5 * (1 - 0.5) * 2.7810836$$

$$\theta_2 = \theta_2 + 0.3 * (0 - 0.5) * 0.5 * (1 - 0.5) * 2.550537003$$

$$\theta_0 = -0.0375, \theta_1 = -0.104290635, \theta_2 = -0.09564513761$$

Nous pouvons répéter ce processus et mettre à jour le modèle pour chaque instance du DataSet. Une seule itération dans le DatSet d'apprentissage s'appelle une epoch, dans notre cas une epoch est équivalente a 10 itérations (10 lignes de notre DataSet).

Le graphe ci-dessous montre l'accuracy du modèle sur 10 epochs.



Les coefficients calculés après 10 epochs de descente de gradient stochastique sont :

$$\theta_0 = -0.4066054641$$

$$\theta_1 = 0.8525733164$$

$$\theta_2 = -1.104746259$$

donc maintenant on peut faire des prédictions via notre modèle, si on prend des valeurs de la 9 ème ligne ($x_1 = 8.675418651$, et $x_2 = -0.2420686549$) :

$$prediction = 1/(1 + e^{(-(-0.4066054641 + 8.675418651 * 0.8525733164 + -0.2420686549) * -1.104746259)})$$

prediction = 0.999 donc class 1

3.4.4 Implémentation avec scikit-learn

Logistic Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from matplotlib.colors import ListedColormap

dataset = pd.read_csv('/Users/macbookair/Documents/cours/cours apprentissage
    machine/datasets/User_Data.csv')

# input
x = dataset.iloc[:, [2, 3]].values
# output
y = dataset.iloc[:, 4].values

xtrain, xtest, ytrain, ytest = train_test_split(
x, y, test_size = 0.25, random_state = 0)

sc_x = StandardScaler()
xtrain = sc_x.fit_transform(xtrain)
xtest = sc_x.transform(xtest)

classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)

y_pred = classifier.predict(xtest)
cm = confusion_matrix(ytest, y_pred)

print ("Confusion Matrix : \n", cm)
print ("Accuracy : ", accuracy_score(ytest, y_pred))
X_set, y_set = xtest, ytest
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                               stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(
    np.array([X1.ravel(), X2.ravel()]).T).reshape(
    X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))
```

```

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

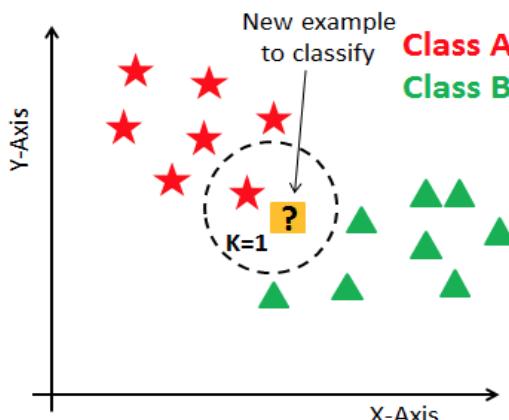
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(['red', 'green'])(i), label = j)

plt.title('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

```

3.5 KNN

K-Nearest Neighbors “KNN”, est l’un des nombreux algorithmes (apprentissage supervisé) utilisés dans l’exploration de données et l’apprentissage automatique, c’est un algorithme de classificateur dans lequel l’apprentissage est basé sur la similarité des données (un vecteur). L’algorithme KNN suppose que des objets similaires existent à proximité. En d’autres termes, des choses similaires sont proches les unes des autres.



L’algorithme Lazy comme KNN ne comporte pas d’étape d’apprentissage. Tous les points de données ne seront utilisés qu’au moment de la prévision. En l’absence d’étape de formation, l’étape de prédiction est coûteuse. Un algorithme d’apprentissage impatient apprend avec empressement pendant l’étape de formation. Pour rechercher des points similaires les plus proches, vous devez calculer la distance entre les points à l’aide de mesures de distance telles que la distance euclidienne, la distance de Hamming, la distance de Manhattan et la distance de Minkowski.

N-dimensional Manhattan distance

$$d(p, q) = \sum_{i=1}^n |p_i - q_i|$$

Dimensional Euclidean distance

$$p = |p_1, p_2, \dots, p_n|$$

$$q = |q_1, q_2, \dots, q_n|$$

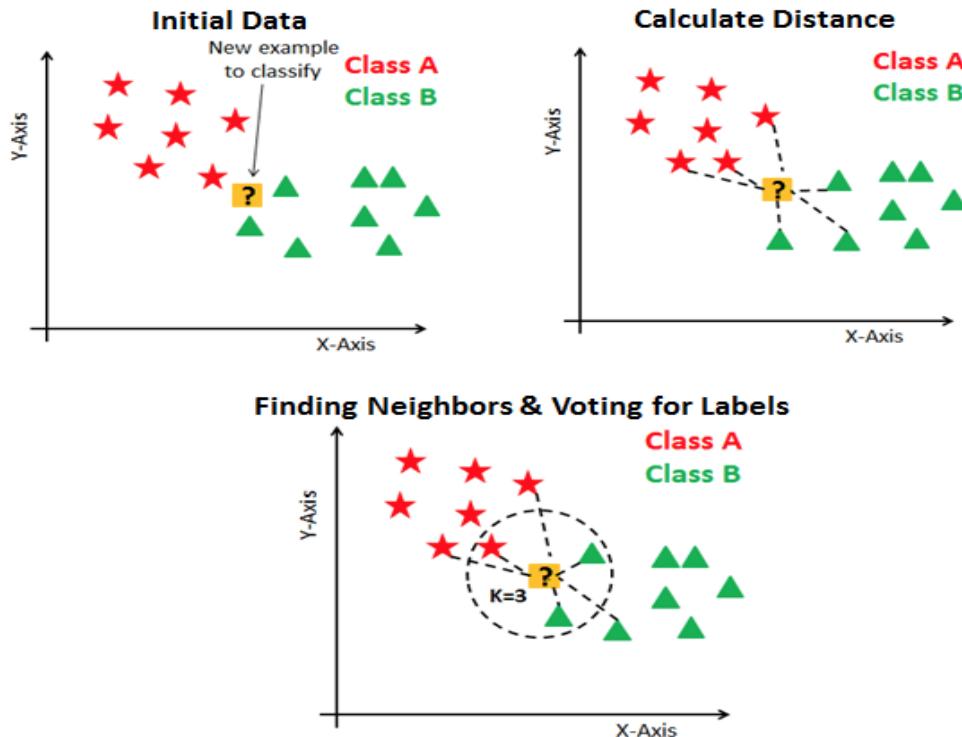
$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2}$$

N-dimensional Minkowski distance

$$d(p, q) = (\sum_{i=1}^n |p_i - q_i|^c)^{1/c}$$

KNN a les étapes de base suivantes :

- Calculer la distance
- Trouver les plus proches voisins
- Votez pour les étiquettes



3.5.1 Comment algorithme KNN fonctionne ?

L'algorithme KNN suit les 7 étapes suivantes :

- Charger les données
- Initialise la valeur de k
- Pour obtenir la classe prévue, itérez de 1 au nombre total de points de données d'entraînement
- Calculez la distance entre les données de test et chaque ligne de données d'entraînement. Ici, nous utiliserons la distance euclidienne comme métrique de distance.
- Triez les distances calculées par ordre croissant en fonction des valeurs de distance.
- Obtenir les k premières lignes du tableau trié
- Obtenir la classe la plus fréquente de ces lignes
- Retourne la classe prédite

Exemple numérique

Nous disposons de données issues de l'enquête par questionnaire (pour demander l'avis des personnes) et de tests objectifs portant sur deux attributs (durabilité à l'acide et force) pour déterminer si un mouchoir en papier spécial est bon ou non. Voici quatre échantillons de formation.

X1 = Acid Durability	X2 = Strength	Y = Classification
7	7	Bad
7	4	Bad
3	4.	Good
1	4	Good

À présent, l'usine fabrique un nouveau tissu qui répond aux tests de laboratoire avec $X1 = 3$ et $X2 = 7$. Sans autre enquête coûteuse, pouvons-nous deviner quelle est la classification de ce nouveau tissu ?

1. Déterminer le paramètre $K = \text{nombre de voisins les plus proches}$ Supposons utiliser $K = 3$
2. Calculez la distance entre l'instance de requête et tous les échantillons d'apprentissage. La coordonnée de l'instance de requête est $(3, 7)$, au lieu de calculer la distance calculée, la distance carrée est plus rapide à calculer (sans racine carrée).

X1 = Acid Durability	X2 = Strength	Distance carrée à l'instance de $(3, 7)$
7	7	$(7-3)^2 + (7-7)^2 = 16$
7	4	$(7-3)^2 + (4-7)^2 = 25$
3	4.	$(3-3)^2 + (7-4)^2 = 9$
1	4	$(1-3)^2 + (4-7)^2 = 13$

On trie la distance et déterminez les voisins les plus proches en fonction de la distance minimale K -ième.

X1	X2	Distance carrée de $3, 7$	Rang	inclus 3-voisins	Catégorie de Y
7	7	$(7-3)^2 + (7-7)^2 = 16$	3	Oui	Bad
7	4	$(7-3)^2 + (4-7)^2 = 25$	4	Non	-
3	4.	$(3-3)^2 + (7-4)^2 = 9$	1	Oui	Good
1	4	$(1-3)^2 + (4-7)^2 = 13$	2	Non	Good

3. En Utilisant la majorité simple de la catégorie des voisins les plus proches comme valeur de prédiction de l'instance de requête.

Nous avons 2 Good et 1 Bad, puisque $2 > 1$, puis nous concluons qu'un nouveau tissu de papier qui a passé avec succès les tests de laboratoire avec $X1 = 3$ et $X2 = 7$ est inclus dans la catégorie Good.

3.5.2 Implémentation avec scikit-learn classification binaire

KNN Binaire

```
# Assigning features and label variables
# First Feature
weather=['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny',
         'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy']
# Second Feature
temp=['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild',
      'Mild', 'Hot', 'Mild']

# Label or target variable
play=['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
```

```

# Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print(weather_encoded)

# converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)

#combinig weather and temp into single listof tuples
features=list(zip(weather_encoded,temp_encoded))

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training sets
model.fit(features,label)

#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print(predicted)

```

3.5.3 Implémentation avec scikit-learn classification multi classes

KNN Multiple

```

#Import scikit-learn dataset library
from sklearn import datasets
#Load dataset
iris = datasets.load_iris()

print(iris.feature_names)
x = iris.data[:, 0:4] # we only take the first four features.
y = iris.target

# Import train_test_split function
from sklearn.model_selection import train_test_split
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3) # 70%
# training and 30% test

```

```

#Import knearest neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier

#Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors=5)

#Train the model using the training sets
knn.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = knn.predict(X_test)

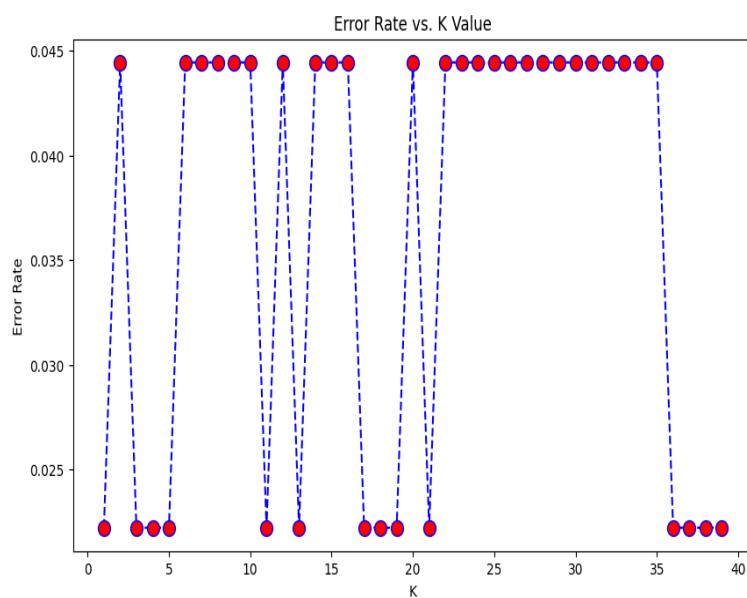
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

```

3.5.4 Comment choisir la valeur optimale de K ?

La question se pose de savoir comment choisir le nombre optimal de voisins ? Et quels sont ses effets sur le classificateur ? Le nombre de voisins (K) dans KNN est un hyperparamètre qu'on doit choisir au moment de la construction du modèle. On peut considérer K comme une variable de contrôle pour le modèle de prédiction.

Des recherches ont montré qu'aucun nombre optimal de voisins ne convient à tous les types de jeux de données. Chaque jeu de données a ses propres exigences. Dans le cas d'un petit nombre de voisins, le bruit aura une influence plus grande sur le résultat et un grand nombre de voisins rend le calcul coûteux. La recherche a également montré qu'un petit nombre de voisins ont un ajustement très flexible, avec un biais faible, mais une variance élevée et un grand nombre de voisins, avec une limite de décision plus lisse, ce qui signifie une variance inférieure mais un biais plus élevé.



Il n'existe pas de méthodes statistiques prédéfinies pour trouver la valeur la plus favorable de K.

- Initialiser une valeur K aléatoire et commencer à calculer.
- Le choix d'une petite valeur de K conduit à des frontières de décision instables.
- Une valeur K substantielle est meilleure pour la classification car elle conduit à lisser les frontières de décision.
- Déterminez un graphique entre le taux d'erreur et K en indiquant les valeurs dans une plage définie. Choisissez ensuite la valeur K ayant un taux d'erreur minimal.

KNN Choix de K

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
#Load dataset
iris = datasets.load_iris()

print(iris.feature_names)
x = iris.data[:, 0:4] # we only take the first four features.
y = iris.target

# Import train_test_split function
from sklearn.model_selection import train_test_split
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3) # 70%
    training and 30% test

#Import knearest neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier

error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:-",min(error_rate),"at K =",error_rate.index(min(error_rate)))

acc = []
# Will take some time
from sklearn import metrics
for i in range(1,40):
    neigh = KNeighborsClassifier(n_neighbors = i).fit(X_train,y_train)

```

```

yhat = neigh.predict(X_test)
acc.append(metrics.accuracy_score(y_test, yhat))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),acc,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:-",max(acc),"at K =",acc.index(max(acc)))

```

3.6 Naive bayes

Naive Bayes Classifier est un algorithme populaire en Machine Learning. C'est un algorithme du Supervised Learning utilisé pour la classification. Il est particulièrement utile pour les problématiques de classification de texte. Un exemple d'utilisation du Naive Bayes est celui du filtre anti-spam.

3.6.1 Modèle discriminatif vs Modèle génératif

Le modèle discriminatif est le modèle dans lequel les algorithmes tentent de classer directement une étiquette en entrée, comme la régression logistique et l'algorithme du perceptron. Le modèle discriminatif n'a pas de concept de ce à quoi l'objet pourrait ressembler. Il se contente de classifier. Il ne peut pas générer une nouvelle image basée sur la classe.

Formellement, c'est $p(y|x; \theta)$ où p peut être n'importe quel modèle de classification tel que le modèle de régression logistique.

D'autre part, le modèle génératif est le modèle qui essaie d'abord d'apprendre à quoi chaque objet peut ressembler. Ensuite, sur la base de l'entrée, il donne une probabilité que l'entrée soit de cette classe. Il dispose des concepts sur ce à quoi l'objet pourrait ressembler. Il peut générer une nouvelle image sur la base des connaissances passées.

$p(y, x; \theta)$: probabilité conjointe , joint probability.

L'exemple classique est le classificateur naïf de Bayes. Dans ce cas, nous avons une classe préalable. Une distribution préalable est essentiellement la connaissance que nous supposons de la distribution des données. Nous supposons simplement que le modèle que nous sélectionnons comme antérieur est un modèle approprié pour expliquer les informations cachées dans l'ensemble de données. À partir de l'ensemble de données, nous pouvons savoir quels paramètres sont les mieux adaptés au modèle que nous avons sélectionné. Avec un modèle antérieur de classe calculé, nous pouvons utiliser la règle de Bayes pour calculer la probabilité d'appartenir à chaque classe, puis choisir celle dont la valeur est la plus élevée. Parallèlement, avec une certaine antériorité, nous pouvons générer un nouvel échantillon à partir du modèle antérieur, puis générer de nouvelles caractéristiques sur la base de l'antériorité choisie. C'est ce qu'on appelle le processus génératif.

3.6.2 Pourquoi le naive bayes ??

Jetons un coup d'œil rapide au théorème de Bayes.

$$p(w_i|x_i) = \frac{p(x_i|w_i) \cdot p(w_i)}{p(x_i)}$$

Ce qui se traduit par :

$$\text{probabilité_postérieure} = \frac{\text{probabilité_conditionnelle} * \text{probabilité_prieurs}}{\text{preuve}}$$

Si nous utilisons le théorème de Bayes comme classificateur, notre but, ou fonction objectif, est de maximiser la probabilité postérieure.

$$\text{la_valeur_de_la_classe_predite} \leftarrow \text{argmax}_{1,2,\dots,j} p(w_i|x_i)$$

Maintenant, parlons des composants individuels. Les prieurs représentent essentiellement notre connaissance experte (ou toute autre connaissance préalable). Le terme d'évidence s'annule car il est constant pour toutes les classes.

Il est temps de parler de la partie "naïve" du "classificateur naïf de Bayes". Ce qui le rend "naïf" est que nous calculons la probabilité conditionnelle (parfois aussi appelée vraisemblance) comme le produit des probabilités individuelles pour chaque caractéristique :

$$p(x|w_j) = p(1|w_j) * \dots * p(d|w_j) = \prod_{k=1}^d p(x_k|w_j)$$

Étant donné que cette hypothèse (l'indépendance absolue des caractéristiques) n'est probablement jamais satisfaite dans la pratique, c'est la partie vraiment "naïve" de Bayes naïf.

3.6.3 Probabilités conditionnelles

Le naive Bayes classifier se base sur le théorème de Bayes. Ce dernier est un classique de la théorie des probabilités. Ce théorème est fondé sur les probabilités conditionnelles. Les probabilité conditionnelle : Quelle est la probabilité qu'un événement se produise sachant qu'un autre événement s'est déjà produit.

Exemple : Supposons qu'on ait une classe de lycéens. Soit et les deux événements suivants :

- l'événement : l'élève est une fille.
- L'événement : l'élève pratique l'allemand.

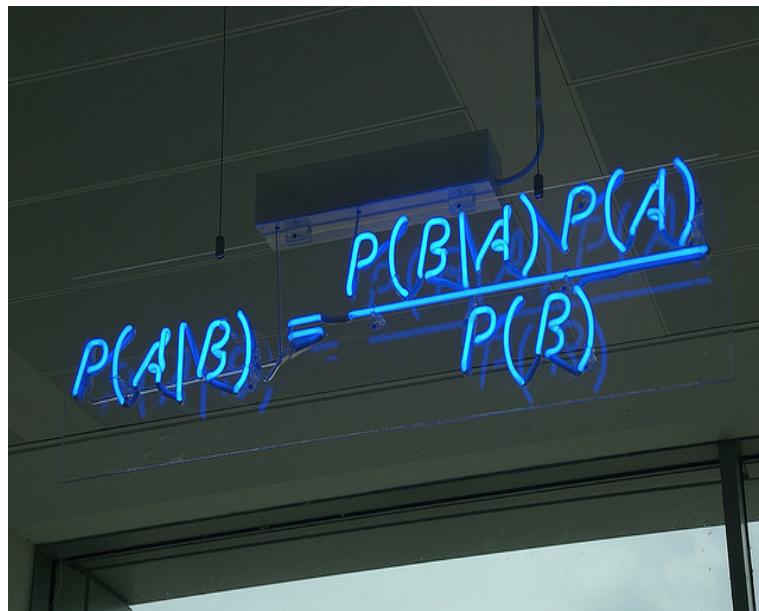
Quelle est la probabilité qu'on choisisse au hasard une fille pratiquant l'allemand ? Le théorème de Bayes permet de calculer ce genre de probabilité. Notons la probabilité d'un événement.

$$P(\text{eleve est une fille ET eleve pratique allemand}) = P(\text{eleve est une fille}) * P(\text{eleve pratique allemand} | \text{eleve est une fille})$$

Cela équivaut à calculer :

$$P(\text{eleve est une fille ET eleve pratique allemand}) = P(\text{eleve pratique allemand}) * P(\text{eleve est une fille} | \text{eleve pratique allemand})$$

Le terme $P(A|B)$ se lit : la probabilité que l'événement A se réalise sachant que l'événement B s'est déjà réalisé. On appelle le terme A : l'évidence. Le terme B s'appelle Outcome.



	Fille (A)	Garçon ($\neg A$)	Totaux
Allemand (B)	10	7	17
Autre langue ($\neg B$)	4	9	13
Totaux	14	16	30

Calculons la probabilité suivante : Quelle est la probabilité qu'on tire au hasard un élève parlant Allemand sachant qu'elle est une fille ? Selon la formule de Bayes on a :

$$P(\text{Allemand}|Fille) = P(A|B) = \frac{P(B \cap A)}{P(A)} = \frac{P(A|B) * P(B)}{P(A)}$$

$P(A)$ est la probabilité de prendre au hasard une fille de la population des élèves de la classe. On appelle $P(A)$ la probabilité antérieure (prior probability).

$$P(A) = \frac{\text{cardinal}(A)}{\text{cardinal}(\Omega)} = \frac{14}{30} = 0.4666$$

$$P(B \cap A) = \frac{\text{cardinal}(B \cap A)}{\text{cardinal}(\Omega)} = \frac{10}{30} = 0.333$$

Ce qui donne :

$$P(B|A) = \frac{\frac{10}{30}}{\frac{14}{30}} = \frac{0.333}{0.4666} = 0.7143$$

Note :

- le cardinal d'un ensemble est le nombre d'éléments dans ce dernier
- $\text{cardinal}(\Omega)$ représente l'ensemble des lycéens de notre exemple (l'univers de probabilités)

3.6.4 Classification multi-classes avec Naive Bayes

Pour mieux comprendre le Naive Bayes Classifier, déroulons le sur un exemple simple. Note : l'exemple ci-dessus est inspiré d'une discussion sur le site StackOverflow. Supposons qu'on ait un jeu de données sur 1000 fruits. On dispose de trois types : Banane, Orange, et "autre". Pour chaque fruit, on a 3 caractéristiques :

- Si le fruit est long ou non
- S'il est sucré ou non
- Si sa couleur est jaune ou non

Notre jeu de données se présente comme suit :

Type	Long	petit (\neg long)	sucré	\neg sucré	Jaune	Pas jaune	Total
Banane	400	100	350	150	450	50	500
Orange	0	300	150	150	300	0	300
Autre fruit	100	100	150	50	50	150	200
Total	500	500	650	350	800	200	1000

Note : le symbole \neg signifie la négation (\neg froid chaud)

L'idée du jeu est de prédire le type d'un fruit (orange, banane ou autre) qu'on n'a pas encore vu. Ceci en se basant sur ses caractéristiques.

Supposons que quelqu'un nous demande de lui donner le type d'un fruit qu'il a. Ses caractéristiques sont les suivantes :

- Il est jaune
- Il est long
- Il est sucré

Pour savoir s'il s'agit d'une banane, ou d'une orange ou d'un autre fruit, il faut qu'on calcule les trois probabilités suivantes :

- $P(\text{Banane}|\text{long}, \text{jaune}, \text{sucré})$: La probabilité qu'il s'agisse d'une banane sachant que le fruit est long, jaune et sucré
- $P(\text{Orange}|\text{long}, \text{jaune}, \text{sucré})$: La probabilité qu'il s'agisse d'une orange sachant que le fruit est long, jaune et sucré
- $P(\text{Autre}|\text{long}, \text{jaune}, \text{sucré})$: La probabilité qu'il s'agisse d'un autre fruit sachant que ce dernier est long, jaune et sucré

Le type du fruit "inconnu" qu'on cherche à classifier sera celui où on a la plus grande probabilité. Selon la formule de Bayes, on a :

$$P(\text{Banane}|\text{long}, \text{jaune}, \text{sucré}) = \frac{P(\text{long}|\text{Banane}) * P(\text{sucré}|\text{Banane}) * P(\text{jaune}|\text{Banane}) * P(\text{Banane})}{P(\text{jaune}) * P(\text{sucré}) * P(\text{long})}$$

Avec notre jeu de données, on peut calculer facilement un certain nombre de probabilités :

$$P(\text{Banane}) = \frac{\text{cardinal}(\text{Banane})}{\text{cardinal}(\text{Tous fruits})} = \frac{500}{1000} = 0.5$$

$$P(Orange) = 0.3$$

$$P(Autrefruit) = 0.2$$

$$P(long) = 0.5$$

$$P(sucre) = 0.65$$

$$P(jaune) = 0.8$$

Calculons maintenant le terme : $P(\text{Long} | \text{Banane})$: Probabilité que le fruit est long sachant qu'il s'agit d'une banane.

$$P(\text{Long}|\text{Banane}) = \frac{\text{cardinal}(\text{BananeEtLong})}{\text{cardinal}(\text{Banane})} = \frac{400}{500} = 0.8$$

Avec la même logique, on peut calculer les probabilités suivantes : $P(\text{Sucre} | \text{Banane})$ et $P(\text{Jaune} | \text{Banane})$. Je vous invite à les calculer comme guise d'entraînement. Maintenant, qu'on a toutes nos probabilités utiles pour notre calcul, on peut calculer la probabilité suivante :

$$P(\text{Banane}|long, jaune, sucre) = \frac{0.8 * 0.7 * 0.9 * 0.5}{0.5 * 0.65 * 0.8} = \frac{0.252}{0.26} = 0.969$$

Avec la même logique on obtient :

$$P(Orange|long, jaune, sucre) = 0$$

$$P(Autrefruit|long, jaune, sucre) = 0.072$$

On remarque que la probabilité que notre fruit soit une banane $P(\text{Banane} | \text{long, jaune, sucre})$ est largement plus grande que celle des autres probabilités. On classifie notre fruit inconnu comme étant une banane.

Inconvénients :

- l'algorithme Naive Bayes Classifier suppose l'indépendance des variables : C'est une hypothèse forte et qui est violée dans la majorité des cas réels.

3.6.5 Exemple Python

Naive Bayes

```
# example of preparing and making a prediction with a naive bayes model
from sklearn.datasets import make_blobs
from scipy.stats import norm
from numpy import mean
from numpy import std

# fit a probability distribution to a univariate data sample
def fit_distribution(data):
    # estimate parameters
    mu = mean(data)
```

```

sigma = std(data)
print(mu, sigma)
# fit distribution
dist = norm(mu, sigma)
return dist

# calculate the independent conditional probability
def probability(X, prior, dist1, dist2):
    return prior * dist1.pdf(X[0]) * dist2.pdf(X[1])

# generate 2d classification dataset
X, y = make_blobs(n_samples=100, centers=2, n_features=2, random_state=1)
# sort data into classes
Xy0 = X[y == 0]
Xy1 = X[y == 1]
# calculate priors
priory0 = len(Xy0) / len(X)
priory1 = len(Xy1) / len(X)
# create PDFs for y==0
distX1y0 = fit_distribution(Xy0[:, 0])
distX2y0 = fit_distribution(Xy0[:, 1])
# create PDFs for y==1
distX1y1 = fit_distribution(Xy1[:, 0])
distX2y1 = fit_distribution(Xy1[:, 1])
# classify one example
Xsample, ysample = X[0], y[0]
py0 = probability(Xsample, priory0, distX1y0, distX2y0)
py1 = probability(Xsample, priory1, distX1y1, distX2y1)
print('P(y=0 | %s) = %.3f' % (Xsample, py0*100))
print('P(y=1 | %s) = %.3f' % (Xsample, py1*100))
print('Truth: y=%d' % ysample)

```

3.6.6 Exemple sickit-learn

Naive Bayes Sklearn

```

# example of gaussian naive bayes
from sklearn.datasets import make_blobs
from sklearn.naive_bayes import GaussianNB
# generate 2d classification dataset
X, y = make_blobs(n_samples=100, centers=2, n_features=2, random_state=1)
# define the model
model = GaussianNB()
# fit the model
model.fit(X, y)
# select a single sample

```

```

Xsample, ysample = [X[0]], y[0]
# make a probabilistic prediction
yhat_prob = model.predict_proba(Xsample)
print('Predicted Probabilities: ', yhat_prob)
# make a classification prediction
yhat_class = model.predict(Xsample)
print('Predicted Class: ', yhat_class)
print('Truth: y=%d' % ysample)

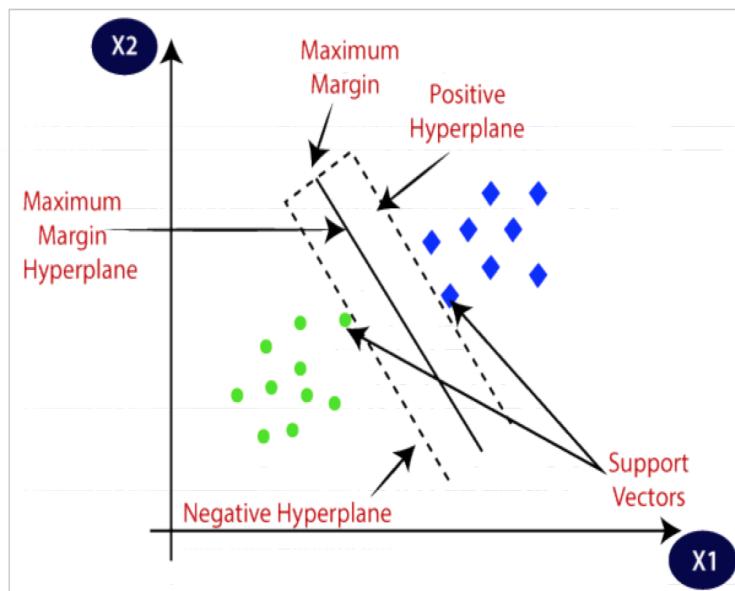
```

3.7 Support Vector Machine

SVM (Vladimir Vapnik) est l'un des algorithmes d'apprentissage supervisé les plus populaires, utilisé pour les problèmes de classification et de régression. Cependant, il est principalement utilisé pour les problèmes de classification dans l'apprentissage automatique.

L'objectif de l'algorithme SVM est de créer la meilleure ligne ou limite de décision capable de séparer l'espace à n dimensions en classes afin que nous puissions facilement placer le nouveau point de données dans la bonne catégorie à l'avenir. Cette frontière de meilleure décision est appelée un hyperplan.

SVM choisit les points/vecteurs extrêmes qui aident à créer l'hyperplan. Ces cas extrêmes sont appelés vecteurs de support et, par conséquent, l'algorithme est appelé machine à vecteur de support. Considérez le diagramme ci-dessous dans lequel deux catégories différentes sont classées à l'aide d'une limite de décision ou d'un hyperplan :



SVM peut être de deux types

- SVM linéaire : SVM linéaire est utilisé pour les données séparables linéairement, ce qui signifie que si un ensemble de données peut être classé en deux classes en utilisant une seule ligne droite, ces données sont alors appelées données séparables linéairement et le classificateur est utilisé appelé classificateur SVM linéaire.

- SVM non linéaire : le SVM non linéaire est utilisé pour les données séparées non linéairement, ce qui signifie que si un ensemble de données ne peut pas être classé à l'aide d'une ligne droite, ces données sont appelées données non linéaires et le classificateur utilisé est appelé non-classificateur SVM linéaire.

Hyperplan et vecteurs de support dans l'algorithme SVM

Hyperplan : il peut y avoir plusieurs lignes/limites de décision pour séparer les classes dans un espace à n dimensions, mais nous devons trouver la meilleure limite de décision qui aide à classer les points de données. Cette meilleure frontière est connue sous le nom d'hyperplan de SVM.

Les dimensions de l'hyperplan dépendent des entités présentes dans le jeu de données, ce qui signifie que s'il y a 2 entités (comme indiqué sur l'image), alors l'hyperplan sera une ligne droite. Et s'il y a 3 caractéristiques, alors l'hyperplan sera un plan à 2 dimensions.

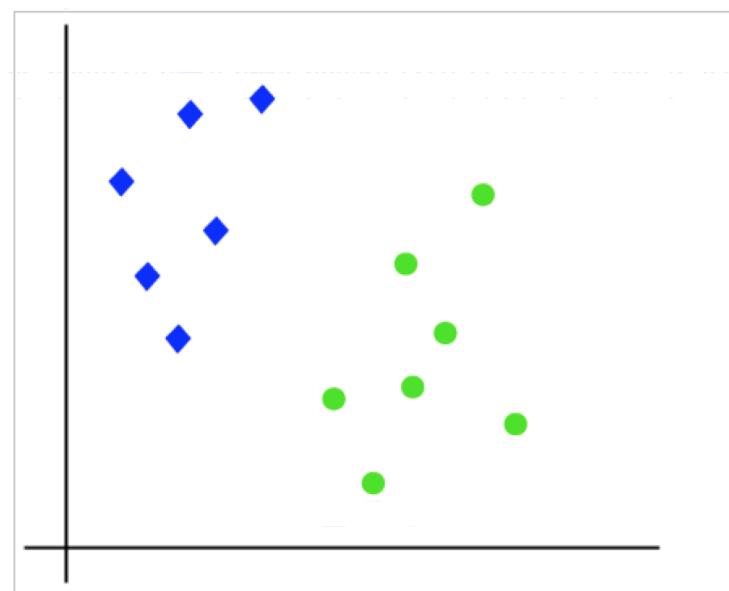
Nous créons toujours un hyperplan qui a une marge maximale, c'est-à-dire la distance maximale entre les points de données.

Support vectors : Les points de données ou vecteurs les plus proches de l'hyperplan et qui affectent la position de l'hyperplan sont appelés vecteurs de support. Puisque ces vecteurs supportent l'hyperplan, donc appelé vecteur de support.

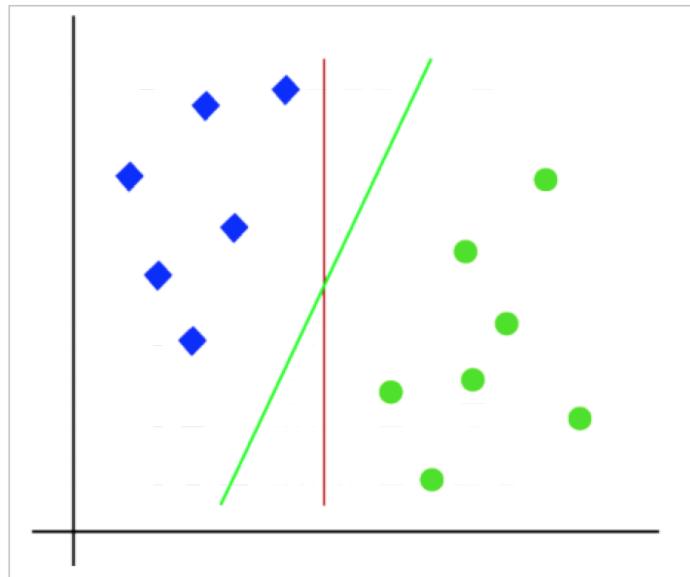
Comment fonctionne SVM ?

3.7.1 SVM linéaire

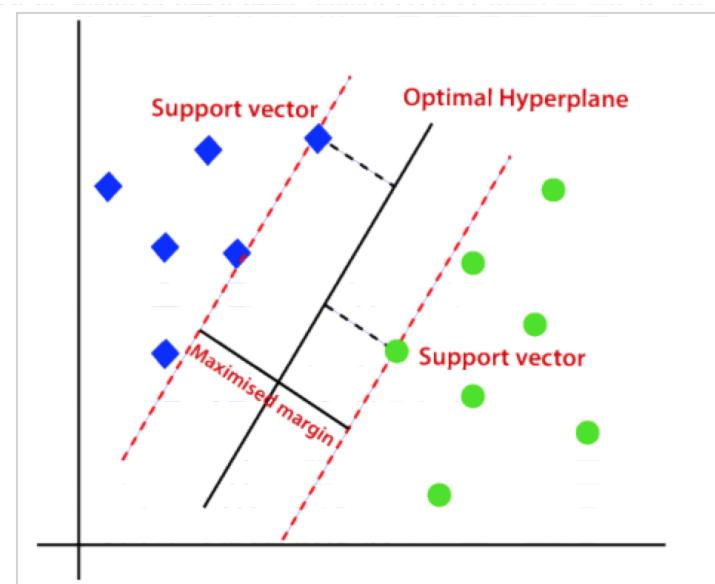
Le fonctionnement de l'algorithme SVM peut être compris à l'aide d'un exemple. Supposons que nous ayons un jeu de données qui a deux balises (vert et bleu) et que le jeu de données a deux caractéristiques x_1 et x_2 . Nous voulons un classificateur capable de classer la paire (x_1, x_2) de coordonnées en vert ou en bleu. Considérez l'image ci-dessous :



Donc, comme il s'agit d'un espace à 2 dimensions, en utilisant simplement une ligne droite, nous pouvons facilement séparer ces deux classes. Mais plusieurs lignes peuvent séparer ces classes. Considérez l'image ci-dessous :

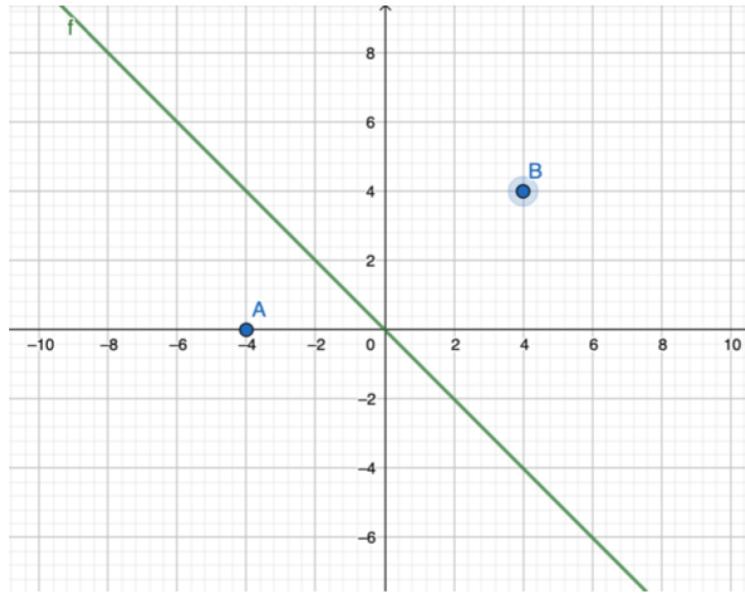


Par conséquent, l'algorithme SVM aide à trouver la meilleure ligne ou limite de décision ; cette meilleure limite ou région est appelée hyperplan. L'algorithme SVM trouve le point le plus proche des lignes des deux classes. Ces points sont appelés vecteurs supports. La distance entre les vecteurs et l'hyperplan est appelée marge. Et le but de SVM est de maximiser cette marge. L'hyperplan avec une marge maximale est appelé l'hyperplan optimal.



Math de SVM linéaire (Prime)

$$Y = w^T X + b \rightarrow Y = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$



$$w^T X + b = 0$$

Si slope = -1 et b = 0 :

$$w^T = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \quad (3.1)$$

$$X = [-4, 0] \quad (3.2)$$

$$w^T X + b = 4 \quad (3.3)$$

positif

Si slope = -1 et b = 0 :

$$w^T = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \quad (3.4)$$

$$X = [4, 4] \quad (3.5)$$

$$w^T X + b = -4 \quad (3.6)$$

négatif

$w^T X + b = 1$ la droite qui passe sur le support vector de la classe positive.

$w^T X + b = -1$ la droite qui passe sur le support vector de la classe négative.

$$d = \frac{|w^T X + B_0|}{\|X\|} \quad (3.7)$$

$$x_1 = w^T X_1 + b = 1$$

$$x_2 = w^T X_2 + b = -1$$

$$x_1 - x_2 \rightarrow w^T(X_1 - X_2) = 2$$

On va éliminer w^T :

$\frac{w^T(X_1 - X_2)}{\|w\|} = \frac{2}{\|w\|} \rightarrow \text{Max}(\frac{2}{\|w\|})$. ou bien $\text{Min}(\|w\|)$, autre écriture maximiser $2/\|w\|^2$ ou bien minimiser $\|w\|^2$, (Hard Margine).

$$\|w\| = \sqrt{(w_1^2 + w_2^2 + \dots + w_n^2)}.$$

$$w^T X + b >= 1 \forall x \in c_1 \dots \beta +$$

$$w^T X + b <= -1 \forall x \in c_2 \dots \beta -$$

Ou bien une forme généralisée : $y_i * w^T x + b >= 1$

Dans le cas où on veut éviter le phénomène de sur apprentissage (SOFT Margine) :

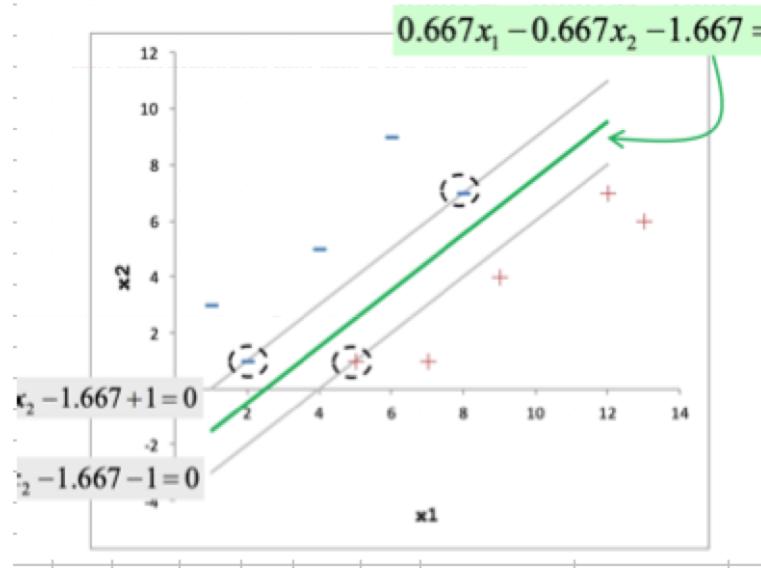
$$\min \frac{1}{2} \|w\|^2 + C \sum_{k=1}^p \zeta_k, C > 0$$

Un nouveau paramètre de régularisation C contrôle le compromis entre la maximisation de la marge et la minimisation de la perte.

Les nouvelles variables ζ_i ajoutent de la flexibilité pour les erreurs de classification du modèle :

Exemple de SVM linéaire (Primal)

X1	X2	Y
1	3	-1
2	1	-1
4	5	-1
6	9	-1
8	7	-1
5	1	1
7	1	1
9	4	1
12	7	1
13	6	1



L'objectif est de calculer les coef Beta :

Selon le graphique on a trois points qui représentent les support vecteurs :

$$S_1 = (2, 1)$$

$$S_2 = (8, 7)$$

$$S_3 = (5, 1)$$

on augmenter les vecteurs en ajoutant 1 : Bais/Inception.

$$\tilde{S}_1 = (2, 1, 1)$$

$$\tilde{S}_2 = (8, 7, 1)$$

$$\tilde{S}_3 = (5, 1, 1)$$

Donc on peut construire un système d'équation linéaire :

$$\sum_i^n \alpha_i \langle x_i \cdot x \rangle$$

$$\alpha_1 \tilde{S}_1 \tilde{S}_1 + \alpha_2 \tilde{S}_2 \tilde{S}_1 + \alpha_3 \tilde{S}_3 \tilde{S}_1 = -1$$

$$\alpha_1 \tilde{S}_1 \tilde{S}_2 + \alpha_2 \tilde{S}_2 \tilde{S}_2 + \alpha_3 \tilde{S}_3 \tilde{S}_2 = -1$$

$$\alpha_1 \tilde{S}_1 \tilde{S}_3 + \alpha_2 \tilde{S}_2 \tilde{S}_3 + \alpha_3 \tilde{S}_3 \tilde{S}_3 = +1$$

$$\alpha_1(2, 1, 1)(2, 1, 1) + \alpha_2(8, 7, 1)(2, 1, 1) + \alpha_3(5, 1, 1)(2, 1, 1) = -1$$

$$\alpha_1(2, 1, 1)(8, 7, 1) + \alpha_2(8, 7, 1)(8, 7, 1) + \alpha_3(5, 1, 1)(8, 7, 1) = -1$$

$$\alpha_1(2, 1, 1)(5, 1, 1) + \alpha_2(8, 7, 1)(5, 1, 1) + \alpha_3(5, 1, 1)(5, 1, 1) = +1$$

$$\alpha_1 6 + \alpha_2 24 + \alpha_3 12 = -1$$

$$\alpha_1 24 + \alpha_2 114 + \alpha_3 48 = -1$$

$$\alpha_1 12 + \alpha_2 48 + \alpha_3 27 = +1$$

$$\alpha_1 = \frac{-17}{6}$$

$$\begin{aligned}\alpha_2 &= \frac{1}{6} \\ \alpha_3 &= 1\end{aligned}$$

$$w = \sum_i^n \alpha_i \tilde{S}_i$$

$$w = \frac{-17}{6}(2, 1, 1) + \frac{1}{6}(8, 7, 1) + (5, 1, 1)$$

$$w = \left(\frac{-17}{3}, \frac{-17}{6}, \frac{-17}{6} \right) + \left(\frac{4}{3}, \frac{7}{6}, \frac{1}{6} \right) + (5, 1, 1)$$

$$w = \begin{pmatrix} -5, 66 + 1, 33 + 5 \\ -2, 83 + 1, 16 + 1 \\ -2, 83 + 0, 166 + 1 \end{pmatrix} \quad (3.8)$$

$$w = \begin{pmatrix} 0, 667 \\ -0, 667 \\ -1, 664 \end{pmatrix} \quad (3.9)$$

Exemple de SVM linéaire (Dual)

On peut définir les support vecteurs par l'intermédiaire des multiplicateurs de Lagrange, à travers une méthode d'optimisation : "Sequential Minimal Optimization".

Pour résoudre le problème de programmation quadratique ci-dessus avec des contraintes d'inégalité, nous pouvons utiliser la méthode des multiplicateurs de Lagrange. La fonction de Lagrange est donc (Dual forme) :

$$L(w, w_0, \alpha) = \frac{1}{2} \|w\|^2 + \sum_i \alpha_i (y_i (w^T x_i + w_0) - 1)$$

Pour résoudre ce problème, nous posons les questions suivantes :

$$\frac{\partial L}{\partial w} = 0, \frac{\partial L}{\partial \alpha} = 0, \frac{\partial L}{\partial w_0} = 0$$

Nous devons maximiser ce qui précède sous réserve de ce qui suit :

$$\frac{\partial L}{\partial w} = 0 \rightarrow w = \sum_i \alpha_i y_i x_i$$

et

$$\frac{\partial L}{\partial w_0} = 0 \rightarrow \sum_i \alpha_i y_i = 0$$

En les substituant au deuxième terme de la fonction de Lagrange, on obtient le problème d'optimisation suivant (problème quadratique), également appelé problème dual :

$$L_d = -\frac{1}{2} \sum_i \sum_k \alpha_i \alpha_k y_i y_k (x_i)^T (x_k) + \sum_i \alpha_i$$

L'avantage de ce qui précède est que nous disposons d'une expression pour w en termes de multiplicateurs de Lagrange. La fonction objective ne comporte aucun terme. Un multiplicateur de Lagrange est associé à chaque point de données. Le calcul de w_0 est également expliqué plus loin.

La classification de tout point de test x peut être déterminée à l'aide de cette expression :

$$y(x) = \sum_i \alpha_i y_i x^T x_i + w_0$$

Une valeur positive de $y(x)$ implique que x appartient à 1 et une valeur négative signifie que x appartient à -1.

Le problème dual est plus facile à résoudre puisqu'il ne comporte que les multiplicateurs de Lagrange. En outre, le fait que le problème dual dépende des produits intérieurs des données d'apprentissage est utile pour étendre les SVM linéaires à l'apprentissage de frontières non linéaires.

Kuhn-Tucker Conditions

En outre, les conditions de Karush-Kuhn-Tucker (KKT) sont satisfaites par le problème d'optimisation sous contrainte ci-dessus, comme indiqué ci-dessous :

$$\alpha_i \geq 0$$

$$y_i y(x_i) - 1 \geq 0$$

$$\alpha_i (y_i y(x_i) - 1) = 0$$

Les conditions KKT imposent que, pour chaque point de données, l'une des conditions suivantes soit remplie :

Le multiplicateur de Lagrange est nul, c'est-à-dire que $\alpha_i = 0$. Ce point ne joue donc aucun rôle dans la classification. OR

$t_s y(x_s) = 1$ et $\alpha_i > 0$: Dans ce cas, le point de données joue un rôle dans la détermination de la valeur de w . Un tel point est appelé vecteur de support.

Calcul de w_0 :

Pour w_0 , nous pouvons sélectionner n'importe quel vecteur de support et résoudre

$$t_s y(x_s) = 1$$

en nous donnant :

$$t_s \left(\sum_i \alpha_i y_i x_s^T x_i + w_0 \right) = 1$$

Sequential Minimal Optimization

L'optimisation minimale séquentielle (SMO) est un algorithme permettant de résoudre le problème de programmation quadratique (QP) qui se pose lors de l'apprentissage des machines à

vecteurs de support (SVM). SMO est un algorithme itératif permettant de résoudre le problème d'optimisation décrit ci-dessus.

L'algorithme procède comme suit :

L'algorithme SMO simplifié prend les paramètres α , α_i et α_j , et les optimise. Pour ce faire, nous itérons sur tout α_i , $i = 1, \dots, M$. Si α_i ne satisfait pas aux conditions de Karush-Kuhn-Tucker dans une certaine tolérance numérique, nous sélectionnons α_j au hasard parmi les $M - 1$ α restants et optimisons α_i et α_j . La fonction suivante nous aidera à sélectionner j au hasard :

Si aucun des α n'est modifié après quelques itérations sur tous les α_i , alors l'algorithme se termine. Nous devons également trouver les bornes L et H :

- if $y(i) \neq y(j)$, $L = \max(0, \alpha_j - \alpha_i)$, $H = \min(C, C + \alpha_j - \alpha_i)$
- if $y(i) = y(j)$, $L = \max(0, \alpha_i + \alpha_j - C)$, $H = \min(C, \alpha_i + \alpha_j)$

Où C est un paramètre de régularisation.

Nous allons maintenant trouver que α_j est donné par :

$$\alpha_j := \alpha_j - \frac{y^{(i)}(E_i - E_j)}{\eta}$$

d'où,

$$E_k = f(x^{(k)}) - y^{(k)}$$

$$\eta = 2 \langle x^{(i)}, x^{(j)} \rangle - \langle x^{(i)}, x^{(i)} \rangle - \langle x^{(j)}, x^{(i)} \rangle$$

d'où

$$\langle X, Y \rangle = X \cdot Y^T = \Sigma X_i Y_i$$

Si cette valeur se trouve en dehors des limites L et H , nous devons réduire la valeur de α_j pour la ramener dans cette fourchette :

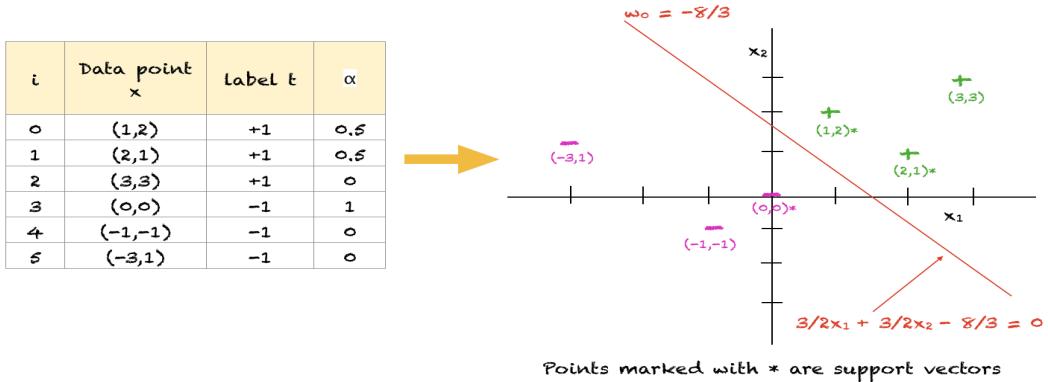
$$\alpha_j = \begin{cases} H & \text{if } \alpha_j > H \\ \alpha_j & \text{if } H \leq \alpha_j \leq L \\ L & \text{if } \alpha_j < L \end{cases}$$

Enfin, nous pouvons trouver la valeur de α_i . Celle-ci est donnée par :

$$\alpha_i := \alpha + y^{(i)} y^{(j)} (\alpha_j - \alpha_j^{(old)})$$

ou $\alpha_j^{(old)}$ is the value of α_j apres optimisation.

Cet exemple vous montrera que le modèle n'est pas aussi complexe qu'il n'y paraît.



SVM avec Sklearn

Linear SVM

```

import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

# importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25,
    random_state=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)

from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)

#Predicting the test set result
y_pred= classifier.predict(x_test)

```

```

#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)

from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].
max() + 1, step = 0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01)
)
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).
reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

```

```

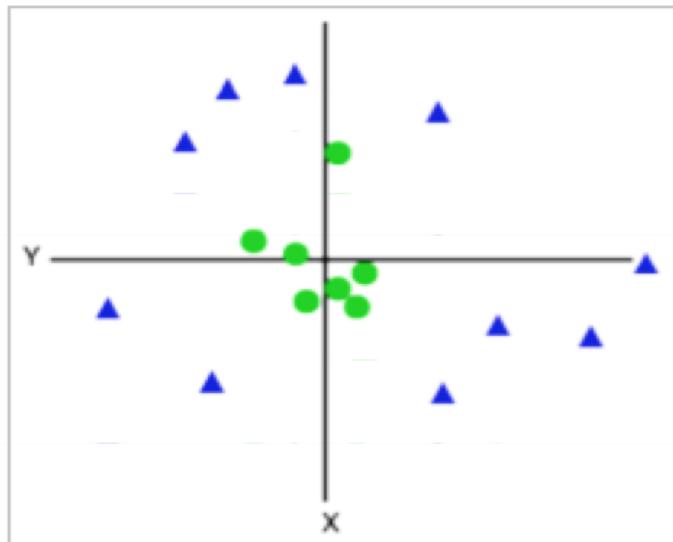
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].
max() + 1, step = 0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01)
)
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).
reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c = ListedColormap((
    'red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()

```

```
mtp.show()
```

3.7.2 SVM Non-linéaire

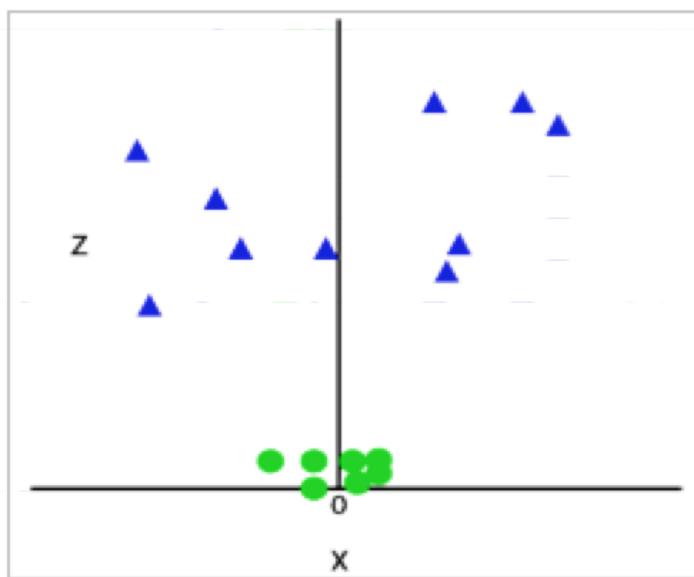
Si les données sont disposées linéairement, nous pouvons les séparer en utilisant une ligne droite, mais pour les données non linéaires, nous ne pouvons pas tracer une seule ligne droite. Considérez l'image ci-dessous :



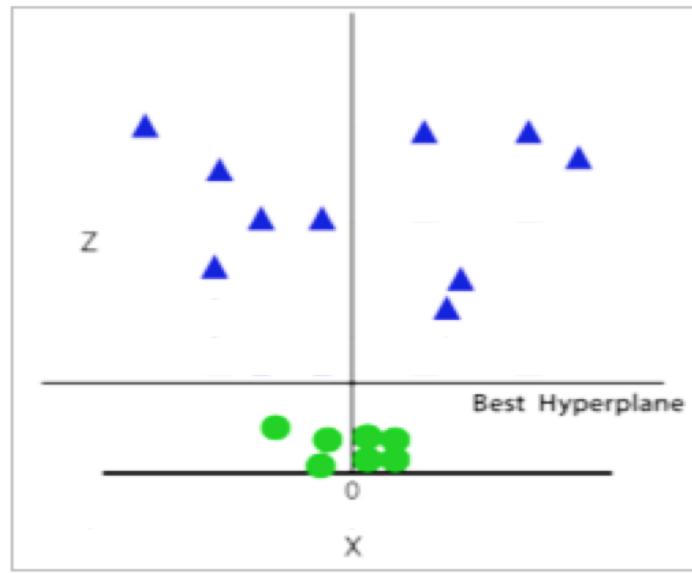
Donc, pour séparer ces points de données, nous devons ajouter une dimension supplémentaire. Pour les données linéaires, nous avons utilisé deux dimensions x et y, donc pour les données non linéaires, nous ajouterons une troisième dimension z. Il peut être calculé comme suit :

$$z = x^2 + y^2$$

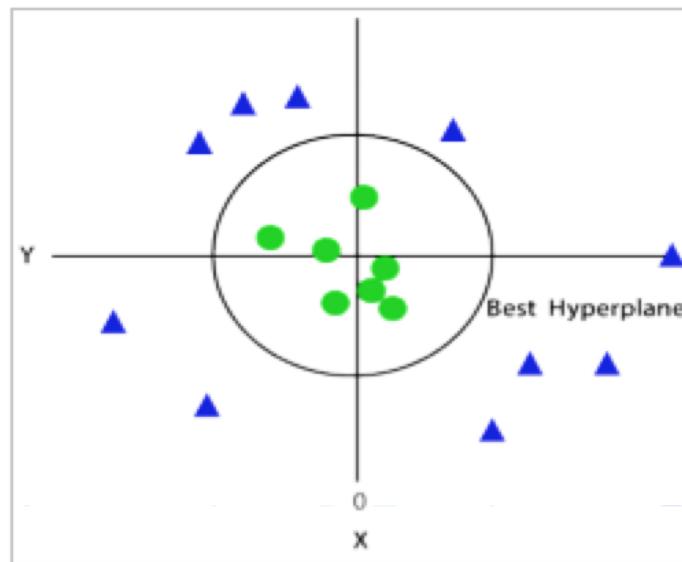
En ajoutant la troisième dimension, l'espace échantillon deviendra comme l'image ci-dessous :



Alors maintenant, SVM divisera les ensembles de données en classes de la manière suivante. Considérez l'image ci-dessous :



Puisque nous sommes dans l'espace 3D, il ressemble donc à un plan parallèle à l'axe des x. Si nous le convertissons dans un espace 2d avec $z=1$, il deviendra alors :



L'exemple le plus simple de fonction noyau est le noyau linéaire :

$$K(x_i, x_j) = x_i^T x_j$$

Le noyau polynomial :

$$K(x_i, x_j) = (x_i^T x_j + 1)^d$$

Le noyau gaussien :

$$K(x_i, x_j) = \exp\left(\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

Le noyau sigmoïde :

$$K(x_i, x_j) = \tanh(\alpha x_i^T x_j + c)$$

```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data" #
    Assign column names to the dataset
colnames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
    # Read dataset to pandas dataframe
irisdata = pd.read_csv(url, names=colnames)

X = irisdata.drop('Class', axis=1)
y = irisdata['Class']

from sklearn.model_selection import train_test_split X_train, X_test, y_train,
y_test = train_test_split(X, y, test_size = 0.20)

#Polynomial Kernel

from sklearn.svm import SVC
svclassifier = SVC(kernel='poly', degree=8)
svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

#Guassian Kernel

from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf')
svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

#Sigmoid Kernel
from sklearn.svm import SVC
svclassifier = SVC(kernel='sigmoid')

```

```

svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

```

Si nous comparons les performances des différents types de noyaux, nous pouvons clairement voir que le noyau sigmoïde est le moins performant. Cela est dû au fait que la fonction sigmoïde renvoie deux valeurs, 0 et 1, elle est donc plus adaptée aux problèmes de classification binaire. Cependant, dans notre cas, nous avions trois classes de sortie.

Parmi le noyau gaussien et le noyau polynomial, nous pouvons voir que le noyau gaussien a atteint un taux de prédiction parfait de 100 % tandis que le noyau polynomial a mal classé une instance. Par conséquent, le noyau gaussien a légèrement mieux performé. Cependant, il n'y a pas de règle absolue quant au noyau qui fonctionne le mieux dans chaque scénario. Il s'agit de tester tous les noyaux et de sélectionner celui avec les meilleurs résultats sur votre jeu de données de test.

3.7.3 Formulation du problème de programmation quadratique avec la cvxopt

Pour Svm, nous devons maximiser :

$$\sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m \phi(x_n, x_m)$$

La fonction de qp dans cvxopt prend la forme de :

$$\frac{1}{2} x^T P x + q^T x$$

qui doit être minimisé , Nous devons donc maintenant minimiser

$$-\sum_{n=1}^N \alpha_n + \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m \phi(x_n, x_m)$$

Ici, X représente les alphas :

$$x = \begin{bmatrix} \alpha_1 \\ \vdots \\ \vdots \\ \alpha_n \end{bmatrix}$$

Comme pour q :

$$q = -[1 \dots 1]$$

Comme Pour P :

$$P = \begin{bmatrix} y_1 y_1 \phi(x_1, x_1) & \dots & y_N y_1 \phi(x_N, x_1) \\ \vdots & \ddots & \vdots \\ y_1 y_N \phi(x_1, x_N) & \dots & y_N y_N \phi(x_N, x_N) \end{bmatrix}$$

Dans le cas d'un noyau linéaire, nous pouvons obtenir k à partir de :

$$\phi(x_m, x_n) = x_m^T * x_n$$

Dans le cas d'un noyau polynomial

$$\phi(x_m, x_n) = (1 + x_m^T * x_n)^d$$

Dans le cas d'un noyau rbf

$$\phi(x_m, x_n) = e^{\frac{-||x_m - x_n||^2}{k*\gamma^2}}$$

k est le nombre de caractéristiques.

L'étape suivante est celle des conditions, nos conditions sont :

$$\sum_{n=1}^N \alpha_n y_n = 0$$

$$0 \leq \alpha_n \leq C$$

Les conditions de la programmation quadratique se présentent sous la forme :

$$Gx \leq h, Ax = b$$

Nous pouvons maintenant trouver A,b,h,G :

$$A = [y_1 \dots \dots y_N]$$

$$b = 0$$

Pour G, b nous devons séparer les conditions, nous avons donc :

$$\alpha_n \leq C, -\alpha_n \leq 0$$

G devient donc

$$G = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 \\ -1 & 0 & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 \end{bmatrix}_{rows=2N, cols=N}$$

h devient donc

$$h = \begin{bmatrix} C \\ \vdots \\ C \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{rows=2N}$$

Et maintenant, puisque nous pouvons représenter q,p,G,A,b :

Nous résolvons la valeur de α et l'utilisons pour calculer les poids et à partir de b.

$$W = \sum_{n=1}^N \alpha_n x_n y_n$$

et b :

$$b = \frac{\sum_{i \in SV} (y_i - \sum_{j \in SV} (\alpha_j y_j \phi(x_j, x_i)))}{\text{len}(SV)}$$

SVM with cvxopt

```
import pandas as pd
import numpy as np;

%pip install cvxopt
from cvxopt import matrix
from cvxopt.solvers import qp

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

class svm_reg :
    def linear_kernal (self,x1 ,x2) :
        return x1.T@x2

    def polynomial_kernel(self,x1, x2):
        return ( x1.T @ x2 +1) ** self.param

    def rbf_kernel(self, x1 , x2) :
        return np.exp(- np.linalg.norm(x1 -x2)**2 / (self.nf * self.param**2))

    def __init__(self,m_type : str = 'linear' , C : float = 6 ,param = 1 ) -> None:
        self.C = C;
        self.param = param
        self.m_type = m_type
        if m_type =='linear' :
            self.kernel = self.linear_kernal
        elif m_type == 'polynomial' :
            self.kernel = self.polynomial_kernel
        elif m_type == 'rbf' :
            self.kernel = self.rbf_kernel

    def fit(self,X :np.array, Y : np.array) :
        ''' Initializes the model and calculates the weights vector and b '''
        n ,self.nf = X.shape
```

```

K = np.zeros((n, n)) # Initialize k matrix (linear model)
for i in range(n):
    for j in range(n):
        K[i,j] = self.kernel(X[i] , X[j])

P = matrix(np.outer(Y,Y) * K) # outer product of Y and itself * the k
matrix

q = matrix(-1*np.ones(n))
b = matrix(0.0)
A = matrix(Y*1.0,(1,n))

g = np.diag(np.ones(n))      # first diagonal matrix (n positive 1)
g2 = -1* np.diag(np.ones(n)) # second diagonal matrix (n negative 1)
g = np.concatenate((g,g2),axis=0) # combine the two matrices vertically
g = matrix(g)

h = self.C * np.ones(n)      # vector of C
h2 = np.zeros(n)            # vector of zeros
h= np.concatenate((h.T,h2.T),axis = 0) # combine vertically
h = matrix(h)

solution = qp(P, q, g, h, A, b) # solve for alpha

alphas = np.ravel(solution['x']) # Get alphas and combine them into an
array

self.w = np.zeros(self.nf)      # calculate W
for alpha , y,x in zip(alphas,Y,X) :
    self.w += alpha *y*x

idxs = alphas > 1e-4
self.svs_x = X[idxs]
self.svs_y = Y[idxs]
self.alphas = alphas[idxs]

# Calculate b
self.b = np.mean([self.svs_y[i] - sum([alpha * y * self.kernel(x,self.svs_x
[i]) for alpha , y ,x in zip(self.alphas , self.svs_y , self.svs_x )]) for i in
range(len(self.svs_x))])

def _predict(self, inpt : np.array)      :
    if self.m_type == 'linear' :
        return self.w@ inpt + self.b
    else :

```

```

        return sum([(alpha*y * self.kernel(sv_x , inpt)) for alpha,y ,sv_x in
zip(self.alphas, self.svs_y , self.svs_x) ]) +self.b

def predict(self, inpt : np.array) :
    ''' takes input vector and calculates the expected scalar result '''
    inpt = np.array(inpt)
    if len(inpt.shape ) == 1 :
        return self._predict(inpt)
    else :
        res = np.zeros(len(inpt))
        for i in range(len(inpt)) :
            res[i] = self._predict(inpt[i])
        return res;

class svm_class :
    ''' One vs all classifier , Uses set of svm_reg models for each class in the
output '''
    def __init__(self,m_type : str = 'linear' , C : float = 6 ,param = 1 ) -> None:
        self.C = C;
        self.param = param
        self.m_type = m_type

    def fit(self, X , Y) :
        classes = set(Y)
        models_dict = dict.fromkeys(classes)
        for cl in classes :
            idxs = (Y== cl)
            Y_modified =-1* np.ones(len(Y))
            Y_modified[idxs] = 1
            models_dict[cl] = svm_reg(m_type= self.m_type ,C = self.C , param= self
.param)
            models_dict[cl].fit(X,Y_modified.T)

        self.models_dict = models_dict

    def _predict(self,inpt) :
        results_dict = dict.fromkeys(self.models_dict.keys())
        for cl in self.models_dict.keys() :
            results_dict[cl] = self.models_dict[cl].predict(inpt)

        return max(results_dict, key=results_dict.get)

```

```

def predict(self,inpt) :
    inpt = np.array(inpt)
    if len(inpt.shape ) == 1 :
        self._predict(inpt)
    else :
        res = np.zeros(len(inpt))
        for i in range(len(inpt)) :
            res[i] = self._predict(inpt[i])
    return res;

#path = 'https://archive.ics.uci.edu/ml/machine-learning-databases/ionosphere/
ionosphere.data'
path = 'ionosphere.data'
data = pd.read_csv(path ,header= None);
data.head()

input_data = data.iloc[:,0:34].copy()
output_data = data.iloc[:,34].copy()
output_data.replace('g',1 ,inplace=True)
output_data.replace('b',-1 ,inplace=True)

input_train ,input_test , output_train , output_test = train_test_split(input_data
, output_data , test_size=1/5, shuffle = 1, random_state=50)

C = 1000
gamma = 0.1
model = svm_class(m_type='rbf' , C= C ,param= gamma)

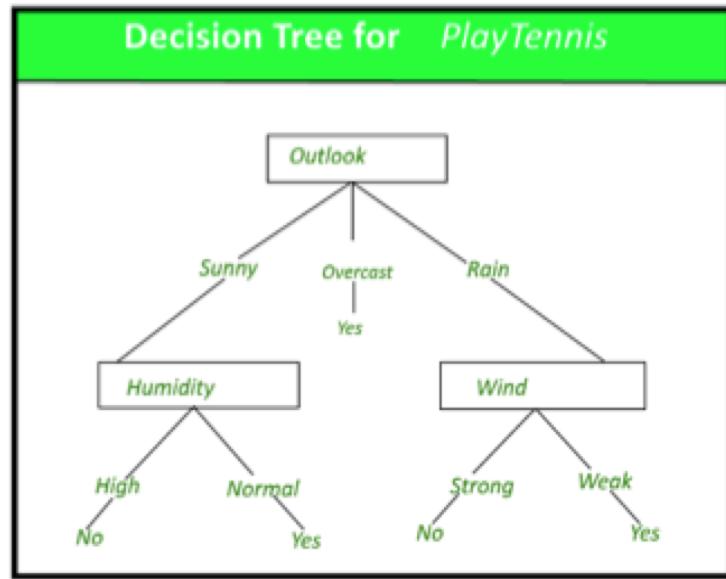
x = np.array(input_train)
y = np.array(output_train)
model.fit(x,y)

res_model = model.predict(np.array(input_test))
print('Model accuracy =' , accuracy_score(res_model,output_test) *100)

```

3.8 Arbre de décision

Un arbre de décision est un outil d'aide à la décision représentant un ensemble de choix sous la forme graphique d'un arbre. Les différentes décisions possibles sont situées aux extrémités des branches, et sont atteints en fonction de décisions prises à chaque étape.



Il a l'avantage d'être lisible et rapide à exécuter. Il s'agit de plus d'une représentation calculable automatiquement par des algorithmes d'apprentissage supervisé.

Dans la littérature il existe pas mal d'algorithmes pour établissement d'un arbre de décision citons : ID3, CART, C4.5.

ID3 (C4.5 / C 5) : ID3 a été développé en 1986 par Ross Quinlan. L'algorithme crée un arbre multi-voies, trouvant pour chaque noeud la caractéristique catégorielle qui générera le gain d'information le plus important pour les cibles catégorielles. Les arbres sont cultivés à leur taille maximale, puis une étape d'élagage est généralement appliquée pour améliorer la capacité de l'arbre à généraliser aux données invisibles. (Entropie et Méthode Gain)

CART : CART est très similaire à C4.5, mais il diffère en ce sens qu'il prend en charge les variables cibles numériques (régression) et ne calcule pas les ensembles de règles. CART construit des arbres binaires à l'aide de la fonctionnalité et du seuil générant le gain d'informations le plus élevé sur chaque noeud. (Index de Gini)

ID 3 exemple numérique

Le tableau suivant indique les facteurs décisionnels pour jouer au tennis à l'extérieur au cours des 14 derniers jours.

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

On peut résumer l'algorithme ID3 comme illustré ci-dessous :

$$Entropy(S) = \Sigma - p(I).logp(I)$$

$$Gain(S, A) = Entropy(S) - \Sigma[p(S|A).Entropy(S|A)]$$

L'entropie est la mesure de l'impureté, du désordre ou de l'incertitude dans un tas d'exemples.

On doit d'abord calculer l'entropie. La colonne Décision comprend 14 instances et comprend deux libellés : oui et non. Il y a 9 décisions étiquetées oui et 5 décisions étiquetées non.

$$Entropy(Decision) = -p(Yes).logp(Yes) - p(No).logp(No)$$

$$Entropy(Decision) = -(9/14).log(9/14) - (5/14).log(5/14) = 0.940$$

Maintenant, nous devons trouver le facteur le plus déterminant pour la prise de décision.

Facteur « « Wind » sur décision

$$Gain(Decision, Wind) = Entropy(Decision) - \Sigma[p(Decision|Wind).Entropy(Decision|Wind)]$$

L'attribut Wind a deux étiquettes : Weak et Strong. Nous réfléchirions à la formule.

$$\text{Gain}(\text{Decision}, \text{Wind}) = \text{Entropy}(\text{Decision}) - [p(\text{Decision}|\text{Wind}=\text{Weak}) \cdot \text{Entropy}(\text{Decision}|\text{Wind}=\text{Weak})] - [p(\text{Decision}|\text{Wind}=\text{Strong}) \cdot \text{Entropy}(\text{Decision}|\text{Wind}=\text{Strong})]$$

Maintenant, on doit calculer $(\text{Decision}|\text{Wind}=\text{Weak})$ et $(\text{Decision}|\text{Wind}=\text{Strong})$ respectivement.

Facteur Weak wind factor sur décision

Il y a 8 instances pour Weak Wind. La décision de 2 éléments est non et 6 éléments sont oui comme illustré ci-dessous. $\text{Entropy}(\text{Decision}|\text{Wind}=\text{Weak}) = - p(\text{No}) \cdot \log p(\text{No}) - p(\text{Yes}) \cdot \log p(\text{Yes})$

$$\text{Entropy}(\text{Decision}|\text{Wind}=\text{Weak}) = - (2/8) \cdot \log(2/8) - (6/8) \cdot \log(6/8) = 0.811$$

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
13	Overcast	Hot	Normal	Weak	Yes

Ici, il y a 6 instances pour Strong wind . La décision est divisée en deux parties égales.

$$\text{Entropy}(\text{Decision}|\text{Wind}=\text{Strong}) = - p(\text{No}) \cdot \log p(\text{No}) - p(\text{Yes}) \cdot \log p(\text{Yes})$$

$$\text{Entropy}(\text{Decision}|\text{Wind}=\text{Strong}) = - (3/6) \cdot \log(3/6) - (3/6) \cdot \log(3/6) = 1$$

On peut maintenant revenir à l'équation Gain (Decision, Wind).

$$\text{Gain}(\text{Decision}, \text{Wind}) = \text{Entropy}(\text{Decision}) - [p(\text{Decision}|\text{Wind}=\text{Weak}) \cdot \text{Entropy}(\text{Decision}|\text{Wind}=\text{Weak})] - [p(\text{Decision}|\text{Wind}=\text{Strong}) \cdot \text{Entropy}(\text{Decision}|\text{Wind}=\text{Strong})] = 0.940 - [(8/14) \cdot 0.811] - [(6/14) \cdot 1] = 0.048$$

Les calculs pour la colonne de Wind sont terminés. Maintenant, nous devons appliquer les mêmes calculs aux autres colonnes pour trouver le facteur le plus déterminant dans la décision.

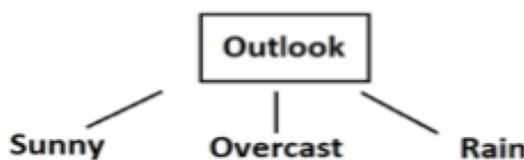
1. Gain(Decision, Outlook) = 0.246
2. Gain(Decision, Temperature) = 0.029
3. Gain(Decision, Humidity) = 0.151

Comme on le voit, le facteur de perspective sur la décision produit le score le plus élevé. C'est pourquoi la décision de perspective apparaîtra dans le nœud racine de l'arbre.

Day	Outlook	Temp.	Humidity	Wind	Decision
2	Sunny	Hot	High	Strong	No
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
14	Rain	Mild	High	Strong	No

Overcast outlook sur décision

Fondamentalement, la décision sera toujours oui si outlook sont overcast.



Sunny outlook sur décision

Day	Outlook	Temp.	Humidity	Wind	Decision
3	Overcast	Hot	High	Weak	Yes
7	Overcast	Cool	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes

Ici, il y a 5 instances pour les perspectives ensoleillées. La décision serait probablement de 3/5 non, 2/5 oui.

1. Gain(Outlook=Sunny|Temperature) = 0.570
2. Gain(Outlook=Sunny|Humidity) = 0.970
3. Gain(Outlook=Sunny|Wind) = 0.019

Maintenant, l'humidité est la décision car elle produit le meilleur score si les perspectives sont ensoleillées.

À ce stade, la décision sera toujours No si Humidity est High.

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

D'un autre côté, la décision sera toujours Yes si Humidity était Normal.

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No

Enfin, cela signifie que nous devons vérifier Humidity et décider si les perspectives sont Sunny.

Day	Outlook	Temp.	Humidity	Wind	Decision
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Rain outlook sur décision

Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

On peut calculer les trois Gain de Outlook=Rain selon , Temperature, Humidity, Wind.

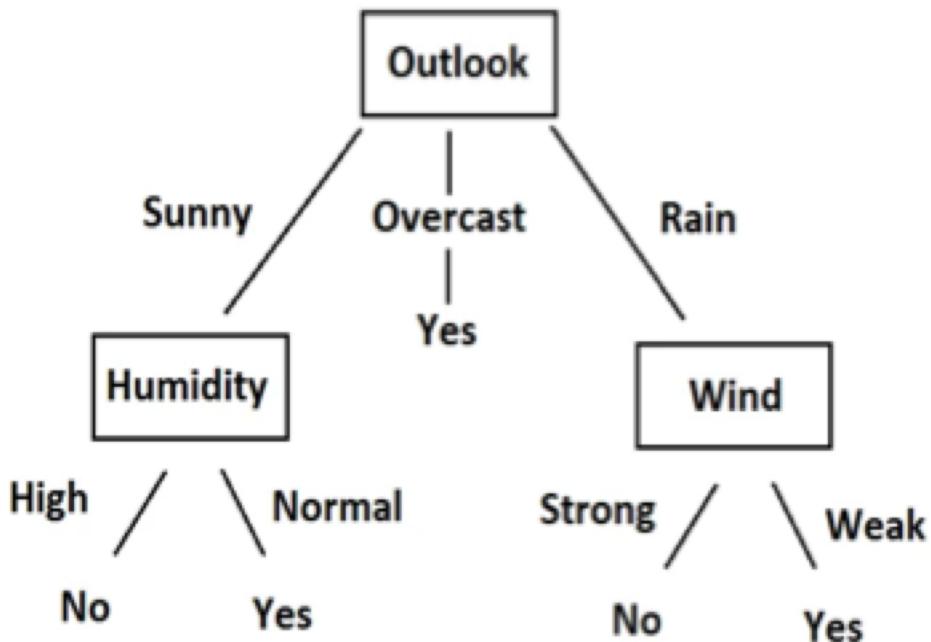
1. Gain(Outlook=Rain | Temperature)
2. Gain(Outlook=Rain | Humidity)
3. Gain(Outlook=Rain | Wind)

Ici, visuellement on peut constater que le Wind produit le score le plus élevé s'il fait Rain. C'est la raison pour laquelle on doit vérifier l'attribut de Wind au 2e niveau si les prévisions sont Rain. Ainsi, il est révélé que la décision sera toujours yes si le Wind est Weak et si Rain.

Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes

Day	Outlook	Temp.	Humidity	Wind	Decision
6	Rain	Cool	Normal	Strong	No
14	Rain	Mild	High	Strong	No

La construction de l'arbre de décision est donc terminée. On peut utiliser les règles suivantes pour la prise de décision.



Arbre de décision

```
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
```

```

from sklearn.model_selection import train_test_split # Import train_test_split
function
from sklearn import metrics #Import scikit-learn metrics module for accuracy
calculation

col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("pima-indians-diabetes.csv", header=None, names=col_names)

#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1) # 70% training and 30% test

# Create Decision Tree classifier object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifier
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

pip install graphviz
pip install pydotplus

from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
filled=True, rounded=True,
special_characters=True, feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())

```

3.9 Ensemble Learning

L'apprentissage d'ensemble est une métaproche générale de l'apprentissage automatique qui vise à améliorer les performances prédictives en combinant les prédictions de plusieurs modèles. Bien qu'il existe un nombre apparemment illimité d'ensembles qu'on peut développer pour le problème de modélisation prédictive, trois méthodes dominent le domaine de l'apprentissage d'ensemble. À tel point que, plutôt que des algorithmes à proprement parler, il s'agit d'un domaine d'étude qui a donné naissance à de nombreuses méthodes plus spécialisées.

Les trois principales classes de méthodes d'apprentissage d'ensemble sont le bagging, le stacking et le boosting, et il est important d'avoir une compréhension détaillée de chaque méthode et de les prendre en compte dans des projets de modélisation prédictive.

3.9.1 Techniques Simple

Vote maximal (Max Voting)

La méthode du vote maximal est généralement utilisée pour les problèmes de classification. Dans cette technique, plusieurs modèles sont utilisés pour faire des prédictions pour chaque point de données. Les prédictions de chaque modèle sont considérées comme un "vote". Les prédictions obtenues par la majorité des modèles sont utilisées comme prédition finale.

Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final Rating
5	4	5	4	4	4

Max Voting

```
from sklearn.ensemble import VotingClassifier
model1 = LogisticRegression(random_state=1)
model2 = tree.DecisionTreeClassifier(random_state=1)
model = VotingClassifier(estimators=[('lr', model1), ('dt', model2)], voting='hard')
model.fit(x_train,y_train) model.score(x_test,y_test)
```

Calcul de la moyenne (Averaging)

Comme pour la technique du vote maximal, plusieurs prédictions sont faites pour chaque point de données dans le calcul de la moyenne. Dans cette méthode, nous prenons la moyenne des prédictions de tous les modèles et l'utilisons pour faire la prédition finale. Le calcul de la moyenne peut être utilisé pour faire des prédictions dans les problèmes de régression ou pour calculer les probabilités dans les problèmes de classification.

Par exemple, dans le cas ci-dessous, la méthode de calcul de la moyenne prendrait la moyenne de toutes les valeurs.

Soit $(5+4+5+4+4)/5 = 4,4$

Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final Rating
5	4	5	4	4	4.4

```

model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()
model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)
pred1=model1.predict_proba(x_test)
pred2=model2.predict_proba(x_test)
pred3=model3.predict_proba(x_test)
finalpred=(pred1+pred2+pred3)/3

```

Moyenne pondérée (Weighted Average) :

Il s'agit d'une extension de la méthode de calcul de la moyenne. Tous les modèles se voient attribuer des poids différents qui définissent l'importance de chaque modèle pour la prédiction. Par exemple, si deux de vos collègues sont des critiques, tandis que d'autres n'ont aucune expérience préalable dans ce domaine, les réponses de ces deux amis auront plus d'importance que celles des autres personnes.

Le résultat est calculé comme suit : $[(5*0,23) + (4*0,23) + (5*0,18) + (4*0,18) + (4*0,18)] = 4,41$.

-	Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final Rating
weight	0,23	0,23	0,18	0,18	0,18	
rating	5	4	5	4	4	4.41

Weighted Average

```

model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()
model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)
pred1=model1.predict_proba(x_test)
pred2=model2.predict_proba(x_test)
pred3=model3.predict_proba(x_test)
finalpred=(pred1*0.3+pred2*0.3+pred3*0.4)

```

3.9.2 Techniques Avancées

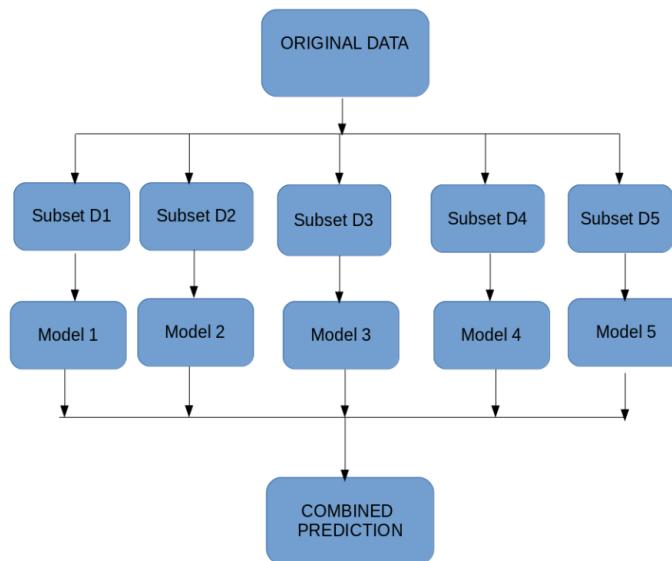
Bagging

L'idée derrière le bagging est de combiner les résultats de plusieurs modèles (par exemple, tous les arbres de décision) pour obtenir un résultat généralisé. Voici une question : Si vous créez tous les modèles sur le même ensemble de données et que vous les combinez, cela sera-t-il utile ? Il y a de fortes chances que ces modèles donnent le même résultat puisqu'ils reçoivent les mêmes données. Comment résoudre ce problème ? L'une des techniques est le bootstrapping.

Le bootstrap est une technique d'échantillonnage qui consiste à créer des sous-ensembles d'observations à partir de l'ensemble de données original, avec remplacement. La taille des sous-ensembles est la même que celle de l'ensemble original.

La technique du bagging (ou Bootstrap Aggregating) utilise ces sous-ensembles (bags) pour se faire une idée juste de la distribution (ensemble complet). La taille des sous-ensembles créés pour le bagging peut être inférieure à celle de l'ensemble original.

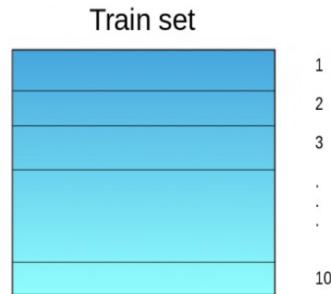
1. Plusieurs sous-ensembles sont créés à partir de l'ensemble de données original, en sélectionnant les observations avec remplacement.
2. Un modèle de base (modèle faible) est créé sur chacun de ces sous-ensembles.
3. Les modèles fonctionnent en parallèle et sont indépendants les uns des autres.
4. Les prédictions finales sont déterminées en combinant les prédictions de tous les modèles.



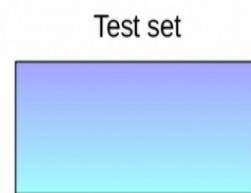
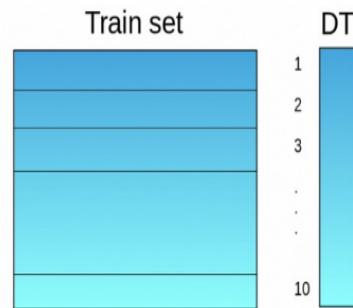
Stacking

Le Stacking est une technique d'apprentissage d'ensemble qui utilise les prédictions de plusieurs modèles (par exemple, arbre de décision, knn ou svm) pour construire un nouveau modèle. Ce modèle est utilisé pour faire des prédictions sur l'ensemble de test.

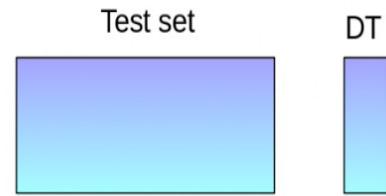
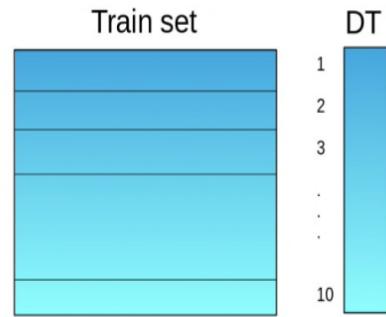
1. Le train est divisé en 10 parties.



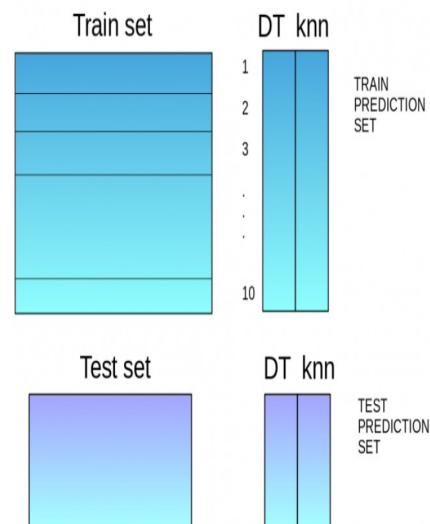
2. Un modèle de base (supposé être un arbre de décision) est ajusté sur 9 parties et des prédictions sont faites pour la 10ème pièce. Cette opération est effectuée pour chaque partie de l'ensemble.



3. Le modèle de base (dans ce cas, l'arbre de décision) est ensuite ajusté sur l'ensemble des données de formation.
4. À l'aide de ce modèle, des prédictions sont faites sur l'ensemble de test.

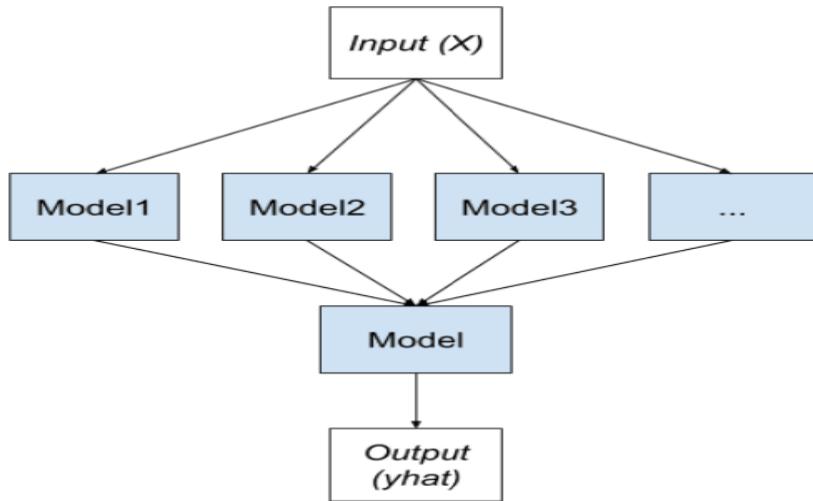


5. Les étapes 2 à 4 sont répétées pour un autre modèle de base (par exemple knn), ce qui permet d'obtenir un autre ensemble de prédictions pour l'ensemble de formation et l'ensemble de test.



6. les prédictions de l'ensemble d'entraînement sont utilisées comme caractéristiques pour construire un nouveau modèle..

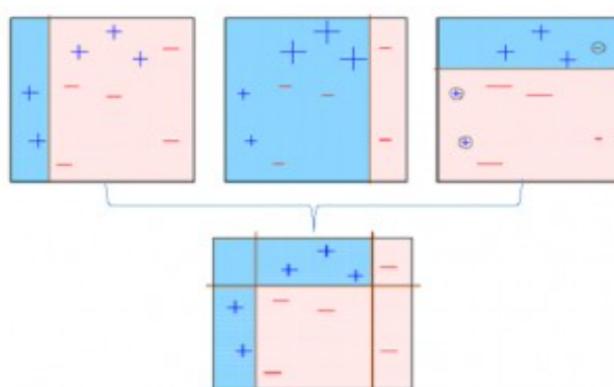
Stacking Ensemble

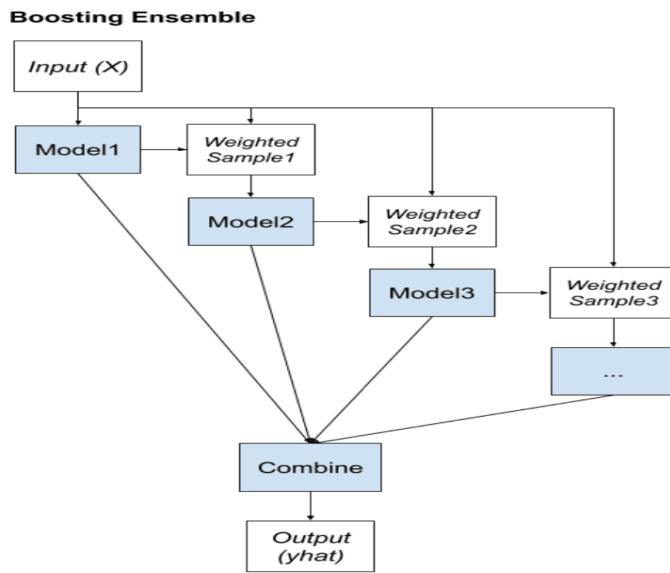


Boosting

Le Boosting est un processus séquentiel dans lequel chaque modèle suivant tente de corriger les erreurs du modèle précédent. Les modèles suivants dépendent du modèle précédent. Comprendons le fonctionnement du boosting dans les étapes suivantes.

1. Un sous-ensemble est créé à partir de l'ensemble de données original.
2. Au départ, tous les points de données reçoivent la même pondération.
3. Un modèle de base est créé sur ce sous-ensemble.
4. Ce modèle est utilisé pour faire des prédictions sur l'ensemble des données.
5. Les erreurs sont calculées à partir des valeurs réelles et des valeurs prédites.
6. Les observations dont les prédictions sont erronées se voient attribuer une pondération plus élevée. (Ici, les trois points bleu-plus mal classés auront une pondération plus élevée).
7. Un autre modèle est créé et des prédictions sont faites sur l'ensemble des données.(Ce modèle tente de corriger les erreurs du modèle précédent).
8. De même, plusieurs modèles sont créés, chacun corrigeant les erreurs du modèle précédent.
9. Le modèle final (apprenant fort) est la moyenne pondérée de tous les modèles (apprenants faibles).





Ainsi, l'algorithme de renforcement combine un certain nombre d'apprenants faibles pour former un apprenant fort. Les modèles individuels n'obtiendraient pas de bons résultats sur l'ensemble des données, mais ils fonctionnent bien sur une partie de l'ensemble. Ainsi, chaque modèle renforce les performances de l'ensemble.

Bagging algorithms

- Bagging meta-estimator
- Random forest

Boosting algorithms

- AdaBoost
- GBM
- XGBM
- Light GBM
- CatBoost

3.9.3 Code source

Bagging

```

# explore random forest tree depth effect on performance
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from matplotlib import pyplot

# get the dataset
def get_dataset():
  
```

```

X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
                           n_redundant=5, random_state=3)
return X, y

# get a list of models to evaluate
def get_models():
    models = dict()
    # consider tree depths from 1 to 7 and None=full
    depths = [i for i in range(1,8)] + [None]
    for n in depths:
        models[str(n)] = RandomForestClassifier(max_depth=n)
    return models

# evaluate a given model using cross-validation
def evaluate_model(model, X, y):
    # define the evaluation procedure
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    # evaluate the model and collect the results
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
    return scores

# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
    # evaluate the model
    scores = evaluate_model(model, X, y)
    # store the results
    results.append(scores)
    names.append(name)
    # summarize the performance along the way
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()

```

Stacking

```

from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression

```

```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import StackingClassifier
from matplotlib import pyplot

# get the dataset
def get_dataset():
    X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
                               n_redundant=5, random_state=1)
    return X, y

# get a stacking ensemble of models
def get_stacking():
    # define the base models
    level0 = list()
    level0.append(('lr', LogisticRegression()))
    level0.append(('knn', KNeighborsClassifier()))
    level0.append(('cart', DecisionTreeClassifier()))
    level0.append(('svm', SVC()))
    level0.append(('bayes', GaussianNB()))
    # define meta learner model
    level1 = LogisticRegression()
    # define the stacking ensemble
    model = StackingClassifier(estimators=level0, final_estimator=level1, cv=5)
    return model

# get a list of models to evaluate
def get_models():
    models = dict()
    models['lr'] = LogisticRegression()
    models['knn'] = KNeighborsClassifier()
    models['cart'] = DecisionTreeClassifier()
    models['svm'] = SVC()
    models['bayes'] = GaussianNB()
    models['stacking'] = get_stacking()
    return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1,
                            error_score='raise')
    return scores

```

```

# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X, y)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()

```

Boosting

```

# example of grid searching key hyperparameters for adaboost on a classification
# dataset
from sklearn.datasets import make_classification
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
# define dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
                           n_redundant=5, random_state=6)
# define the model with default hyperparameters
model = AdaBoostClassifier()
# define the grid of values to search
grid = dict()
#The number of trees can be set via the "n_estimators" argument and defaults to 50.
grid['n_estimators'] = [10, 50, 100, 500]
grid['learning_rate'] = [0.0001, 0.001, 0.01, 0.1, 1.0]
# define the evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define the grid search procedure
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,
                           scoring='accuracy')
# execute the grid search
grid_result = grid_search.fit(X, y)
# summarize the best score and configuration
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
# summarize all scores that were evaluated
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):

```

```
print("%f (%f) with: %r" % (mean, stdev, param))
```

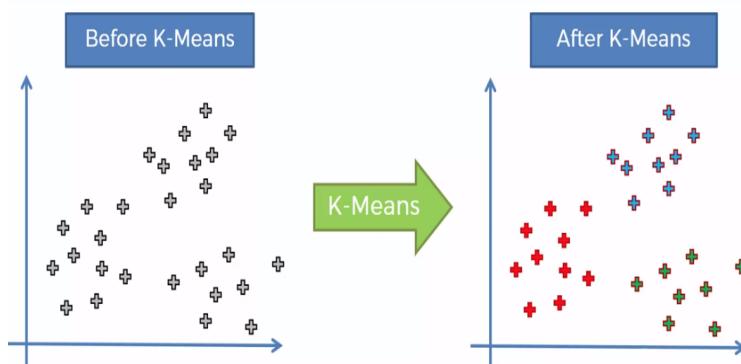
Apprentissage non supervisé

L'apprentissage non supervisé (parfois dénommé « Clustering ») est une méthode d'apprentissage automatique. Il s'agit pour un logiciel de diviser un groupe hétérogène de données, en sous-groupes de manière que les données considérées comme les plus similaires soient associées au sein d'un groupe homogène et qu'au contraire les données considérées comme différentes se retrouvent dans d'autres groupes distincts ; l'objectif étant de permettre une extraction de connaissance organisée à partir de ces données.

4.1 Kmeans

La classification K-means est un type d'apprentissage non supervisé, qui est utilisé lorsque vous avez des données non étiquetées (c'est-à-dire des données sans catégories ou groupes définis). Le but de cet algorithme est de trouver des groupes dans les données, avec le nombre de groupes représentés par la variable K. L'algorithme fonctionne de manière itérative pour affecter chaque point de données à l'un des K groupes en fonction des caractéristiques fournies. Les points de données sont regroupés en fonction de la similarité des fonctionnalités. Les résultats de l'algorithme de regroupement des moyennes K sont :

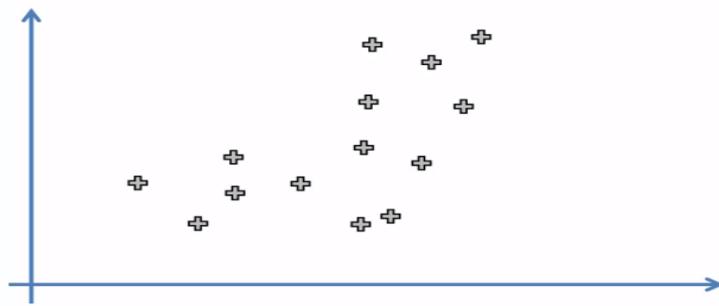
- Les centroïdes des grappes K, qui peuvent être utilisés pour étiqueter de nouvelles données
- Étiquettes pour les données d'apprentissage (chaque point de données est attribué à un seul cluster)



4.1.1 Comment Kmeans marche

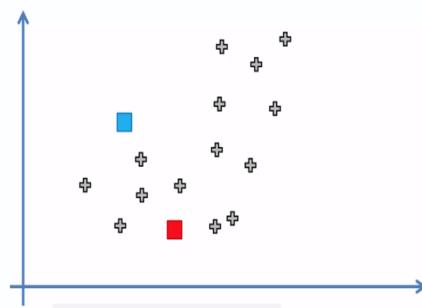
Il commence par choisir le nombre de K grappes. Le K représente le nombre de clusters que l'algorithme trouverait dans l'ensemble de données. Maintenant, choisir le bon K est très important. Parfois, le K est clairement visible à partir du jeu de données lors de la visualisation. Cependant, la plupart du temps, ce n'est pas le cas et nous verrons très bientôt comment choisir la bonne valeur de K.

STEP 1: Choose the number K of clusters: K = 2



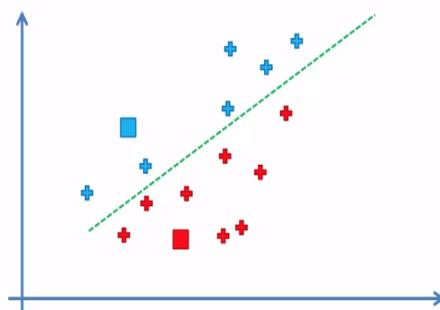
La deuxième étape consiste à attribuer K points aléatoires en tant que centroïdes. Ces points K peuvent être des points du jeu de données ou extérieurs. Il y a une chose à noter cependant. L'initialisation aléatoire des centroïdes peut parfois provoquer une interruption d'initialisation aléatoire, ce que nous verrions bientôt dans cette section.

STEP 2: Select at random K points, the centroids (not necessarily from your dataset)



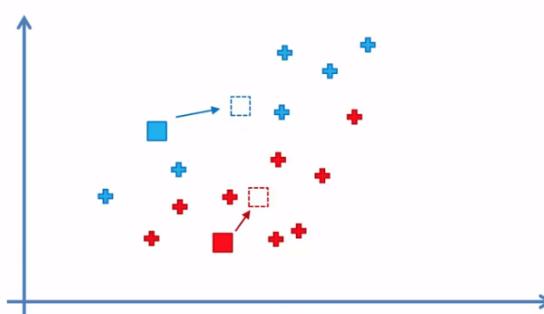
Dans la troisième étape, les points de jeu de données seraient alloués au centroïde le plus proche d'eux.

STEP 3: Assign each data point to the closest centroid ➔ That forms K clusters



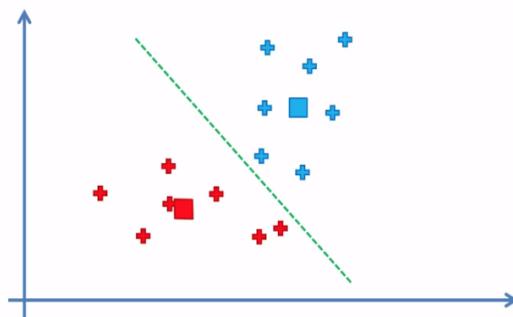
La quatrième étape consiste à calculer le centroïde des groupes individuels et à y placer l'ancien centroïde.

STEP 4: Compute and place the new centroid of each cluster



La cinquième étape consiste à réaffecter des points comme à l'étape 3. Si une réaffectation est effectuée, nous devons revenir à la quatrième étape. Si aucune réaffectation n'a lieu, nous pouvons dire que notre modèle a convergé et qu'il est prêt.

STEP 5: Reassign each data point to the new closest centroid.
If any reassignment took place, go to STEP 4, otherwise go to FIN.

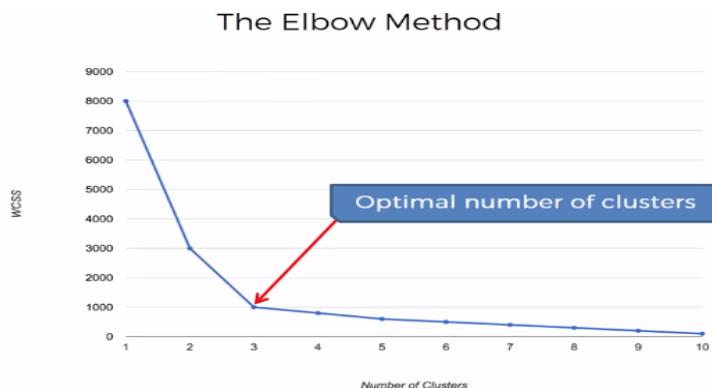


4.1.2 Choisir le bon K

$$WCSS = \sum_{P_i \text{incluster}1} distance(p_i, c_1)^2 + \sum_{P_i \text{incluster}2} distance(p_i, c_2)^2 + \sum_{P_i \text{incluster}3} distance(p_i, c_3)^2$$

Pour évaluer le choix de K, on utilise un paramètre appelé WCSS.

La méthode Elbow est ensuite utilisée pour choisir la meilleure valeur de K. Dans la description ci-dessous, nous pouvons voir qu'après 3, il n'y a pas de diminution significative du WCSS, alors 3 est la meilleure. Par conséquent, il se forme une forme de coude et c'est généralement une bonne idée de choisir le numéro où ce coude est formé. Il y aurait de nombreuses fois où le graphique ne serait pas aussi intuitif, mais avec la pratique, cela devient plus facile.



4.1.3 Exemple sickit-learn

Kmeans Sklearn

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import matplotlib.pyplot as plt
import seaborn as sns

# Importing the dataset
dataset = pd.read_csv('/Users/macbookair/Documents/cours/cours apprentissage
    machine/datasets/Mall_Customers.csv', index_col='CustomerID')
```

```

dataset.drop_duplicates(inplace=True)

# using only Spending_Score and income variable for easy visualisation
X = dataset.iloc[:, [2, 3]].values

# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    # inertia method returns wcss for that model
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10,5))
sns.lineplot(range(1, 11), wcss,marker='o',color='red')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

# Visualising the clusters
plt.figure(figsize=(15,7))
sns.scatterplot(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], color = 'yellow', label = 'Cluster 1',s=50)
sns.scatterplot(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], color = 'blue', label = 'Cluster 2',s=50)
sns.scatterplot(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], color = 'green', label = 'Cluster 3',s=50)
sns.scatterplot(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], color = 'grey', label = 'Cluster 4',s=50)
sns.scatterplot(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], color = 'orange', label = 'Cluster 5',s=50)
sns.scatterplot(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color = 'red',
                label = 'Centroids',s=300,marker=',')
plt.grid(False)
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()

```

4.2 Expectation Maximization EM

4.2.1 1D - Mixture Model

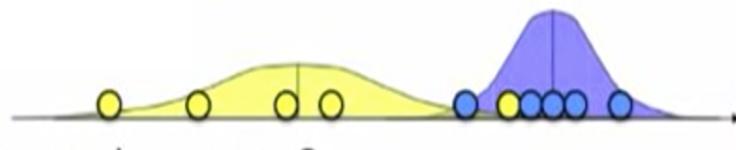
Mixture model est un modèle statistique permettant de modéliser différentes sous-populations dans la population globale sans que ces sous-populations soient identifiées dans les données par une variable observée.

Supposons qu'on a les observations : x_1, \dots, x_n .

- $k=2$, deux distributions gaussiens (a, b) avec μ, σ^2 inconnues.
- estimation insignifiant si l'on connaît la source de chaque observation.

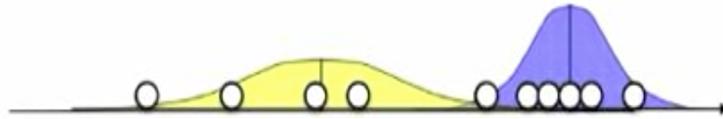
$$\mu_b = \frac{x_1 + \dots + x_n}{n_b}$$

$$\sigma_b^2 = \frac{(x_1 - \mu_b)^2 + \dots + (x_n - \mu_b)^2}{n_b}$$



Et si on ne connaît pas la source ?

si on ne connaît les Params de la distribution gaussienne (μ, σ^2), peut-on deviner si le point est susceptible d'être a ou b ?



$$p(b|x_i) = \frac{p(x_i|b)p(b)}{p(x_i|b)p(b) + p(x_i|a)p(a)}$$

$$p(x_i|b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp\left(-\frac{(x_i - \mu_b)^2}{2\sigma_b^2}\right)$$

4.2.2 Fonctionnement de l'algorithme EM

Probleme "chicken first or egg"

- Connaitre (μ_a, σ_a^2) et (μ_b, σ_b^2) pour deviner la source des points.
- Connaitre la source pour estimer (μ_a, σ_a^2) et (μ_b, σ_b^2) .

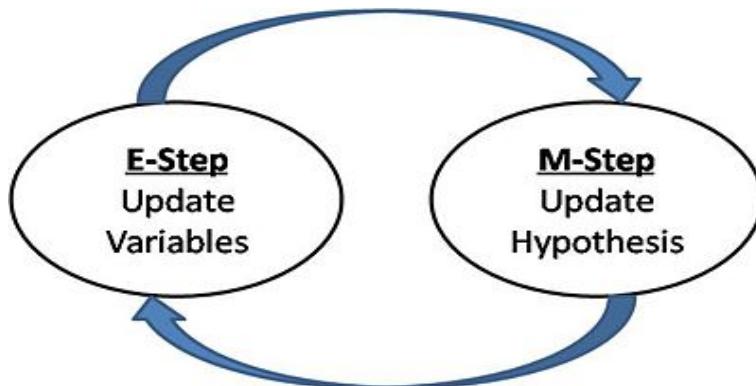
L'algorithme d'expectation-maximisation, est une approche d'estimation du maximum de vraisemblance en présence de variables latentes.

L'algorithme EM est une approche itérative qui alterne entre deux modes. Le premier mode tente d'estimer les variables manquantes ou latentes, appelées étape d'estimation ou étape E. Le deuxième mode tente d'optimiser les paramètres du modèle pour mieux expliquer les données, appelé étape de maximisation ou étape M.

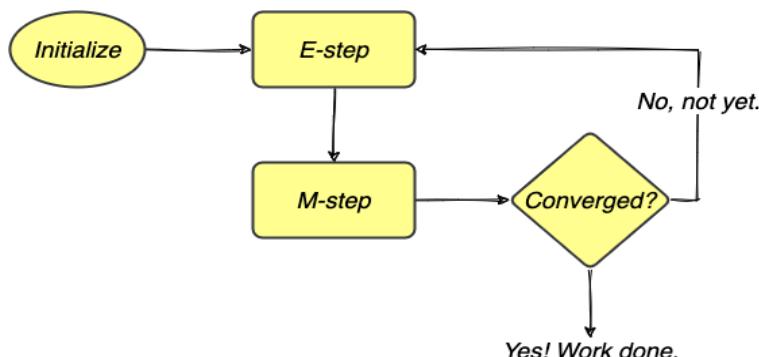
- Commencer avec deux gaussiennes placées au hasard (μ_a, σ_a^2) et (μ_b, σ_b^2) .
- Pour chaque point $p(b|x_i)$ =Est-ce que ça a l'air de venir de b ?
- ajuster les (μ_a, σ_a^2) et (μ_b, σ_b^2) pour s'adapter aux points qui leur sont attribués.
- itérer jusqu'à la convergence

1. E-Step. Estimez les variables manquantes dans l'ensemble de données.
2. M-Step. Maximisez les paramètres du modèle en présence des données.
3. L'algorithme EM peut être appliqué assez largement, bien qu'il soit peut-être plus connu en apprentissage automatique
4. pour être utilisé dans des problèmes d'apprentissage non supervisés, tels que l'estimation de la densité et le regroupement.

L'application la plus discutée de l'algorithme EM est le clustering avec un modèle de mélange "Mixture model".

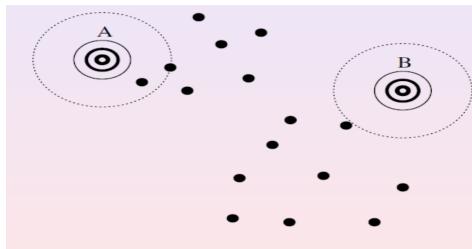


1. Initialement, un ensemble de valeurs initiales des paramètres est considéré. Un ensemble de données observées incomplètes est fourni au système en supposant que les données observées proviennent d'un modèle spécifique.
2. L'étape suivante est appelée «attente» - étape ou étape E. Dans cette étape, nous utilisons les données observées afin d'estimer ou de deviner les valeurs des données manquantes ou incomplètes. Il est essentiellement utilisé pour mettre à jour les variables.
3. L'étape suivante est appelée étape de «maximisation» ou étape M. Dans cette étape, nous utilisons les données complètes générées à l'étape précédente «Attente» afin de mettre à jour les valeurs des paramètres. Il est essentiellement utilisé pour mettre à jour l'hypothèse.
4. Maintenant, à la quatrième étape, il est vérifié si les valeurs convergent ou non, si oui, puis arrêtez sinon répétez les étapes 2 et 3, c'est-à-dire «Expectation» - étape et «Maximisation» - étape jusqu'à ce que la convergence se produise.



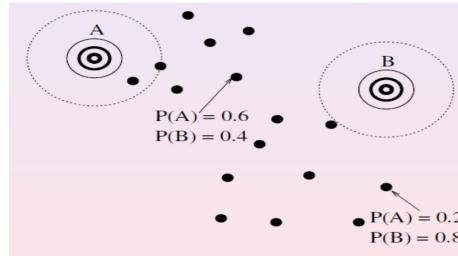
The EM for the GGM (graphical view 1)

Hidden variable: for each point, which Gaussian generated it?



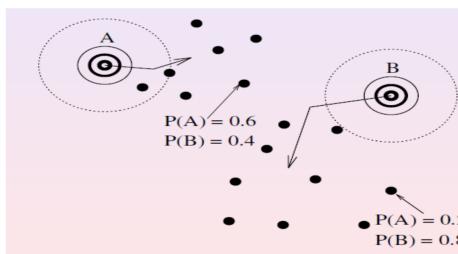
The EM for the GGM (graphical view 2)

E-Step: for each point, estimate the probability that each Gaussian generated it.



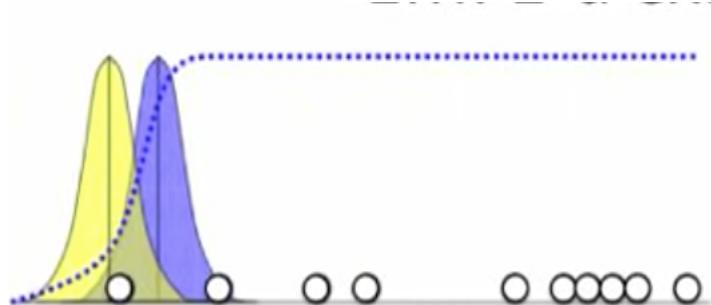
The EM for the GGM (graphical view 3)

M-Step: modify the parameters according to the hidden variable to maximize the likelihood of the data (and the hidden variable).

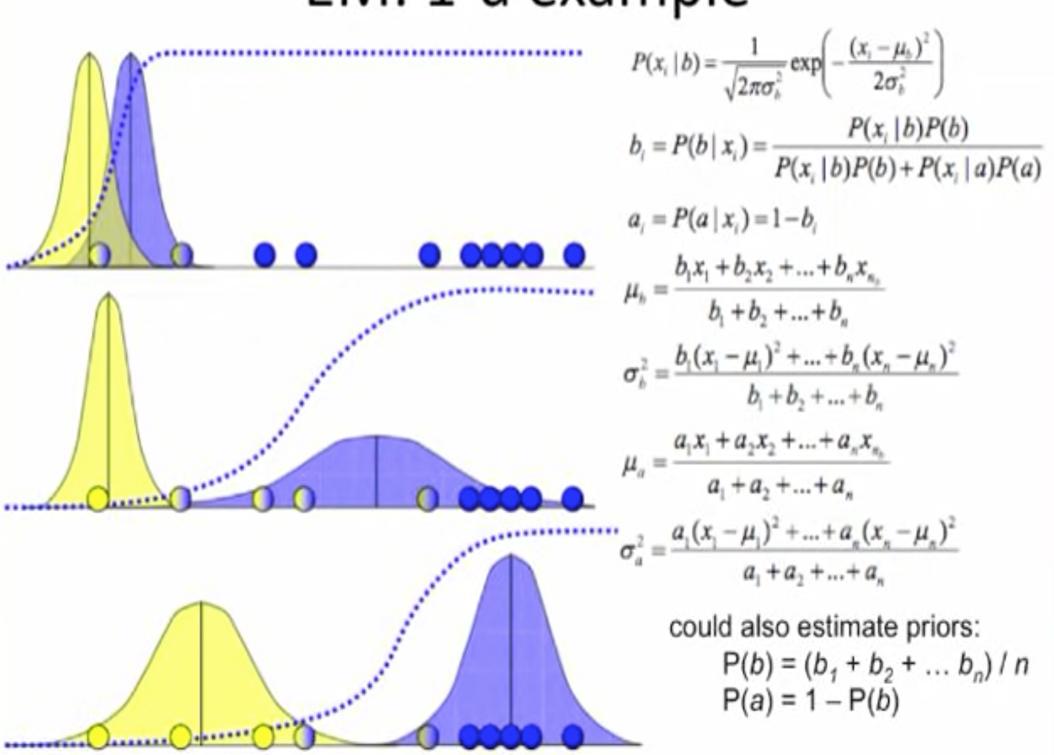


4.2.3 Exemple EM : 1D

Commencer avec deux gaussiennes placées au hasard :



Claculer "Bayesian Posterior" $p(b|x_i)$ et claculer la variance :



4.2.4 Convergence de EM

see <https://www.statlect.com/fundamentals-of-statistics/EM-algorithm>

4.2.5 Exemple Python

EM Clustering

```

# example of a bimodal constructed from two gaussian processes
import numpy as np
import pandas as pd
from sklearn.mixture import GaussianMixture
import os
import matplotlib.pyplot as plt
import seaborn as sns

# Importing the dataset
dataset = pd.read_csv('/Users/macbookair/Documents/cours/cours apprentissage
    machine/datasets/Mall_Customers.csv', index_col='CustomerID')
dataset.drop_duplicates(inplace=True)

# using only Spending_Score and income variable for easy visualisation
X = dataset.iloc[:, [2, 3]].values

# fit model
model = GaussianMixture(n_components=4, init_params='random')
model.fit(X)
# predict latent values

```

```

yhat = model.predict(X)

# check latent value for first few points
print(yhat[:100])

# check latent value for last few points
print(yhat[-100:])

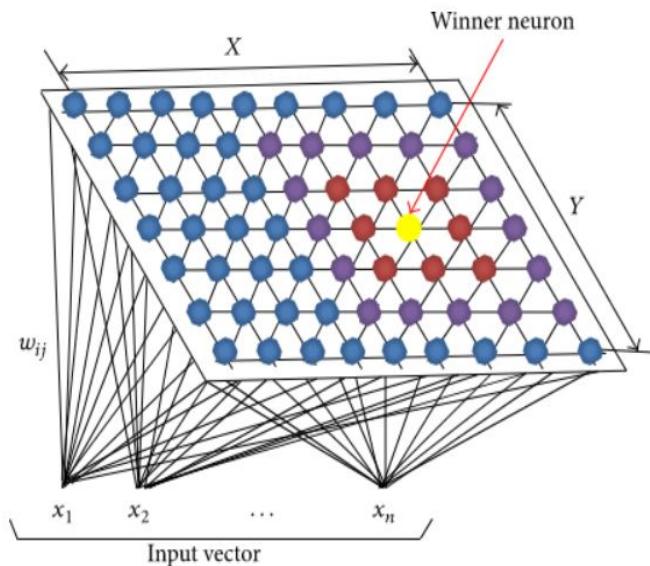
frame = pd.DataFrame(X)
frame['cluster'] = yhat
frame.columns = ['Age', 'Annual_Income', 'cluster']

color=['blue', 'green', 'red', 'black']

for k in range(0,4):
    data = frame[frame["cluster"]==k]
    plt.scatter(data["Age"],data["Annual_Income"],c=color[k])
plt.show()

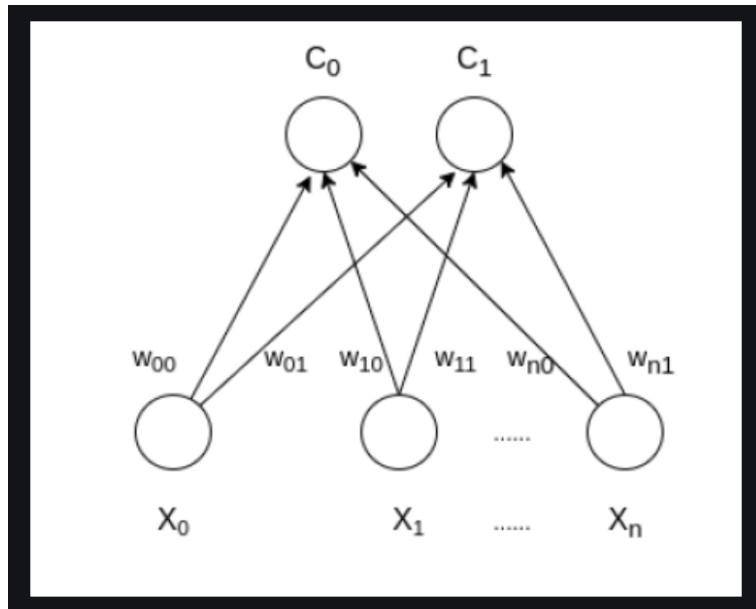
```

4.3 Self Organized Map / Kohonen Map



SOM est un type de réseau neuronal artificiel qui s'inspire également des modèles biologiques des systèmes neuronaux des années 1970. Il suit une approche d'apprentissage non supervisé et forme son réseau à l'aide d'un algorithme d'apprentissage compétitif. SOM est utilisé pour les techniques de regroupement et de mise en correspondance (ou de réduction de la dimensionnalité) afin de mettre en correspondance des données multidimensionnelles avec des données de dimension inférieure, ce qui permet de réduire des problèmes complexes pour en faciliter l'interprétation. La SOM comporte deux couches, l'une étant la couche d'entrée et l'autre la couche de sortie.

L'architecture de la carte auto-organisatrice avec deux segments et n caractéristiques d'entrée d'un échantillon est présentée ci-dessous :



4.3.1 Fonctionnement des SOMs

Contrairement à de nombreux autres types de réseaux, un SOM n'a pas besoin de spécifier une sortie cible. Au lieu de cela, lorsque les poids des noeuds correspondent au vecteur d'entrée, cette zone du treillis est optimisée de manière sélective pour ressembler davantage aux données de la classe à laquelle appartient le vecteur d'entrée. À partir d'une distribution initiale de poids aléatoires, et après de nombreuses itérations, le SOM finit par s'installer dans une carte de zones stables. Chaque zone est en fait un classificateur de caractéristiques, de sorte que vous pouvez considérer la sortie graphique comme une sorte de carte de caractéristiques de l'espace d'entrée.

Algorithme

1. Initialisation des poids w_{ij} une valeur aléatoire peut être supposée. Initialiser le taux d'apprentissage α .
2. : Calcul de la distance euclidienne quadratique.

$$D(j) = \sum (w_{ij} - x_i)^2$$

où i=1 à n et j=1 à m

3. : trouver l'indice J, lorsque $D(j)$ est minimal, qui sera considéré comme l'indice gagnant.
4. : pour chaque j dans un voisinage spécifique de j et pour tous les i, calculer le nouveau poids.

$$w_{ij}(new) = w_{ij}(old) + \alpha[x_i - w_{ij}(old)]$$

5. : Mettre à jour la règle d'apprentissage en utilisant :

$$\alpha(t+1) = 0,5 * t$$

6. : Test de la condition d'arrêt (Nombre d'iterations).

```

import math

class SOM:

    # Function here computes the winning vector
    # by Euclidean distance
    def winner(self, weights, sample):

        D0 = 0
        D1 = 0

        for i in range(len(sample)):

            D0 = D0 + math.pow((sample[i] - weights[0][i]), 2)
            D1 = D1 + math.pow((sample[i] - weights[1][i]), 2)

        # Selecting the cluster with smallest distance as winning cluster

        if D0 < D1:
            return 0
        else:
            return 1

    # Function here updates the winning vector
    def update(self, weights, sample, J, alpha):
        # Here iterating over the weights of winning cluster and modifying them
        for i in range(len(weights[0])):
            weights[J][i] = weights[J][i] + alpha * (sample[i] - weights[J][i])

        return weights

# Driver code

def main():

    # Training Examples ( m, n )
    T = [[1, 1, 0, 0], [0, 0, 0, 1], [1, 0, 0, 0], [0, 0, 1, 1]]

    m, n = len(T), len(T[0])

    # weight initialization ( n, C )
    weights = [[0.2, 0.6, 0.5, 0.9], [0.8, 0.4, 0.7, 0.3]]

```

```

# training
ob = SOM()

epochs = 3
alpha = 0.5

for i in range(epochs):
    for j in range(m):

        # training sample
        sample = T[j]

        # Compute winner vector
        J = ob.winner(weights, sample)

        # Update winning vector
        weights = ob.update(weights, sample, J, alpha)

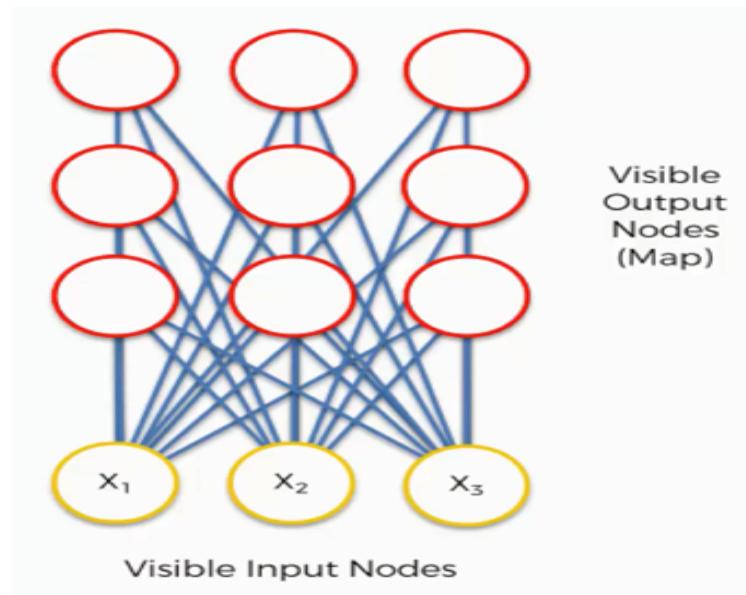
    # classify test sample
    s = [0, 0, 0, 1]
    J = ob.winner(weights, s)

    print("Test Sample s belongs to Cluster : ", J)
    print("Trained weights : ", weights)

if __name__ == "__main__":
    main()

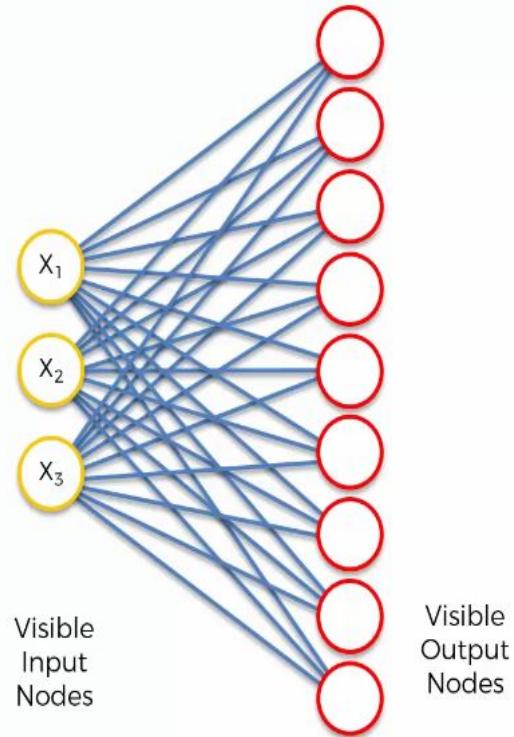
```

4.3.2 SOM : L'algorithme d'apprentissage en détail



Nos vecteurs d'entrée correspondent à trois caractéristiques et nous avons neuf nœuds de sortie.

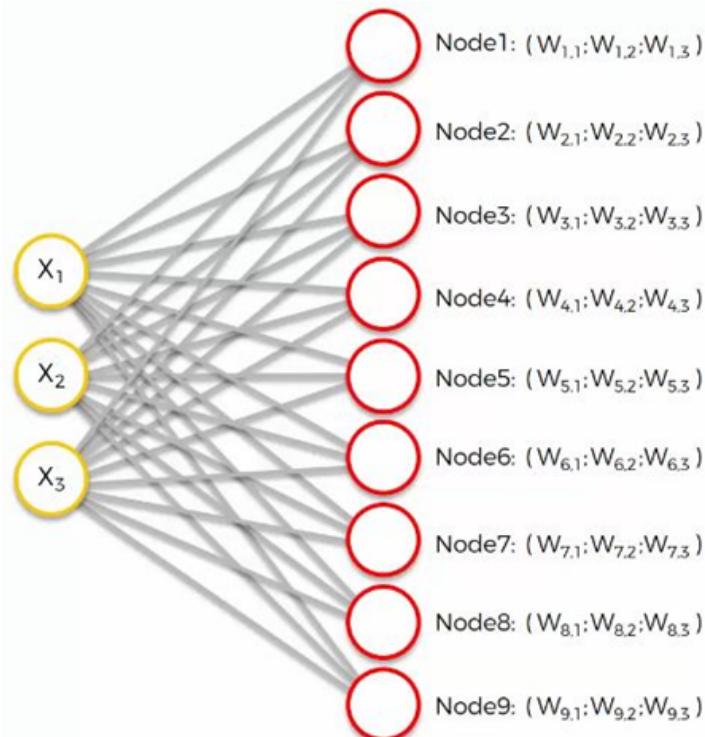
Considérons la structure de l'auto-organisation qui a 3 nœuds d'entrée visibles et 9 sorties qui sont connectées directement à l'entrée comme le montre la figure ci-dessous.



Les valeurs de nos nœuds d'entrée sont les suivantes : $x_1 = 0, 7$, $x_2 = 0, 6$, $x_3 = 0, 9$

Étape 1 : Initialisation des poids

Nous avons initialisé au hasard les valeurs des poids (proches de 0 mais pas 0).



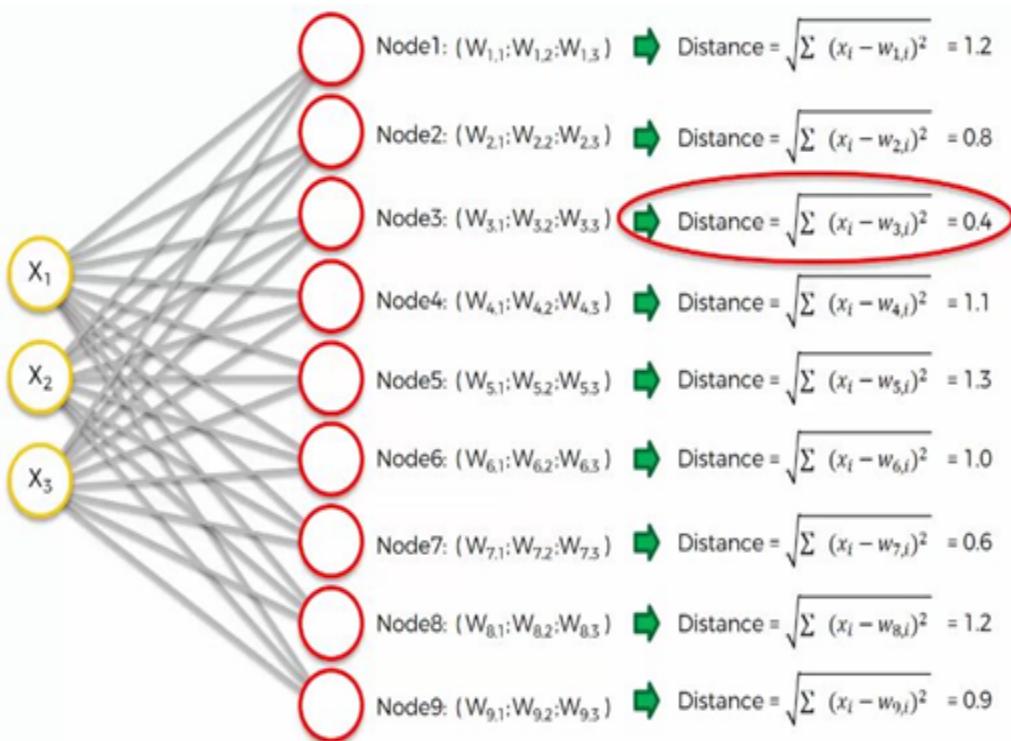
$W_{1,1}=0.31$	$W_{1,2}=0.22$	$W_{1,3}=0.10$
$W_{2,1}=0.21$	$W_{2,2}=0.34$	$W_{2,3}=0.19$
$W_{3,1}=0.39$	$W_{3,2}=0.42$	$W_{3,3}=0.45$
$W_{4,1}=0.25$	$W_{4,2}=0.32$	$W_{4,3}=0.62$
$W_{5,1}=0.24$	$W_{5,2}=0.31$	$W_{5,3}=0.16$
$W_{6,1}=0.52$	$W_{6,2}=0.33$	$W_{6,3}=0.42$
$W_{7,1}=0.31$	$W_{7,2}=0.22$	$W_{7,3}=0.10$
$W_{8,1}=0.12$	$W_{8,2}=0.41$	$W_{8,3}=0.19$
$W_{9,1}=0.34$	$W_{9,2}=0.40$	$W_{9,3}=0.51$

Etape 2 : Calcul de la meilleure unité d'appariement (Best Matching Unit)

Pour déterminer la meilleure unité de correspondance, une méthode consiste à parcourir tous les noeuds et à calculer la distance euclidienne entre le vecteur de poids de chaque noeud et le vecteur d'entrée actuel. Le noeud dont le vecteur de poids est le plus proche du vecteur d'entrée est désigné comme BMU.

$$distance = \sum_{i=0}^{i=n} (X_i - W_i)^2$$

Où X est le vecteur d'entrée actuel et W le vecteur de poids du noeud.
Calculons l'unité de meilleure correspondance à l'aide de la formule de distance.

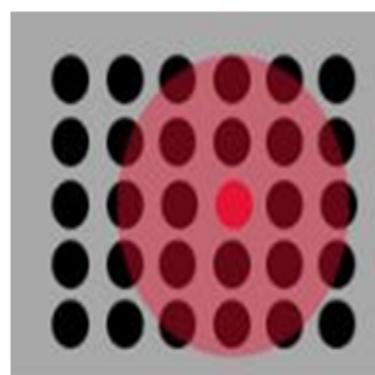


Comme nous pouvons le constater, le noeud numéro 3 est le plus proche avec une distance de 0,4. Nous appellerons ce noeud notre BMU (best-matching unit).

Étape 3 : Calcul de la taille du voisinage autour de la BMU

À chaque itération, une fois que le BMU a été déterminé, l'étape suivante consiste à calculer quels sont les autres noeuds qui se trouvent dans le voisinage du BMU.

l'étape suivante consiste à calculer quels sont les autres noeuds qui se trouvent dans le voisinage du BMU. Les vecteurs de poids de tous ces noeuds seront modifiés à l'étape suivante. on calcule le rayon du voisinage, puis il suffit d'appliquer la méthode de Pythagore pour déterminer si chaque noeud se trouve ou non à l'intérieur de la distance radiale.



On peut voir que le voisinage montré ci-dessus est centré autour de la BMU (point rouge) et englobe la plupart des autres noeuds et le rayon du cercle.

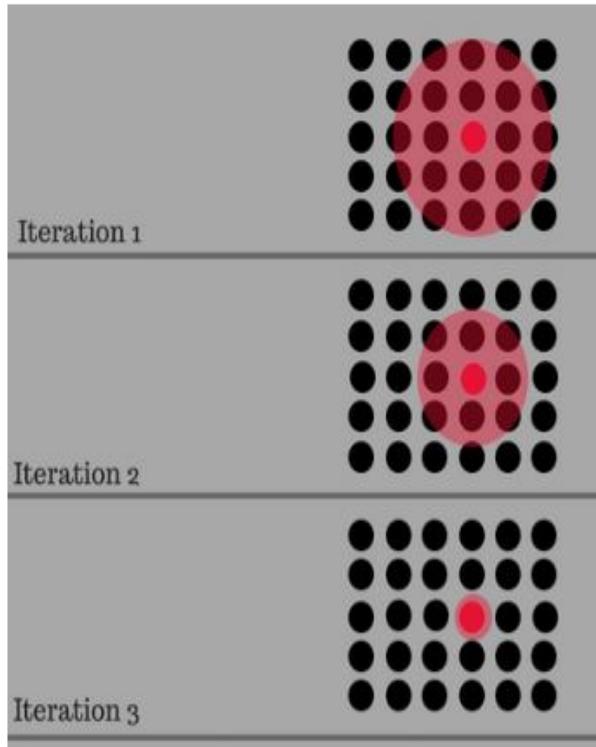
La taille du voisinage autour de la BMU diminue avec une fonction de décroissance exponentielle. Elle diminue à chaque itération jusqu'à ce qu'elle atteigne uniquement la BMU.

$$\sigma(t) = \sigma_0 \exp\left(\frac{-t}{\lambda}\right)$$

σ_0 la largeur de la maille à $t=0$

t le pas de temps actuel d'où $t = 0, 1, 2, \dots, n$

λ constante temps



Au fil du temps, le voisinage se réduira à la taille d'un seul nœud, la BMU. Maintenant que nous connaissons le rayon, il suffit d'itérer sur tous les nœuds du treillis pour déterminer s'ils se trouvent ou non dans le rayon. Si un nœud se trouve dans le voisinage, son vecteur de poids est ajusté comme suit à l'étape 4.

Étape 4 : Ajustement des poids

chaque nœud dans le voisinage du BMU (y compris le BMU) voit son vecteur de poids ajusté selon l'équation suivante :

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha_t [x_i - w_{ij}(\text{old})]$$

Ainsi, dans notre exemple, le nœud 4 est l'unité la mieux adaptée (comme vous pouvez le voir à l'étape 2), ce qui correspond à leurs poids :

$$W_{3,1} = 0.39, W_{3,2} = 0.42, W_{3,3} = 0.45$$

$$\text{Input vector : } x_1 = 0.7, x_2 = 0.6, x_3 = 0.9$$

learning rate = 0.5

Pour $W_{3,1} = 0.39 + 0.5 (0.7-0.39) = 0.545$

Pour $W_{3,2} = 0.42 + 0.5 (0.6-0.42) = 0.51$

Pour $W_{3,3} = 0.45 + 0.5 (0.9-0.45) = 0.675$

La décroissance du taux d'apprentissage est calculée à chaque itération à l'aide de l'équation suivante :

$$\alpha(t) = \alpha_0 \exp\left(\frac{-t}{\lambda}\right)$$

Au fur et à mesure de la formation, le voisinage se réduit progressivement. À la fin de la formation, la taille des voisinages est nulle.

alpha₀ : taux d'apprentissage à t = 0.

4.3.3 Code source Python

SOM Sklearn

```
import numpy as np
from sklearn.datasets import load_digits
from som import Som
from pylab import plot, axis, show, pcolor, colorbar, bone

digits = load_digits()
data = digits.data
labels = digits.target

som = Som(16, 16, 64, sigma=1.0, learning_rate=0.5)
som.random_weights_init(data)
print("Initiating SOM.")
som.train_random(data, 10000)
print("\n. SOM Processing Complete")

bone()
pcolor(som.distance_map().T)
colorbar()

labels[labels == '0'] = 0
labels[labels == '1'] = 1
labels[labels == '2'] = 2
```

```

labels[labels == '3'] = 3
labels[labels == '4'] = 4
labels[labels == '5'] = 5
labels[labels == '6'] = 6
labels[labels == '7'] = 7
labels[labels == '8'] = 8
labels[labels == '9'] = 9

markers = ['o', 'v', '1', '3', '8', 's', 'p', 'x', 'D', '*']
colors = ["r", "g", "b", "y", "c", (0,0.1,0.8), (1,0.5,0), (1,1,0.3), "m", (0.4,0.6,0)]
for cnt,xx in enumerate(data):
    w = som.winner(xx)
    plot(w[0]+.5,w[1]+.5,markers[labels[cnt]],
         markerfacecolor='None', markeredgecolor=colors[labels[cnt]],
         markersize=12, markeredgewidth=2)
axis([0,som.weights.shape[0],0,som.weights.shape[1]])
show()

```

4.4 PCA

Une méthode d'apprentissage automatique importante pour la dimensionnalité est appelée analyse en composantes principales « Principal Component Analysis / PCA ».

C'est une méthode qui utilise de simples opérations matricielles d'algèbre linéaire et calcule une projection des données d'origine dans le même nombre ou dans des dimensions inférieures.

La dimensionnalité fait référence au nombre de caractéristiques associées à chaque mesure de données. Cela correspond aux colonnes d'une donnée tabulaire.

Lorsqu'un jeu de données est au-delà de la 2D ou de la 3D, il serait difficile de le visualiser. Mais la plupart des ensembles de données utilisés pour l'analyse ont une très grande dimension, disons des dizaines voire des centaines. Alors, comment gérons-nous ce scénario ? C'est là que le concept de réduction de dimensionnalité entre en jeu.

La réduction de la dimensionnalité dans les statistiques et l'apprentissage automatique est le processus par lequel le nombre de variables aléatoires considérées est réduit en obtenant un ensemble de quelques variables principales.

La méthode PCA peut être décrite et mise en œuvre à l'aide des outils de l'algèbre linéaire.

PCA est une opération appliquée à un DataSet, représentée par une matrice A de n x m, qui donne une projection de A que nous appellerons B.

4.4.1 Exemple numérique de PCA : de n = 2 vers n =1

DataSet :

X	4	8	13	7
Y	11	4	5	14

Etape 1 n = 2 propriétés et Nombre des exemples = 4

Etape 2 moyenne x = 32/4 = 8 et moyenne y = 35/4 = 8.5

Etape 3 Le calcule de la matrice covariance (x,y) et Les paires possibles : (x,x) , (x,y), (y,x),(y,y).

$$cov(x, x) = \frac{\sum(x_{ik} - Moy(x_i))(x_{jk} - Moy(x_j))}{N - 1}$$

ou bien

$$cov(x, x) = \frac{\sum(x_{ik} - Moy(x_i))^2}{N - 1}$$

$$cov(x, x) = 1/3 * ((4 - 8)^2 + (8 - 8)^2 + (13 - 8)^2 + (7 - 8)^2) = 14$$

$$cov(x, y) = 1/3*((4-8)*(11-8, 5)+(8-8)*(4-8, 5)+(13-8)*(5-8, 5)+(7-8)*(14-8, 5)) = -11$$

$$cov(x, y) = cov(y, x) = -11$$

$$cov(y, y) = 1/3 * ((11 - 8, 5)^2 + (4 - 8, 5)^2 + (5 - 8, 5)^2 + (14 - 8, 5)^2) = 23$$

Matrice de covariance :

$$\begin{pmatrix} cov(x, x) & cov(x, y) \\ cov(y, x) & cov(y, y) \end{pmatrix}$$

$$\begin{pmatrix} 14 & -11 \\ -11 & 23 \end{pmatrix}$$

Etape 4

Valeurs propres et vecteurs propres

Valeur propre : $\det(S - \lambda I) = 0$ d'où S = matrice cov , I matrice d'identité 2*2 .

$$\det = \begin{vmatrix} 14 - \lambda & -11 \\ -11 & 23 - \lambda \end{vmatrix} = 0$$

$$(14 - \lambda) * (23 - \lambda) - (-11 * -11) = \lambda^2 - 37\lambda + 201 = 0$$

deux racines.

$$\lambda_1 > \lambda_2$$

$$\lambda_1 = 30, 38$$

et

$$\lambda_2 = 6, 615$$

Vecteur propre de

$$\lambda_1 : (S - \lambda_1 I) * e_1 = 0$$

d'où e_1 matrice 2*1 [u1 ,u2]

$$\begin{vmatrix} 14 - \lambda_1 & -11 \\ -11 & 23 - \lambda_1 \end{vmatrix} * \begin{vmatrix} u_1 \\ u_2 \end{vmatrix} = 0$$

$$\begin{vmatrix} (14 - \lambda_1)u_1 - 11u_2 \\ -11u_1 + (23 - \lambda_1)u_2 \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \end{vmatrix}$$

$$\begin{vmatrix} -16, 38u_1 - 11u_2 \\ -11u_1 - 7, 38u_2 \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \end{vmatrix}$$

$$-16, 38u_1 - 11u_2 = 0$$

$$u_2 = \frac{-11, 38}{11} u_1$$

$$u_2 = -1, 48u_1$$

On substitue u_2 dans la deuxieme equation :

$$-11u_1 - 7, 38(-1, 48)u_1 = 0$$

$$-11u_1 + 11u_1 = 0$$

Infinite de solutions

Donc si on prend $u_1 = t$ alors $u_2 = -1, 48 * t$

le vecteur propre de λ_1 si on prend $t = 11$ est :

$$u_1 = 11$$

et

$$u_2 = -1, 48 * 11 = -16.17$$

Normalisation de vecteur propre :

e1 :

$$\frac{11}{\sqrt{(11^2 + (-16.17)^2)}} = 0, 55$$

$$\frac{-16.17}{(\sqrt{(11^2 + (-16.17)^2)})} = -0, 83$$

de la même manier en calcule le vecteur propre de λ_2 : e2 :

0,83 0,55

Etape 5

la nouvelle data set avec une seul dimension.

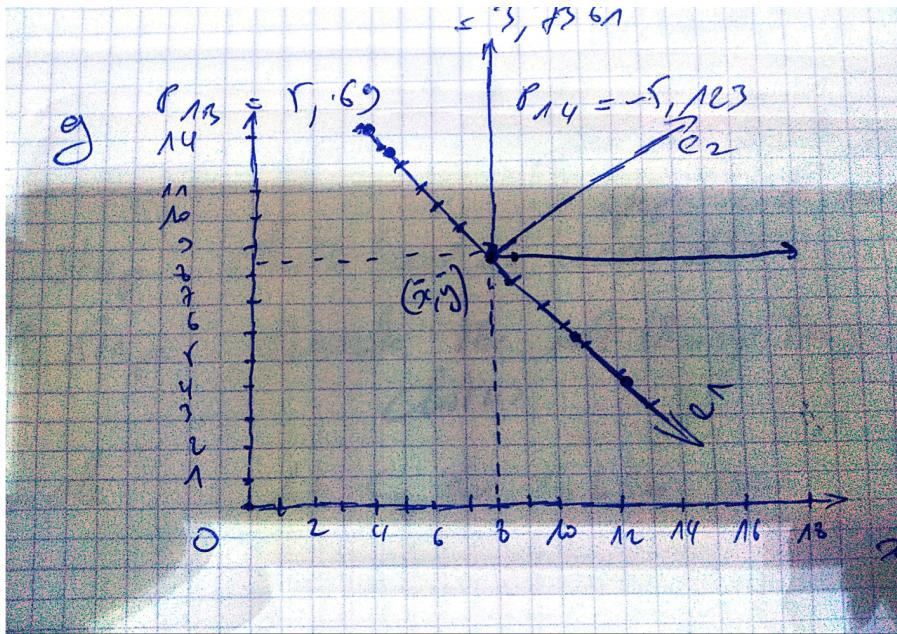
Propriété	ex1	ex2	ex3	ex4
PC1	P11	P12	P13	P14

$$P11 = e^T * \begin{pmatrix} xi - moyenne(x) \\ yi - moyenne(y) \end{pmatrix}$$

$$P11 = [0, 55 - 0.83] * \begin{pmatrix} 4 - 8 \\ 11 - 8.5 \end{pmatrix} = -4.305$$

$$p12 = 3, 73, p13 = 5, 69, p14 = 3, 17$$

Propriété	ex1	ex2	ex3	ex4
PC1	-4,305	3,73	5,69	-5,12



4.4.2 Implémentation python

PCA 1

```

from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# calculate the mean of each column
M = mean(A.T, axis=1)
print(M)
# center columns by subtracting column means
C = A - M
print(C)
# calculate covariance matrix of centered matrix
V = cov(C.T)
print(V)
# eigendecomposition of covariance matrix
values, vectors = eig(V)

print(vectors)
print(values)

```

```
# project data
P = vectors.T.dot(C.T)
print(P.T)
```

4.4.3 Implémentation sickit-learn

PCA sKlearn

```
# Principal Component Analysis
from numpy import array
from sklearn.decomposition import PCA
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# create the PCA instance
pca = PCA(2)
# fit on data
pca.fit(A)
# access values and vectors
print(pca.components_)
print(pca.explained_variance_)
# transform data
B = pca.transform(A)
print(B)
```

4.4.4 Exemple complet

PCA complet

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn import preprocessing
import matplotlib.pyplot as plt

url = "/Users/macbookair/Documents/cours/cours apprentissage machine/datasets/iris.
       data"
df = pd.read_csv(url, names=['sepal length', 'sepal width', 'petal length', 'petal
       width', 'target'])
from sklearn.preprocessing import StandardScaler

features = ['sepal length', 'sepal width', 'petal length', 'petal width']

x = df.loc[:, features].values
y = df.loc[:, ['target']].values
```

```

x = StandardScaler().fit_transform(x)

pca = PCA(n_components=2)

principalComponents = pca.fit_transform(x)

principalDataframe = pd.DataFrame(data = principalComponents, columns = ['PC1', 'PC2'])

targetDataframe = df[['target']]

newDataframe = pd.concat([principalDataframe, targetDataframe], axis = 1)

print(newDataframe)

plt.scatter(principalDataframe.PC1, principalDataframe.PC2)
plt.title('PC1 against PC2')
plt.xlabel('PC1')
plt.ylabel('PC2')

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')

ax.set_title('Plot of PC1 vs PC2', fontsize = 20)

targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

colors = ['r', 'g', 'b']

for target, color in zip(targets,colors):
    indicesToKeep = newDataframe['target'] == target
    ax.scatter(newDataframe.loc[indicesToKeep, 'PC1'],
              newDataframe.loc[indicesToKeep, 'PC2'],
              c = color,
              s = 50)
ax.legend(targets)
ax.grid()
plt.show()

print(pca.explained_variance_ratio_)

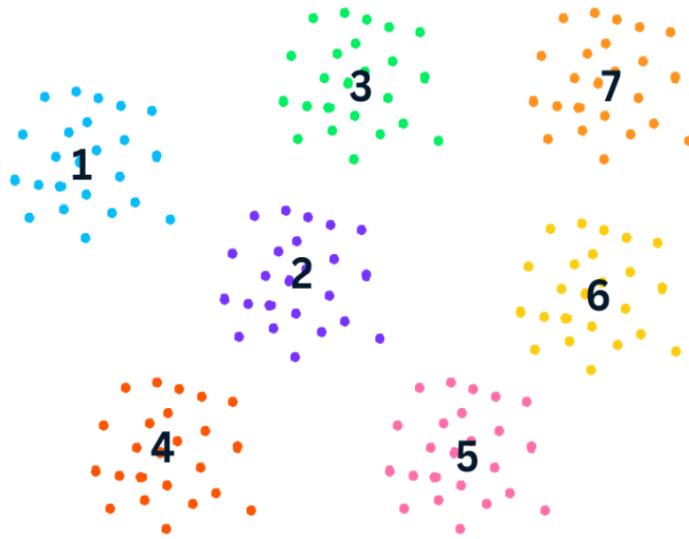
```

Ces valeurs montrent que la première composante principale PC1 explique 72,77% de la variation des données d'origine, tandis que la seconde composante principale explique 23,03 % de la variation des données d'origine.

En conclusion, cela signifie que les données 4 dimensions originales peuvent être réduites en toute sécurité à 2 dimensions à l'aide de PCA, car le DataSet ne peut être expliqué que par deux composants !

4.5 TSNE

Apprendre à visualiser des données de haute dimension dans un espace de basse dimension en utilisant une technique non linéaire de réduction de la dimensionnalité.



Le t-SNE (t-distributed Stochastic Neighbor Embedding) est une technique non supervisée de réduction de la dimensionnalité non linéaire pour l'exploration des données et la visualisation des données à haute dimension. La réduction non linéaire de la dimensionnalité signifie que l'algorithme nous permet de séparer des données qui ne peuvent pas être séparées par une ligne droite.

t-SNE donne une impression et une intuition sur la façon dont les données sont organisées dans les dimensions supérieures. Il est souvent utilisé pour visualiser des ensembles de données complexes en deux et trois dimensions, ce qui permet de mieux comprendre les modèles et les relations sous-jacentes dans les données.

4.5.1 t-SNE vs PCA

Le t-SNE et PCA sont des techniques de réduction dimensionnelle qui ont des mécanismes différents et fonctionnent mieux avec différents types de données.

PCA (analyse des composantes principales) est une technique linéaire qui fonctionne mieux avec des données ayant une structure linéaire. Elle cherche à identifier les composantes principales sous-jacentes des données en les projetant sur des dimensions inférieures, en minimisant la variance et en préservant les grandes distances entre paires.

Le t-SNE est une technique non linéaire qui vise à préserver les similarités par paire entre les points de données dans un espace de dimension inférieure. Le t-SNE s'attache à préserver les petites distances par paire, tandis que PCA vise à maintenir les grandes distances par paire afin de maximiser la variance.

En résumé, PCA préserve la variance des données, tandis que le t-SNE préserve les relations entre les points de données dans un espace de dimension inférieure, ce qui en fait un bon algorithme

pour la visualisation de données complexes de haute dimension.

4.5.2 Fonctionnement du t-SNE

L'algorithme t-SNE trouve la mesure de similarité entre les paires d'instances dans un espace de dimension supérieure et inférieure. Il tente ensuite d'optimiser deux mesures de similarité. Tout cela se fait en trois étapes.

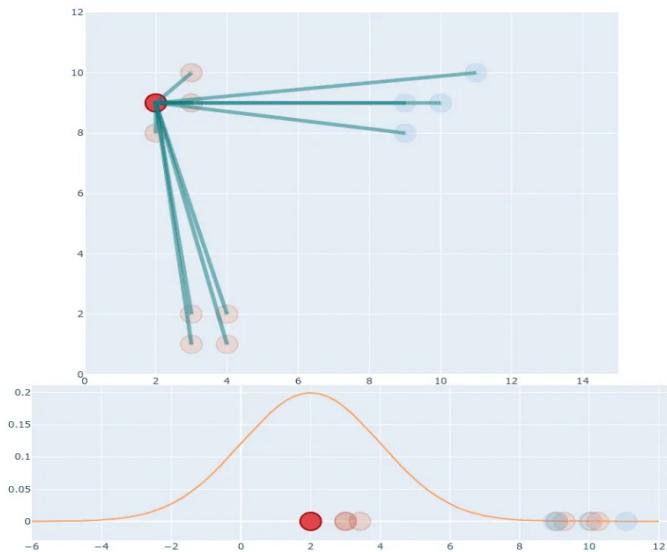
1. L'algorithme t-SNE modélise un point sélectionné en tant que voisin d'un autre point dans les dimensions supérieures et inférieures. Il commence par calculer une similarité par paire entre tous les points de données dans l'espace à haute dimension à l'aide d'un noyau gaussien. Les points éloignés les uns des autres ont une probabilité plus faible d'être choisis que les points proches les uns des autres.
2. Ensuite, l'algorithme tente de cartographier les points de données à haute dimension dans un espace à plus faible dimension tout en préservant les similitudes par paire.
3. Pour ce faire, il minimise la divergence (KL Divergence) entre la distribution de probabilité de l'espace de haute dimension d'origine et celle de l'espace de basse dimension. L'algorithme utilise la descente de gradient pour minimiser la divergence. L'intégration en basse dimension est optimisée jusqu'à un état stable.

Le processus d'optimisation permet la création de grappes et de sous-grappes de points de données similaires dans l'espace de dimension inférieure, qui sont visualisées pour comprendre la structure et les relations dans les données de dimension supérieure.

4.5.3 Algorithme

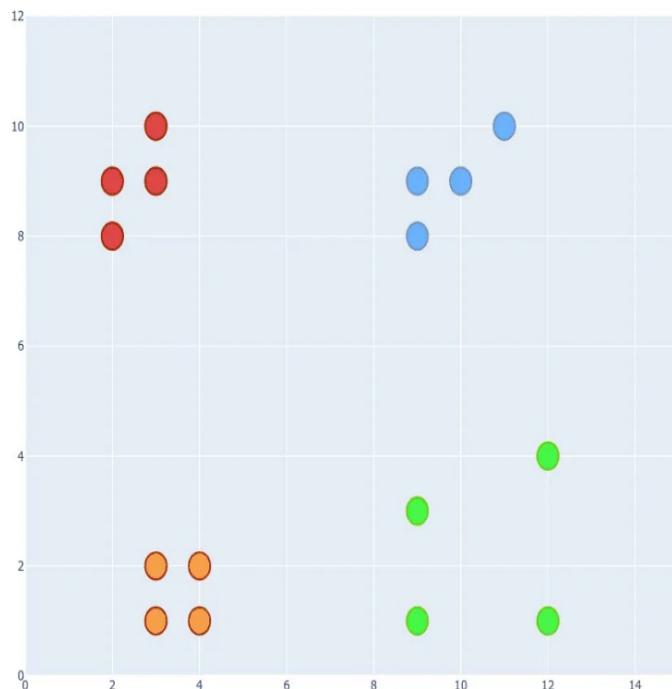
1. Distribution de probabilité (Similarité dans une dimension supérieur) : Distribution de probabilité qui représente les similitudes entre voisins.
la similarité du point de données x_j avec le point de données x_i est la probabilité conditionnelle $p_{j|i}$ que x_i choisisse x_j comme voisin. A travers la distribution gaussien :

$$g(|x_i - x_j|)$$

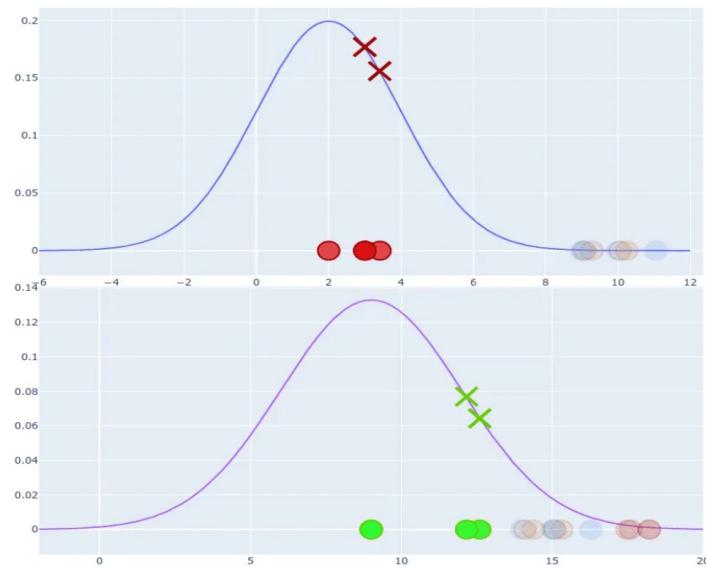


Cluster dispersées et variance

Jusqu'à présent, nos clusters étaient étroitement liés au sein de leur groupe. Que se passe-t-il si nous avons une nouvelle cluster comme celle-ci ?



Nous devrions être en mesure d'appliquer le même processus qu'auparavant.



Nous n'avons pas encore terminé. On peut distinguer les points similaires et non similaires, mais les valeurs absolues des probabilités sont beaucoup plus faibles que dans le premier exemple (comparez les valeurs de l'axe des ordonnées).

Nous pouvons y remédier en divisant la valeur actuelle de la projection par la somme des projections.

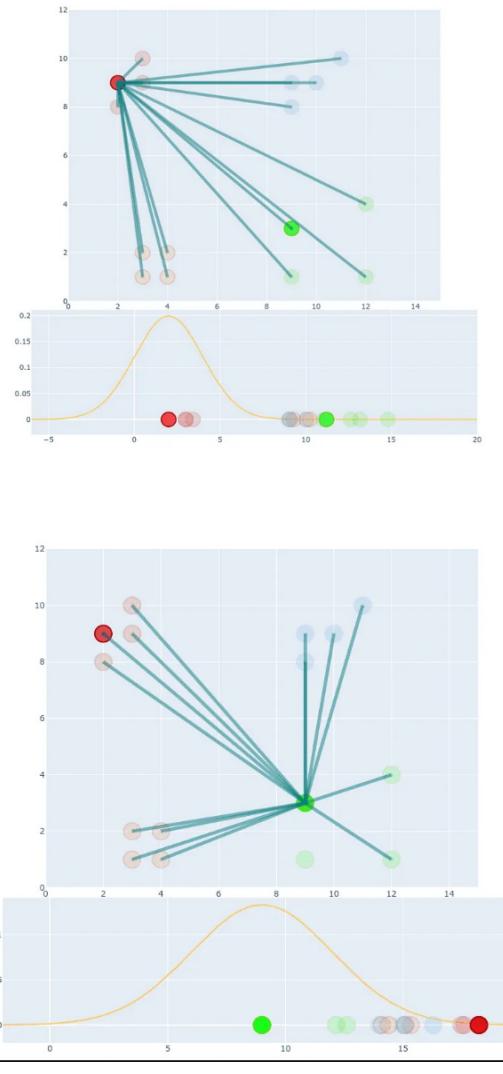
$$p_{i|j} = \frac{g(|x_i - x_j|)}{\sum_{i \neq j} g(|x_i - x_j|)}$$

C'est un bon endroit pour mentionner que $p_{i|i}$ est défini pour être égal à 0, et non à 1.

$$\sum_j p_{i|j} = 1$$

Traiter des distances différentes

Si nous prenons deux points et essayons de calculer la probabilité conditionnelle entre eux, les valeurs de $p_{i|j}$ et $p_{j|i}$ seront différentes :



La raison en est qu'ils proviennent de deux distributions différentes. Quelle est celle que nous devrions choisir pour le calcul ?

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}$$

Où N est un nombre de dimensions. Pour fournir une représentation fidèle en dimension inférieure, nous avons un objectif principal à l'esprit : les points proches doivent rester proches, les points éloignés doivent rester éloignés.

Le t-SNE atteint cet objectif en modélisant l'ensemble de données avec une distribution de probabilité agnostique, en trouvant une approximation de dimension inférieure avec une distribution qui correspond étroitement. Il a été introduit par Laurens van der Maaten et Geoffrey Hinton dans leur article Visualizing High-Dimensional Data Using t-SNE.

Étant donné que nous souhaitons également capturer une éventuelle structure de cluster sous-jacente, on définit une distribution de probabilités sur la courbe avec cette formule générale :

Eq 1 :

$$p_{i|j} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq j} \exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

Qu'est-ce que la perplexité

La perplexité est une mesure de l'information. Dans le t-SNE, la perplexité est utilisée pour

définir le nombre de voisins les plus proches pris en compte. (Shannon Entropy)
En t-SNE, les valeurs typiques de la perplexité se situent entre 5 et 50.

Création d'un espace à faible dimension

La partie suivante du t-SNE consiste à créer un espace à faible dimension avec le même nombre de points que dans l'espace d'origine. Les points doivent être répartis de manière aléatoire dans le nouvel espace. L'objectif de cet algorithme est de trouver une distribution de probabilité similaire dans un espace à faible dimension.

Eq 4 :

$$q_{i|j} = \frac{(1 + \|x_i - x_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|x_k - x_l\|^2)^{-1}}$$

Descente de gradient (Optimisation)

Pour optimiser cette distribution, le t-SNE utilise la divergence de Kullback-Leibler entre les probabilités conditionnelles $p_{j|i}$ et $q_{j|i}$.

$$C = D_{KL}(P||Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Dérivée de KL :

Eq 5 :

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|x_k - x_l\|^2)^{-1}$$

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

Data: data set $X = \{x_1, x_2, \dots, x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations T , learning rate η , momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, \dots, y_n\}$.

```

begin
    compute pairwise affinities  $p_{j|i}$  with perplexity  $Perp$  (using Equation 1)
    set  $p_{ij} = \frac{p_{ji} + p_{ij}}{2n}$ 
    sample initial solution  $\mathcal{Y}^{(0)} = \{y_1, y_2, \dots, y_n\}$  from  $\mathcal{N}(0, 10^{-4}I)$ 
    for  $t=1$  to  $T$  do
        compute low-dimensional affinities  $q_{ij}$  (using Equation 4)
        compute gradient  $\frac{\delta C}{\delta y_i}$  (using Equation 5)
        set  $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta y_i} + \alpha(t) (\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)})$ 
    end
end

```

4.5.4 Python code source

Tsne

```

rom sklearn.manifold import TSNE from keras.datasets import mnist from sklearn.
datasets import load_iris from numpy import reshape
import seaborn as sns import pandas as pd iris = load_iris()
x = iris.data y = iris.target
tsne = TSNE(n_components=2, verbose=1, random_state=123) z = tsne.fit_transform(x)
df = pd.DataFrame() df["y"] = y df["comp-1"] = z[:,0] df["comp-2"] = z[:,1]

```

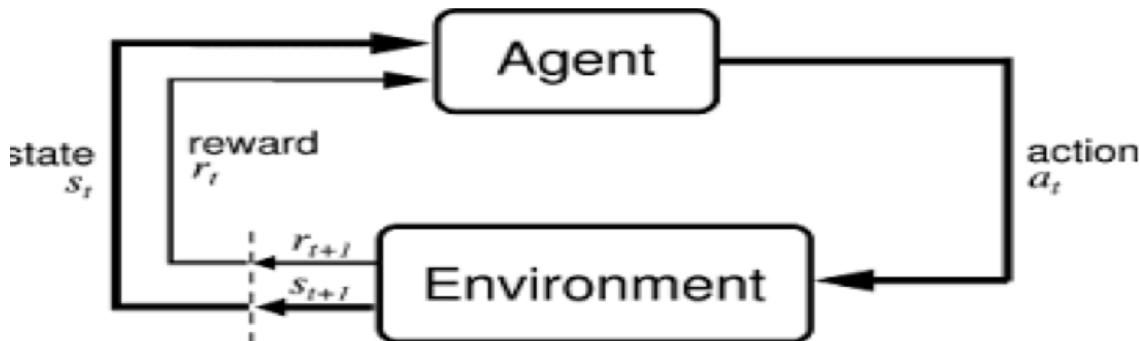
```
sns.scatterplot(x="comp-1", y="comp-2", hue=df.y.tolist(),
palette=sns.color_palette("hls", 3), data=df).set(title="Iris data T-SNE projection")
```

Tsne

```
from sklearn.manifold import TSNE from keras.datasets import mnist from sklearn.
datasets import load_iris from numpy import reshape
import seaborn as sns import pandas as pd
(x_train, y_train), (_, _) = mnist.load_data() x_train = x_train[:3000]
y_train = y_train[:3000]
print(x_train.shape)
# MNIST is a three-dimensional data, we'll reshape it into the two-dimensional one.
print(x_train.shape)
x_mnist = reshape(x_train, [x_train.shape[0], x_train.shape[1]*x_train.shape[2]])
print(x_mnist.shape)
tsne = TSNE(n_components=2, verbose=1, random_state=123) z = tsne.fit_transform(
    x_mnist)
df = pd.DataFrame()
df["y"] = y_train
df["comp-1"] = z[:,0] df["comp-2"] = z[:,1]
sns.scatterplot(x="comp-1", y="comp-2", hue=df.y.tolist(),
palette=sns.color_palette("hls", 10), data=df).set(title="MNIST data T-SNE
projection")
```

Apprentissage par renforcement

L'apprentissage par renforcement est l'étude de la prise de décision au fil du temps avec des conséquences. Le secteur a mis au point des systèmes permettant de prendre des décisions dans des environnements complexes sur la base d'un retour d'information externe. Il peut être compris à l'aide des concepts d'agents, d'environnements, d'états, d'actions et de récompenses.



Il existe plusieurs algorithmes au niveau d'apprentissage par renforcement citant : Q-learning, MDP, SARSA, DQ-Learning, etc.

Agent : un agent prend des actions ; par exemple, un drone effectuant une livraison ou Super Mario naviguant dans un jeu vidéo. L'algorithme est l'agent.

Action (A) : A est l'ensemble de tous les mouvements possibles que l'agent peut effectuer. Une action est presque explicite, mais il convient de noter que les agents choisissent généralement parmi une liste d'actions discrètes et possibles.

Facteur de remise « Discount rate » : le facteur de remise est multiplié par les récompenses futures découvertes par l'agent afin d'atténuer l'effet de ces récompenses sur le choix d'action de l'agent. Pourquoi ? Il est conçu pour que les récompenses futures valent moins que les récompenses immédiates ;

Environnement : Le monde à travers lequel l'agent se déplace. L'environnement prend l'état et l'action actuels de l'agent en entrée et renvoie en sortie la récompense de l'agent et son état suivant.

État (S) : Un État est une situation concrète et immédiate dans laquelle l'agent se trouve ; c'est-à-dire un lieu et un moment spécifiques, une configuration instantanée qui met l'agent en relation avec d'autres choses importantes telles que des outils, des obstacles, des ennemis ou des prix.

Récompense (R) : Une récompense est la rétroaction par laquelle nous mesurons le succès ou l'échec des actions d'un agent dans un état donné. Par exemple, dans un jeu vidéo, lorsque Mario touche une pièce, il gagne des points.

Politique (π) : La politique est la stratégie que l'agent utilise pour déterminer la prochaine action en fonction de l'état actuel. Il associe les états aux actions, les actions qui promettent la plus haute récompense.

Valeur (V) : Le rendement à long terme attendu avec remise, par opposition à la récompense à court terme R. $V\pi(s)$ est défini comme le rendement à long terme attendu de l'état actuel selon la politique π . Nous actualisons les récompenses, ou diminuons leur valeur estimée, à mesure qu'elles se produisent.

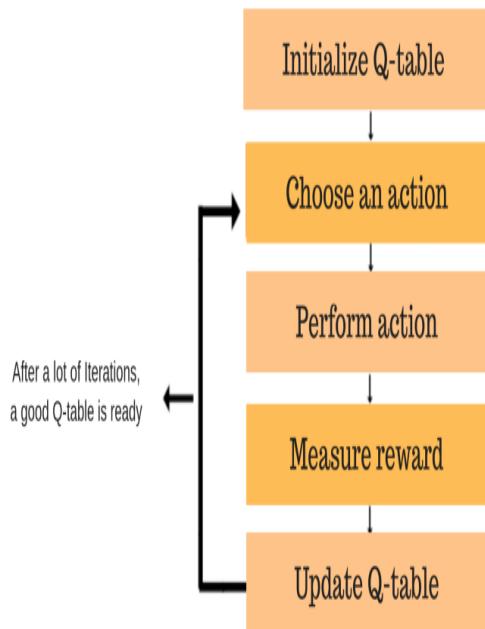
Q-value ou action-value (Q) : Q-value est similaire à valeur, sauf qu'il prend un paramètre supplémentaire, l'action en cours a . $Q\pi(s, a)$ fait référence au retour à long terme d'une action entretenant une action sous la politique π à partir de l'état actuel s .

Trajectoire : une séquence d'états et d'actions qui influencent ces états.

5.1 Q-Learning

Q-Learning est une forme de base d'apprentissage par renforcement qui utilise les valeurs Q (également appelées valeurs d'action) pour améliorer de manière itérative le comportement de l'agent d'apprentissage.

Algorithme Q-Learning :



Mathématiques : l'algorithme Q-Learning :

La fonction Q utilise l'équation de Bellman et prend deux entrées : état (s) et action (a). Il retourne la récompense future attendue de cette action à cet état.

$$Q^\pi(s_t, a_t) = \underbrace{E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots]}_{\text{Expected discounted cumulative reward}} \underbrace{| s_t, a_t]}_{\text{Given the state and action}}$$

↓ ↓ ↓
 Q-Values for the state given a particular state Expected discounted cumulative reward Given the state and action

Nous pouvons voir cette fonction Q comme un lecteur qui fait défiler la Q-table pour trouver la ligne associée à notre état et la colonne associée à notre action. Il renvoie la valeur Q de la cellule correspondante. C'est la "récompense future attendue".

Pour mettre à jour la fonction Q (s, a), on utilise « the Bellman equation » :

$$\text{New } Q(s, a) = Q(s, a) + \alpha [R(s, a) + \gamma \max_{a'} Q'(s', a') - Q(s, a)]$$

- New Q Value for that state and the action
- Learning Rate
- Reward for taking that action at that state
- Current Q Values
- Maximum expected future reward given the new state (s') and all possible actions at that new state.
- Discount Rate

Le discount rate est une valeur entre 0 et 1.

En utilisant la fonction ci-dessus, nous obtenons les valeurs de Q pour les cellules du tableau. Lorsque nous commençons, toutes les valeurs de la table Q sont des zéros.

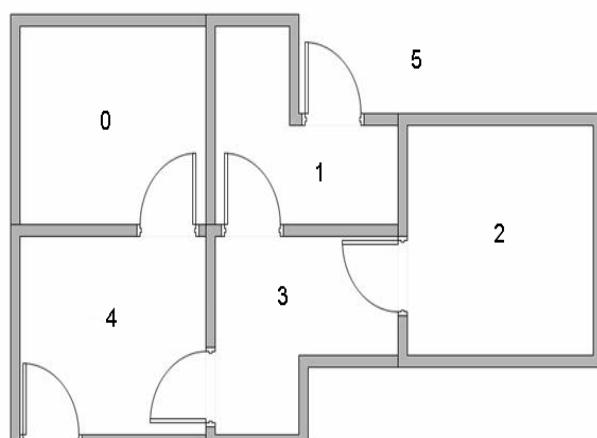
Il existe un processus itératif de mise à jour des valeurs. Alors que nous commençons à explorer l'environnement, la fonction Q nous donne de meilleures approximations en mettant à jour en permanence les valeurs Q du tableau.

Enfin... récapitulons :

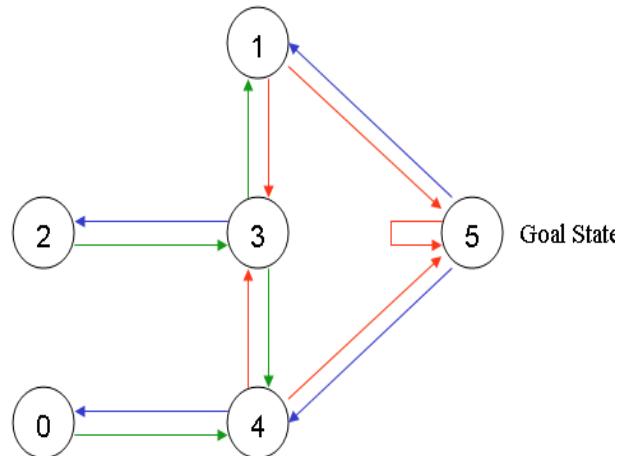
- Q-Learning est un algorithme d'apprentissage par renforcement basé sur les valeurs qui est utilisé pour trouver la politique de sélection d'action optimale à l'aide d'une fonction Q.
- Notre objectif est de maximiser la fonction de valeur Q.
- La table Q nous aide à trouver la meilleure action pour chaque état.
- Il est utile de maximiser la récompense escomptée en sélectionnant la meilleure des actions possibles.
- Q (état, action) renvoie la récompense future attendue de cette action à cet état.
- Cette fonction peut être estimée à l'aide de Q-Learning, qui met à jour de manière itérative Q (s, a) à l'aide de l'équation de Bellman.
- Initialement, nous explorons l'environnement et mettons à jour le Q-Table. Lorsque la Q-Table est prête, l'agent commence à exploiter l'environnement et à prendre de meilleures actions.

5.1.1 Exemple Q-Learning pas à pas

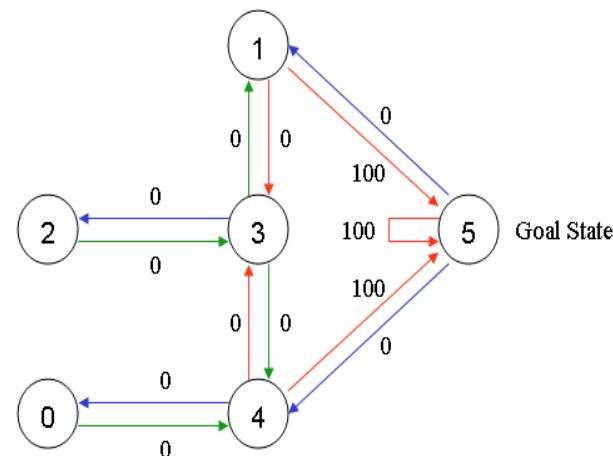
Supposons que nous ayons 5 pièces dans un bâtiment reliés par des portes, comme indiqué dans la figure ci-dessous. Nous numérotions chaque pièce de 0 à 5. L'extérieur du bâtiment peut être considéré comme une grande pièce (5). Notez que les portes 1 et 4 mènent au bâtiment depuis la salle 5 (à l'extérieur).



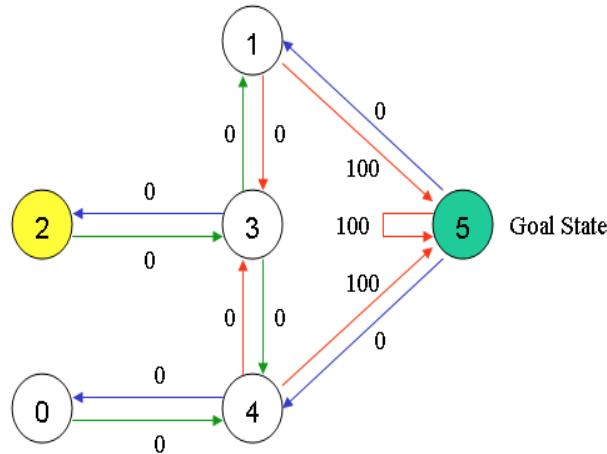
Nous pouvons représenter les pièces sur un graphique, chaque pièce en tant que nœud et chaque porte en tant que lien.



La salle des objectifs est le numéro 5. Pour définir cette salle comme objectif, nous allons associer une valeur de récompense à chaque porte (c'est-à-dire un lien entre les nœuds). Les portes qui mènent immédiatement au but ont une récompense instantanée de 100. Les autres portes qui ne sont pas directement connectées à la salle cible n'ont aucune récompense. Les portes étant à double sens (0 à 4 et 4 à 0), deux flèches sont attribuées à chaque pièce. Chaque flèche contient une valeur de récompense instantanée, comme indiqué ci-dessous :



On peut imaginer que notre agent comme un robot virtuel stupide qui peut apprendre par l'expérience. L'agent peut passer d'une pièce à l'autre mais n'a aucune connaissance de l'environnement ni quelle séquence de portes mène à l'extérieur. Supposons que nous voulions modéliser une sorte d'évacuation simple d'un agent de n'importe quelle pièce du bâtiment. Supposons maintenant que nous ayons un agent dans la salle 2 et que nous voulions que l'agent apprenne à atteindre l'extérieur de la maison (5).



Supposons que l'agent se trouve à l'état 2. À partir de l'état 2, il peut passer à l'état 3 car l'état 2 est connecté à 3. À partir de l'état 2, l'agent ne peut pas passer directement à l'état 1 car il n'y a pas de porte directe reliant la pièce 1 et 2 (donc, pas de flèches). A partir de l'état 3, il peut aller à l'état 1 ou 4 ou revenir à 2 (regardez toutes les flèches à propos de l'état 3). Si l'agent est à l'état 4, les trois actions possibles doivent passer à l'état 0, 5 ou 3. Si l'agent est à l'état 1, il peut passer à l'état 5 ou 3. A partir de l'état 0, il ne peut aller que Retour à l'état 4. Nous pouvons placer le diagramme d'état et les valeurs de récompense instantanée dans le tableau de récompense suivant, "matrice R".

$$R = \begin{matrix} & \text{Action} \\ \text{State} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{matrix}$$

Nous allons commencer par définir la valeur du paramètre d'apprentissage $\Gamma = 0,8$ et l'état initial en tant que Salle 1. Initialise la matrice Q en tant que matrice à zéro :

$$Q = \begin{matrix} & \text{Action} \\ \text{State} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

Maintenant, imaginons ce qui se passerait si notre agent était à l'état 5. Regardez la sixième ligne de la matrice de récompense R (c'est-à-dire l'état 5). Il y a 3 actions possibles : passer à l'état 1,

4 ou 5. $Q(\text{état}, \text{action}) = R(\text{état}, \text{action}) + \text{Gamma} * \text{Max}[Q(\text{état suivant}, \text{toutes les actions})]$

$$Q(1, 5) = R(1, 5) + 0,8 * \text{Max}[Q(5, 1), Q(5, 4), Q(5, 5)] = 100 + 0,8 * 0 = 100$$

L'état suivant, 5, devient maintenant l'état actuel. Parce que 5 est l'état de l'objectif, nous avons terminé un épisode. Le cerveau de notre agent contient maintenant une matrice Q actualisée :

$$Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \left[\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \right] \end{matrix}$$

Pour le prochain épisode, nous commençons par un état initial choisi au hasard. Cette fois, nous avons l'état 3 comme notre état initial. Regardez la quatrième rangée de la matrice R ; il y a 3 actions possibles : passer à l'état 1, 2 ou 4. Par sélection aléatoire, nous choisissons d'aller à l'état 1 comme action. Maintenant, nous imaginons que nous sommes dans l'état 1. Regardez la deuxième rangée de la matrice de récompense R (c'est-à-dire l'état 1). Il y a 2 actions possibles : passer à l'état 3 ou à l'état 5. Ensuite, nous calculons la valeur Q : $Q(\text{état}, \text{action}) = R(\text{état}, \text{action}) + \text{Gamma} * \text{Max}[Q(\text{état suivant}, \text{toutes les actions})]$

$$Q(3, 1) = R(3, 1) + 0,8 * \text{Max}[Q(1, 3), Q(1, 5)] = 0 + 0,8 * \text{Max}(0, 100) = 80$$

Nous utilisons la matrice Q mise à jour du dernier épisode. $Q(1, 3) = 0$ et $Q(1, 5) = 100$. Le résultat du calcul est $Q(3, 1) = 80$ car la récompense est égale à zéro. La matrice Q devient :

$$Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \left[\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \right] \end{matrix}$$

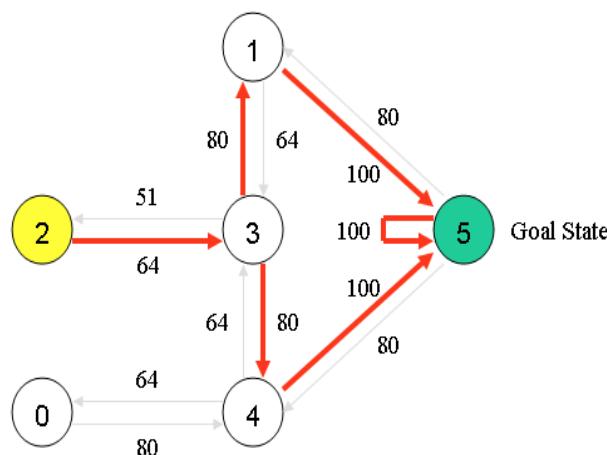
Si l'agent en apprend davantage au fil des épisodes, il atteindra enfin les valeurs de convergence dans la matrice Q comme :

$$Q = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 400 & 0 \\ 1 & 0 & 0 & 0 & 320 & 0 & 500 \\ 2 & 0 & 0 & 0 & 320 & 0 & 0 \\ 3 & 0 & 400 & 256 & 0 & 400 & 0 \\ 4 & 320 & 0 & 0 & 320 & 0 & 500 \\ 5 & 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix}$$

Cette matrice Q peut ensuite être normalisée (c'est-à-dire convertie en pourcentage) en divisant toutes les entrées non nulles par le nombre le plus élevé (500 dans ce cas) :

$$Q = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 80 & 0 \\ 1 & 0 & 0 & 0 & 64 & 0 & 100 \\ 2 & 0 & 0 & 0 & 64 & 0 & 0 \\ 3 & 0 & 80 & 51 & 0 & 80 & 0 \\ 4 & 64 & 0 & 0 & 64 & 0 & 100 \\ 5 & 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix}$$

Une fois que la matrice Q se rapproche suffisamment d'un état de convergence, nous savons que notre agent a appris les chemins les plus optimaux vers l'état d'objectif. Tracer les meilleures séquences d'états est aussi simple que de suivre les liens avec les valeurs les plus élevées dans chaque état.



5.1.2 Implémentation python

Q Learning

```
import numpy as np
```

```

import pylab as plt
import random

# List of points that are mapped to each other.
# Our goal is to get from any point to the goal point, in this case 5

environment = [(0,4),(4,3),(2,3),(1,3),(4,5),(1,5)]

## Rewards Matrix

# We have 6 possible states and 6 possible actions
# We have 6 actions because, lets say from 3, we can choose an action to go to
# state 2, 1 or 4.
# Lets look at it another way, we have the following possible actions
# Up, Down, Front, back, slant in, slant out ( Ok bad choice of names, but I am
# creatively impaired, sooo ...)

states = 6
actions = 6
goal = 5

R = np.matrix(np.ones(shape=[states,actions]))
R = R * -1 # We assign -1 to every value

for row in environment:
    R[row] = 0
    if row[1] == goal:
        R[row] = 100
    row = row[::-1]
    R[row] = 0

R[goal,goal] = 100

## Q matrix : Our brain
# Initially, we will have an empty Q matrix, which as we iterate through the
# different possible states and actions
# the RL process will populate with values it can use to make decisions.

Q = np.matrix(np.zeros(shape=[states,actions]))

print("The rewards Matrix is \n \n {}".format(R))
print("\n ")
print("The Q table Matrix is \n \n {}".format(Q))

# Now let us create the structure of the simple Q Learner
print("1) Please refer to the image above for better clarity \n")

# Current State

```

```

currentstate = 1

# Let us get all possible actions for current state
# From the rewards function, we can see which direction we can go in

def available_actions(state):
    rewards_row_state = R[state,:]
    av_actions = np.where(rewards_row_state >= 0)[1] # In our case, any value ge 0 is a
        # possible action we can take from our current state
    return av_actions
actions = available_actions(currentstate)
print("2) As we can see, the available actions from state {} is {}".format(
    currentstate,actions))

# Lets say, we choose one action from the available actions, so we choose randomly

def choose_random_action(actions):
    action = random.choice(actions)
    return action

action = choose_random_action(actions)
print("3) The action we have chosen to take is to go to {}, from state {} \n".
    format(action,currentstate))

# we now have information about our current state, current action we are taking,
# and possible next states and actions we can take

gamma = 0.8

print("4) Here we have pause a little and introduce a hyper parameter called Gamma"
    )
print(" Gamma is a paramter that is used to control if the model needs to focus on
    immediate reward or future rewards \n")

# Now we develop our Q update function

print("5) The Equation for our Q learner is as follows")
print(" Q[state,action] = R[state,action] + gamma * Max(Q[next state,all_actions])"
    )

def Q_update(state,action,gamma):
    # First the max Q part
    Max_Q = np.max(Q[action,:])
    Q[state,action] = R[state,action] + gamma * Max_Q
    return Q

```

```

Q = Q_update(currentstate,action,gamma)

for i in range(100):
    current_state = random.choice([0,1,2,3,4,5]) # Step 1
    actions = available_actions(current_state) # Step 2
    action = choose_random_action(actions) # Step 3
    Q = Q_update(current_state,action,gamma) # Step 4
    # We can normalize Q Matrix values for a better idea of what is going on

Q = Q/np.max(Q)*100

print ("Updated Q After 100 episodes")
print(Q)

# run after training
currentstate = 2
tracking = []
tracking.append(currentstate)

while currentstate != goal:
    nextstep = np.where(Q[currentstate,:] == np.max(Q[currentstate,:]))[1]
    tracking.append(nextstep[0])
    if len(nextstep) > 1:
        nextstep = random.choice(nextstep)
    else:
        nextstep = nextstep
    currentstate = nextstep
print("The most efficient path from state {} is {}".format(tracking[0],tracking))

```

Bibliographie

https://github.com/jeffheaton/t81_558_deep_learning/blob/master/t81_558_class01_intro_python.ipynb <https://github.com/machinelearningmindset/machine-learning-course>
<https://openclassrooms.com/courses/initiez-vous-au-machine-learning> <https://machinelearningmastery.com/simple-linear-regression-tutorial-for-machine-learning>
<http://onlinestatbook.com/2/regression/intro.html> <https://www.edvancer.in/>
<http://step-step-guide-to-execute-linear-regression-python> <https://dataaspirant.com/2014/>
<http://12/20/linear-regression-implementation-in-python> <https://machinelearningmastery.com/>
<http://com/implement-simple-linear-regression-scratch-python> <https://www.dezyre.com/>
<http://data-science-in-r-programming-tutorial/linear-regression-tutorial> <https://medium.com/>
<http://com/simple-ai/linear-regression-intro-to-machine-learning-6-6e320dbdaf06> <http://www.ozzieliu.com/tutorials/Linear-Regression-Gradient-Descent.html> <http://www.ozzieliu.com/2016/02/09/gradient-descent-tutorial> <http://www.kdnuggets.com/2016/11/>
<http://linear-regression-least-squares-matrix-multiplication-concise-technicaloverview.html> <http://http://www.real-statistics.com/multiple-regression/least-squares-method-multiple-regression>
https://www.youtube.com/watch?v=Qa_FI92_qo8 <https://mariuszprzydatek.com/2014/11/11/iterative-dichotomiser-3-id3-algorithm-decision-trees-machine-learning>
<https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python>
<https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148>
<https://towardsdatascience.com/understanding-logistic-regression-9b02c2aec102>
SkitLearnLinearregression:<https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-p>
SkitLearnMultiplelinearregression:<https://www.kaggle.com/striderking/>
linear-regression-with-multiple-variable
PerofrmancesMetricsClassification:<https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d43208>
PerofrmancesMetricsRegression:<https://becominghuman.ai/understand-regression-performance-metric>
Featuresselection:<https://machinelearningmastery.com/feature-selection-machine-learning-python>
Dataanalysis:<https://medium.com/@harrypotter0/an-introduction-to-data-analysis-with-pandas-27ec>
Dataanalysis:<https://www.kaggle.com/kashnitsky/topic-1-exploratory-data-analysis-with-pandas>
projetmachinelearning:<https://machinelearningmastery.com/machine-learning-in-python-step-by-step>
sklearnlinearregression:<https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-py>
logisticregression:<https://machinelearningmastery.com/logistic-regression-tutorial-for-machine-learni>
<https://machinelearningmastery.com/linear-regression-tutorial-using-gradient-descent-for-machine-learni>
KNN:<https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn>
KNN:https://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html
logisticregression:https://www.geeksforgeeks.org/ml-logistic-regression-using-python
DecisionTree:https://dataaspirant.com/2017/02/01/decision-tree-algorithm-python-with-scikit-learn
Kmeans:https://www.datascience.com/blog/k-means-clustering
Kmeans:https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering
Kmeans:https://blog.floydhub.com/introduction-to-k-means-clustering-in-python-with-scikit-learn
DecisionTree:https://sefiks.com/2017/11/20/a-step-by-step-id3-decision-tree-example
DecisionTree:https://sefiks.com/2018/08/27/a-step-by-step-cart-decision-tree-example
ANN:https://medium.com/technology-invention-and-more/everything-you-need-to-know-about-artificial-neural-networks-57fac18
ANN:<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example> ANN:
<https://stackabuse.com/introduction-to-neural-networks-with-scikit-learn> Kmeans:<https://www.kaggle.com/shrutimechlearn/step-by-step-kmeans-explained-in-detail/data> PCA:

<https://www.kindsongenius.com/2019/01/12/principal-components-analysis-pca-in-python-step-by-step/>
[PCA:https://machinelearningmastery.com/calculate-principal-component-analysis-scratch-python/](https://machinelearningmastery.com/calculate-principal-component-analysis-scratch-python/)
[Qlearning:http://mnemstudio.org/path-finding-q-learning-tutorial.htm](http://mnemstudio.org/path-finding-q-learning-tutorial.htm)
[Q-Learniqg:https://github.com/SuryaKari/Q_Learning/blob/master/Simple_Q_Learner.ipynb](https://github.com/SuryaKari/Q_Learning/blob/master/Simple_Q_Learner.ipynb) Qlearning:<https://rubikscode.net/2019/06/24/introduction-to-q-learning-with-python-and-open-ai-gym> Qlearning:<https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc>
[Overfitting:https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning](https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning)
<https://machinelearningmastery.com/compare-machine-learning-algorithms-python-scikit-learn>
<https://machinelearningmastery.com/prepare-data-machine-learning-python-scikit-learn>
<https://machinelearningmastery.com/evaluate-performance-machine-learning-algorithms-python-using-metrics>
<https://machinelearningmastery.com/understand-machine-learning-data-descriptive-statistics-python>
<https://machinelearningmastery.com/feature-selection-machine-learning-python> https://
<https://machinelearningmastery.com/visualize-machine-learning-data-python-pandas> Qlearning:
<https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d07f3>
<https://www.geeksforgeeks.org/ml-expectation-maximization-algorithm> https://
<https://www.kaggle.com/charel/learn-by-example-expectation-maximization> https://
<https://machinelearningmastery.com/expectation-maximization-em-algorithm> https://
<https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python> https://
<https://machinelearningmastery.com/imbalanced-classification-with-python-7-day-mini-course>
<https://tivadaradanka.com/blog/how-tsne-works> https://<https://towardsdatascience.com/t-sne-clearly-explained-d84c537f53a> https://<https://www.datatechnotes.com/2020/11/tsne-visualization-example-in-python.html> https://<https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/#:~:text=Ensemble%20learning%20refers%20to%20algorithms,discussed%20and%20used%20in%20practice.> https://https://github.com/EzzatEsam/SVM-Implementation-Python-QuadraticProgramming/blob/master/svm_quad.ipynb