# Petite introduction aux méthodes de l'IA

Première partie

G. Barmarin

Aujourd'hui les techniques mises en œuvre pour faire « réfléchir » nos machines sont multiples:

- Logique floue
- Algorithme génétiques
- Data mining
- Inférence bayésienne
- Réseaux de neuronnes
- Apprentissage automatique
- …

Même si actuellement, ce sont l'apprentissage automatique et plus particulièrement les réseaux de neurones et le deep learning qui bouleversent ce domaine.

## Un petit mot quand même sur la logique floue (fuzzy logic)

La logique floue est une logique polyvalente où les valeurs de vérité des variables — au lieu d'être vrai ou faux — sont des réels entre 0 et 1.

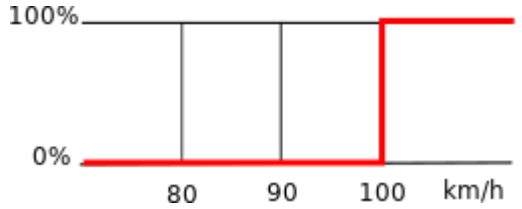
En ce sens, elle étend la logique booléenne classique avec des valeurs de vérités partielles.
Elle consiste à tenir compte de divers facteurs numériques pour aboutir à une décision qu'on souhaite acceptable.

Elle est formalisée par une *théorie mathématique des ensembles flous* de Lotfi Zadeh (1965) qui présente une extension de la théorie des ensembles classiques aux *ensembles définis de façon imprécise*.

La logique floue remplace la valeur de vérité d'une proposition à choisir dans {*vrai*,*faux*} par un degré de vérité, à choisir par exemple dans [0, 1]. En logique floue, il y a donc des degrés dans la satisfaction d'une condition.

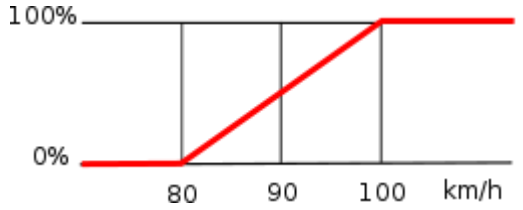
Prenons l'exemple de la vitesse d'un véhicule sur une route où la vitesse normale est de 90 km/h. La vitesse est considérée élevée au-dessus de 100 km/h et réglementaire en dessous de 80 km/h. On souhaite caractériser la vitesse du véhicule en répondant par exemple à la question « La vitesse est-elle élevée ? ».

Dans ce modèle de réglementation routière, en logique booléenne, la réponse à la question s'énonce de la manière suivante: La vitesse est élevée à 100 % au-dessus de 100 km/h et à 0 % en dessous.



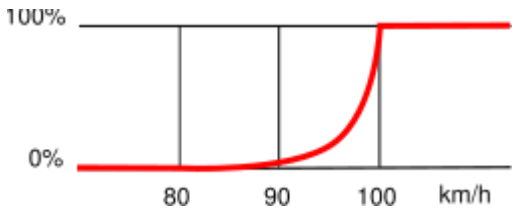

*Logique booléenne*

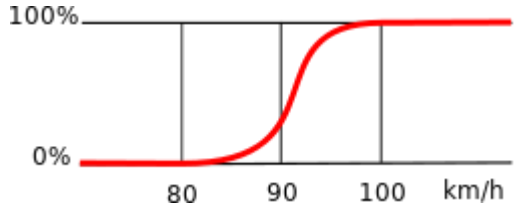
En logique floue, on autorise différents degrés d'énoncés de réponse à la question « La vitesse est-elle élevée ? » :
La vitesse est réglementaire en dessous de 80 km/h. On peut donc dire qu'en dessous de 80 km/h, la vitesse est élevée avec un taux de confiance de 0 %.
La vitesse est élevée au-dessus de 100 km/h. La vitesse est non réglementaire avec un taux de confiance de 100 % au-dessus de 100 km/h.
Aux stades intermédiaires, on considère que la vitesse est non réglementaire à 50 % de confiance à 90 km/h et à 25 % de confiance à 85 km/h.

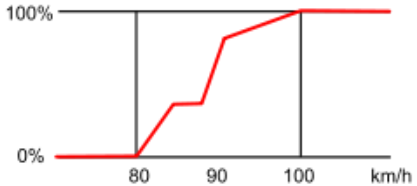



*Logique floue*

Il n'est pas obligatoire que la transition soit linéaire. Des transitions hyperboliques (comme une sigmoïde ou une tangente hyperbolique), exponentielle, gaussienne (dans le cas d'un état moyen) ou de toute autre nature sont utilisables.



Le degré de vérité d'une relation floue entre deux ou t objets est le degré d'appartenance de la paire ou du t-uple à l'ensemble flou associé à la relation.

Soit la relation est1 (est-un/est-une). Pour dire que toute chaise est un meuble et que 30 % des meubles sont des chaises, on accordera un degré de vérité de 1 à est1(chaise, meuble) et un degré 0,3 à est1(meuble, chaise).

La connaissance topographique qu'un agent possède d'un monde clos pourra s'appuyer sur l'attribution de degrés de vérité à une relation du style x est_plus_près_de y que_de z, construite et/ou affinée par apprentissage.

De façon générale, les relations floues permettront de coder des connaissances graduées,

En définissant les opérateurs logiques de base de la façon suivante:

ET = min OU = max NON(x) = 1-x

il devient possible de représenter toutes les opérations logiques de base en logique floue :

| Opérateur | Abréviation | Application |
|---|---|---|
| OU | OR | A *OU* B $= max(A, B)$[9] |
| ET | | A *ET* B $= min(A, B)$ |
| NON | | *NON* A $= 1 - A$ |
| OU Exclusif | OUEX | A *OUEX* B $= max(A, B) - min(A, B)$ [réf. nécessaire] |
| NI | NON-OU | A *NI* B $= 1 - max(A, B)$ |
| ON | NON-ET | A *ON* B $= 1 - min(A, B)$ |
| EQV | NON-OUEX | A *EQV* B $= 1 + min(A, B) - max(A, B)$ |
| REPETEUR | REP | *REP* A $= A$ |

# L'apprentissage automatique

## Définition

L'apprentissage automatique peut-être vu comme l’ensemble des techniques permettant à une machine d’apprendre à réaliser une tâche sans avoir à la programmer explicitement pour cela.
(*Définition d'Antoine Gaudelas*)

L'apprentissage automatique ou apprentissage statistique est un domaine de l'intelligence artificielle qui concerne la conception, l'analyse, le développement et l'implémentation de méthodes permettant à un ordinateur d'évoluer par un processus systématique, et ainsi de remplir des tâches difficiles ou problématiques à remplir par des moyens algorithmiques plus classiques.
(*Définition adaptée de Wikipédia*)

Le type de tâches traitées consiste généralement en des problèmes de classification de données:

Voici 50 photos de ma fille, voici maintenant toutes les photos de mon album, retrouve celles ou se trouve ma fille

Voici 10000 personnes avec leurs caractéristiques (âge, localisation, taille, profession...), rassemble-les en 12 groupes cohérents partageant les mêmes points communs

Voici les paramètres de vol des avions de ma flotte avec les pannes survenues à chaque vol, quand il s'en produit. Maintenant voici les derniers vols réalisés par cet avion, indique-moi quand arrivera la prochaine panne et sur quel élément elle se produira.

Voici les livres écrits par Victor Hugo, voici une nouvelle dont nous recherchons l'auteur, est-elle de lui ?

Voici des mesures scientifiques réalisées sur un échantillon de données, elles semblent répondre à une loi Mathématique, essaye de l'approximer par des fonctions Mathématiques puis prédit les valeurs des prochaines données

## Présentation de l'apprentissage automatique

Quand on parle d'apprentissage automatique, on parle de différents types d'apprentissages, parmi les plus répandus on citera:

- supervisé,
- non supervisé,
- par renforcement,
- par transfert

Yann Le Cun qui est considéré comme l'un des inventeurs du Deep Learning résume ainsi ces différentes classes:

*« La majorité des types d'apprentissages chez l'homme et l'animal sont non supervisés. Si l'apprentissage automatique était un gâteau, l'apprentissage non supervisé serait ce gâteau, l'apprentissage supervisé serait le nappage du gâteau et l'apprentissage par "renforcement" serait la cerise sur le gâteau. Nous savons faire le nappage et la cerise, mais nous ne savons pas faire le gâteau. »*

certains ont des avis plus nuancés.

# Apprentissage supervisé

L'apprentissage supervisé permet de répondre à des problématiques de classification et de régression.

L'idée consiste à associer un label (une étiquette) à des données sur lesquelles vous possédez des mesures.

•Si les labels sont discrets (des libellés ou valeurs finies) on parlera de classification.

•Si au contraire les labels sont continus (comme l'ensemble des nombres réels), on parlera de régression.

## Classification

La classification consiste à *donner des étiquettes* à ses données:

•Vous disposez d'un ensemble de données connues que vous avez déjà classé (photos, plantes, individus...)
•Vous souhaitez, à partir de cette première classification, dite connaissance, classer de nouveaux éléments

Certains logiciels d'albums photos utilisent ce type d'apprentissage pour classer vos images:

•Ils vous permettent de désigner un ensemble de photos contenant votre enfant et d'indiquer où se trouve ce dernier dans ces images. *C'est la phase d'apprentissage*
•Puis vous lui dites, voici ma collection de 15000 photos, retrouve toutes celles qui contiennent mon enfant
Le logiciel analyse alors votre collection et tente de retrouver celles qui présentent une similitude avec le jeu de données que vous lui avez enseigné. Certains logiciels vous indiquent même où se situe votre enfant dans l'image.
*C'est la phase de prédiction*

Vous trouverez facilement de nombreux exemples sur Internet, notamment en Python, comme celui-ci utilisant un réseau de neurones.

Ce type de classification permet de répondre à de nombreux problèmes d'identification : reconnaissance de plantes, de personnes, de produits, reconstitution de valeurs manquantes (en remplacement d'une interpolation) etc...
Il peut utiliser différents types d'algorithmes, comme *les plus proches voisins*, *les machines à vecteurs de supports*, *les arbres décisionnels*, ...

## Apprentissage non supervisé

L'apprentissage non supervisé répond au même besoin de classification de données.

Mais contrairement à l'apprentissage supervisé, vous ne possédez pas de données déjà classées/connues servant de base à la prédiction.

Vous utilisez ce type d'algorithme pour répondre à 2 types de tâches:

•La classification de données, mais comme vous n'avez pas encore les labels, on parle alors de *clustering* c'est à dire de regroupement.

•L'autre tâche consiste à réduire le nombre de dimensions de vos données tout en conservant une variation/un regroupement semblable aux données d'origine, on parle alors de *réduction de dimension (dimensionality reduction)*

# Clustering

Avec ce type d'apprentissage il n'y a qu'une phase de prédiction.

Vous alimentez l'algorithme de toutes vos données et lui demandez de les répartir en N groupes.
L'algorithme tentera alors de créer des groupes pour lesquels les paramètres de chaque donnée sont les plus similaires.

Quelques exemples d'applications:

Classer les cultures d'une région:
Vous disposez des métriques sur les parcelles agricoles d'une région (teneur en nitrates, phosphates, salinité, surface, haies, ...) et vous savez qu'il y a 12 cultures différentes.
Vous allez demander à votre classifieur de créer 12 groupes en les classant selon des critères de ressemblance.
Vous possédez des statistiques sur une population (salaire, localité, âge, profession, nombre d'enfants), vous souhaitez les regrouper en différentes catégories sociales.

Les algorithmes des plus proches voisins, arbres de décision, les réseaux de neurones, la propagation par affinité sont généralement utilisés pour ce type de traitement.

# Réduction de dimension

Ici vous avez un problème un peu différent : chaque information/donnée que vous souhaitez traiter possède trop de paramètres, soit pour être visualisée, soit pour être traitée dans des temps raisonnables et sur des machines limitées en ressources.

Vous devez donc diminuer le nombre de ces paramètres tout en conservant une cohérence sur leurs variations globale. De sorte que si vous regroupiez vos données en utilisant tous les paramètres vous obtiendriez sensiblement le même regroupement qu'avec les paramètres réduits.

Un usage type consiste à visualiser les groupes des données dans un espace en 2D ou 3D.

Un algorithme fréquemment utilisé est l'analyse en composantes principales (PCA), mais il en existe d'autres.

# Apprentissage par renforcement

Ce type d'apprentissage s'applique plus à des problèmes d'optimisations.
L'idée étant de faire prendre des décisions à un système pour obtenir un résultat qui soit le meilleur possible.

Ce type d'algorithme est très inspiré d'études du comportement en biologie animale ou psychologie. Il est d'ailleurs étroitement lié à ces disciplines.

Pour cela l'algorithme va appliquer des règles sur son environnement pour arriver au résultat attendu. Il dispose de la faculté de mesurer l'impact de la règle sur l'environnement : on se rapproche du but ou on s'en écarte.

A partir de là, il peut donc se constituer une base de connaissances des gains réalisés par chaque action qui l'aidera à améliorer ses décisions et ainsi trouver les meilleures manières d'atteindre son but.

Les algorithmes ne garantissent pas forcément d'obtenir le meilleur résultat, mais de s'en approcher. Là où ils commencent à se complexifier c'est que l'obtention de petits gains immédiats ne doit pas empêcher de chercher des gains plus forts qui ne s'obtiennent qu'après plusieurs actions ayant entraîné une suite de pertes.

# Apprentissage par renforcement

Différents algorithmes sont disponibles pour ce type d'apprentissage, comme le Q-Learning ou Monte-Carlo, SARSA, mais ce sont les réseaux de neurones qui semblent être les plus pratiqués t.

Exemples d'utilisation:

La robotique: faire se déplacer un robot dans des conditions mouvantes
Optimiser le déplacement des ascenseurs
Router des paquets réseau
Jeux de stratégie
Chemin parcouru par les fourmis pour acheminer leur nourriture à la fourmilière

Pour creuser le sujet:

- Apprentissage par renforcement – de la théorie à la pratique
- Un cours de l'INRIA et quelques autres
- Des vidéos de robots réalisant diverses tâches grâce à l'apprentissage par renforcement

# Apprentissage par transfert

L'apprentissage par transfert vise à utiliser les connaissances d'un jeu de tâches sources pour non seulement influencer l'apprentissage mais aussi améliorer les performances sur une autre tâche cible.

Il consiste en quelque sorte à utiliser les connaissances acquises pour les ré-appliquer dans un autre environnement.

Pour être efficace l'environnement cible ne doit pas être trop différent de celui des tâches sources, sinon des transferts négatifs seront réalisés menant au contraire du résultat recherché.

Un exemple d'utilisation consisterait à transférer des tâches d'un réseau de neurones utilisé pour la reconnaissance manuscrite de l'écriture dans une langue pour l'appliquer à une autre langue, même très différente (français/japonais).

# Les principaux algorithmes de l’apprentissage

Chaque type d'apprentissage peut s'appuyer sur différents algorithmes :

- Linear Regression (régression linéaire),
- Logistic Regression (régression logistique),
- Decision Tree (arbre de décision),
- Random forest (forêts d'abres/arbes aléatoires)
- SVM (machines à vecteur de support),
- Naive Bayes (classification naïve bayésienne),
- KNN (Plus proches voisins),
- Gradient Boost & Adaboost,
- Dimensionality reduction
- Q-Learning
- Réseaux de neurones
- ...

# La régression linéaire

La régression linéaire consiste à déterminer une équation de droite/plan qui se rapproche au plus près de l'ensemble des points étudiés. L'idée étant de déterminer les coefficients a et b de l'équation

$$y = F(x) = ax + b + \epsilon(x)$$

- *y* est la variable que l'on cherche à calculer, prédire, elle est dite **endogène** (dépendante).
- *x* est la variable prédictive, elle est dite **exogène** (indépendante).
- *a* et *b* sont les paramètres (les coefficients) du modèle.

Dans le cas de la régression simple, *a* est appelée la **pente** et *b* est appelée **la constante**. ε est le bruit généré lors de la mesure et perturbant la bonne identification de la relation.

On utilise ensuite cette équation pour prédire de nouvelles données.
Ce système n'est efficace que si les relations entre vos données sont simples.
Dans le cas de courbes plus complexes qu'une droite ou qu'un plan on se tournera vers des solutions polynomiales et gaussiennes.

Les algorithmes de type SVM sont très adaptés pour ces cas plus avancés.

# La régression logistique

Les régressions logistiques, aussi appelées *modèle logit*, sont un cas particulier des régressions linéaires.

Cela consiste à modéliser l'effet d'un vecteur de données/paramètres aléatoires sur une variable binomiale, c'est à dire pouvant avoir seulement 2 états, Vrai/Faux (0/1), d'où le terme *logistique*.

*y* est la variable expliquée et les vecteurs *(x1, x2, ..., xn)* les variables prédictives/explicatives.

Le site Wikipédia fournit une description complète du modèle probabiliste associé. Il s'agit bien d'une régression linéaire même si le fait de n'avoir que 2 valeurs possibles pour '*y*' fait penser à un problème de classification.

Elle est utilisée dans de multiples cas (Wikipédia):

•En médecine, elle permet par exemple de trouver les facteurs qui caractérisent un groupe de sujets malades par rapport à des sujets sains.
•Dans le domaine des assurances, elle permet de cibler une fraction de la clientèle qui sera sensible à une police d'assurance sur tel ou tel risque particulier.
•Dans le domaine bancaire, pour détecter les groupes à risque lors de la souscription d'un crédit.
•En économétrie, pour expliquer une variable discrète. Par exemple, les intentions de vote aux élections.

# Les arbres de décision

L'application Akinator vous amuse en devinant un personnage auquel vous pensez.
Cette application est basée sur le principe des arbres de décision:

- A chaque noeud une question est posée permettant de limiter l'ensemble des solutions restantes en 2 parties disjointes et les plus importantes possibles. (principe de la dichotomie, calcul de la médiane).
- Puis au noeud suivant une autre question est posée pour limiter de la même manière l'ensemble des solutions restantes
- Jusqu'à ce qu'il n'en reste plus qu'une seule.

Dans le cas de l'apprentissage automatique les arbres de décision sont construits par l'algorithme pour essayer de diviser ainsi les données sur les paramètres des vecteurs:

- Chaque nœud interne décrit un test sur un paramètre d'apprentissage
- Chaque branche représente un résultat du test
- Chaque feuille contient la valeur de la variable cible
    - une étiquette de classe pour les arbres de classification
    - une valeur numérique pour les arbres de régression

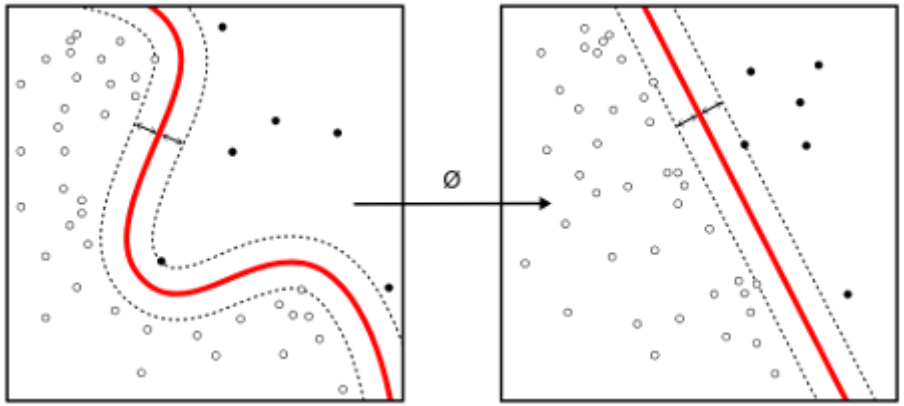
La pertinence de l'algorithme construisant l'arbre se mesurera à sa capacité de trouver les paramètres qui permettent de maximiser le partage à chaque noeud.
Les arbres de décision s'utilisent en apprentissage supervisé.
L'article Wikipédia fourni en lien décrit bien les contraintes techniques de construction de ces arbres et

Random forest

Les forêts d'arbres décisionnels/aléatoires sont basées sur le concept de bagging/inférence statistique et d'arbres décisionnels.
L'idée étant d'apprendre sur de multiples arbres de décision travaillant sur des sous-ensembles de données les plus indépendants possible.
Cela permet de régler plusieurs problèmes inhérents aux arbres de décision uniques comme l'altération du résultat selon l'ordre des paramètres prédicteurs dans les noeuds, ou encore de réduire leur compléxité.

# Les machines à vecteur de support (SVM)

Les machines à vecteur de support sont très utilisées dans les problèmes de régression et offrent une extension aux régressions linéaires lorsque les données présentes des niveaux de séparation *plus tordus* !




Elles permettent de calculer des données quand leurs labels ne sont pas séparables par une équation linéaire. Elles proposent de séparer les données avec des équations plus riches comme des polynomes, gaussiennes, etc...
Elles connaissent un très grand succès, pour de multiples raisons:

- Elles peuvent travailler avec des données disposant d'un très grand nombre de paramètres
- Elles utilisent peu d'hyperparamètres
- Elles garantissent de bons résultats théoriques
- Elles peuvent égaler ou dépasser en performance les réseaux de neurones ou modèles gaussiens

## KNN

L'algorithme des plus proches voisins est relativement simple.

Une de ses forces est de ne calculer aucune information dans le processus d'apprentissage.

Il recherche les N plus proches voisins (par un calcul de distance) entre la donnée à prédire et les données connues. Il retourne alors la classe de la majorité des voisins.

Assez simple à mettre en oeuvre il peut générer beaucoup de calculs et ne pas être adapté à de fortes volumétries ; notamment si le nombre de paramètres est très grand.

## Gradient Boost & Adaboost

Le gradient boosting est un algorithme s'appliquant aux problèmes de classification et régression.

L'idée est d'améliorer la prédiction et la vitesse en combinant un ensemble d'algorithmes d'apprentissages plus simples au travers d'un arbre de décision.

Le travail revient alors à identifier la fonction permettant de maximiser le choix des différents algorithmes.

AdaBoost est une variante du gradient boost. Il combine via une somme pondérée le résultat de différents algorithmes d'apprentissage plus simples. Il est adaptatif dans le sens où il peut jouer sur le poids des différents algorithmes simples en fonction de la qualité de leurs résultats.

## Naive Bayes

La classification naïve bayésienne est basée sur le théorème de Bayes permettant de déterminer la distribution d'une loi binomiale.

Il s'agit d'une équation décrivant la relation entre des probabilités conditionnelles de quantités statistiques.

Elle s'inscrit dans le groupe des classifieurs linéaires.

Nous souhaitons ici trouver la probabilité d'un label à partir de paramètres observés, noté *P(L|params)*
Le théorème de Bayes permet de calculer cette information:

$$P(L \mid \text{params}) = \frac{P(\text{params} \mid L)P(L)}{P(\text{prams})}$$

La génération d'un modèle sur cette loi se fait pour chaque label et peut être ardue.
Le modèle est dit naïf car il simplifie grandement cette tâche en procédant à plusieurs approximations naïves. Il est de ce fait très rapide et c'est un bon modèle pour commencer une classification.

## Réseaux de neurones et deep learning

Les réseaux de neurones tentent de reproduire le fonctionnement des neurones biologiques.

Pour vulgariser leur fonctionnement:

•il s'agit d'un graphe ou chaque nœud représente un neurone
•chaque neurone reçoit une information quand il est stimulé
•il ne la transmet (généralement en la modifiant ou complétant) aux neurones auxquels il est connecté que si celle-ci dépasse un certain seuil

Ils sont généralement très rapides et permettent des mécanismes *perceptifs* indépendants du concepteur de l'algorithme. Ils ouvrent la voie au raisonnement de la machine.

L'algorithme porte sur la construction d'un automate dont les états peuvent évoluer:
•Les valeurs seuils peuvent être adaptées au fil des informations traitées en utilisant des systèmes correctifs de récompense/blâme par exemple
•Les paramètres *synaptiques* peuvent aussi varier
•Tout comme les connexions entre neurones
Dans ces réseaux la phase d'apprentissage vise à faire converger les paramètres des données vers une classification optimale.
Ils nécessitent énormément de données d'apprentissage et ne sont pas adaptés à tous les problèmes notamment si le nombre de paramètres en entrée est trop faible.

## Réseaux de neurones et deep learning

Le terme *Deep learning* désignes des réseaux de neuronnes juxtaposés, ou encore constitués de plusieurs couches.

Il s'inspire, entre autres, des dernières avancées en neuroscience et des modèles de communication de notre système nerveux.

Certains l'associent aussi à une modélisation offrant un plus haut niveau d'abstraction des données pour offrir de meilleures prédictions.

Le deep learning est particulièrement efficace sur le traitement de l'image, du son et de la vidéo. On le retrouve dans les domaines de la santé, robotique, vision par ordinateur, ...

Du fait de leur grand besoin en données d'apprentissage ils sont très coûteux en ressources matérielles.

# Scikit-learn

pip install scikit-learn
pip install pandas

# Machine Learning et Python

De multiples librairies existent:

- Annoy, librairie extrêmement rapide implémentant la recherche des plus proches voisins
- Caffe, Deep learning framework
- Chainer, Framework intuitif pour les réseaux de neurones
- neon, Deep Learning framework extrêmement performant
- NuPIC, Plateforme d'IA implémentant les algorithmes d'apprentissage HTM
- Shogun, Large Scale Machine Learning Toolbox
- TensorFlow, Réseau de neurones disposant d'une API de haut niveau
- Torch, Framework d'algorithmes d'apprentissage très performant disposant de binding Python
- Theanets, deep learning
- ...

Elles sont toutes généralement de grande qualité et utilisées dans des environnements professionnels.

Toutefois, Scikit-Learn est probablement la plus populaire des librairies disponibles pour ce langage. Elle possède un grand nombre de fonctionnalités spécialisées dans l'analyse de données et le data Mining qui en font un outil de choix pour les chercheurs et développeurs.

## Scikit-learn (sklearn)

Scikit-learn, encore appelé sklearn, est la bibliothèque la plus puissante et la plus robuste pour le machine learning en Python.

Elle fournit une sélection d'outils efficaces pour l'apprentissage automatique et la modélisation statistique, notamment la classification, la régression et le clustering via une interface cohérente en Python.

Cette bibliothèque, qui est en grande partie écrite en Python, s'appuie sur NumPy, SciPy et Matplotlib.

Pourquoi utiliser Scikit-learn ?
Il n'y a pas beaucoup de pages sur Internet où l'on peut réellement trouver les raisons pour lesquelles Scikit-learn est devenu populaire parmi les scientifiques des données, mais après des recherches avancées, nous avons compris pourquoi il est si populaire. Les principales forçes de Scikit learn sont :

Licence BSD : Scikit-learn possède une licence BSD ; par conséquent, il existe une restriction minimale sur l'utilisation et la distribution du logiciel, ce qui le rend libre d'utilisation pour tous.

Facile à utiliser : la popularité de Scikit-learn est due à la facilité d'utilisation qu'il offre.

Documentation détaillée : Il propose également une documentation détaillée de l'API à laquelle les utilisateurs peuvent accéder à tout moment sur le site Web, ce qui les aide à intégrer l'apprentissage automatique dans leurs propres plateformes.

Utilisation intensive dans l'industrie : Scikit-learn est largement utilisé par diverses organisations pour prédire le comportement des consommateurs, identifier les activités suspectes, et bien plus encore.

Algorithmes d'apprentissage automatique : Scikit-learn couvre la plupart des algorithmes d'apprentissage automatique via un énorme soutien communautaire : la possibilité d'effectuer des tâches d'apprentissage automatique à l'aide de Python a été l'une des principales raisons de la popularité de Scikit-learn, car Python est facile à apprendre et à utiliser (Apprenez Python ici) et dispose déjà d'une vaste communauté d'utilisateurs qui peuvent désormais effectuer de l'apprentissage automatique sur une plateforme avec laquelle ils sont à l'aise.

Qui utilise Scikit-learn ?
Sckit-learn est utilisé dans de nombreux secteurs comme outil principale pour la mise en place de modèles prédictives. Je détaille dans cette partie comment certaines des entreprises les plus connues l'utilisent comme outil phare dans leurs systèmes de recommandations et de prédictions des risques :

Spotify : Scikit-learn est beaucoup utilisé pour les recommandations musicales chez Spotify.

Inria : à l'INRIA, scikit-learn est utilisé pour soutenir la recherche fondamentale de pointe dans de nombreuses équipes : Parietal pour la neuro-imagerie, Lear pour la vision par ordinateur, Visages pour l'analyse d'images médicales, Privatics pour la sécurité.

Booking.com : Booking.com utilisent des algorithmes d'apprentissage automatique pour de nombreuses applications différentes, telles que la recommandation d'hôtels et de destinations à leurs clients, la détection de réservations frauduleuses ou la programmation de leurs agents du service clientèle. Scikit-learn est l'un des outils qu'ils utilisent pour mettre en œuvre des algorithmes standard pour les tâches de prédiction.

BNP Paribas Cardif : BNP Paribas Cardif utilise scikit-learn pour plusieurs de ses modèles d'apprentissage automatique en production. Leur communauté interne de développeurs et de data scientists utilise scikit-learn depuis 2015, pour plusieurs raisons : la qualité des développements, de la documentation et de la gouvernance des contributions, et la taille même de la communauté de contributeurs. Ils utilisent des pipelines de scikit-learn dans leur gouvernance interne du risque de modèle comme l'une de leurs bonnes pratiques pour diminuer les risques opérationnels et le risque d'overfitting.

Installer Scikit-learn
La façon la plus simple d'installer non seulement Scikit-learn, mais aussi Python et ses paquets les plus populaires (IPython, SciPy NumPy, Matplotlib) est d'utiliser Anaconda, une distribution Python multiplateforme (Linux, macOS, Windows) pour l'analyse de données et le calcul scientifique.

Les instructions d'installation pour Anaconda sont disponibles ici: https://docs.continuum.io/anaconda/install/

Après l'installation d'Anaconda, les commandes suivantes vous permettront  d'acquérir tout l'environnement nécessaire pour travailler avec Scikit-learn.

# Étude de cas : classification des plantes d'iris

Voici le **Hello World** de la Data Science : l'étude des plantes d'iris.

Nous allons donc utiliser le jeu de données des plantes d'iris.

Cet ensemble de données se compose de quatre champs, à savoir la longueur du sépale, la largeur du sépale, la longueur du pétale et la largeur du pétale.

Il contient également une super classe qui contient trois classes différentes, *Iris setosa, Iris versicolor* et *Iris virginica*.
Il s'agit essentiellement des espèces de plantes Iris, et les données de notre ensemble de données, c'est-à-dire les plantes Iris, sont divisées en trois classes.

Nous allons importer cet ensemble de données et ensuite exécuter des algorithmes d'apprentissage automatique sur celui-ci.

## Importation du jeu de données

Comme nous l'avons mentionné précédemment, l'ensemble de données que nous allons utiliser dans cette étude de cas est l'ensemble de données des plantes d'iris. Scikit learn Python est livré avec cet ensemble de données, nous n'avons donc pas besoin de le télécharger depuis une autre source. Nous allons importer l'ensemble de données directement, mais avant cela, nous devons importer Scikit learn et Pandas à l'aide des commandes suivantes :

```
import sklearn
import pandas as pd
```

Après avoir importé sklearn, nous pouvons facilement importer l'ensemble de données à partir de celui-ci, en utilisant la commande suivante :

```
from sklearn.datasets import load_iris
```

Nous avons importé avec succès l'ensemble de données des plantes Iris de sklearn. Nous devons maintenant importer Pandas, car nous allons charger les données importées dans un DataFrame Pandas et utiliser les fonctions head() et tail() de Python Pandas pour afficher le contenu du DataFrame. Voyons comment convertir cet ensemble de données en un DataFrame Pandas.

**Exploration des données**

Nous allons charger les données importées dans un DataFrame Pandas et utiliser les fonctions head() et tail() de Pandas pour afficher le contenu du DataFrame.

```
plantes_iris = load_iris() df_plantes_iris =
pd.DataFrame(plantes_iris.data, columns=plantes_iris.feature_names)
```

Nous avons maintenant un DataFrame nommé *df_plantes_iris* qui contient l'ensemble de données des plantes d'iris importé de Scikit-learn sous forme de tableau.
Toutes les opérations de Machine Learning seront appliquées sur ce DataFrame.

Affichons les enregistrements de ce DataFrame en utilisant la fonction *head()* :

```
df_plantes_iris.head()
```

La fonction head(), lorsqu'elle est utilisée sans argument, affiche les cinq premières lignes du DataFrame. Cependant, nous pouvons passer n'importe quel argument entier pour afficher le même nombre de lignes du DataFrame.

Voici le résultat:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) |
|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 |

La fonction tail(), lorsqu'elle est utilisée sans aucun argument, affiche les cinq dernières lignes du DataFrame.

```
df_plantes_iris.tail()
```

Comme pour la fonction head(), nous pouvons passer n'importe quel nombre entier comme argument pour afficher le même nombre d'enregistrements depuis la fin.
L'affichage de la commande ci-dessus est le suivant :

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) |
|---|---|---|---|---|
| **145** | 6.7 | 3.0 | 5.2 | 2.3 |
| **146** | 6.3 | 2.5 | 5.0 | 1.9 |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 |
| **148** | 6.2 | 3.4 | 5.4 | 2.3 |
| **149** | 5.9 | 3.0 | 5.1 | 1.8 |

Comme la fonction tail() affiche les derniers enregistrements du DataFrame, nous pouvons voir que le numéro d'index de la dernière ligne est 149. En revanche, lorsque nous utilisons la fonction head(), le numéro d'index de la première ligne est 0, c'est-à-dire que le nombre total d'entrées est de 150 ou qu'un total de 150 enregistrements est présent dans l'ensemble de données des plantes d'iris.

Maintenant, voyons comment nous pouvons vérifier les types de données des champs présents dans le DataFrame.

```
df_plantes_iris.dtypes
```

Résultat affiché :

*sepal length (cm)  float64*
*sepal width (cm)  float64*
*petal length (cm)  float64*
*petal width (cm)  float64*
*dtype: object*

Ainsi, en utilisant *dtypes*, on peut lister les différentes colonnes du DataFrame, ainsi que leurs types de données Python respectifs.

## Visualisation des données

Après avoir effectué l'exploration de nos données, nous allons maintenant créer des graphiques pour représenter visuellement les données de notre ensemble de données, ce qui nous aidera à découvrir d'autres informations cachées.

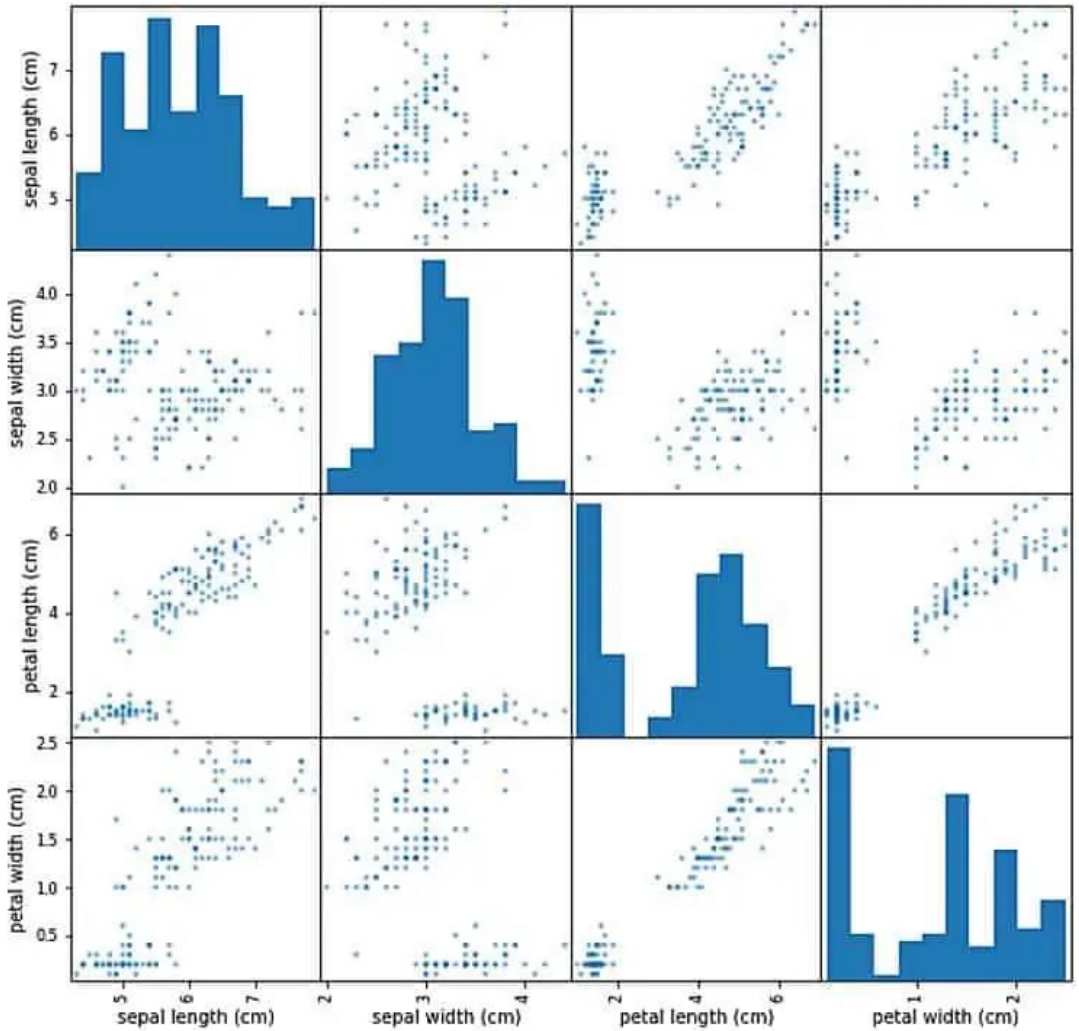
Python dispose de nombreuses bibliothèques qui fournissent des fonctions permettant de réaliser des visualisations de données sur des ensembles de données.

On peut utiliser l'extension *.plot* de Pandas pour créer un nuage de points des caractéristiques ou des champs de notre ensemble de données les uns par rapport aux autres, mais on doit également importer python *matplotlib* qui fournit une API orientée objet pour intégrer les graphiques dans les applications.

```
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
scatter_matrix(df_plantes_iris,figsize=(10,10))
plt.show()
```

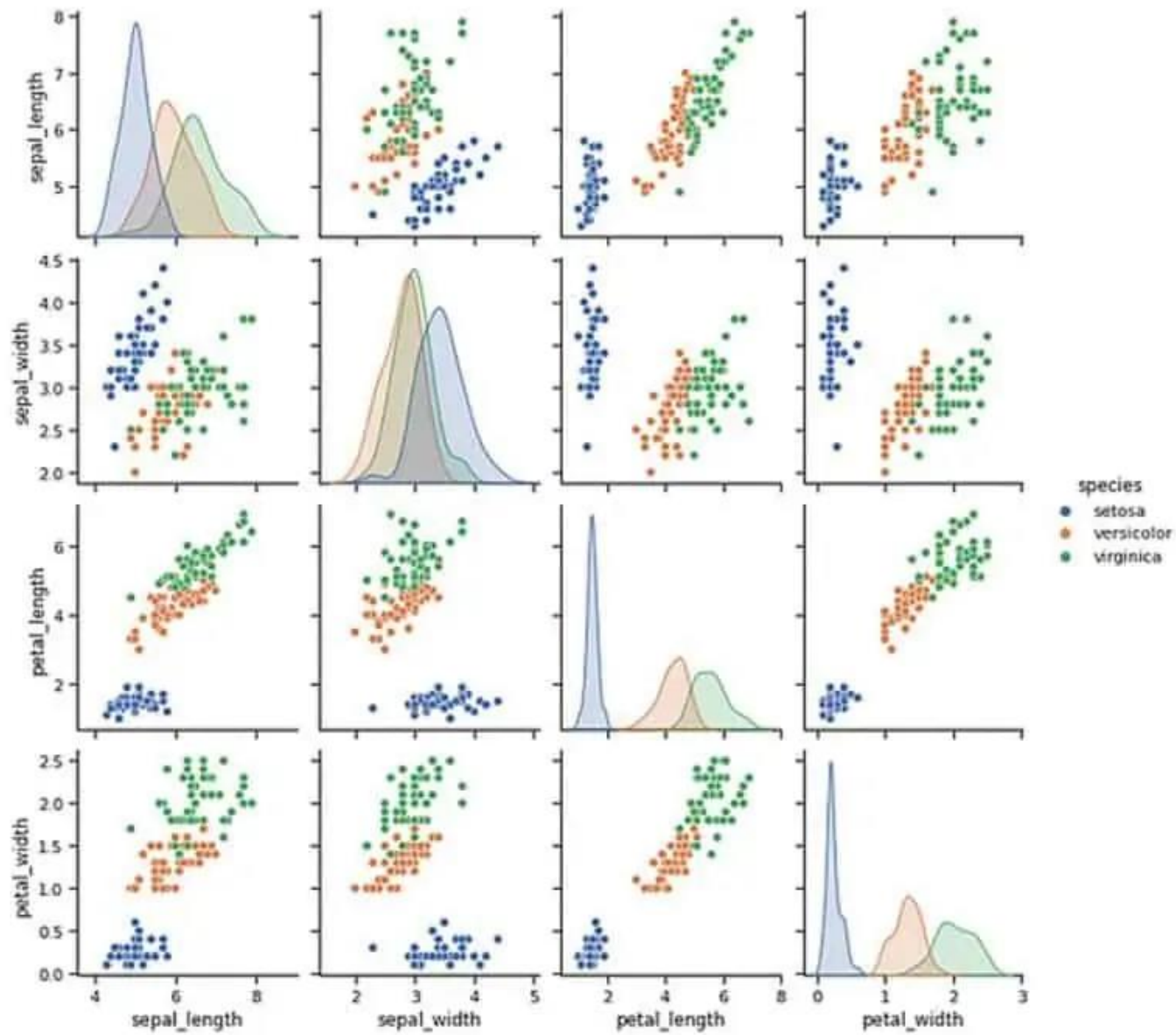
Résultat:

diagrammes de paires de toutes les caractéristiques de l'ensemble de données les unes par rapport aux autres.



# Une autre vue avec le package seaborn

```
<seaborn.axisgrid.PairGrid at 0x7f3fcf0220d0>
```



# Apprentissage et prédiction

Le nuage de points que nous avons créé n'est utile que dans une certaine mesure. Il est évident qu'il existe un regroupement des espèces de plantes Iris en différentes classes, et il montre également qu'il existe certaines relations entre les champs ou les caractéristiques. Mais il est difficile d'indiquer quelle classe représente quel type et quel point de données représente quelle espèce de fleur dans le nuage de points, en raison de la monotonie de la distribution des couleurs dans les points de données.

Heureusement, nous pouvons rectifier et surmonter ce problème en utilisant le module Seaborn pour la visualisation de données en Python. C'est exactement ce que nous avons fait en créant un diagramme en paires de l'ensemble de données donné à l'aide de Seaborn. Nous pouvons nous référer à ce diagramme pour tirer les conclusions et les prédictions.
Pour effectuer des prédictions sur cet ensemble de donnés les étapes sont :

1. Sélection des champs
2. Préparation des données
3. Ensemble d'entraînement et ensemble de test
4. Construire un modèle et choisir un classificateur
5. Tester notre modèle

# Sélection des champs

Il faut décider quelles caractéristiques ou quels champs de l'ensemble de données nous allons utiliser pour mettre en œuvre l'apprentissage automatique et faire des prédictions. Nous devons sélectionner les caractéristiques qui ont le plus de sens pour le modèle d'apprentissage automatique.

Pourquoi sélectionner des caractéristiques ?
On pourrait penser raisonnablement qu'on peut simplement utiliser toutes les caractéristiques pour notre modèle d'apprentissage automatique et laisser le modèle faire le travail pour nous en déterminant quelle caractéristique est la plus pertinente ?

La réponse à cette question est que toutes les caractéristiques ne servent pas d'information. L'ajout de fonctionnalités dans le seul but d'avoir des données dans le modèle rendra ce dernier inutilement lent et moins efficace. Le modèle s'embrouillera dans l'abondance de données inutiles et essaiera de les intégrer à lui-même, ce qui est inutile. C'est pourquoi nous devons sélectionner les caractéristiques qui seront utilisées dans le modèle.

Dans le diagramme de paires que nous avons créé à l'aide de scatter_matrix, nous pouvons remarquer que les caractéristiques petal length (cm) et petal width (cm) sont regroupées en groupes assez bien définis.

On remarque également que la frontière entre Iris versicolor et Iris virginica semble floue, ce qui pourrait poser problème à certains classificateurs. Nous devrons garder cela à l'esprit. Mais ces caractéristiques donnent toujours le regroupement le plus visible entre les espèces ; nous utiliserons donc ces deux caractéristiques pour notre modèle d'apprentissage automatique.

## Préparation des données

Pour l'instant, nous avons les données dans un DataFrame Pandas.
Avant de commencer avec le modèle d'apprentissage automatique, nous devons convertir les données en tableaux NumPy, car sklearn fonctionne bien avec les données sous forme de tableaux NumPy. Il ne fonctionne pas avec les DataFrame de Pandas.

Ceci peut être fait en utilisant la commande suivante :

```
import numpy as np
labels = np.asarray(plantes_iris.target)
```

L'outil, LabelEncoder() peut encoder les chaînes d'étiquettes en représentations numériques.
Il parcourt l'étiquette et convertit la première chaîne unique en 0, puis la suivante en 1, et ainsi de suite. Voici comment l'utiliser :

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(labels)
labels = le.transform(labels)
```

## Préparation des données

Maintenant, nous allons supprimer tous les éléments dont nous n'avons pas besoin de notre DataFrame en utilisant la méthode drop() :

```
df_selected1 = df_plantes_iris.drop(['sepal length (cm)', 'sepal
width (cm)', 'target'], axis=1)
```

Après cela, les seules caractéristiques qui nous restent sont la longueur et la largeur des pétales.

À l'aide de la commande suivante, nous allons convertir les caractéristiques numériques en tableaux d'étiquettes.

```
df_features = df_selected1.to_dict(orient='records')
from sklearn.feature_extraction import DictVectorizer
vec = DictVectorizer()
features = vec.fit_transform(df_features).toarray()
```

## Ensemble d’entraînement et ensemble de test

L’étape suivante consiste à diviser les données en un ensemble d’entraînement et un ensemble de test.

Là encore, sklearn dispose d’un outil pour le faire, appelé train_test_split.
Tout ce que nous avons à faire est de l’importer et de l’utiliser comme suit :

```
from sklearn.model_selection import train_test_split
features_train, features_test, labels_train, labels_test =
train_test_split(
features, labels, test_size=0.20, random_state=0)
```

Nos ensembles de test et d’apprentissage sont prêts.

Maintenant, effectuons la classification en utilisant des algorithmes ou des approches d’apprentissage automatique, puis comparerons la précision de test de tous les classificateurs sur les données de test.

## Construire un modèle et choisir un classificateur

Scikit-learn est fourni avec un organigramme qui aide les utilisateurs à décider quel algorithme d'apprentissage automatique conviendra le mieux à leur ensemble de données.

Nous allons l'utiliser comme référence pour identifier l'algorithme que nous devons utiliser sur nos données de test. L'organigramme est disponible sur le site officiel de Scikit-learn.

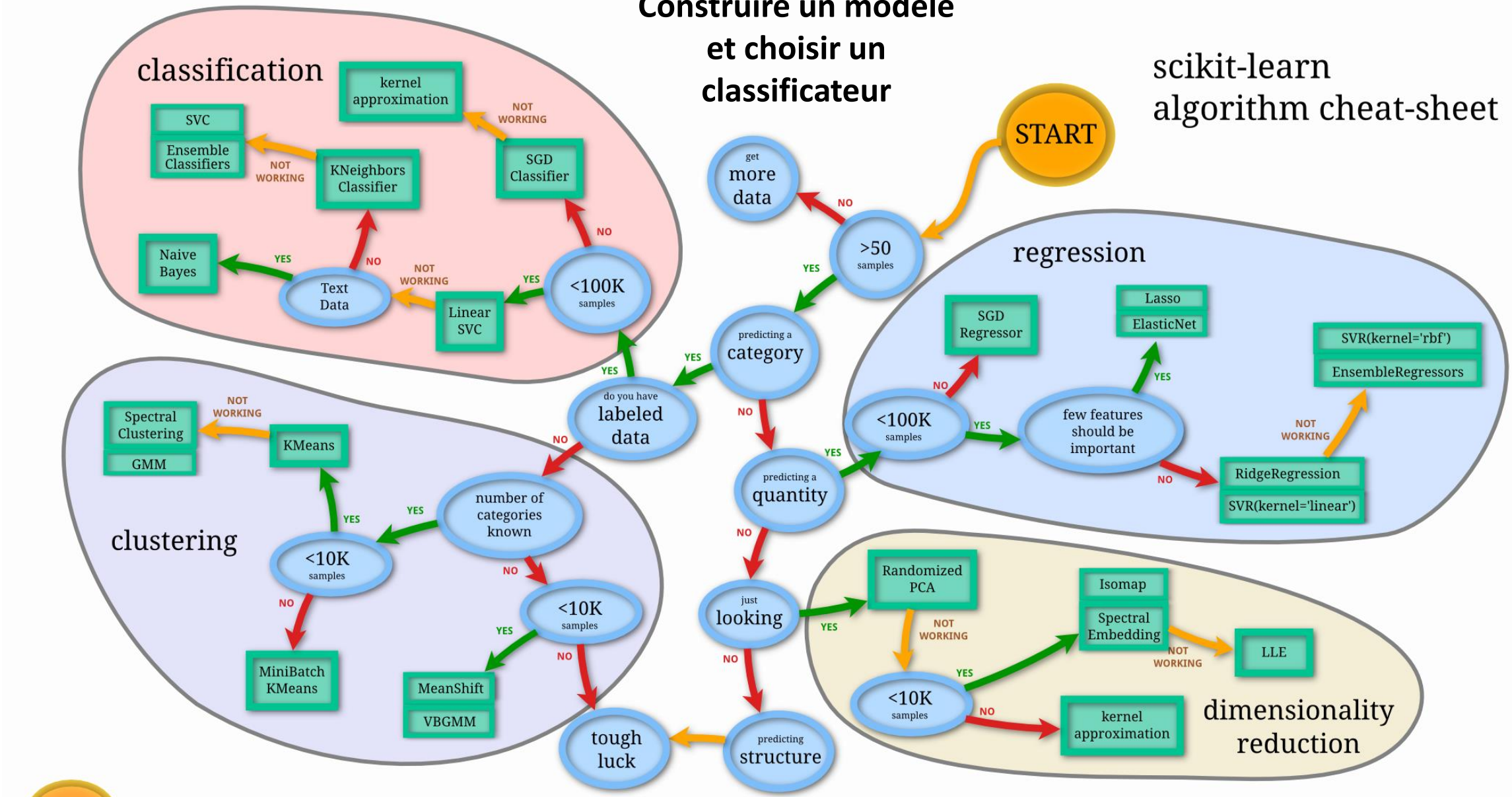
À l'aide de la liste suivante, voyons dans quelle catégorie nous nous situons :

- le nombre d'échantillons : notre nombre d'échantillons est supérieur à 50 et inférieur à 100 000
- les données sont-elles étiquetées ? nous avons des données étiquetées
- la catégorie est-elle prédite ? nous avons des prédictions sur la catégorie des plantes d'iris.

Donc, en parcourant l'organigramme, nous pouvons essayer les algorithmes suivants sur notre ensemble de test :

- SVM (Support Vector Machine)
- K-Nearest Neighbors Classifier

# Construire un modèle et choisir un classificateur




https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

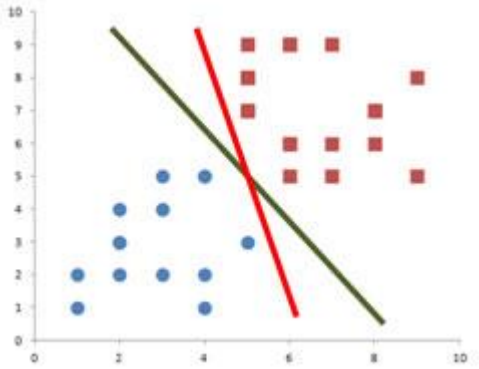
# SVM (Support Vector Machine)

Dans le domaine de l'apprentissage automatique, le SVM ou Support Vector Machine est un algorithme d'apprentissage dans lequel l'algorithme analyse les données et construit un modèle qui est utilisé principalement pour les techniques de classification ou de régression de l'apprentissage automatique. Ici, dans notre cas, nous utilisons le modèle SVM pour la classification.

Calcul de la précision en utilisant l'ensemble de test :

```
from sklearn.svm import SVC
svm_model_linear = SVC(kernel = 'linear', C =
1).fit(features_train, labels_train)
svm_predictions = svm_model_linear.predict(features_test)
accuracy = svm_model_linear.score(features_test, labels_test)
print("Précision du test:",accuracy)
```

Sortie : Précision du test : 1.0



Calcul de la précision à l'aide de l'ensemble d'entraînement :

```
from sklearn.svm import SVC
svm_model_linear = SVC(kernel = 'linear', C = 1).fit(features_train,
labels_train)
svm_predictions = svm_model_linear.predict(features_train)
accuracy = svm_model_linear.score(features_train, labels_train)
print("Précision entrainement:",accuracy)
```

Sortie: Précision entrainement: 0.9583333333333334

Maintenant, nous pouvons utiliser la précision de l'ensemble d'entraînement et la précision de l'ensemble de test que nous avons calculées pour déterminer dans quelle mesure notre modèle est surajusté en comparant ces deux précisions.

La suradaptation d'un modèle est une condition ou une erreur de modélisation où la fonction s'adapte trop étroitement à un ensemble limité de points de données.

Comme vous pouvez le voir, il n'y a pas de grande différence entre la précision du test et celle de l'entraînement, c'est-à-dire que notre modèle n'est pas surajusté.

Classificateur K-Nest Neighbors (voisins les plus proches)
KNN ou K-nearest neighbors est une méthode d'apprentissage non paramétrique en apprentissage automatique, principalement utilisée pour les techniques de classification et de régression. Il est considéré comme l'un des algorithmes les plus simples de l'apprentissage automatique.

Calcul de la précision en utilisant l'ensemble de test :

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 7).fit(features_train, labels_train)
accuracy = knn.score(features_test, labels_test)
print("Précision du test:", accuracy)
```

Sortie

```
Précision du test: 1.0
```

Calcul de la précision à l’aide de l’ensemble d’entraînement :

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 7).fit(features_train, labels_train)
accuracy = knn.score(features_train, labels_train)
print("Précision entrainement:", accuracy)
```

Sortie

```
Précision entrainement: 0.958333333333334
```

Là encore, nous pouvons utiliser la précision de l’ensemble d’entraînement et celle de l’ensemble de test pour déterminer si le modèle est surajusté.

## Tester notre modèle

Nous pouvons maintenant faire des prédictions sur des données inédites ou nouvelles.
Par exemple, si nous choisissons une nouvelle fleur d'iris au hasard et que nous mesurons les 2 valeurs des caractéristiques, notre modèle peut maintenant prédire l'espèce de la fleur sur la base de ce qu'il a appris de l'ensemble de données d'apprentissage.

Supposons que les valeurs des caractéristiques mesurées soient les suivantes :

longueur du pétale = 3,3 cm
largeur du pétale = 2,9 cm

Enregistrons les valeurs des caractéristiques dans un objet appelé X_new et utilisons l'estimateur knn pour prédire le code de l'espèce :

```
X_new = [[3.3, 2.9]]
print(knn.predict(X_new))
print(plantes_iris.target_names[knn.predict(X_new)])
```

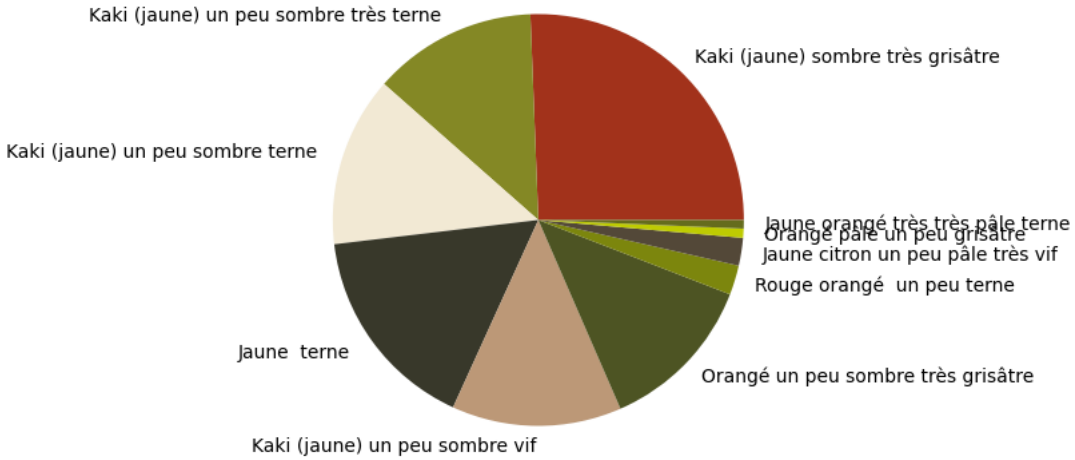
Notre modèle a prédit que la nouvelle fleur appartient au type d'Iris Verscicolor, qui a une valeur de réponse de 1.

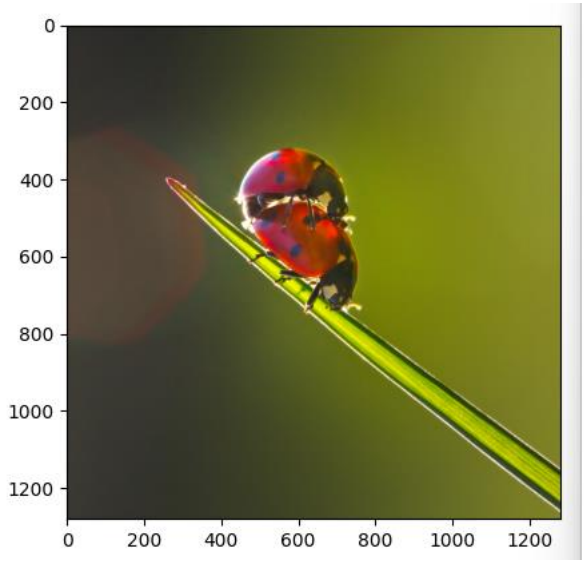
# K-Means

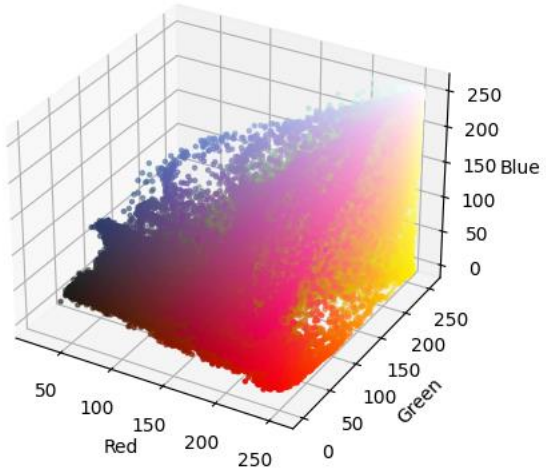
K-Means est un algorithme de clustering en Unsupervised Learning.

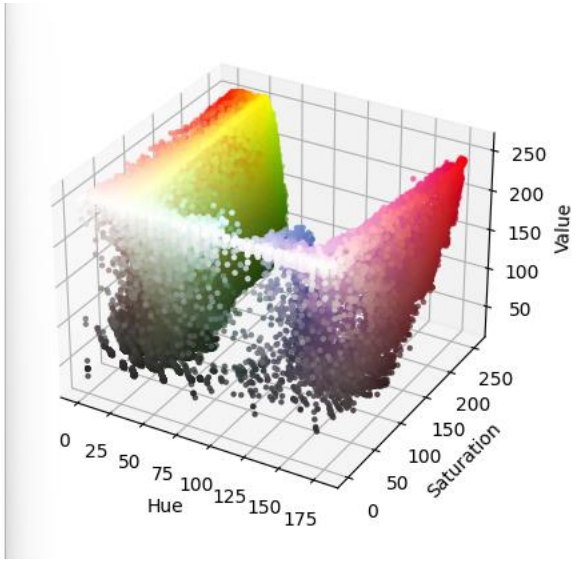
On lui donne un ensemble d'éléments (des données),
et un nombre de groupes K.
K-means va segmenter en K groupes les éléments.

Le groupement s'effectue en minimisant la distance
euclidienne entre le centre du cluster et un élément donné.




Exemple de la classification de couleurs présentes dans une image (Analyse -des-couleurs-principales-image.py)

**Pour aller plus loin**

•https://makina-corpus.com/data-science/initiation-au-machine-learning-avec-python-la-theorie

•Wikipédia renferme de bonnes références sur l'intelligence artificielle

•Le site O'Reilly tient un blog sur l'IA et propose quelques livres gratuits intéressants

- The New Artificial Intelligence Market
- Intelligence Artifical Now
- What Is Artificial Intelligence

•AFIA, l'Association Française pour l'Intelligence Artificielle

•Le livre Python Data Science Handbook entièrement en notebooks