Majeure Machine Learning

Techniques

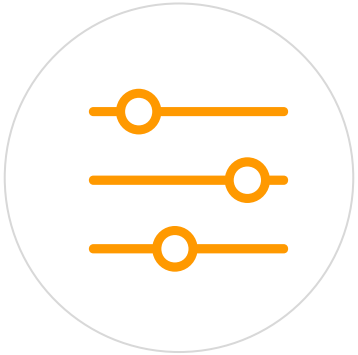
Contenu

- Différents algorithmes supervisés
- Méthodes ensemblistes
- Différents algorithmes non-supervisés
- Hyperparameters Tuning

Ce que vous devrez savoir faire

- Comprendre le principe de fonctionnement de certains algorithmes de ML
- Comprendre les méthodes ensemblistes
- Etre capable de choisir les bonnes techniques
- Comprendre les différentes techniques d'Hyperparameters Tuning

Apprentissage Supervisé



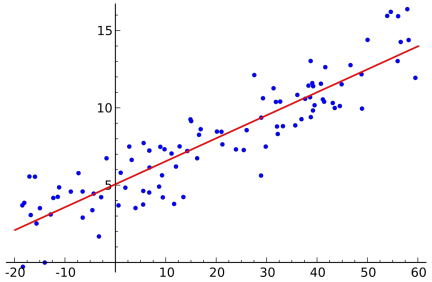
Algorithmes paramétriques

Définition : Nombre de paramètres est fixe et donc la complexité ne dépend pas du nombre de lignes des données

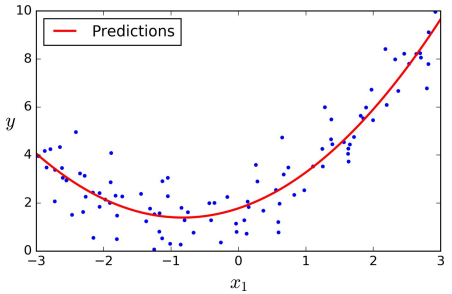
Régression

Régression

- Linéaire : $\hat{y}(x) = \theta_1 x_1 + \theta_0$



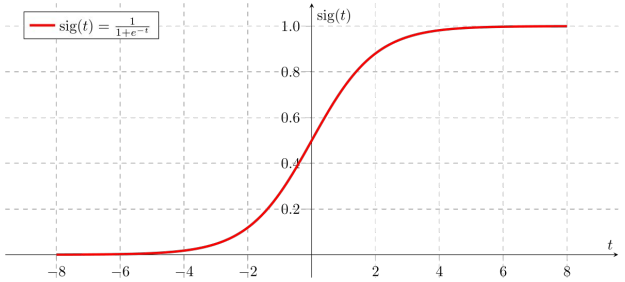
- Polynomiale : $\hat{y}(x) = \theta_2 x^2 + \theta_1 x_1 + \theta_0$



Classification

- Logistique : $\hat{y}(x) = \sigma(\theta_1 x_1 + \theta_0)$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Naive* Bayes

Théorème de Bayes :

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

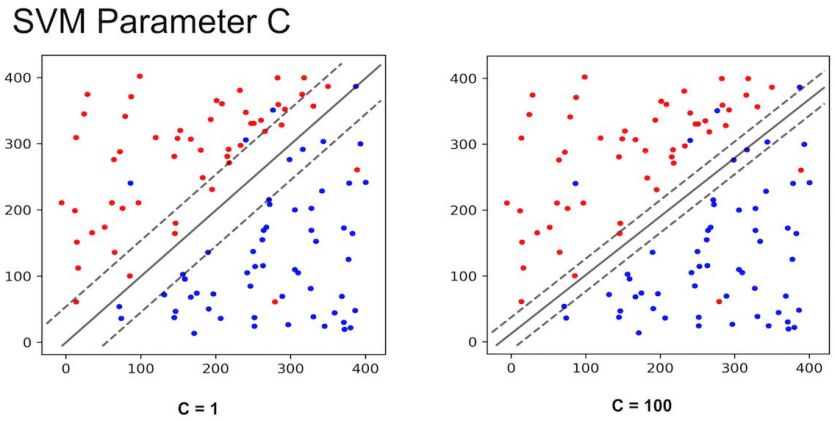
**Naive => Fait l'hypothèse que les features sont indépendantes les unes des autres.*

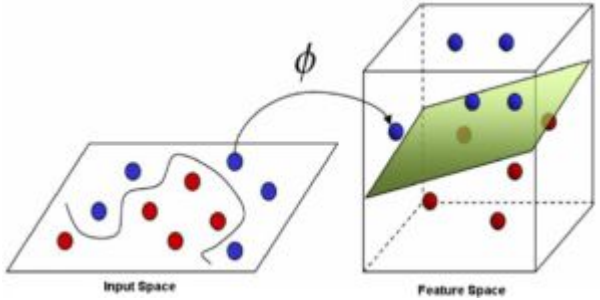
Peu probable mais marche plutôt bien quand même

- $P(X_1 \dots X_n|Y) = \prod_{i=1}^{n} P(X_i|Y)$ => Likelihood
- P(Y) = 1/n_classes => Prior
- P(X_n) = Gaussian, Bernoulli… => Evidence

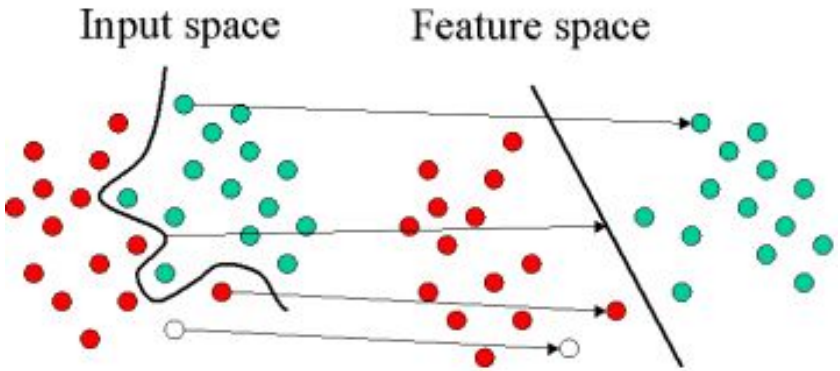
SVM (Support Vector Machine)

Marge





Kernel

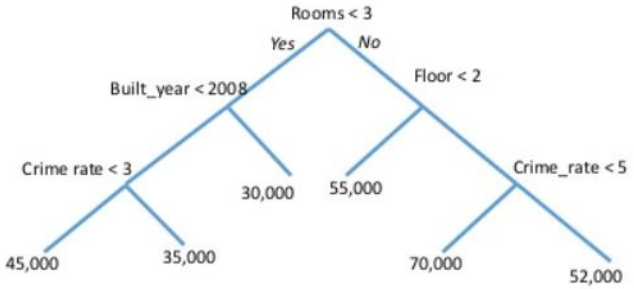


Algorithmes non-paramétriques

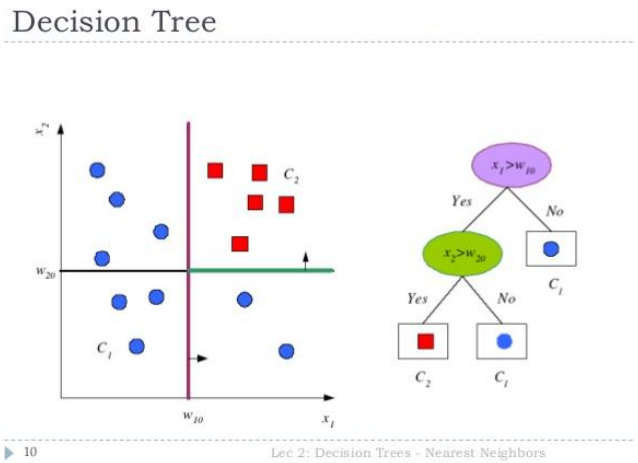
Définition : Nombre de paramètres et donc complexité du modèle dépendante du nombre de lignes

Arbre de décision (CART)

Régression



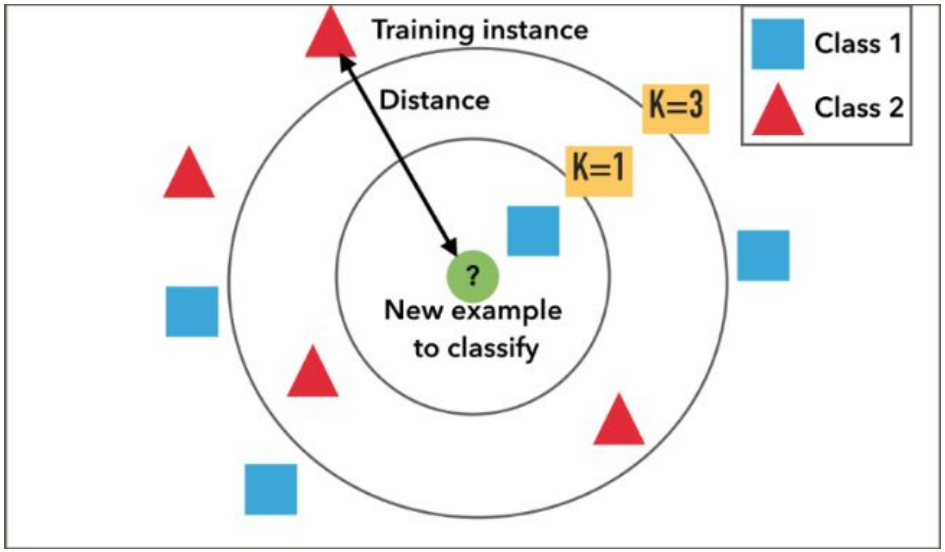
Classification



Tester toutes les séparations possibles
*Trouver la feature et la valeur qui sépare le mieux les données**
Séparer par cette valeur
Entrer dans une des séparations
Si Nb_element > min_element_par_feuille :
    Recommencer le process
Sinon :
    Si nb_elements > 0:
        Prédiction = mean(elements) / majorité(elements)
    Sinon :
        Prédiction = élément
    Changer de séparation ou remonter

* Cost function :
- Classification : Gini criterion
- Régression : Somme des erreurs au carré

K-Nearest-Neighbors



Prédiction = moyenne / majorité des k plus proches voisins

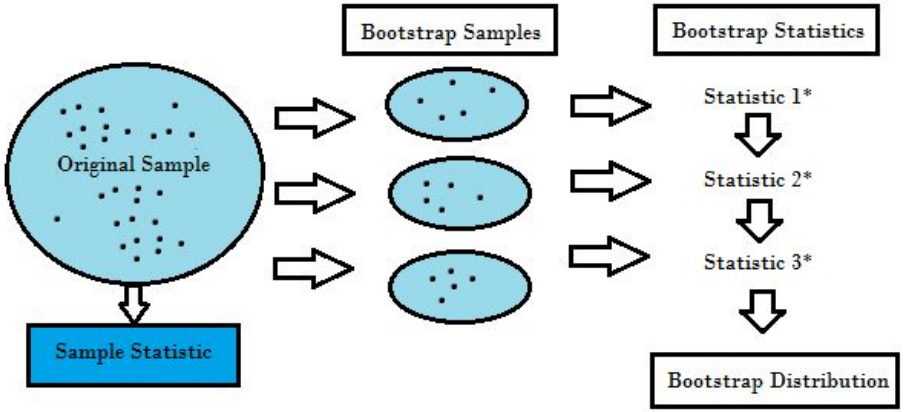
Algorithmes ensembliste

Même avec un seul algorithme on peut améliorer sa performance à l’aide d’un ensemble

Bootstrapping

Objectif : Pouvoir générer plusieurs ensembles d'entraînement

$D_{bootsrap} \leftarrow \{\}$
pour N *itérations*
  choisir aléatoirement et uniformément un entier n *parmi* $\{1, \ldots N\}$
  $D_{bootsrap} \leftarrow D_{bootsrap} \cup \{(x_n, t_n)\}$
$retourner D_{bootsrap}$

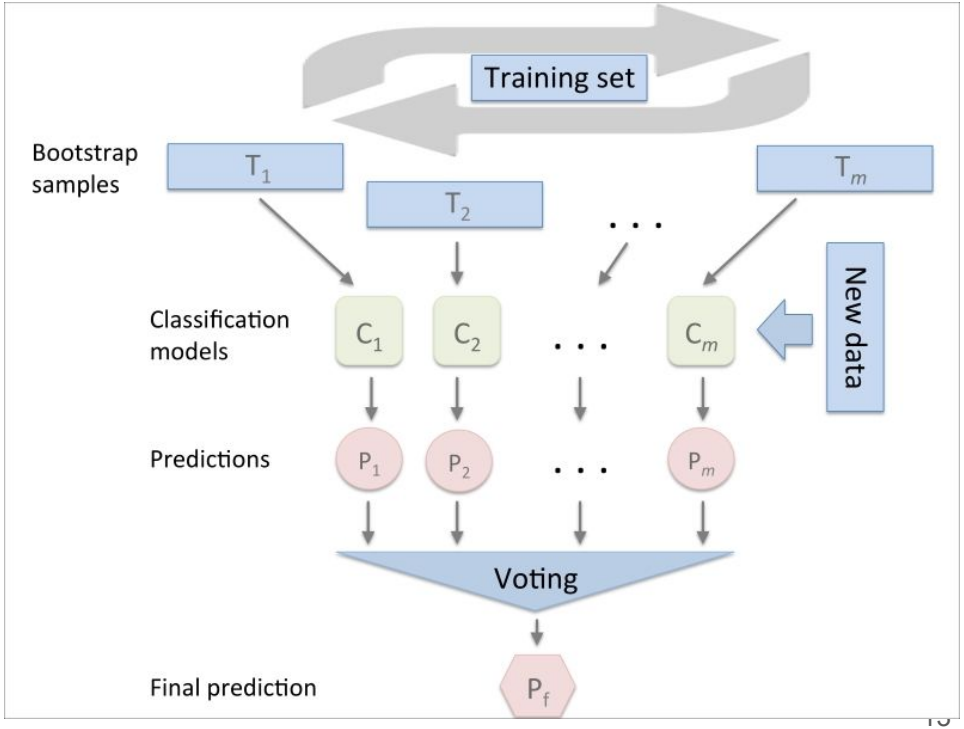


Combinaisons de modèles - Bagging (bootstrap aggregaation)

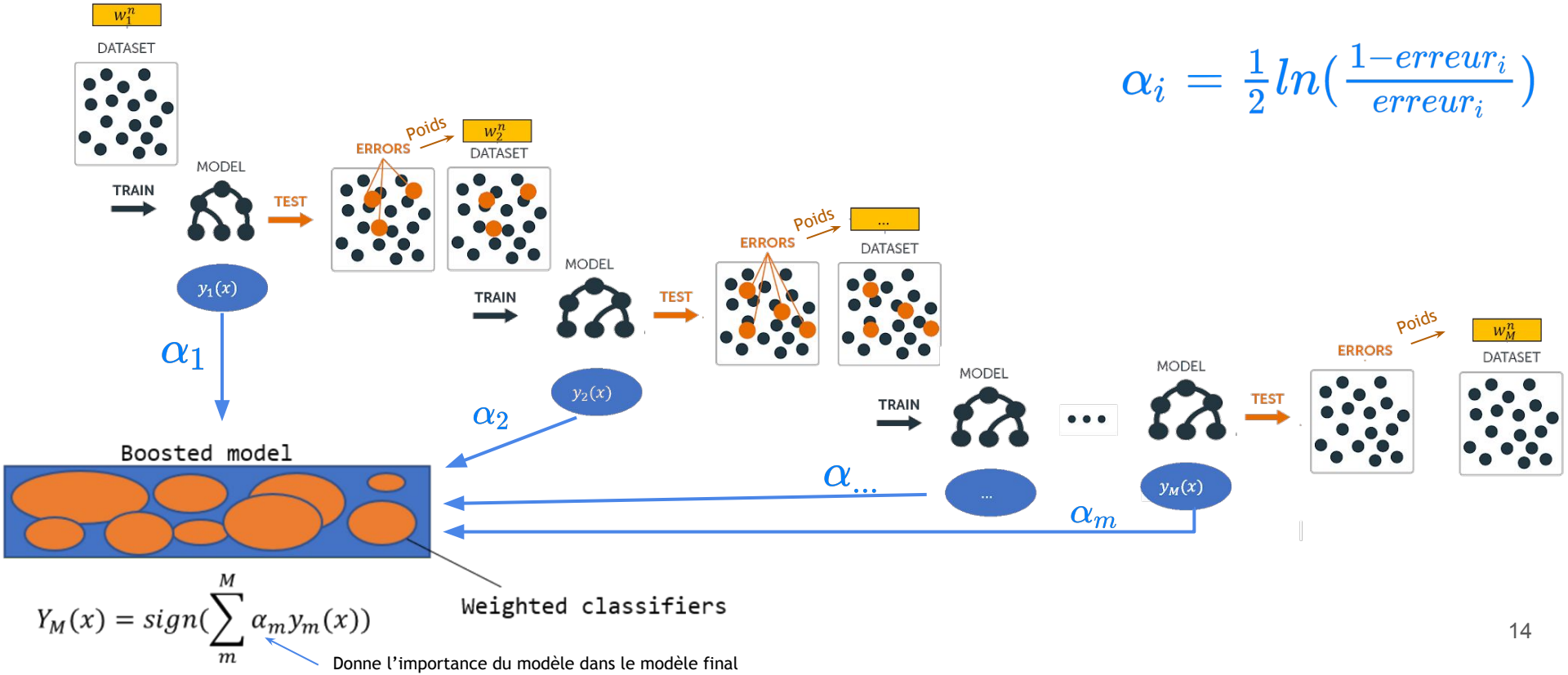
A utiliser lorsque les modèles ont une forte capacité

- ***Bagging :*** entraîne M modèles avec un algorithme donné, sur M ensembles de données *bootstrap*
  - pour $m = 1, ..., M$
    - génère un ensemble de données *bootstrap* $\mathcal{D}_{\text{bootstrap}}$ à partir de $\mathcal{D}$
    - entraîner un modèle $y_m(\mathbf{x})$ sur $\mathcal{D}_{\text{bootstrap}}$
  - retourner le modèle ensemble (comité)

$$y_{\text{COM}}(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x})$$



Combinaisons de modèles - Boosting



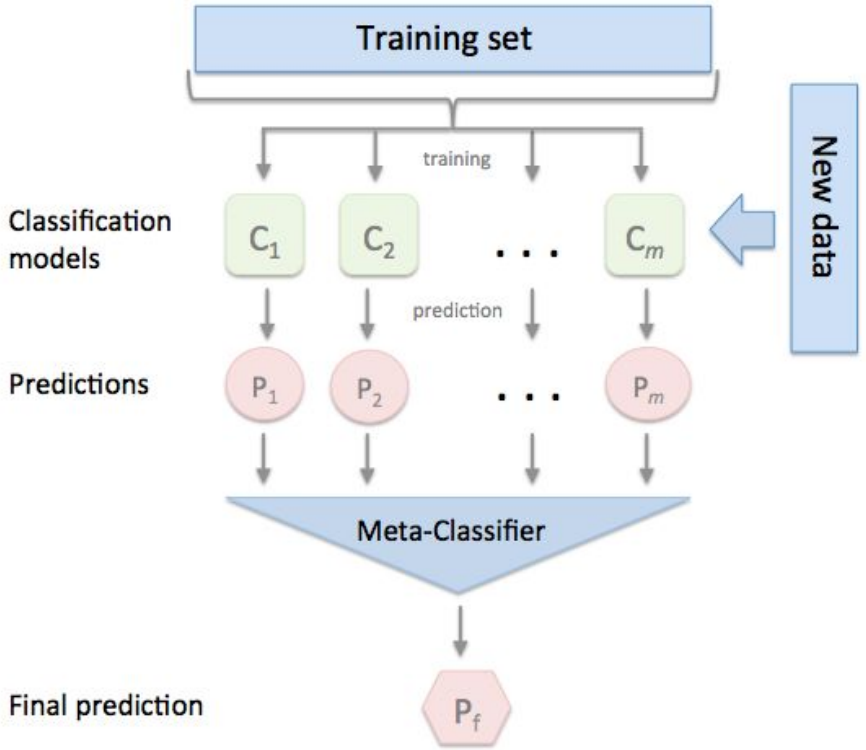
$$\alpha_i = \frac{1}{2} ln\left(\frac{1-erreur_i}{erreur_i}\right)$$

$$Y_M(x) = sign\left(\sum_m^M \alpha_m y_m(x)\right)$$

Donne l'importance du modèle dans le modèle final

Combinaisons de modèles - Stacking

Meta-Classifier : c'est un modèle qui va prendre en entrée les prédictions des sous-modèles (peu importe les types de modèles) et qui va apprendre à les combiner pour en obtenir un résultat optimum

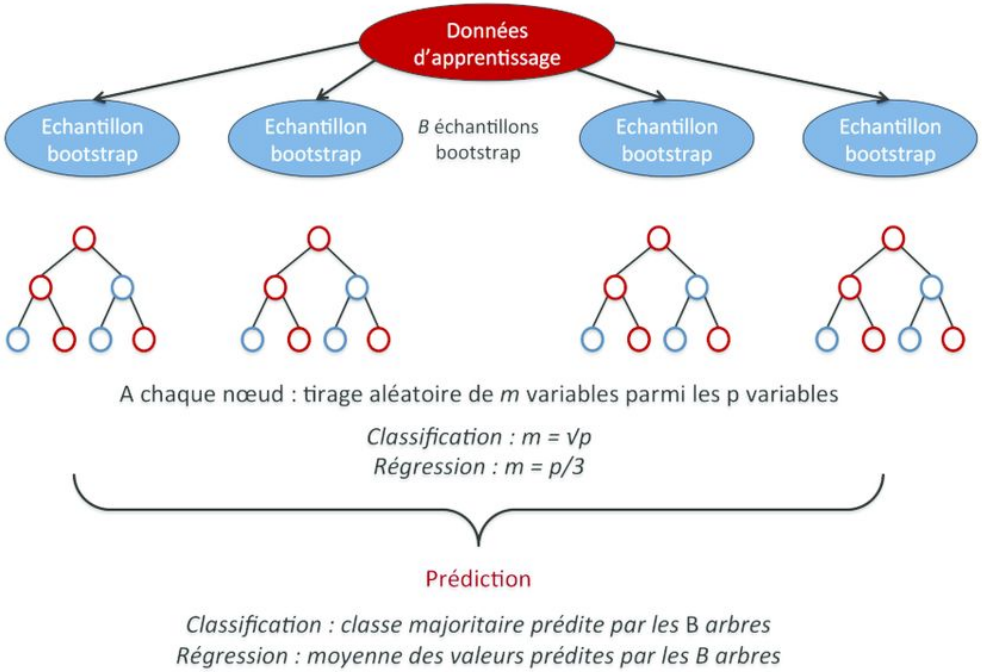


source

Random forest

Random forest = tree bagging + feature sampling

Feature sampling => donner accès à différentes informations selon les arbres (explication jeu du “qui est-ce”)



Gradient Boosting

Gradient Boosting = Descente de Gradient + Boosting

Dans le **boosting** l'importance du vote de chaque classifier est calculé par (selon sa performance) :

$$\alpha_i = \frac{1}{2} ln(\frac{1-erreur_i}{erreur_i})$$

Le **Gradient Boosting** calcule ce poids différemment. On applique alors une descente de gradient afin d'optimiser ce poids pour chaque arbre afin d'obtenir le meilleur "meta-model"

Fonction de coût

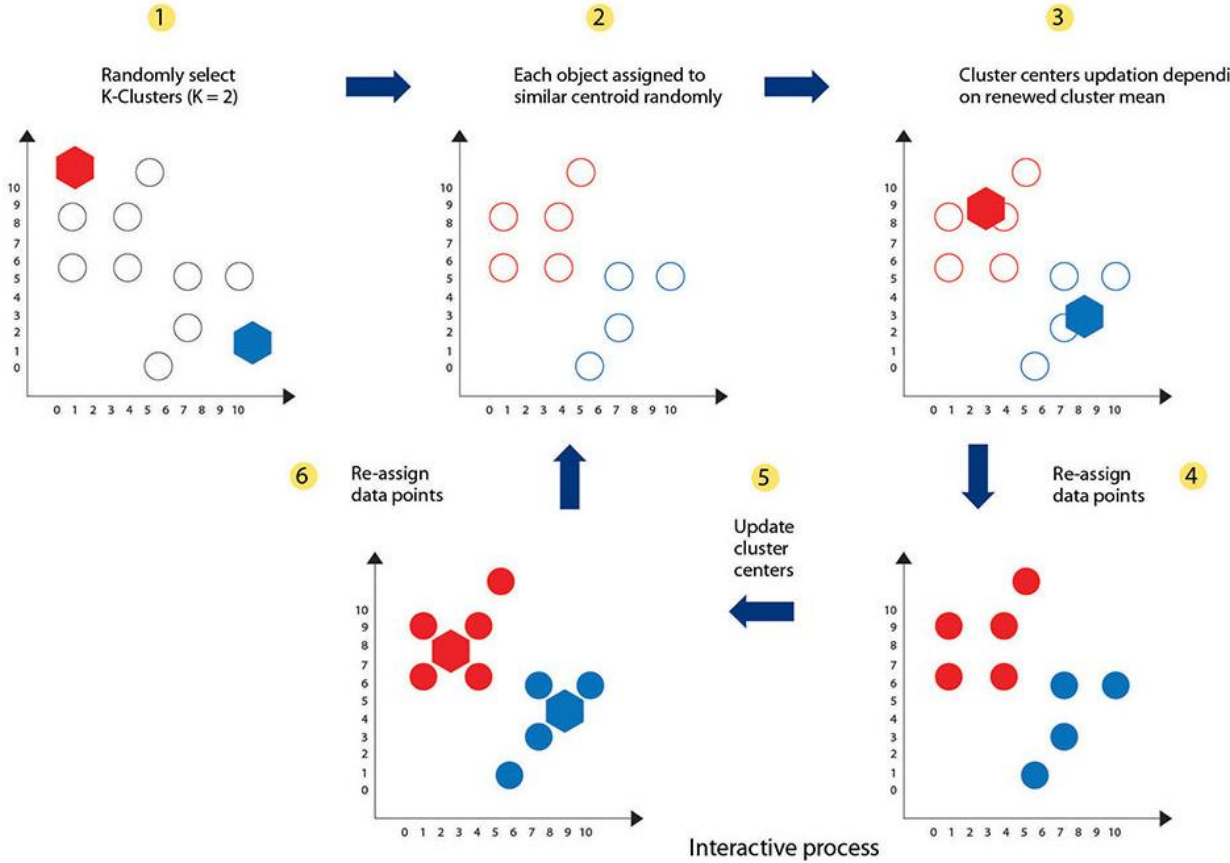
Comparatif

Algorithme	Hyperparamètres	Avantages	Inconvénients
Régression	- Degré si polynomiale - Type de régularisation - Coefficient de régularisation	- Rapide - Explicable	- Peu adaptée aux problèmes complexes et non linéaires
Arbre de décision	- Nombre minimum d'éléments de feuille - Profondeur maximum	- Non-linéaire - Explicable - Rapide	- Enclin à overfitting - Performances moyennes
Naive Bayes	- Loi de la probabilité d'évidence	- Rapide - Nécessite peu d'exemples - Efficace	- Hypothèse d'indépendance forte - Performances moyennes
SVM	- Kernel - Coefficient de régularisation	- Très performant - Applicable à tous les problèmes	- Long et coûteux
KNN	- K	- Très rapide	- Performances moyennes
Random Forest	- Nombre d'arbres + Hyperparamètres des arbres	- Très performant - Applicable à tous les problèmes	- Long et coûteux - Enclin à overfitting
Gradient Boosting	- Fonction de coût + Hyperparamètres des arbres	- Très (Très) performant - Applicable à tous les problèmes	- Très long et coûteux

Apprentissage Non-Supervisé

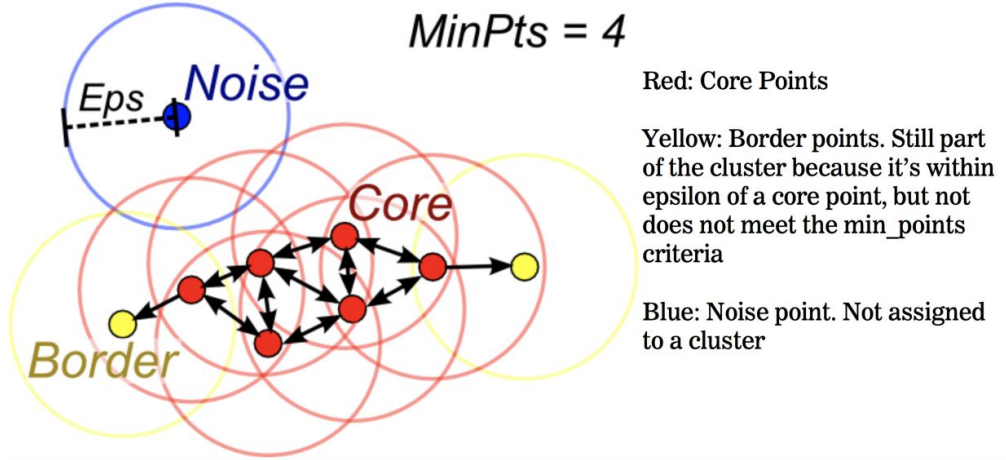
Clustering

K-Means

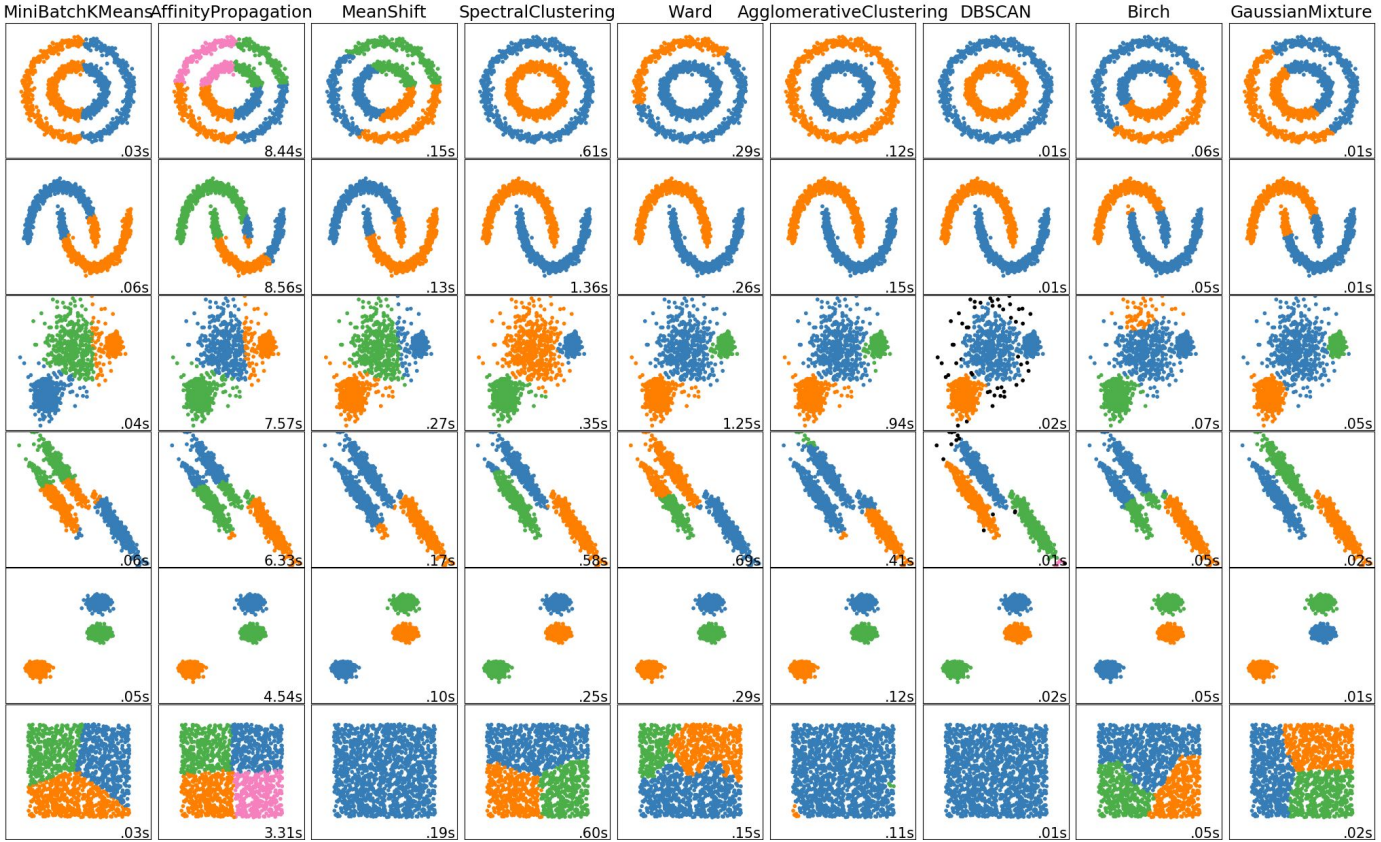

1
Randomly select
K-Clusters (K = 2)
2
Each object assigned to
similar centroid randomly
3
Cluster centers updation depending
on renewed cluster mean
4
Re-assign
data points
5
Update
cluster
centers
6
Re-assign
data points
Interactive process

DBSCAN (Density-Based Spatial Clustering of Applications with noise)

- Détecte automatiquement les clusters
- Se base sur la densité
- Et sur un nombre minimum d'éléments
- Détecte des outliers
- Peut trouver des formes de clusters non-circulaires



Autres


MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
AgglomerativeClustering
DBSCAN
Birch
GaussianMixture
.03s
8.44s
.15s
.61s
.29s
.12s
.01s
.06s
.01s
.06s
8.56s
.13s
1.36s
.26s
.15s
.01s
.05s
.01s
.04s
7.57s
.27s
.35s
1.25s
.94s
.02s
.07s
.05s
.06s
6.33s
.17s
.58s
.69s
.41s
.01s
.05s
.02s
.05s
4.54s
.10s
.25s
.29s
.12s
.02s
.05s
.01s
.03s
3.31s
.19s
.60s
.15s
.11s
.01s
.05s
.02s

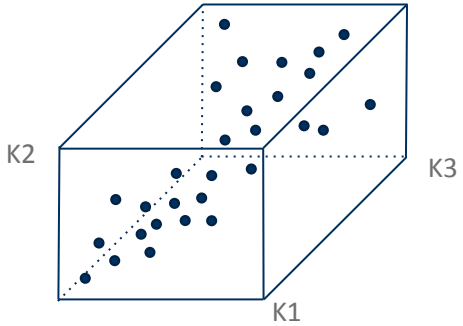
Réduction de dimensions

Motivation

X1	X2	X3	...	Xn
v11	v12	v13	...	v1n
v21	v22	v23	...	v2n
...	...	...	...	...
vm1	vm2	vm3	...	vmn

Pas de visualisation possible

K1	K2	K3
u11	u12	u13
u21	u22	u23
...	...	...
um1	um2	um3



+

Amélioration du temps d'apprentissage

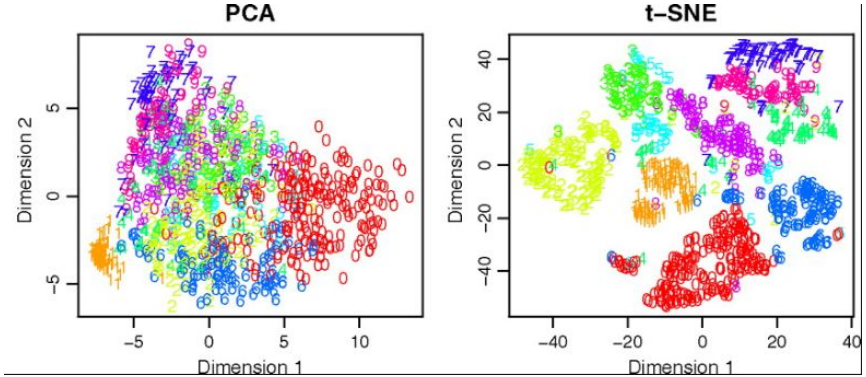
Deux exemples de solutions

PCA
(Principal Component Analysis)

- Procédé d'algèbre linéaire
- Simple et efficace
- Peut être limité

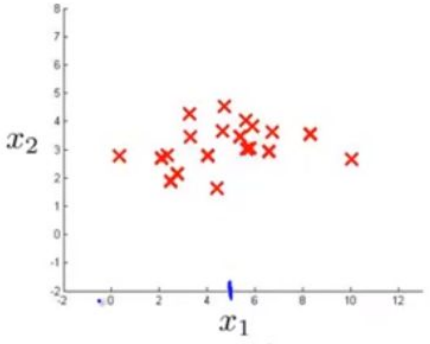
T-SNE
(T-Distributed Stochastic Neighbor Embedding)

- Utilisation d'embeddings
- Adapté à quelques milliers de lignes
- Lourd

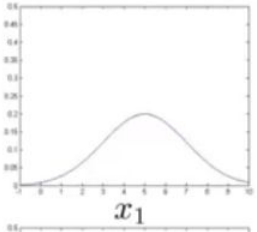


Détection d’anomalie

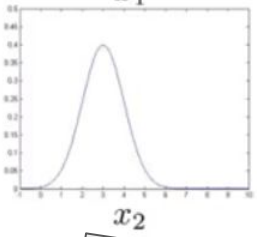
Gaussian Detection



Calcul de la moyenne et de la variance par feature

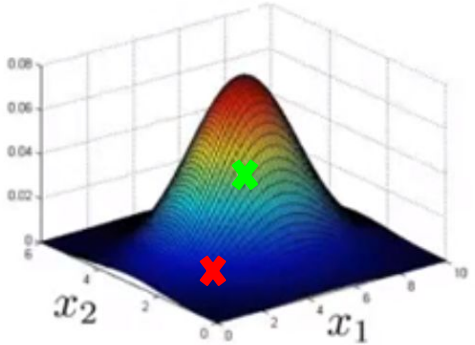


$$p(x_1; \mu_1, \sigma_1^2)$$



$$p(x_2; \mu_2, \sigma_2^2)$$

Produit des probabilités



Principe de l'algorithme:

- Pour chaque feature:
  - On calcule la moyenne et la variance
  - On en déduit une loi normale
- On calcule la probabilité finale en multipliant les lois normales précédemment calculées
- On fixe une valeur de seuil (valeur arbitraire)
- Si la probabilité de notre exemple est inférieure à la valeur de seuil alors on la détecte comme anomalie

Hyperparameters Tuning

Problématique

Exemple :

Algorithme A

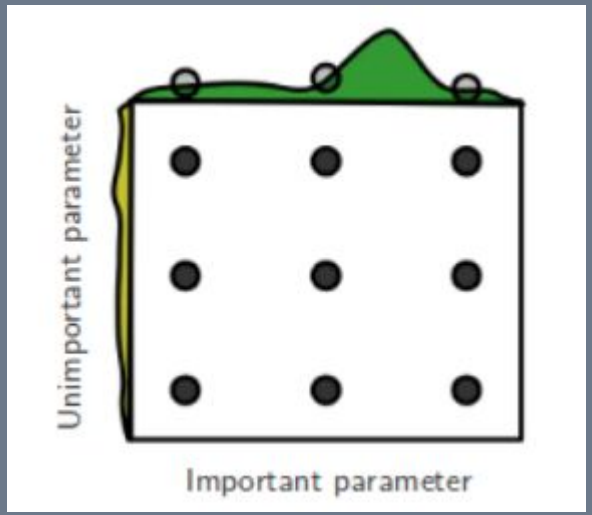
- Hyperparam 1 : 0.0001 -> 1000
- Hyperparam 2 : 2 -> 200
- Hyperparam 3 : 1 -> 4

Objectif : 2 combinaisons

=> Comment choisir les valeurs optimales ?

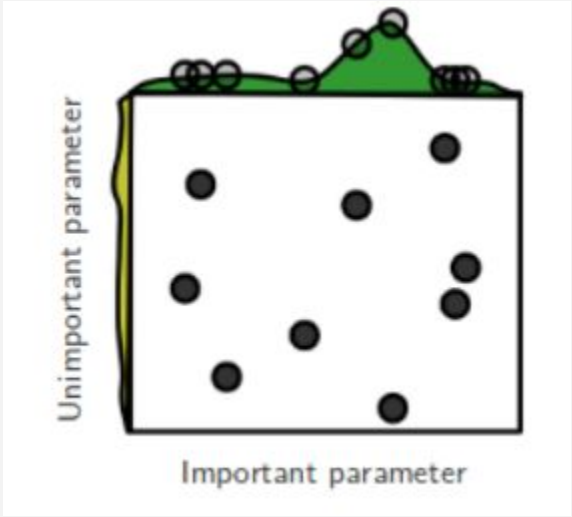
Grid search

- Hyperparam 1 : [0.1, 1, 10]
- Hyperparam 2 : [5, 75, 150]
- Hyperparam 3 : [1, 2, 4]



Random search

- Hyperparam 1 : norm(0.01, 10)
- Hyperparam 2 : randint(5, 200)
- Hyperparam 3 : randint(1, 4)



Tuning

Exemple :

Algorithme A

- Hyperparam 1 : 0.0001 -> 1000
- Hyperparam 2 : 2 -> 200
- Hyperparam 3 : 1 -> 4

Objectif : 2 combinaisons

Grid Search :

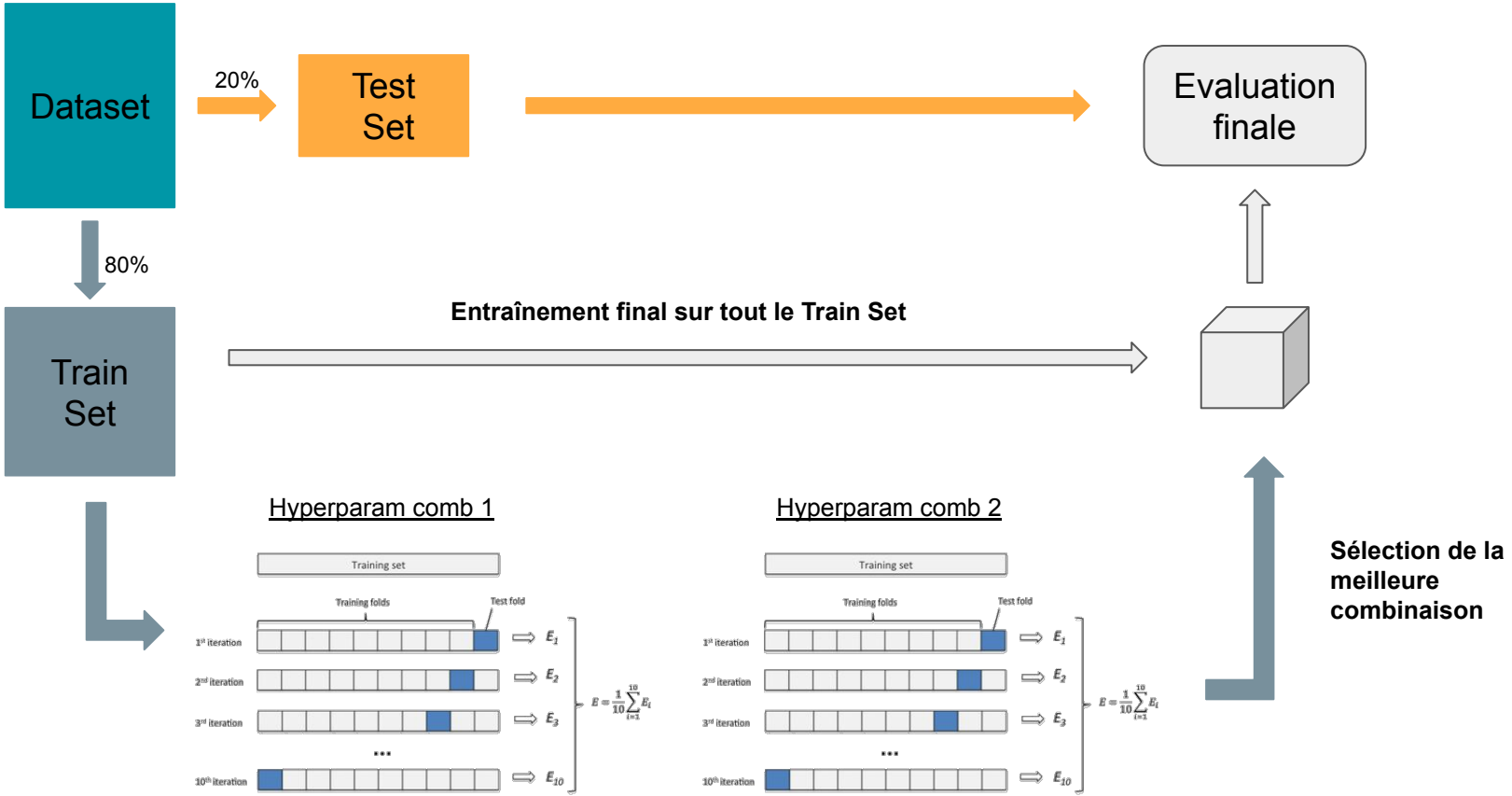
- H1 = 0.1, H2 = 5, H3 = 1
- H1 = 1, H2 = 150, H3 = 4

Random Search :

- H1 = 8.5, H2 = 112, H3 = 3
- H1 = 0.5, H2 = 65, H3 = 2

=> Le random Search permet d'explorer plus et est moins biaisé par les hypothèses faites par la personne mettant en place l'algorithme

Tuning + Cross Validation



Fin du chapitre 4