Nature des données

Variable cible binaire… => classification !!!

Variables descriptives : on a que des variables quanti ??? les dates sont à considérer comment ?? 🡺  « Recodage » en enlevant les tirets et en ayant un int du coup on n’a que des variables quanti maintenant !!!

Un mot pour chaque méthode :

**Arbres** :

Critère de séparation des feuilles :

* Gini ; Entropie ; Variance à minimiser

la profondeur maximal de l’arbre  
le nombre minimum d’exemples dans une feuille  
le nombre minimum d’exemples pour effectuer une séparation  
le gain minimum à chaque séparation

**Gradient tree boosting** :

Le gradient boosting est une méthode ensembliste séquentielle (opposée aux méthodes parallèles) à base d’arbre. Comme son nom l'indique, la méthode ensembliste construit un **modèle additif** qui combine des modèles avec des performances faibles **(apprenants faibles)** afin d’obtenir un modèle prédictif plus efficace, et séquentiel car des arbres (apprenants faibles) seront créés les uns à la suite des autres, et chaque nouvel arbre aura pris compte des erreurs de prédiction de l’arbre précédent. L’erreur de prédiction est calculée entre les données réelles et les données expérimentales prédites par chaque nouvel apprenant faible. Cette erreur se calcule à l’aide d’une **fonction de perte** (ou de coût) que l’on cherche a minimiser. Pour la problématique de classification avec le gradient boosting, la fonction de perte que nous chercherons à optimiser sera la **déviance binomiale**log(1+exp(−2*yf*))

« fitter les résidus » = corriger les erreurs faites par les précédentes itérations

Hyperparamètres = Learning rate, nombre d’apprenant faible, taille des apprenant faibles

**Adaboost** :

Le mauvais résultat reçoit un poids accru dans le gradient de boosting.

**La perte exponentielle**,

**Bagging** :

Foret aléatoire : pour rendre plus stable le modèle et minimiser la variance

On apprend plusieurs arbres différents en utilisant des informations différentes en tirage aléatoire

On combine ensuite tous ces arbres

**Plus on a de diversité, plus on gagne en stabilité :** meilleur en généralisation

On peut en plus tuner les hyperparamètre sur le nb de variables a considérer etc… c’est ca qui fait que c’est efficace

**SVM** :

Essayer avec un svm linaires, puis avec un SVM a noyau si la solution n’est pas linéairement séparable

Un SVM permet de construire un hyperplan (droite affine) qui sépare des données en 2 parties. Cet hyperplan est construit de façon à être la plus éloignées possible des observations

Cette façon de faire permet d’avoir une méthode avec une qualité de prévision est la plus grande possible.

La marge géométrique m d’un échantillon linéairement séparable est donnée par la plus  
petite distance d’un point de l’échantillon à la frontière de décision. On cherche à maximiser cette distance car ça nous permettra de maximiser la confiance et donc minimiser la probabilité d’erreur associée au classifieur.

Hyperparamètre : C et noyau

**KNN** :

On cherche à prédire la classe d’une nouvelle observation en observant ses k plus proches voisins pour lesquels on connait déjà l’étiquette. On donnera comme classe a cette nouvelle observation, la classe la plus représentée (résultat majoritaire des statistiques des classes d’appartenance) des *k* plus proches voisins de cette nouvelle observation.

Méthode non paramétrique donc pas besoins de vérifier des conditions sur nos données pour utiliser la méthode.

Avantage : méthode très simple !!

Choix des k voisins par validation croisée.

ATTENTION !!! si le jeu de données est déséquilibréé au niveau des classes 🡺 mauvaise prediction car prediction de toujours la classe majoritaire !!! Solution =

1. Abstraction dans la représentation des données ( ???)
2. Mettre en place un système de pondération sur les voisins pour mesurer leur influence contributive (par exemple poids=1/*d*, ou *d* est la distance de l'élément, à classer ou à pondérer, de ce voisin).

Les différentes distnaces possibles pour les KNN :

* Distances disponibles pour les données quantitatives (métriques): **Euclidienne**, Minkowski, Manhatan, Tchebychev, Canberra.
* Distances disponibles pour les données quantitatives (noyaux): linéaire, sigmoïde, logarithmique, puissance, Gaussienne, Laplacienne.

**K-means** :

Méthode de partitionnement directe, non supervisé. Construire une partition en k classes.

Clustering mais pour quoi faire ??? sachant que nous on a un jeu de données étiqueté !!!

Regrouper des individus ou objets ayant les caractéristiques les plus proches au sein de K groupes.

* Dans la détection des fraudes, K-means peut aussi contribuer à identifier des actions potentiellement malhonnêtes en fonction de leur proximité avec des groupes de profil renvoyant à un modèle frauduleux.

Algorithme principe :

Ces groupes (clusters) sont constitués par itération successive de leur centre de gravité sur le graphique. La méthode ? On détermine d'abord un nombre de groupes à identifier. On appelle ce nombre K. On positionne ensuite aléatoirement leur centre de gravité (ou centroïde) sur le plan. Puis on associe chaque point aux centres de gravité dont il est le plus proche. On calcule ensuite le véritable centre de gravité de chaque groupe ainsi créé. Puis on le recalcule pour l'ensemble des points du nuage. Et ainsi de suite, jusqu'à ce que les centres de gravité soient parfaitement équilibrés. C'est ce que l'on appelle la convergence de l'algorithme.

**Sampling** :

<https://www.kaggle.com/code/rafjaa/resampling-strategies-for-imbalanced-datasets/notebook>

**RNN** :

<https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/>

**ADL** :

Voir papier ricco pour explications si ça fonctionne… et pour les éventuels hyperparamètres

**ADQ** :

Voir papier ricco pour explications si ça fonctionne… et pour les éventuels hyperparamètres

**LOF** :

Détection de point aberrants ??? pourquoi faire ???? … supprimer les valeurs bizarres du jeu de donnés ou justement trouver les observations qui peuvent potentiellement être des fraudes ?

Méthode non supervisée

**auto-encodeurs**

Méthode non supervisé

**LOGIT** :

**Cost-sensitive learning** :

On pondère les erreurs

Si on souhaite faire en sorte que notre algorithme se focalise sur les exemples de la classe minoritaire, il va falloir lui indiquer que mal classer un tel exemple va entrainer une valeur importante de la loss, et que, au contraire, s’il classe mal un exemple de la classe majoritaire, cela entraînera une conséquence moindre.

Modifier le poids de chaque classe sur le substitue de taux d’erreur

Attribuer un poids à chaque entrée de la matrice de confusion (cout a l’échelle de chaque classe)

CFN = n/m et CFP = p/m

CFN > CFP

Poids global de l’ensemble de + = poids global de l’ensemble des –

Attribuer un poids a chaque exemple de mon jeu de données. Pour notre cas mettre un plus gros poids pour les fraudes avec un gros montant par rapports aux petites fraudes.

**Méthodes à noyau** :

Si notre problème de classification ne peut pas se résoudre avec une méthode linéaire, on utilise des méthodes a noyau. Le principe est d’intégrer à notre modèle, plusieurs fonctions de bases, par ex :

* Noyau polynomial :

Hyperparamètre = M. plus M est grand plus le modèle a de la capacite (M détermine la longueur du vecteur phi. Plus phi est grand, plus le problème est facile à résoudre. Enjeu est de trouver la valeur de M pour laquelle on a l’erreur minimal. Attention si M est trop grand, on ne sera trop loin sur la fonction d’erreur pour être a son minimum)

Hyperparamètre = c.

* Noyau gaussien :

Hyperparamètre = . Plus est petit, plus le modele a de la capacité. Si est trop grand, on va toujours prédire la même chose (sous apprentissage) !!!! permet de mettre de l’importance sur les points proches (en distance entre x et x’) quand on calcule le noyau (et donc quand on va faire la prédiction). Si est très petit, on va utiliser donc très peu de données x en entrée pour faire une prédiction 🡺 ça entraine une fonction de prédiction très **variable/ flexible** ! Attention à ne pas tendre sur du sur apprentissage sur trop petit

Ces fonctions de bases seront passées dans le jeu de données pour ramener notre problème de classification en quelque chose de résolvable avec une fonction linéaire.

En gros, on convertie un algorithme linéaire en un algo non linéaire car notre problème est resolvable de façon non linéaire.

Les fonctions de bases sont définies **implicitement**

Dans ces méthodes on cherche à déterminer K la matrice de gram, qui se compose de tous les noyaux calculés grâce a la comparaison/ combinaison des toutes les observation 2 à 2.

Hyperparamètre lambda pour la régularisation.

**F-Mesure**:

**AUC ROC** :

<https://boostedml.com/2019/07/handling-imbalanced-classification-datasets-in-python-choice-of-classifier-and-cost-sensitive-learning.html>

<https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/>