**L2-MI Mini projet Challenge:** crédit**,Groupe:** audace

**Membres:** OUERD Kenza, ROINEL Aymeric, KUTU Azowa, WANG Ruiwen ,LU Jin

**Challenge URL:**

**Repo GitHub du projet**

**Présentation et mise en contexte:**

Nous avons choisi le projet “Give me some credit” qui consiste à prévoir si un client sera ou non capable de rembourser son crédit.

Pour cela nous aurons accès à 121500 exemples d’entraînement (représentant 121500 clients), possédant chacun 56 caractéristiques, et indiquant si le client en question a pu rembourser son crédit.

Trois groupes travailleront chacun sur une classe différente. Le premier s’occupera du preprocessing (préparation des données), et utilisera principalement le jeu de données d’entraînement. Le deuxième du classifieur (choix des modèles et entraînement de celui-ci), et utilisera quant à lui un jeu de donnée de validation et de test (en plus des données d'entraînement). Le troisième groupe aura comme objectif de faire la visualisation des données et des résultats avant et après la passage par le classifieur.

**Rôle du binôme (Roucairol et Roinel), preprocessing:**

Le rôle de ce binôme est de préparer au mieux le jeu de données utilisé par le classifieur. Chaque client possède 56 caractèristiques, et il est évident que celle-ci ne seront pas toutes utiles pour notre classification.

Pour commencer nos recherches nous utiliserons un algorithme d’élimination des caractéristiques à faible variance *(voir algo plus bas)*. En effet, la variance représentant la dispersion de données, nous allons alors pouvoir avoir une première idée des caractéristiques les moins importantes d’un client. Nous utiliserons ensuite d’autres algorithme ou méthodes de SkLearn pour améliorer au mieux notre sélection. Le regroupement de certaines données à l’influence similaire est aussi envisagé, tout comme l’ajustement du rapport client\_qui\_paye/client\_qui\_ne\_paye\_pas dans le jeu de données d’entraînement (car 90% ou plus des clients payent leur crédit).

**Rôle du binôme (Jin et Ruiwen), classifieur:**

Notre rôle est de classifier les données. Pour cela, nous récupérons les données qui viennent d’être traitées par l'autre binôme de notre groupe.

Nous allons ensuite comparer plusieurs algorithmes possibles (algorithmes sur sklearn ou une combinaison de plusieurs algorithmes) sur les données d’entrainement et par cross-validation pour trouver le plus adapté pour classer nos données. On pourra choisir l’algorithme avec le ou les meilleurs résultats sur les valeurs que l’on va regarder ou bien l’algorithme parmi les meilleurs qui reste le plus constant.

**Rôle du binôme (michel et kenza), visualisation:**

Notre rôle est de présenter les données et nos résultats de manière lisible et compréhensible. Pour cela nous avons décidé d’imager les différentes étapes importantes de notre modèle d’apprentissage. La première étape est avant l’entrainement des données ce qui, permettra de représenter les données et les corrélations qu’il pourrait y avoir entre nos différents paramètres. La seconde étape nous permettra de représenter les lignes de décisions qui séparerons nos classes. Notre problème étant la classification binaire des données nous devons faire en sorte de différencier les classes. Pour cela nous utiliserons surtout la bibliothèque de fonction matplotlyb. Nous avons choisi de représenter de représenter la corrélation des paramètres par une heapmaps et nos données en utilisant la fonction scatter de mathplotlyb.

**Résultats préliminaires**

Pour déterminer un résultat préliminaire, on choisit la modèle d'apprentissage “ random forest” comme exemple (voir l’algorithme 2). En utilisant cette méthode, on voit un nombre d'erreurs assez faible. (0.01 environ)

Mais la performance n'est pas encore idéale et on peut améliorer notre algorithme parce que le score n'est pas encore idéal (0.86 environ). Et de plus, le temps d'exécution est un peu trop long. Donc on doit essayer d'autres algorithmes ou combiner plusieurs algorithmes afin d'éviter d'avoir long temps d'exécution.

On a aussi essayé 4 autres algorithmes et on les a résumés sous forme d'un tableau à la fin du rapport.

**Tableaux sur les points bonus:**

Tableau 1 :Statistiques sur les données

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Dataset** | **Num. examples** | **Num. variables** | **Sparsity** | **Has categorical variable?** | **Has missing data?** | **Num. examples in each class** |
| **train** | 121500 | 56 | 0 | no | no | inconnu |
| **validation** | 15000 | 56 | 0 | no | no | inconnu |
| **test** | 13500 | 56 | 0 | no | no | inconnu |

En utilisant la bibliothèque “data manager”, on peut lire directement des informations sur le dataset. Mais le nombre d’exemples dans chaque classe (granted, non-granted) n’est pas figuré dans l’affichage des informations.

**Tableau 2 :Résultats préliminaires**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Methods** | **Naive Bayes** | **Linear Regression** | **Decision Tree** | **Random Forest** | **Perceptron** |
| **Training** | 0.07 | 0.06 | 0.00 | 0.01 | 0.09 |
| **CV** | 0.07 | 0.06 | 0.10 | 0.07 | 0.10 |
| **Validation** | Non-essayé | 0.62 | non-essayé | non-essayé | non-essayé |

Après l’apprentissage de chaque algorithmes et la validation, on fait afficher le “accuracy\_score” chaque fois et on obtient donc le taux d’erreurs de chaque algo.

En comparant ces 5 différentes méthodes, on voit que le Decision Tree a une meilleure performance pour traiter notre problème (parmi ces 5 méthodes).

Comme il y a une contrainte de nombre de soumissions par jour sur le site codalab, on n’a pas tout essayé les 5 algorithmes sur codalab.

**Algorithme et références :**

Algorithme 1, élimination des caractéristiques à faible variance :

Nécessite :

- t la taille de l’échantillon à modifier

- v le seuil de variance pour laquelle cette caractéristique sera supprimée (0 par défaut)

- C le tableau des valeurs de la caractéristique à tester (de taille t)

fonction seuil\_var\_elim(C, v) :

t = taille de C

somme = 0

somme\_pour\_variance = 0

pour i allant de 0 à t-1 avec un pas de 1 :

somme = somme + C[i]

fin pour

moyenne = somme/t

pour i allant de 0 à t-1 avec un pas de 1 :

somme\_pour\_variance = somme\_pour\_variance + (C[i] + moyenne)^2

fin pour

variance = somme\_pour\_variance/t

si variance < v :

supprimer C des données

fin si

Fin fonction

Algorithm 2 Random Forest

Précondition: un training set S := (x1, y1), . . . ,(xn, yn), features F, et nombre d’arbres dans forest B.

function RandomForest(S , F)

H ← ∅

for i ∈ 1, . . . , B do

S(i) ← A bootstrap sample from S // un exemple de S

hi ← RandomizedTreeLearn(S(i), F)

H ← H ∪ {hi}

end for

return H

end function

function RandomizedTreeLearn(S , F)

At each node:

f ← very small subset of F // f un sous-dataset très petit

Split on best feature in f // séparer le meilleur “feature” dans f

return The learned tree

15 end function