Classification de données clients pour des prêts bancaires

Python & Machine Learning

Anis BOUCHEKHIMA

2019 / 2020

DONNÉES

Fichiers d'entrée / de sortie

Les données utilisées dans ce projet sont répartis en trois fichiers d'entrée:

- **`bank_train_data.csv`**
- **`bank_train_labels.csv`**
- **`bank_test_data.csv`**

Il y'a un seul à générer:

- **`bank_test_labels.csv`**, contenant les prédictions obtenues en sortie du modèle.

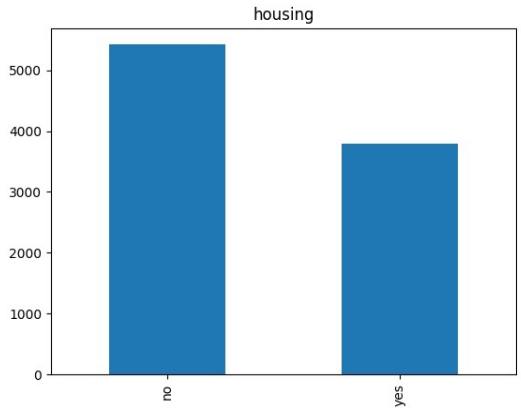
Caractéristiques

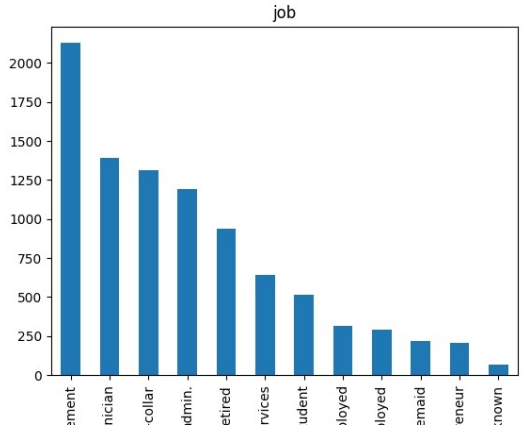
En total, les fichiers comporte 16 caractéristiques potentielles, dont 6 attributs continues et 10 attributs discrètes.

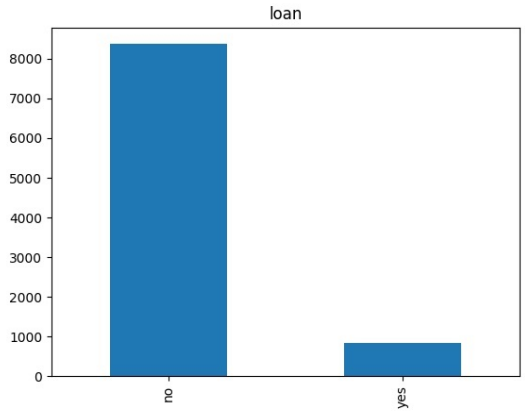
Attributs continues :
age, balance, duration, campaign, pdays, previous.

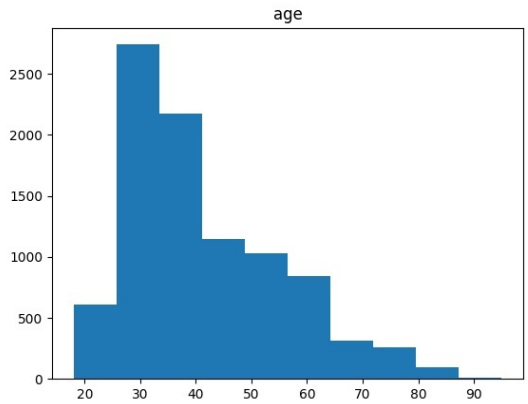
Attributs discrets :
job, marital, education, contact, day, month, poutcome, default, housing, loan.

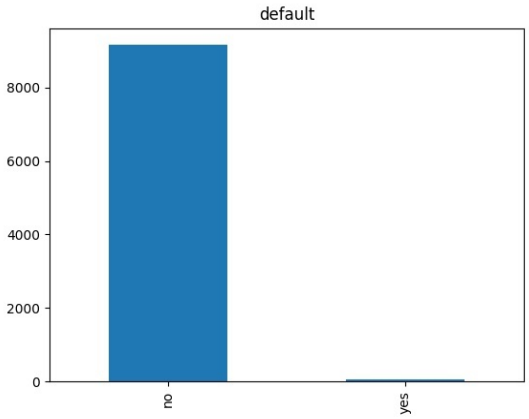
Les figures qui suivent représentés des statistiques sur quelques attributs (colonnes) des données fournies.
(générées par le fichier `analysis.py`)











Division des données

Afin de diviser les données aléatoirement, on mélange les données et on prend les échantillons de manière à ce que 50% des données ont la classe 1,
et les autres ont la classe 0.

MÉTHODES UTILISÉES

K Plus Proches Voisins

Cette méthode nous permet de prédire la classe qu'on cherche (`has_subscribed`) d'un client donné en trouvant la classe la plus présente parmi les k plus proches voisins.

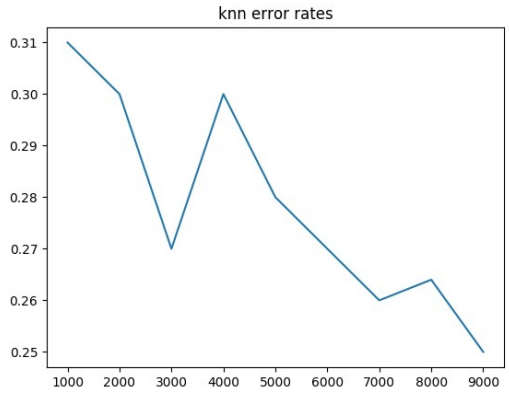
Dans un premier temps, on prends 1000 échantillons pour l'entraînement (des voisins dans ce cas) et 100 échantillons pour le test.

Résulats :

Taux d'erreur	Faux positifs	Faux négatifs
34%	14	20

Ensuite, on fait varier la valeur de k de 2 à 20. En changeant la valeur de k, on remarque que le changement de taux d'erreur est négligeable. Par la suite, on prend la valeur 10 pour k.

Maintenant, on fait varier la taille des échantillons de 1000 à 9000 et on obtient la figure qui suit.



Cette fois, on remarque que plus on augmente la taille des échantillons pris pour l'entraînement, plus le taux diminue.

Pour obtenir les prédictions finales à soumettre, on prend 9200 échantillons pour l'entraînement et on prend la valeur 10 pour k.

Le score obtenu est de 0.827
donc le taux d'erreur est égale à 0.173

Régression Linéaire

La classe d'un échantillon dans cette méthode, se prédit en utilisant la descente du gradient à travers plusieurs epoch d'entraînement.

Contrairement au valeurs des datasets vue en TP4 de ce module, les valeurs du dataset de ce mini projet peuvent être suffisamment large pour avoir un débordement de mémoire lors de calcul des exponentiels dans la fonction softmax.

```
RuntimeWarning: overflow encountered in exp
```

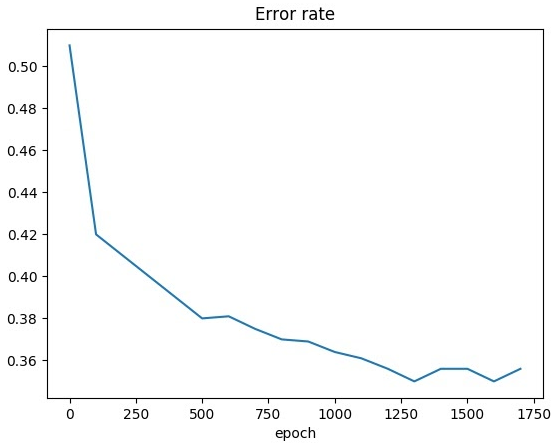
Pour résoudre ce problème, on normalise les valeurs en soustrayant la valeur maximale de chaque ligne.

Un autre problème rencontré lors du développement de ce modèle d'apprentissage est les valeurs nulles dans la matrice **`f`** obtenue de la fonction **`output`**.

`RuntimeWarning: divide by zero encountered in log`

Pour cela, on utilise la fonction `np.clip` pour avoir des valeurs très petites 10^{-7} au lieu des valeurs nulles.

On prend par la suite 5000 échantillons pour l'entraînement et on obtient les taux erreur des données de test montrés dans la figure qui suit.



On remarque que le taux d'erreur ne change pas pratiquement si on fait varier le nombre d'échantillons pris pour l'entraînement.

Pour obtenir les prédictions finales à soumettre, on prend 9200 échantillons pour l'entraînement.

Le score obtenu est de 0.9232
donc le taux d'erreur est égale à 0.0768

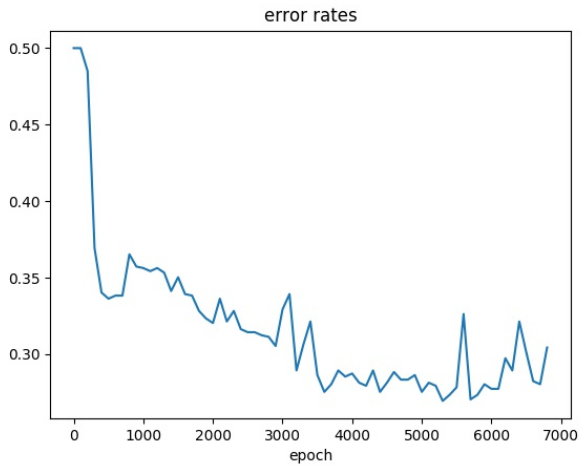
On teste encore une fois en prenant que 5000 échantillons:

Le score obtenu est de 0.9092
donc le taux d'erreur est égale à 0.098
ce qui est incohérent avec le taux d'erreur calculé qui était de 0.35.

Réseau de Neurons

Pour le modèle des réseaux de neurons, la bibliothèque PyTorch a été utilisée pour implémenter un modèle avec une seule couche cachée (Single-Layer Artificial Neural Network).

Comme on fait pour le modèle de régression linéaire On prend par la suite 5000 échantillons pour l'entraînement et on obtient les taux erreur des données de test montrés dans la figure qui suit.



Le score obtenu est de 0.8950
donc le taux d'erreur est égale à 0.1050

Et pour ce modèle aussi, on remarque des incohérence entre le taux d'erreur calculé (0.30) et le score obtenue sur la plateforme Codalab.

Amélioration des modèles

Pour améliorer ces modèles, on peut prévoir d'autres mesures pour filtrer les données d'entraînement, par exemple éliminer les données aberrantes ou bien éliminer les attributs (colonnes) susceptible à ne pas être importants.

Pour le modèle de réseau de neurons , on pourra ajouter plusieurs couches cachées.