Projet Decision Tree et Random Forest

Dans ce projet, nous allons utiliser Python pour résoudre un problème de classification binaire en utilisant à la fois un arbre de décision et une forêt aléatoire. Nous comparerons ensuite leurs résultats et verrons lequel des deux convient le mieux à notre problème. Il s'agit d'un problème de classification binaire où nous devons déterminer si une personne doit recevoir un prêt ou non en fonction d'un certain ensemble de caractéristiques.

Étape 1 : Chargement des bibliothèques et de l'ensemble de données

Commençons par importer les bibliothèques Python et la dataset:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import f1_score

from sklearn.model_selection import train_test_split

# Importing dataset

df=pd.read_csv('dataset.csv')

df.head()
```

L'ensemble de données comprend 614 lignes et 13 caractéristiques, dont l'historique de crédit, l'état civil, le montant du prêt et le sexe. Ici, la variable cible est Loan_Status, qui indique si une personne doit recevoir un prêt ou non.

Étape 2 : Prétraitement des données

Dans cette section, nous allons traiter les variables catégorielles des données et imputer les valeurs manquantes.

```
# Prétraitement des données et imputation des valeurs nulles

# Codage des étiquettes

df['Gender']=df['Gender'].map({'Male':1, 'Female':0})

df['Married']=df['Married'].map({'Yes':1, 'No':0})

df['Education']=df['Education'].map({'Graduate':1, 'Not Graduate':0})

df['Dependents'].replace('3+',3,inplace=True)
```

```
df['Self_Employed']=df['Self_Employed'].map({'Yes':1, 'No':0})

df['Property_Area']=df['Property_Area'].map({'Semiurban':1, 'Urban':2, 'Rural':3})

df['Loan_Status']=df['Loan_Status'].map({'Y':1, 'N':0})

#Imputation par valeur nulle

rev_null=['Gender', 'Married', 'Dependents', 'Self_Employed', 'Credit_History', 'LoanAmount', 'Loan_Amount_Term']

df[rev_null]=df[rev_null].replace({np.nan:df['Gender'].mode(),

                                   np.nan:df['Marié'].mode(),

                                   np.nan:df['Dependents'].mode(),

                                   np.nan:df['Self_Employed'].mode(),

                                   np.nan:df['Credit_History'].mode(),

                                   np.nan:df['LoanAmount'].mean(),

                                   np.nan:df['Loan_Amount_Term'].mean()})
```

Étape 3 : Création des ensembles d'entraînement et de test

Maintenant, divisons l'ensemble de données dans un rapport 80:20 pour les ensembles d'entraînement et de test respectivement :

```
X=df.drop(columns=['Loan_ID','Loan_Status']).values

Y=df['Loan_Status'].values

X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=42)
```

Affichage de la forme des ensembles créés (Données d'entrainement et Données de test)*:*

```
print('Shape of X_train=>',X_train.shape)

print('Shape of X_test=>',X_test.shape)

print('Shape of Y_train=>',Y_train.shape)

print('Shape of Y_test=>',Y_test.shape)
```

Génial ! Maintenant nous sommes prêts pour l'étape suivante où nous allons construire l'arbre de décision et le mode forêt aléatoire.

Étape 4 : Construction et évaluation du modèle

Puisque nous disposons des ensembles d'entrainement et de test, il est temps d'entraîner nos modèles et de classer les demandes de prêt. Tout d'abord, nous allons former un arbre de décision sur cet ensemble de données.

```
# Building Decision Tree

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(criterion = 'entropy', random_state = 42)

dt.fit(X_train, Y_train)

dt_pred_train = dt.predict(X_train)
```

Ensuite, nous allons évaluer ce modèle en utilisant le score F1. Le score F1 est la moyenne harmonique de la précision et du rappel donnée par la formule.

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

Évaluons la performance de notre modèle en utilisant le score F1 :

```
# Evaluating on Test set

dt_pred_test = dt.predict(X_test)

print('Testing Set Evaluation F1-Score=>',f1_score(Y_test,dt_pred_test))
```

Construction d'un modèle de forêt aléatoire

```
#Building  Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(criterion = 'entropy', random_state = 42)

rfc.fit(X_train, Y_train)

# Evaluating on Training set
```

rfc_pred_train = rfc.predict(X_train)

print('Training Set Evaluation F1-Score=>',f1_score(Y_train,rfc_pred_train))

 # Evaluating on Test set

rfc_pred_test = rfc.predict(X_test)

print('Testing Set Evaluation F1-Score=>',f1_score(Y_test,rfc_pred_test))

Ici, nous pouvons clairement voir que le modèle de forêt aléatoire est beaucoup plus performant que l'arbre de décision dans l'évaluation hors échantillon.

Alors, lequel choisir : l'arbre de décision ou la forêt aléatoire ?

La pertinence est améliorée en utilisant l'algorithme Random Forest, donc, on va utiliser Random Forest pour faire la prédiction.

Random Forest convient aux situations où nous disposons d'un grand ensemble de données

Les arbres de décision sont beaucoup plus faciles à interpréter et à comprendre. Comme une forêt aléatoire combine plusieurs arbres décisionnels, elle devient plus difficile à interpréter. La bonne nouvelle, c'est qu'il n'est pas impossible d'interpréter une forêt aléatoire. Voici un article qui traite de l'interprétation des résultats d'un modèle de forêt aléatoire.

https://www.analyticsvidhya.com/blog/2019/08/decoding-black-box-step-by-step-guide-interpretable-machine-learning-models-python/?utm_source=blog&utm_medium=decision-tree-vs-random-forest-algorithm

En outre, la forêt aléatoire a un temps d'apprentissage plus élevé qu'un arbre de décision simple. Vous devez prendre cela en considération car lorsque nous augmentons le nombre d'arbres dans une forêt aléatoire, le temps nécessaire pour former chacun d'entre eux augmente également. Cela peut souvent être crucial lorsque vous travaillez dans des délais serrés dans le cadre d'un projet d'apprentissage automatique.

Malgré l'instabilité et la dépendance à un ensemble particulier de caractéristiques, les arbres de décision sont vraiment utiles car ils sont plus faciles à interpréter et plus rapides à former. Toute personne ayant très peu de connaissances en science des données peut également utiliser les arbres de décision pour prendre des décisions rapides basées sur les données.