M1Info

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Université Paris 8 - Vincennes à Saint-Denis

M1 MIASHS : Big Data et fouille de données

Projet : Random Forest

PANCHALINGAMOORTHY Gajenthran, LABAT Kaxandra

Organisme d'accueil :	Université Paris 8
Cours :	Décision et parcours espace de données

Chapitre 1

Introduction

1.1 Exercice 1

Quelles les prétraitements obligatoires avant d'appliquer l'algorithme Random Forest de Python (Package SkLearn) ?

Avant de commencer à appliquer l'algorithme Random Forest, il faut tout d'abord commencer par importer le package `sklearn` et réaliser la commande suivante afin d'importer le module Random Forest : `from sklearn.ensemble import RandomForestClassifier`. De plus, pour appliquer notre algorithme Random Forest, il faudra éventuellement un ensemble de données pour pouvoir l'exploiter. C'est pourquoi nous allons nous charger d'importer le contenu du fichier *CarteBanquaire.csv*. La méthode `read_csv` du module `pandas`, fera largement l'affaire pour lire ce fichier.
Dans le cas où nous souhaiterons vérifier notre classifieur, nous séparerons notre base de données en 2 parties : une partie pour l'apprentissage et l'autre pour la phase d'entraînement, à l'aide de `train_test_split` provenant de `sklearn.model_selection`. À noter qu'il est important de bien distinguer la colonne *Class* lors de la séparation des données.

Liste des modules utilisés :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

1.2 Exercice 2

Trouver le meilleur paramétrage afin d'avoir le taux d'erreur minimum sur le classifieur

Un des prétraitements intéressants avant l'utilisation finale du Random Forest, semble être le `GridSearchCV`. Cet élément là, va nous permettre de pouvoir utiliser les paramètres les plus efficaces pour notre modèle afin d'obtenir les meilleurs résultats. Tout d'abord, avant d'utiliser le `GridSearchCV`, le choix des paramètres pour les tests sont à fixer. Le nombre de paramètres à tester influencera sur le temps d'exécution. Les paramètres qui semblent les plus judicieux pour le `GridSearchCV` paraissent être `n_estimator`, `max_depth`, `min_samples_split` et `min_samples_leaf`. Nous pourrions en rajouter d'autres, mais cela prendrait trop de temps.

- `n_estimators` représente le nombre d'arbres de décision dans le modèle. Il a donc un impact important sur la qualité de l'apprentissage des données ; en effet, plus la forêt a d'arbres, plus la qualité de l'apprentissage sera importante. Mais cela a aussi un effet sur le temps d'exécution de notre programme qui est un élément à prendre en compte : il est donc essentiel de trouver la bonne valeur pour trouver le juste milieu entre le coût et la performance.

- `max_depth` représente la profondeur de chaque arbre dans la forêt. Plus l'arbre est profond, plus on aura d'informations à propos des données.

- `min_samples_split` représente le nombre minimal d'échantillons nécessaire pour diviser un noeud interne. Plus cette valeur augmente, plus chaque arbre dans la forêt considérera plus d'échantillons à chaque noeud. Nous avons le choix entre mettre un flottant ou un entier pour cette valeur : si il s'agit d'un entier, cela correspondra au nombre minimum d'échantillons ; si il s'agit d'un flottant, cela correspondra au pourcentage d'échantillons.

- `min_samples_leaf` représente le nombre minimal d'échantillons nécessaire pour être au niveau d'un noeud feuille.

```
param_grid = {
    "n_estimators": [10, 50, 100],
    "max_depth": [5, 8, 15],
    "min_samples_split": [2, 5, 10, 30],
    "min_samples_leaf": [1, 2, 5, 10]
}
```

Pour traduire ce dictionnaire `param_grid`, les clés correspondent aux paramètres de Random Forest, et les valeurs sont des tableaux dans lesquels chaque valeur est une valeur d'un paramètre. Par exemple, nous allons tester, RandomForestClassifier avec le paramètre `n_estimators` qui prendra les valeurs 10, 50, 100. Outre la spécification de l'estimator, en l'occurence ici notre Random Forest, et les paramètres dont nous voulons tester, il existe d'autres paramètres du `GridSearchCV` que nous pouvons régler.
Vous trouverez un aperçu de l'ensemble des résultats du `GridSearchCV` sur le fichier *gridsearchcv.txt*.

```
def tuning_rfc_param(param_grid, rfc, X, y):
    CV_rfc = GridSearchCV(
        estimator=rfc,
        param_grid=param_grid,
        verbose=1,
        n_jobs=-1,
        cv=2)

    CV_rfc.fit(X, y)
```

Liste des modules utilisés :

```
from sklearn.model_selection import GridSearchCV
```

1.3 Exercice 3

Quel est le taux d'erreur de votre modèle random forest ?

Avec le `GridSearchCV`, nous remarquons que le taux d'erreur de notre meilleur classifieur est de 0.999567 avec les paramètres suivants : `{'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}`.

1.4 Exercice 4

Quel est le meilleur classifieur (ou estimator) de votre random forest ?

Pour récupérer les estimators de notre Random Forest, il faut d'abord connaître la valeur que nous avons attribuée à `n_estimator` ; en effet, comme nous l'avons vu précédemment `n_estimator` nous indique le nombre d'arbres utilisés pour le modèle, et plus précisément il nous indique le nombre de `DecisionTreeClassifier` que l'on va utiliser. On peut accéder à chaque arbre avec l'attribut `estimators`

de `RandomForestClassifier`, une fois la classification réalisée ; cet attribut nous renvoie une liste de `DecisionTreeClassifier` dont le nombre d'éléments est équivalent à `n_estimator`. Pour savoir lequel des estimators est le plus performant, nous allons tester chacun des estimators avec notre test de données et utiliser une méthode permettant de calculer les taux d'erreur comme `MSE` (Mean Squared Error), `MAE` (Mean Absolute Error), `RMSE` (Root Mean Squared Error)...

```
def calculate_estimators_errors(rfc, X, y, debug=True):
    estimator_errors = []
    for tree in rfc.estimators_:
        pred = 1.0 - tree.score(X, y)
        estimator_errors.append(pred)

    error_min = estimator_errors.index(min(estimator_errors))
    error_max = estimator_errors.index(max(estimator_errors))
```

On obtient ainsi l'estimator le plus performant à l'indice 35 avec un taux d'erreur de 0.0006056722523107713.

Liste des modules utilisés :

```
from sklearn.metrics import mean_squared_error
```

1.5 Exercice 5

Affichez le taux d'erreur de tous les estimator de votre modèle Random Forest

Voici la liste du taux d'erreur des estimators du Random Forest :

```
[0.0007548958507061788, 0.0007812294268936036, 0.0009655644602055774,
 0.000974342318934719, 0.0007724515681644619, 0.0007548958507061788,
 0.0007197844157896123, 0.0008338965792684533, 0.0008690080141850197,
 0.0007636737094353204, 0.0008075630030810285, 0.0007636737094353204,
 0.0007987851443518868, 0.0007110065570604707, 0.0007636737094353204,
 0.0008602301554558781, 0.0009831201776638607, 0.0007548958507061788,
 0.0006232279697690545, 0.0007724515681644619, 0.000702228698331329,
 0.0008338965792684533, 0.0007110065570604707, 0.0009480087427472942,
 0.0008777858729141613, 0.0009128973078307278, 0.00081634086181017,
 0.0009480087427472942, 0.0006320058284981962, 0.0006934508396021874,
 0.00081634086181017, 0.0009918980363930023, 0.0008777858729141613,
```

```
0.0008075630030810285, 0.0008690080141850197, 0.0006056722523107713,
0.0007636737094353204, 0.0009128973078307278, 0.0007285622745187538,
0.0008514522967267364, 0.00081634086181017, 0.0007461179919770371,
0.0007373401332478955, 0.0008338965792684533, 0.0007548958507061788,
0.0007724515681644619, 0.00081634086181017, 0.00081634086181017,
0.0007461179919770371, 0.0008690080141850197]
```

1.6 Exercice 6

Prenez deux estimators de votre modèle Random Forest, l'estimator le plus performant et l'estimator le moins performant. Quel est le taux d'erreur des deux estimatiors

Il s'agit du même procédé que pour chercher l'estimator le plus performant

```
def calculate_estimators_errors(rfc, X, y, debug=True):
    estimator_errors = []
    for tree in rfc.estimators_:
        pred = 1.0 - tree.score(X, y)
        estimator_errors.append(pred)

    error_min = estimator_errors.index(min(estimator_errors))
    error_max = estimator_errors.index(max(estimator_errors))
```

On obtient donc un taux d'erreur de 0.0006056722523107713 pour l'estimator le plus performant contre 0.0009918980363930023 pour l'estimator le moins performant

1.7 Exercice 7

Quelle est la différence entre la moyenne des taux d'erreur des deux estimators et le taux d'erreur de votre meilleur modèle random Forest ?

Soit `M_EE`, la moyenne du taux d'erreur des deux estimators, `ER`, le taux d'erreur du Random Forest et `DIFF_E`, la différence entre `ER` et `M_EE`
`M_EE =`

$$\frac{0.0006056722523107713 + 0.0009918980363930023}{2} \tag{1.1}$$

```
= 0.0007987851443518868
ER = 0.00036867006662394777
```

```
DIFF_E = 0.0007987851443518868 - 0.00036867006662394777 = 0.000430115077727939
```

On remarque grâce à cette valeur, on obtient souvent la même valeur (en répétant cette action plusieurs, nous pouvons plus ou moins arriver à cette conclusion, avec une valeur avoisinant les 0.0004). Ainsi la différence entre la moyenne des taux d'errreur des deux estimators et le taux d'erreur de notre modèle peut se percevoir sur le fait que le taux d'erreur du modèle correspond plus à une moyenne de l'ensemble des arbres plutôt qu'un de ces deux estimators.

1.8 Exercice 8

Affichez les deux arbres correspondants au deux estimators

Pour pouvoir visualiser un arbre, il faudra utiliser la méthode `export_graphviz` de `sklearn.tree`. Cette dernière peut prendre des arguments tels que l'estimator qu'on souhaite visualiser, le fichier de sortie pour stocker l'arbre, les fonctionnalités (dans notre cas V1, V2, V3... V20, Time et Amount), et la classe qu'on obtient.

```
def vizualize_estimators(estimators, out_files, feature_names, class_names):
    for i in range(0, len(estimators)):
        export_graphviz(
            estimators[i],
            out_file=out_files[i],
            feature_names=feature_names,
            class_names=class_names
        )
```

Cependant, il faut prendre en compte que cette fonction nous donne un fichier *.dot* que l'on peut convertir en *.png* pour le visualiser.

Liste des modules utilisés :

```
from sklearn.tree import export_graphviz
```

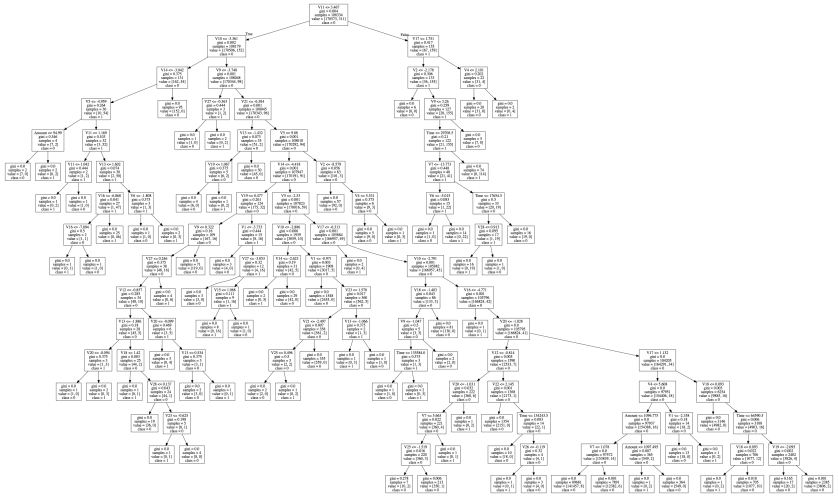

FIGURE 1.1 – Arbre pour l'estimator pour le plus performant

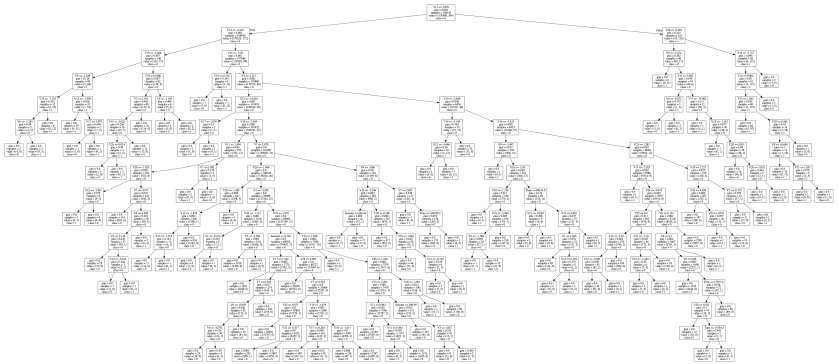

FIGURE 1.2 – Arbre pour l'estimator pour le moins performant

1.9 Exercice 9

Ecrire la liste des règles de décision extraites des deux estimator sous forme de Si A , B => C (x %) ?

Vous pouvez trouver les règles de decision des deux arbres dans les fichiers suivants : *worst_tree_rules.txt* et *best_tree_rules.txt*.

1.10 Exercice 10

Quelles les règles dont un besoin le mauvais estimator pour qu'il s'approche des performances du bon estimator ?

Les règles permettant d'améliorer un mauvais estimator sont principalement le choix des premiers attributs pour la racine de l'arbre ; cela va aider à ce que la structure de l'arbre soit la plus optimale posible. De plus le pruning de certaines branches peut permettent d'obtenir un estimator plus efficace.