

Méthodes d'ensemble

TP 1 – Random Forest

Bienvenue dans le TP sur la mise en œuvre d'une forêt aléatoire à l'aide de l'ensemble de données Titanic ! Dans cet atelier, vous apprendrez à créer et à évaluer un modèle de forêt aléatoire à l'aide du langage de programmation Python.

Une forêt aléatoire est un type de méthode d'apprentissage d'ensemble qui combine plusieurs arbres de décision pour faire des prédictions. C'est un choix populaire pour une variété de tâches, y compris la classification et la régression. Dans ce laboratoire, nous utiliserons l'ensemble de données Titanic, qui contient des informations sur les passagers du Titanic et s'ils ont survécu ou non. Notre objectif sera de construire un modèle de forêt aléatoire capable de prédire la survie d'un passager compte tenu de certains attributs tels que son âge, son sexe et sa classe.

Avant de commencer, assurez-vous que les bibliothèques nécessaires sont installées et que vous avez téléchargé le jeu de données Titanic.

Bonne chance et amusez-vous bien!

Exercice 1 : Random Forest

1. Importer les librairies

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
```

2. Importer les données du Titanic, et filtrer les observations avec NA

```
df = pd.read_csv('./data/titanic.csv')

string_list = [each_string.lower() for each_string in df.columns]
df.columns = string_list
df.dropna(inplace=True)

df.head(5)
```

3. Prenez connaissance des quelques features dans le dataframe,

Quelle autre visualisation pourriez-vous proposer ?

```
print('----Classes----')
print('Classes: ', df['pclass'].unique())
print('Passagers by class:\n', df['pclass'].value_counts().values)
print('----Genre----')
print('Genre: ', df['sex'].unique())
print('Passagers by sex:\n', df['sex'].value_counts().values)

df['age'].hist(bins=50);
```

4. Preprocessing des données : binarisations des quelques features catégoriels
Quelle autre méthode de transformation des features catégoriels connaissez-vous ?

```

#binarize columns
X = df[['pclass', 'sex', 'age', 'embarked']].copy()

#using sklearn to binarize
from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()

#transforming sex
X['sex'] = lb.fit_transform(X['sex'])
X['embarked'] = lb.fit_transform(X['embarked'])

X.head(5)

```

5. Importer des librairies Sklearn pour la modélisation, et faire le split train et test
La base de test est juste 10% des données, quel est le meilleur split pour ce problème ?

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

#split train-test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)

```

6. Créer une fonction pour imprimer les performances des modèles à construire

```

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    """
    print the accuracy score, classification report and confusion matrix of classifier
    """
    if train:
        """
        training performance
        """
        print("Train Result:\n")
        print("accuracy score: {0:.4f}\n".format(accuracy_score(y_train, clf.predict(X_train))))
        print("Classification Report: \n {}\n".format(classification_report(y_train, clf.predict(X_train))))
        print("Confusion Matrix: \n {}\n".format(confusion_matrix(y_train, clf.predict(X_train))))

        res = cross_val_score(clf, X_train, y_train, cv=10, scoring='accuracy')
        print("Average Accuracy: \t {0:.4f}".format(np.mean(res)))
        print("Accuracy SD: \t\t {0:.4f}".format(np.std(res)))

    elif train==False:
        """
        test performance
        """
        print("Test Result:\n")
        print("accuracy score: {0:.4f}\n".format(accuracy_score(y_test, clf.predict(X_test))))
        print("Classification Report: \n {}\n".format(classification_report(y_test, clf.predict(X_test))))
        print("Confusion Matrix: \n {}\n".format(confusion_matrix(y_test, clf.predict(X_test))))

```

7. Définir et entrainer un Random Forest avec des paramètres par défaut.
Jouer avec l'hyperparamètre n_estimators, comment est la performance avec 10, 100 et 200 arbres ?

```

### defining the model
rf_clf = RandomForestClassifier(random_state=42, n_estimators=50)

### fitting
rf_clf.fit(X_train, y_train)

```

8. Imprimer les performances en train et test

```

#performances en train
print_score(rf_clf, X_train, y_train, X_test, y_test, train=True)

#performances en test
print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)

```

9. Visualiser l'importance des features

```
sorted_idx = rf_clf.feature_importances_.argsort()
plt.barh(X_train.columns[sorted_idx], rf_clf.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

Exercise 2 – Grid search

1. Importer les librairies Sklearn

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
```

2. Définir un nouveau Random Forest

```
# defining new Random forest model
rf_clf = RandomForestClassifier(random_state=42)
```

3. Déclarer les hyperparamètres à optimiser

```
#parameters to optimize
params_grid = {"max_depth": [1, 10], # tree depth
               "min_samples_split": [2, 3, 10], #The minimum number of samples required to split an internal node
               "min_samples_leaf": [1, 3, 10], #minimum number of samples required to be at a leaf node
               "bootstrap": [True, False], #bootstrap samples are used when building trees
               "criterion": ['gini', 'entropy'],
               "n_estimators": [10, 20, 30]} #splitting criteria
```

4. Définir le gridsearch et lancer l'entraînement

Quelles autres valeurs peuvent prendre le paramètre scoring ?

```
#defining grid search from sklearn
grid_search = GridSearchCV(rf_clf, params_grid,
                           n_jobs=-1, cv=5,
                           verbose=1, scoring='accuracy')

grid_search.fit(X_train, y_train)
```

5. Imprimer le meilleur score et le meilleur modèle

```
grid_search.best_score_

grid_search.best_estimator_.get_params()
```