Université de Lorraine – M2 informatique - Spécialité IL+ MLFS + AVR

TP 2

Premiers pas avec Python & Scikit-Learn

Installation

Librairies utiles pour Python (>= 3.6):
- pandas
- numpy
- sklearn
- matplotlib

Le plus pratique est d'utiliser Jupyter Notebook
Le mieux (en attendant d'avoir un Jupyter hub à la FST) c'est d'installer Anaconda (https:// www.anaconda.com/products/individual), pour une installation facilitée de Jupyter et d'autres outils pour Python (et R). Il faut compter près de 5Go pour Anaconda…

Une fois Anaconda installé, ouvrir le client web pour Jupyter et créer un nouveau notebook avec (un noyau) Python. Le mien (sur MacOS) avait l'URL http://localhost:8888/tree

Partie 1 : Tutorial

Suivre le petit tutorial Oreilly sur Arche pour créer votre premier notebook (en y insérant les commentaires qui aident à comprendre les étapes et vous permettront de réutiliser les commandes) et le sauvegarder.

Partie 2 : Construction et visualisation d'arbres de décision et encodage de données

a) Charger les données breast-cancer et utiliser Pandas pour explorer ces données

Le fichier **breast-cancer.csv** est sur Arche (à déposer dans votre dossier Jupyter). L'objectif avec ces données et de prédire la rémission d'un cancer du sein (reccurrence_events) à partir de certaines variables.

```
import pandas as pd
import sklearn as sl

df=pd.read_csv("fichier.csv", sep=",")

df

type(df)

#afficher les n premières lignes du fichier
df.head(n)

#afficher les n dernières lignes du fichier
df.tail(n)
```

```
#afficher les types des attributs
df.dtypes

#dimension de la table de données
df.shape

#afficher les valeurs une colonne
df['nom_colonne']

#afficher les valeurs distinctes d'une colonne
df['nom_colonne'].unique()

#accéder aux éléments de la table par leurs index
df.iloc[0] -- première ligne

df.iloc[-1] -- dernière ligne


df.iloc [:2,:2] – deux premières lignes et deux premières colonnes

df.iloc[:,:-1] – toute la table sauf la dernière colonne

#statistiques descriptives des variables catégorielles (nominales)
df.describe (include="object")


#statistiques descriptives des variables numériques
df.describe ()

# ajouter une colonne comme une expression sur des colonnes existantes

df["nouv_colonne"]=df["col1"]*df["col2"]


# Modifier les valeurs d'une colonne cl2 sur une condition sur col1

df.loc[df["col1"] == "valeur", "col2"] = valeur

# Faire la jointure de data frames

Inner_join = df1.merge (df2, on="ID", how="inner")

# enregistrer le contenu d'un data frame dans un fichier

df.to_csv ("fich.csv")
```

b) Tester et comprendre les différentes façons d'encoder les variables nominales en nombres :

- `sklearn.preprocessing.LabelEncoder`
- `sklearn.preprocessing.OrdinalEncoder`
- `sklearn.preprocessing.LabelBinarizer`
- `sklearn.preprocessing.OneHotEncoder`

en les appliquant à un attribut pertinent de votre choix.

```
import pandas as pd
import sklearn as sl
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder

#appliquer LabelEncoder sur la colonne class

le = LabelEncoder ()
df ["class_le"] = le.fit_transform (df ["class"])
df
df.class_le.unique()

#appliquer OneHotEncoder sur la colonne class

ohe = OneHotEncoder (sparse=False)
ohe_resultat = ohe.fit_transform (df ["class"])

df_ohe = pd.dataFrame (ohe_resultat,
columns=ohe.get_feature_names(["class"])
df_ohe

#Appliquer OrdinaleEncoder sur une colonne taille

Tailles=["petit", "moyen", "grand", "énorme"]
oe=OrdinalEncoder (categories=[tailles])
oe.fit_transform (df[["taille"]])       -- ne modifie pas df
df["taille"]= oe.fit_transform (df[["taille]])

-- cette fois la colonne est encodée
```

Ce serait trop long d'encoder l'ensemble des variables en numériques ou en variables binaires (indicatrices avec *OneHotEncoder*). Il faudrait remplacer les intervalles par leur milieu, encoder les variables booléennes (par 0, 1 pour no/false, yes/true) etc.

c) Construire un arbre de décision à partir des données **Glass.csv**. Le visualiser.

d) Calculer la performance du modèle en utilisant la fonction score. La métrique par défaut est l'accuracy mais vous pouvez opter pour une autre métrique (`precision_score` ou `recall_score`)

e) Utiliser la validation croisée (folds = 10) pour évaluer le taux d'erreur (`model_selection.cross_val_score(model, x, y, cv = K)`.

f) Tester une recherche sur grille (grid search) pour sélectionner les meilleurs hyperparamètres : `sklearn.model_selection.GridSearchCV(dtree_model, param_grid, nfolds)`

En donnant comme paramètres `param_grid = {'criterion':['gini','entropy'], 'max_depth': np.arange(3, 15)}` et `nfolds = 10`

Visualiser le nouvel arbre de décision. Quelle est sa performance ?