Introduction à Python pour la Data Analyse

Partie 2 – Préparation de données et Machine Learning

Angelo.Steffenel@univ-reims.fr

Partie 1 – Environnement

Accéder à https://repl.it/languages/python3
Vous pouvez utiliser l'interface sans créer un compte. L'avantage du compte est de pouvoir enregistrer le code que vous avez fait dans ce cours.

Partie 2 – Récupération du dataset avec Pandas

Vous devez charger un dataset sur les passagers du Titanic (*déjà vu ?*)
Pour cela, vous allez importer la bibliothèque pandas avec l'alias **pd** et charger un DataFrame **titanic** à partir d'un fichier csv. Attention, ce fichier a un séparateur ";" à renseigner :

```
import pandas as pd
# la commande suivante sert à afficher toutes les colonnes (pas de …)
pd.options.display.max_columns=100

titanic = pd.read_csv("https://bit.ly/2C2Tup2", sep=';')
```

Votre première tâche est celle d'utiliser ***.head()*** et ***.tail()*** pour afficher la première et la dernière entrée de ce dataset. Obtenez aussi des informations avec ***.info()*** et ***.describe()***

Partie 3 – Types de données

Grâce à ***.info()*** , vous observez que certaines colonnes sont reconnues avec le type "Object", alors qu'on pourrait les transformer en "category". Modifiez le type de donnée des colonnes 'sex', 'pclass', 'survived' et 'embarked' en utilisant ***.astype("category")***.

Si vous voulez connaître combien d'individus sont recensés dans chaque catégorie, vous pouvez utiliser la fonction ***.value_counts()*** appliquée à chaque colonne.

Ex : `titanic['sex']=titanic['sex'].astype("category")`

Partie 4 – Données manquantes et hors-format

Si vous avez fait attention à la sortie de ***.info()***, la colonne 'age' manque certaines cases (des valeurs NaN sont utilisés à leur place). De plus, certaines âges sont représentés en tant que valeurs réelles (0,85 par exemple), mais utilisant une virgule au lieu d'un point (ce qui est attendu par l'ordinateur).

Utiliser, à tour de rôle, les fonctions suivantes pour remplacer la virgule, remplir les cases vides avec la valeur "28" et finalement changer le type de données en float32.

```
titanic['age']=titanic['age'].str.replace(',','.')
titanic['age']=titanic['age'].fillna(28)
titanic['age']=titanic['age'].astype("float32")
```

Étudiez la sortie de **.info()** et **.describe()** après ces modifications.

Partie 5 – supprimer les colonnes "inutiles"

Certaines colonnes ne seront pas utilisées dans notre exercice, vous pouvez les supprimer avec la fonction ***.drop()***. On donnera comme paramètres un array avec les noms des colonnes à supprimer ('body','ticket','fare','cabin','name','boat','home.dest'), ainsi que l'option "axis=1" pour indiquer qu'on souhaite supprimer les colonnes.

Partie 6 – Graphique "camembert"

On veut afficher sur un graphique "camembert" les distributions des colonnes 'survived' et 'pclass'. Pour cela, on utilisera la bibliothèque Matplotlib.

```
import matplotlib.pyplot as plt
sizes=titanic['survived'].value_counts()
fig1, ax1 = plt.subplots()
ax1.pie(sizes, autopct='%1.1f%%', shadow=True, startangle=90)
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a
circle.
plt.show()
```

Si l'image ne s'affiche pas sur l'écran, vous pouvez enregistrer l'image avec plt.savefig('camembert.png') et le visualiser en cliquant dessus.

Partie 7 - BoxPlot

Maintenant, on fera un graphique "boxplot" à partir de la colonne "age". Bien qu'il soit possible de le faire avec Matplotlib, il est bien plus simple (et joli) de le faire en utilisant la bibliothèque Seaborn.

```
import seaborn as sns
sns.boxplot(titanic['age'],orient='v')
plt.show()
```

Si on veut rajouter d'autres colonnes, il suffit de les rajouter :

```
sns.boxplot(titanic['pclass'],titanic['age'],orient='v')
plt.show()
```

Partie 8 - Histogrammes

Toujours intéressés par la colonne "age", nous voulons maintenant faire un histogramme. Avec Matplotlib, il suffit d'indiquer la source des données et le nombre de 'barres' :

```
plt.hist(titanic['age'],bins=8)
plt.show()
```

Partie 9 - Pairplots

Parfois on cherche des corrélations entre les variables. Les pairplots sont des importantes aides visuelles, croisant différentes variables dans un même écran.
Utiliser le code ci-dessous pour générer l'image (on la sauvegardera dans un fichier car elle est trop grande pour la petite fenêtre d'affichage) :

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.pairplot(titanic,vars=['pclass','sex','age','embarked','survived'])
plt.savefig('pairplot.png')
```

Partie 10 – Données catégorisées

Au lieu d'afficher les âges, nous voulons les regrouper en 4 groupes : enfants, jeunes, adultes et seniors. Utiliser la fonction ***.cut()*** pour générer une nouvelle colonne avec ces catégories :

```
titanic['classes'] = pd.cut(titanic["age"], bins=4,
labels=['enfants','jeunes','adultes','seniors']).astype('category')
```

Faire un graphique camembert avec la répartition des classes.

Partie 11 – Machine Learning pt 1

On a fini la préparation des données, maintenant on souhaite tester différentes méthodes de machine learning. Nous allons commencer par la transformation des données quantitatives ('sex', 'pclass', etc.) grâce à ***.get_dummies()*** :

```
dummies=['pclass','sex','embarked','survived']
intercept=pd.get_dummies(titanic,columns=dummies,drop_first=True)
```

Finalement, nous allons générer un dataset d'entraînement et autre de test. Pour cela, on sépare les colonnes avec les variables d'entrée et la colonne à prédire. Ensuite, on générera les datasets en réservant 25% des données pour la partie test :

```
from sklearn.model_selection import train_test_split
titaX = intercept.drop(intercept.columns[-1],axis=1)
titaY = intercept[intercept.columns[-1]]
X_train, X_test, Y_train, Y_test = train_test_split(titaX, titaY,
test_size=0.25)
```

Partie 12 – Régression Linéaire simple

On démarrera par une régression linéaire simple (multivariate). Grâce à Scikit-Learn, on a déjà un "regresseur" tout prêt à être utilisé. On affichera les "poids" de chaque variable puis on fera des prédictions avec le dataset de test, et afficher la comparaison sous forme graphique.

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, Y_train)

coeff_df = pd.DataFrame(regressor.coef_, titaX.columns,
columns=['Coefficient'])
print(coeff_df)

Y_pred = regressor.predict(X_test)
dif = pd.DataFrame({'Actual': Y_test, 'Predicted': Y_pred})
dif1 = dif.head(25)

import matplotlib.pyplot as plt
```

```
dif1.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.savefig('diff.png')
```

Partie 13 – K plus proches voisins et Random Forests

Comme les prédictions avec la régression linéaire sont loin d'être précises, on essayera avec deux autres méthodes, les KNN et les RandomForests.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

modelknn=KNeighborsClassifier()
modelknn.fit(X_train,Y_train)

modelrf=RandomForestClassifier()
modelrf.fit(X_train,Y_train)

y_pred_knn = modelknn.predict(X_test)
dif = pd.DataFrame({'Actual': Y_test, 'Predicted': y_pred_knn})
dif1 = dif.head(25)

dif1.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.savefig('diffknn.png')

y_pred_rf = modelrf.predict(X_test)
dif = pd.DataFrame({'Actual': Y_test, 'Predicted': y_pred_rf})
dif2 = dif.head(25)

dif2.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.savefig('diffrf.png')
```

Pour finir, vous pouvez afficher différentes métriques (MSE, MAE, RMSE) :

```
import sklearn.metrics as metrics
import numpy as np
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test,
y_pred_rf))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test,
y_pred_rf))
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(Y_test, y_pred_rf)))
```

Tout ça n'est juste que le début. Si vous connaissez R, ça peut paraître parfois plus fastidieux, mais par la suite il est extrêmement facile d'accélérer votre code avec des GPUs (bibliothèques Dask, Numba, RAPIDS - https://rapids.ai/), se lancer dans le Deep Learning avec TensorFlow/Keras ou explorer la puissance de plusieurs machines grâce à Spark.