

BIG DATA & DATA SCIENCE

USE CASE 5 : Projet Big Bridge – Déterminer si un profil est à risque ou non

Co-financé par :

Use cases réalisés par les masters :

Sommaire

- Présentation du use case
- Récupération des données
- Préparation des data frames d'entraînement et de test
- Définition du modèle et des couches
- Apprentissage et test
- La fonction `__main__`
- Visualisation des résultats

1. Présentation du use case

Ce use case porte sur le projet Big Bridge et l'analyse de profil de patients afin de savoir s'ils présentent ou non un risque d'après leur profil, selon le taux de pollution.

La suite de ce use case est à réaliser sur un notebook GoogleColab : <https://colab.research.google.com>

Le notebook complet est disponible sur le lien suivant :

<https://colab.research.google.com/drive/1nSrRbXPOFOhyoCzbuBrjxFOkfhYf5OgK?usp=sharing>

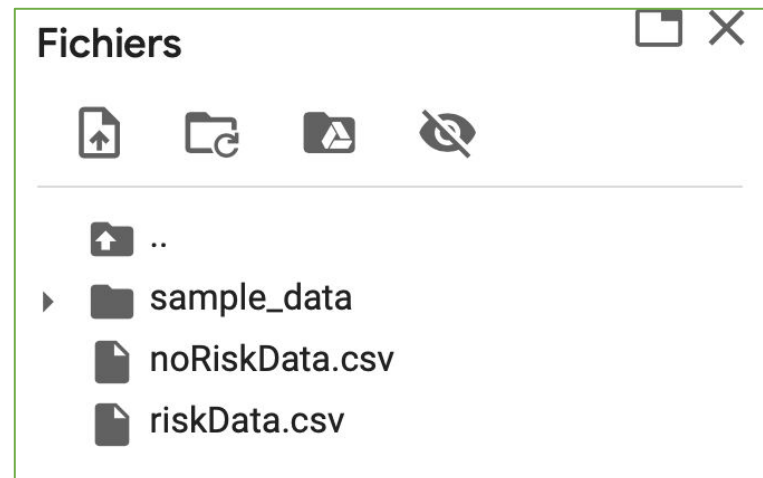
2. Récupération des données

Télécharger les fichiers csv noRiskData.csv et riskData.csv sur le lien suivant :

<https://drive.google.com/file/d/1dH7ab1brLrD6agdjDAPUd0iQLYDf660-/view?usp=sharing>

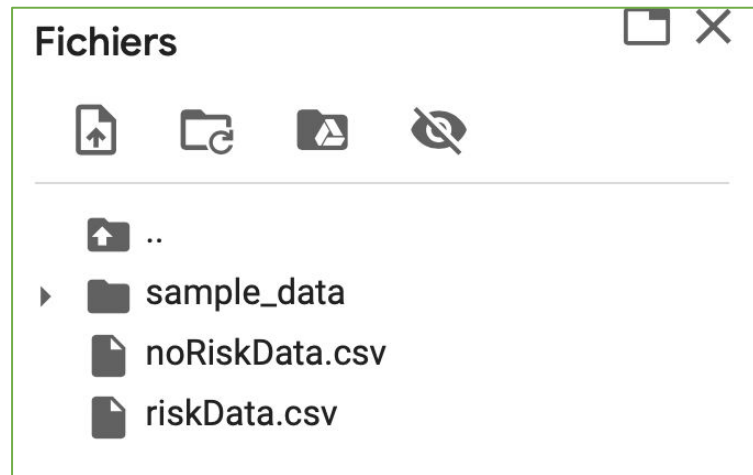
<https://drive.google.com/file/d/1igONsDzNiCUVcgSyF0LLiCyblZqKqkGF/view?usp=sharing>

Les copier dans l'environnement GoogleColab du nouveau notebook que vous venez de créer :



2. Récupération des données

Les copier dans l'environnement GoogleColab du nouveau notebook que vous venez de créer :



Ces deux fichiers contiennent les profils des personnes à risque et les profils des personnes ne présentant pas de risque sur lesquels nous allons faire de la classification supervisée.

2. Récupération des données


Dans un premier temps on importe toutes les librairies nécessaires au projet :

```
▶ import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
from sklearn.metrics import cohen_kappa_score
```

2. Récupération des données

Dans une classe Deep, on définit une fonction deepLearning où on va tout d'abord récupérer les deux jeux de données risk et noRisk.

On va rajouter une colonne « type » qui sera notre variable cible avec pour valeur 1 pour les personnes du dataframe risk et 0 pour le dataframe noRisk .

```
 class Deep :  
    def deepLearning(self):  
        risk = pd.read_table("riskData.csv", sep=";", header=0)  
        noRisk = pd.read_table("noRiskData.csv", sep=";", header=0)  
        # Add `type` column to `riskPeople` with value 1  
        risk['type'] = 1  
  
        # Add `type` column to `noRisk` with value 0  
        noRisk['type'] = 0
```

3. Préparation des data frames d'entraînement et de test

On crée nos ensembles d'apprentissage et de test :

```
# Append `noRisk` to `Risk`
peoples = risk.append(noRisk, ignore_index=True)
print(peoples)
corr = peoples.corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)

# Specify the data
X=peoples.iloc[:,0:10]

# Specify the target labels and flatten the array
y= np.ravel(peoples.type)

# Split the data up in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```


4. Définition du modèle et des couches

On standardise les ensembles et on crée le modèle d'apprentissage avec une couche d'entrée, une couche cachée et une couche de sortie :

```
# Define the scaler
scaler = StandardScaler().fit(X_train)

# Scale the train set
X_train = scaler.transform(X_train)

# Scale the test set
X_test = scaler.transform(X_test)
# Initialize the constructor
model = Sequential()

# Add an input layer
model.add(Dense(12, activation='relu', input_shape=(10,)))
#test with tanh

# Add one hidden layer
model.add(Dense(8, activation='relu'))

# Add an output layer
model.add(Dense(1, activation='sigmoid'))
```

5. Apprentissage et test

```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=50, batch_size=1, verbose=1)

y_pred = model.predict(X_test)
print("y_pred")
print(y_pred[:10])
print("y_test")
print(y_test[:10])

score = model.evaluate(X_test, y_test, verbose=1)
print("score:")
print(score)
```

6. La fonction `__main__`

Sous la classe `Deep`, on crée la fonction `__main__` qui permettra de lancer et d'exécuter les fonctions précédemment définies dans la classe `Deep` :

```
if __name__ == '__main__':  
    deep = Deep()  
    deep.deepLearning()
```

Attention ici à l'indentation : le `if` doit être aligné avec la définition de la classe `Deep`.

7. Visualisation des résultats

Une fois le code lancé, nous pouvons voir la prédiction dans l'onglet d'exécution qui s'affiche en dessous :

```
y_pred
[[5.0150574e-13]
 [1.0000000e+00]
 [1.0000000e+00]
 [4.5713042e-37]
 [1.6111731e-31]
 [1.1490740e-18]
 [9.9999714e-01]
 [1.4658486e-16]
 [0.0000000e+00]
 [7.9238570e-14]]
y_test
[0 1 1 0 0 0 1 0 0 0]
12/12 [=====] - 0s 2ms/step - loss: 0.1992 - accuracy: 0.9779
score:
[0.19915826618671417, 0.9779005646705627]
```

Nous pouvons constater que les personnes à risques sont prédites avec réussite ($y_{\text{test}} = 1$) avec une précision de 0.97.

Références

- [RapportTemin] Rapport de stage 2018, Big Bridge, TEMIN Alison