Université Abdelmalek Essaadi Faculté des Sciences et techniques de Tanger Département Génie Informatique

Cycle Ingénieur : LSI s4

Machine Learning

Pr. EL AACHAK LOTFI

2019/2020

Compte Rendue

Atelier 2 « **Classification** »

Réalisé par :

CHAOUKI Mouad

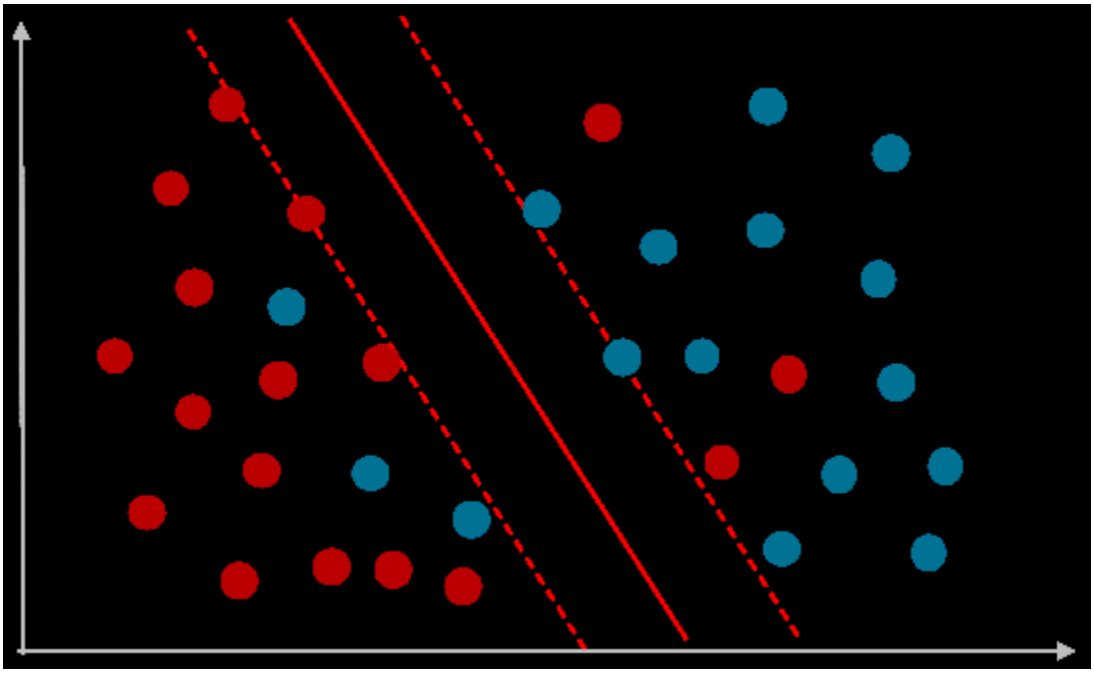
EL KTIBI El Hassan

Encadré par :

Pr.ELAACHAK Lotfi

Objectif :

L'objective principal de cet atelier est de pratiquer les concepts de la classification, en traitant les données d'une Data Sets, ainsi d'évaluer les algorithmes pour construire le modèle adéquat à notre problématique.



Outlies:

Python, Pandas, Sklearn, matplotlib.

Ateliers code sources :

Partie 1 -- Data Visualisation et Feat

Partie 2 -- Classification choix

Data Sets :

Diabetic Data Set: https://www.kaggle.com/kumargh/pimaindiansdiabetescsv

Partie 1 – Data Visualisation et Feature Selection et Normalisation) :

Visualisation :

Pour la visualisation des données. Il faut tout d'abord, importer les Datasets en utilisant le paquetage « ***Pandas » :***

```
DS_FOLDER_PATH = "E:/MDoc/Cycle Courses/Semestres/2nd Year/s4/Machine Learning
 -- Aachak/Ateliers/Atelier 2 Classification/"
CSF_FILE = 'DATA SET -- pima-indians-diabetes.csv'
URL = DS_FOLDER_PATH + CSF_FILE

#importer le datasets
ds = pd.read_csv(URL)
```

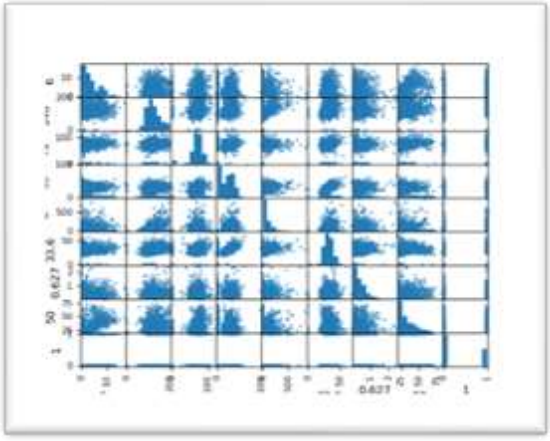
Puis, on peut afficher notre Datasets en détails, par la fonction ***describe(),*** qui va nous permettre de faire une description à notre Datasets ***:***

```
#description du data set
print(ds.describe())
```

	6	148	72	35	0	33.6	0.627	50	1
count	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000
mean	3.842243	120.859192	69.101695	20.517601	79.903520	31.990482	0.471674	33.219035	0.348110
std	3.370877	31.978468	19.368155	15.954059	115.283105	7.889091	0.331497	11.752296	0.476682
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243500	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	32.000000	32.000000	0.371000	29.000000	0.000000
75%	6.000000	140.000000	80.000000	32.000000	127.500000	36.600000	0.625000	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

On projette toutes les valeurs du Data sets en deux variables X et Y mais cette on spécifie l'ensemble des colonnes au variable X. Puis on utilise l'outil 'scatter_matrix' du paquetage « pandas.plotting» pour afficher l'histogramme des différents colonnes.

```
#Visualisation du dataset
scatter_matrix(ds)
plt.show()
```



Sélection :

Pour faire la sélection, il faut tout d'abord remplir le data frame de tel sort on peut projeter les valeurs de data sets en deux variables X et Y

```
#remplir dataframe
```

```
names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin'
, 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Class']
dataframe = pd.read_csv(URL, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
```

Et puis, on peut faire la séléction :

- Élimination récursive de caractéristiques «Recursive Feature Elimination, RFE » :

```
#Élimination récursive de caractéristiques «Recursive Feature Elimination»
print("\n\nImportance des caractéristiques « Feature Importance , FI» :")
model = LogisticRegression()
rfe = RFE(model, 4)
fit = rfe.fit(X, Y)
print("\nNum Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

On aura le résultat suivant :

```
Num Features: 4
Selected Features: [ True  True False False False  True  True False]
Feature Ranking: [1 1 3 4 5 1 1 2]
```

On peut voir que RFE a choisi les 4 principales fonctionnalités : ***Pregnancies, Glucose, DPF et Age***

- Analyse des composants principaux « Principal Component Analysis, PCA » :

```
#Analyse des composants principaux « Principal Component Analysis, PCA »
print("\n\nImportance des caractéristiques « Feature Importance , FI» :")
pca = PCA(n_components=3)
fit = pca.fit(X)
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)
```

On aura le résultat suivant :

```
Explained Variance: [0.88854663 0.06159078 0.02579012]
[[-2.02176587e-03  9.78115765e-02  1.60930503e-02  6.07566861e-02
   9.93110844e-01  1.40108085e-02  5.37167919e-04 -3.56474430e-03]
 [-2.26488861e-02 -9.72210040e-01 -1.41909330e-01  5.78614699e-02
   9.46266913e-02 -4.69729766e-02 -8.16804621e-04 -1.40168181e-01]
 [-2.24649003e-02  1.43428710e-01 -9.22467192e-01 -3.07013055e-01
   2.09773019e-02 -1.32444542e-01 -6.39983017e-04 -1.25454310e-01]]
```

- Importance des caractéristiques « Feature Importance , FI» :

```
#Importance des caractéristiques « Feature Importance , FI»
print("\n\nImportance des caractéristiques « Feature Importance , FI» :")
model = ExtraTreesClassifier()
model.fit(X, Y)
print(model.feature_importances_)
```

On aura le résultat suivant :

```
Importance des caractéristiques « Feature Importance , FI» :
[0.11042056 0.23426956 0.09853991 0.07925164 0.07692991 0.1405244
 0.12115918 0.13890483]
```

Normalisation :

La normalisation est une technique souvent appliquée dans le cadre de la préparation de données pour l'apprentissage automatique. Le but de la normalisation est de changer les valeurs des colonnes numériques du jeu de données en une échelle commune.

En utilisant l'api « ***sklearn*** » , on peut importer les paquetages suivantes :

```
#Normalisation
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
```

Puis on fait la normalisation, selon différent niveaux :

```
#MinMaxScaler
minmaxScaler = MinMaxScaler()
minmaxScaler.fit(X)
print("\n\n MinMaxScaler aprés la transformation :")
print(minmaxScaler.transform(X))

#RobustScaler
robustScaler = RobustScaler().fit(X)
print("\n\n RobustScaler aprés la transformation :")
print(robustScaler.transform(X))

#StandardScaler
standardScaler = StandardScaler()
standardScaler.fit(X)
print("\n\n StandardScaler aprés la transformation :")
print(standardScaler.transform(X))

#Normalizer
normalizer = Normalizer().fit(X)
print("\n\n Normalizer aprés la transformation :")
print(normalizer.transform(X))
```

Et comme résultat, on aura la normalisation de notre data sets de la forme suivante :

```
MinMaxScaler aprés la transformation :
[[0.35294118 0.74371859 0.59016393 ... 0.50074516 0.23441503 0.48333333]
 [0.05882353 0.42713568 0.54098361 ... 0.39642325 0.11656704 0.16666667]
 [0.47058824 0.91959799 0.52459016 ... 0.34724292 0.25362938 0.18333333]
 ...
 [0.29411765 0.6080402  0.59016393 ... 0.390462   0.07130658 0.15      ]
 [0.05882353 0.63316583 0.49180328 ... 0.4485842  0.11571307 0.43333333]
 [0.05882353 0.46733668 0.57377049 ... 0.45305514 0.10119556 0.03333333]]
```

```
RobustScaler aprés la transformation :
[[ 0.6        0.75151515  0.         ...  0.17204301  0.66535948
   1.23529412]
 [-0.4       -0.77575758 -0.33333333 ... -0.58064516 -0.05620915
   0.11764706]
 [ 1.         1.6        -0.44444444 ... -0.93548387  0.78300654
   0.17647059]
 ...
 [ 0.4        0.0969697   0.         ... -0.62365591 -0.33333333
   0.05882353]
 [-0.4        0.21818182 -0.66666667 ... -0.20430108 -0.06143791
   1.05882353]
 [-0.4       -0.58181818 -0.11111111 ... -0.17204301 -0.1503268
  -0.35294118]]
```

```
Normalizer aprés la transformation :
[[0.03355237 0.82762513 0.40262844 ... 0.18789327 0.00350622 0.27960308]
 [0.008424   0.71604034 0.55598426 ... 0.22407851 0.00295683 0.26114412]
 [0.04039768 0.92409698 0.32318146 ... 0.11765825 0.00339341 0.16159073]
 ...
 [0.02691539 0.65135243 0.38758161 ... 0.14103664 0.00131885 0.16149234]
 [0.00665306 0.83828547 0.39918356 ... 0.20025708 0.00232192 0.31269379]
 [0.00791454 0.73605211 0.55401772 ... 0.24060198 0.00249308 0.18203439]]
```

Partie 2 -- Classification choix de algorithme adéquat):

Pour la partie du regression et avant faire l'apprentissage, il faut qu'on devisé notre Datasets en deux catégories, selon l'indice ***' test_size '***:

- Une pour faire l'apprentissage du notre modèle.
- L'autre faire pour faire le test et l'valuation de notre modèle, c-à-d comparer les données réelles avec les données que notre modèle a prédit.

```
DS_FOLDER_PATH = "E:/MDoc/Cycle Courses/Semestres/2nd Year/s4/Machine Learning
 -- Aachak/Ateliers/Atelier 2 Classification/"
CSF_FILE = 'DATA SET -- pima-indians-diabetes.csv'
URL = DS_FOLDER_PATH + CSF_FILE

#importer le datasets au dataframe
names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin'
, 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Class']
dataframe = pd.read_csv(URL, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
```

Ensuite, on lance crée nos modèles et les remplir avec la fonction ***fit()*** qui se trouve dans l'api « ***sklearn.linear_model*** »

```
#-----------------Application des algorithmes-------------------------------|
print("\n\n -----------Application des algorithmes-----------")
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,random
_state=0)

# Fit the model on training set
print("\n\nTraining des modèles:")
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

svc = SVC()
svc.fit(X_train, y_train)

nb = GaussianNB()
nb.fit(X_train, y_train)
```

Puis on crée un tableau des modèles et des noms des fichiers pour enregistrées notre modèle grâce à la bibliothèque « ***pickle ».***

```
#Save&Load
import pickle

# prepare models
models = []
models.append(('Linear Discriminant Analysis', lda))
models.append(('K-Nearest Neighbors', knn))
models.append(('Classification and Regression Trees', dt))
models.append(('Naive Bayes', nb))

#prepare filenames
filenames = []
filenames.append(("Model-KNeighborsClassifier_model.sav",knn))
filenames.append(("Model-LinearDiscriminantAnalysis_model.sav",lda))
filenames.append(("Model-DecisionTreeClassifier_model.sav",dt))
filenames.append(("Model-SVC_model.sav",svc))
filenames.append(("Model-GaussianNB_model.sav",nb))

# save the model to disk
print("\n\nEnregistrement des modèles ")
for file, model in filenames:
        pickle.dump(model, open(file, 'wb'))
```

Après l'enregistrement, on évalue notre modèles en utilisant les algorithmes de sélection

```
# prepare models
models = []
models.append(('Linear Discriminant Analysis', lda))
models.append(('K-Nearest Neighbors', knn))
models.append(('Classification and Regression Trees', dt))
models.append(('Naive Bayes', nb))

#prepare filenames
filenames = []
filenames.append(("Model-KNeighborsClassifier_model.sav",knn))
filenames.append(("Model-LinearDiscriminantAnalysis_model.sav",lda))
filenames.append(("Model-DecisionTreeClassifier_model.sav",dt))
filenames.append(("Model-SVC_model.sav",svc))
filenames.append(("Model-GaussianNB_model.sav",nb))
```

puis on lance une boucle qui va parcourir toutes les modèles et appliquer les différent algorithmes de sélection :

```
# prepare scorings
scoring = []
scoring.append(('Accuracy','accuracy'))
scoring.append(('Logloss','neg_log_loss'))
scoring.append(('Area Under ROC Curve','roc_auc'))

seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
```

```
#-------------Évaluation des modèles en utilisant des métriques--------------|

print("\n\n-----------Evaluation des modèles-----------")

for method, score in scoring:
        msg = "\n\t%s :" % (method)
        print(msg)
        for name, model in models:
                result = model_selection.cross_val_score(model, X, Y, cv=kfold
,scoring=score)
                msg = "\n\t\t%s : %f (%f)" % ( name, result.mean(), result.std
())
                print(msg)
#Confusion Matrix
print("\nConfusion Matrix :")
for name, model in models:
        predicted = model.predict(X_test)
        matrix = confusion_matrix(y_test, predicted)
        msg = "\n%s :" % (name)
        print(msg)
        print(matrix)
#Classification Report
print("\nClassification Report :")
for name, model in models:
        predicted = model.predict(X_test)
        report = classification_report(y_test, predicted)
        msg = "\n%s :" % (name)
        print(msg)
        print(report)
```

Et on aura résultas ci-dessus, et choisit le modèl qui a une résultat plus proche à 1

----------Evaluation des modèles----------

Accuracy :

Linear Discriminant Analysis : 0.773462 (0.051592)

K-Nearest Neighbors : 0.726555 (0.061821)

Classification and Regression Trees : 0.691302 (0.069249)

Naive Bayes : 0.755178 (0.042766)

Logloss :

Linear Discriminant Analysis : -0.485655 (0.064135)

K-Nearest Neighbors : -1.799319 (1.044532)

Classification and Regression Trees : -10.976749 (2.115891)

Naive Bayes : -0.618809 (0.171077)

Area Under ROC Curve :

Linear Discriminant Analysis : 0.828667 (0.043611)

K-Nearest Neighbors : 0.752003 (0.065476)

Classification and Regression Trees : 0.655075 (0.066525)

Naive Bayes : 0.818336 (0.045341)

Confusion Matrix :

Linear Discriminant Analysis :
[[140 17]
 [34 40]]

K-Nearest Neighbors :
[[134 23]
 [35 39]]

Classification and Regression Trees :
[[126 31]
 [30 44]]

```
Naive Bayes :
[[138  19]
 [ 36  38]]

Classification Report :

Linear Discriminant Analysis :
              precision    recall  f1-score   support

         0.0       0.80      0.89      0.85       157
         1.0       0.70      0.54      0.61        74

    accuracy                           0.78       231
   macro avg       0.75      0.72      0.73       231
weighted avg       0.77      0.78      0.77       231

K-Nearest Neighbors :
              precision    recall  f1-score   support

         0.0       0.79      0.85      0.82       157
         1.0       0.63      0.53      0.57        74

    accuracy                           0.75       231
   macro avg       0.71      0.69      0.70       231
weighted avg       0.74      0.75      0.74       231

Classification and Regression Trees :
              precision    recall  f1-score   support

         0.0       0.81      0.80      0.81       157
         1.0       0.59      0.59      0.59        74

    accuracy                           0.74       231
   macro avg       0.70      0.70      0.70       231
weighted avg       0.74      0.74      0.74       231

Naive Bayes :
              precision    recall  f1-score   support

         0.0       0.79      0.88      0.83       157
         1.0       0.67      0.51      0.58        74

    accuracy                           0.76       231
   macro avg       0.73      0.70      0.71       231
weighted avg       0.75      0.76      0.75       231
```

Et par consequent, on trouve que le modèle ***« Linear Discriminant Analysis »*** est le plus adapté pour cette data sets.

Pour mieux conclure cette comparaison, on fait la comparaison en utilise le spot checking

```
----------Comparaison en utilisant spot checking ----------

Linear Discriminant Analysis : (0.773462)

K-Nearest Neighbors : (0.726555)

Classification and Regression Trees : (0.693900)

Naive Bayes : (0.755178)
```

Et on trouve que le « ***LDA*** » a un scoring de 0.77.

```
#------------chargement et prédections des modèles------------------|

# load the model from disk
print("\n\n-----------Chargement et prédections des modèles-----------")
for file, model in filenames:
        loaded_model = pickle.load(open(file, 'rb'))
        result = loaded_model.score(X_test, y_test)
        print("\n %s \n\t (%f)"%(file,result))
```

Avec les résultats suivantes :

```
----------Chargement et prédections des modèles----------

Model-KNeighborsClassifier_model.sav
    (0.748918)

Model-LinearDiscriminantAnalysis_model.sav
    (0.779221)

Model-DecisionTreeClassifier_model.sav
    (0.735931)

Model-SVC_model.sav
    (0.753247)

Model-GaussianNB_model.sav
    (0.761905)
```