Classification avec le jeu de données TITANIC

L'objectif est de poursuivre le TP précédente où de nombreux prétraitements ont été effecutés en ingénierie des données pour pouvoir faire un modèle de prédiction des survivants ou non.

Executer les importations suivantes :

In [1]:

```
import pandas as pd
import numpy as np
import sklearn
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

#Sickit learn met régulièrement à jour des versions et indique des futurs v
#ces deux lignes permettent de ne pas les afficher.
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

Si vous n'avez pas terminé le TP précédent vous pouvez récupérer le fichier titanic2.csv qui contient les différentes transformations. Le mettre dans le répertoire Dataset.

In [2]:

```
df=pd.read_csv('Dataset/titanic2.csv', sep=';')
df.head()
```

Out[2]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	4	1	0	0	2
1	1	1	0	0	1	0	3	0
2	1	3	0	6	0	0	0	2
3	1	1	0	6	1	0	3	2
4	0	3	1	6	0	0	1	2

Il faut maintenant définir les variables d'apprentissage et la variable à prédire. Ici Suvived est la variable à prédire. Attention elle est positionné à la première colonne.

In [3]:

```
array = df.values
X = array[:,1:8]
y = array[:,0]

```

Découper le jeu de données en jeu de test et jeu d'apprentissage. Prenez 30% du jeu de données pour le test.

In [4]:

```

validation_size=0.3 #30% du jeu de données pour le test

testsize= 1-validation_size
seed=30
X_train, X_test, y_train, y_test = train_test_split(X, y,
                    train_size=validation_size,
                    random_state=seed,test_size=testsize)


```

Essai d'un classifieur

Avec le classifieur GaussianNB effectuer une première prédiction en donnant la valeur de l'accuracy.

In [5]:

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

result = clf.predict(X_test)

print('\n accuracy :',
      accuracy_score(result, y_test),'\n')
```

```
accuracy : 0.7155963302752294
```

Afficher la matrice de confusion et le classification report.

In [6]:

```
conf = confusion_matrix(y_test, result)
print ('\n matrice de confusion \n',conf)

print ('\n',classification_report(y_test,
                                  result))
```

```
matrice de confusion
[[57 17]
[14 21]]

              precision    recall  f1-score   support

           0       0.80      0.77      0.79        74
           1       0.55      0.60      0.58        35

   micro avg       0.72      0.72      0.72       109
   macro avg       0.68      0.69      0.68       109
weighted avg       0.72      0.72      0.72       109
```

Effectuer une cross validation avec 10 splits (Kfold)

In [7]:

```
seed=7
k_fold = KFold(n_splits=10,
               shuffle=True, random_state=seed)
```

Appliquer le classifieur GaussianNB et donner les différentes accuracy pour les 10 évaluations.

In [8]:

```
clf = GaussianNB()

scoring = 'accuracy'
score = cross_val_score(clf, X, y,
                        cv=k_fold, scoring=scoring)

print('Les différentes accuracy pour les 10 évaluations sont : \n',score,'
print ('Accuracy moyenne : ',score.mean(),
       ' standard deviation', score.std())

```

```
Les différentes accuracy pour les 10 évaluations sont :
 [0.8125     0.8125     0.75       0.75       0.875      0.86666667
 0.73333333 0.8        0.73333333 0.73333333]

Accuracy moyenne :  0.7866666666666667  standard deviation 0.0519013
48730066764
```

Essai de plusieurs classifieurs

Utiliser à présent différents classifieurs : KNeighborsClassifier, DecisionTreeClassifier, GaussianNB, SVC et RandomForestClassifier.

In [9]:

```
seed = 7
scoring = 'accuracy'
models = []
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
models.append(('RFO', RandomForestClassifier()))
```

Quels sont les résultats des classifications ? Quel est le classifieur le plus efficace ?

In [10]:

```
results = []
names = []
for name,model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(model, X,
                                 y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(),
                           cv_results.std())
    print(msg)
```

```
KNN: 0.647083 (0.109117)
CART: 0.728333 (0.050649)
NB: 0.768750 (0.072821)
SVM: 0.730000 (0.071317)
RFO: 0.774167 (0.099055)
```

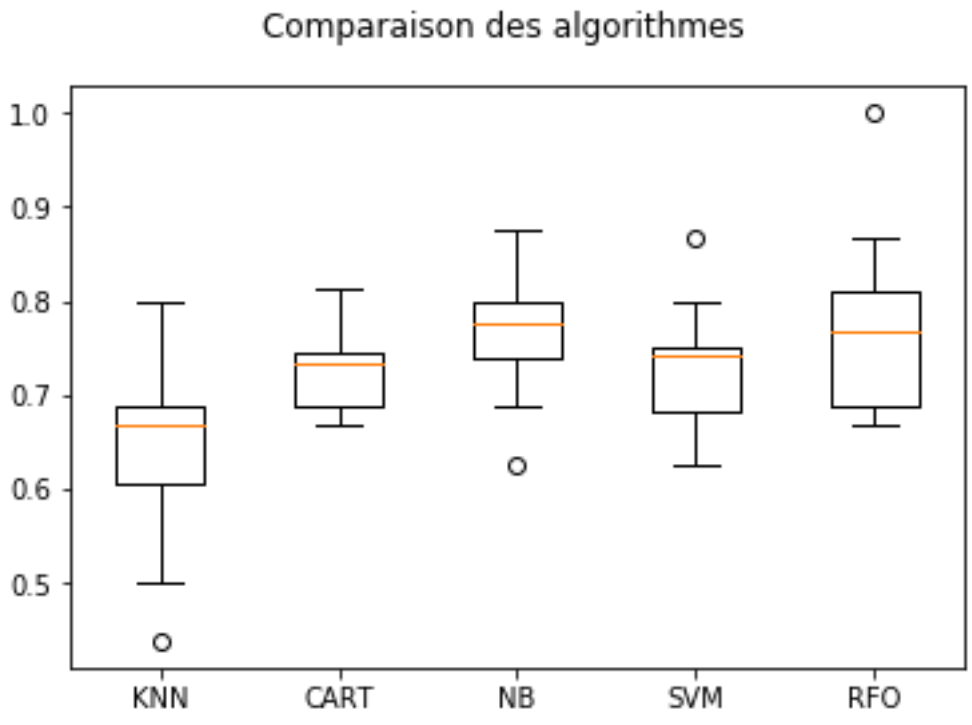
A l'aide de boxplot afficher les résultats des différents classifieurs.

In [11]:

```
fig = plt.figure()
fig.suptitle('Comparaison des algorithmes')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
```

Out[11]:

```
[Text(0, 0, 'KNN'),
 Text(0, 0, 'CART'),
 Text(0, 0, 'NB'),
 Text(0, 0, 'SVM'),
 Text(0, 0, 'RFO')]
```



Le classifieur RandomForestClassifier donne de meilleurs résultats. A l'aide de GridSearch, évaluer les différents paramètres :
grid_param = {'n_estimators': [4, 6, 9],
'max_features': ['log2', 'sqrt','auto'],
'criterion': ['entropy', 'gini'],
'max_depth': [2, 3, 5, 10],
'min_samples_split': [2, 3, 5],
'min_samples_leaf': [1,5,8]
}

In [12]:

```
grid_param = {'n_estimators': [4, 6, 9],
              'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10],
              'min_samples_split': [2, 3, 5],
              'min_samples_leaf': [1,5,8]
             }

gd_sr = GridSearchCV(estimator=RandomForestClassifier(),
                     param_grid=grid_param,
                     scoring='accuracy',
                     cv=5,
                     n_jobs=-1,
                    iid=True,
                     return_train_score=True)


```

Quelle est la meilleure accuracy et pour quels paramètres ?

In [13]:

```
gd_sr.fit(X_train, y_train)
print ('meilleur score ',
       gd_sr.best_score_,'\n')
print ('meilleurs paramètres',
       gd_sr.best_params_,'\n')
print ('meilleur estimateur',
       gd_sr.best_estimator_,'\n')
```

```
meilleur score  0.782608695652174

meilleurs paramètres {'criterion': 'entropy', 'max_depth': 2, 'max_f
eatures': 'sqrt', 'min_samples_leaf': 5, 'min_samples_split': 2, 'n_
estimators': 4}

meilleur estimateur RandomForestClassifier(bootstrap=True, class_wei
ght=None, criterion='entropy',
            max_depth=2, max_features='sqrt', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=4, n_jobs=Non
e,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

GaussianNB obtient un assez bon score mais il n'a pas d'hyperparamètres à rechercher. Il est suivi par DecisisionTreeClassifier et SVC.

Appliquer GridSearchCV pour ces deux classifieurs.

In [14]:

```
from sklearn.tree import DecisionTreeClassifier


grid_param = {
    'max_depth': [1,2,3,4,5,6,7,8,9,10],
    'criterion': ['gini', 'entropy'],
    'min_samples_leaf': [1,2,3,4,5,6,7,8,9,10]
}


gd_sr = GridSearchCV(estimator=DecisionTreeClassifier(),
                     param_grid=grid_param,
                     scoring='accuracy',
                     cv=10,
                     n_jobs=-1,
                    iid=True,
                    return_train_score=True)

gd_sr.fit(X_train, y_train)

gd_sr.fit(X_train, y_train)
print ('meilleur score ',gd_sr.best_score_,'\n')
print ('meilleurs paramètres', gd_sr.best_params_,'\n')
print ('meilleur estimateur',gd_sr.best_estimator_,'\n')
```

```
meilleur score  0.782608695652174
```

```
meilleurs paramètres {'criterion': 'gini', 'max_depth': 1, 'min_samp
les_leaf': 1}

meilleur estimateur DecisionTreeClassifier(class_weight=None, criter
ion='gini', max_depth=1,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=None,
            splitter='best')
```

In [15]:

```
grid_param = {
    'C': [0.001, 0.01, 0.1, 1, 10],
    'gamma' : [0.001, 0.01, 0.1, 1],
    'kernel': ['linear','rbf']}

gd_sr = GridSearchCV(estimator=SVC(),
                     param_grid=grid_param,
                     scoring='accuracy',
                     cv=5,
                     n_jobs=1,
                     iid=True,
                    return_train_score=True)

gd_sr.fit(X_train, y_train)

print ('meilleur score ',
       gd_sr.best_score_,'\n')
print ('meilleurs paramètres',
       gd_sr.best_params_,'\n')
print ('meilleur estimateur',
       gd_sr.best_estimator_,'\n')


```

```
meilleur score  0.7391304347826086
meilleurs paramètres {'C': 1, 'gamma': 0.001, 'kernel': 'linear'}

meilleur estimateur SVC(C=1, cache_size=200, class_weight=None, coef
0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.001, kernel='line
ar',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Vous constatez que RandomForestClassifier et DecisionTreeClassifier peuvent avoir la même accuracy.
Faire un gridsearch avec les deux classifieurs en prenant les paramètres précédents.

In [16]:

```
classifiers = {
    'RandomForestClassifier': RandomForestClassifier(),
    'DecisionTreeClassifier': DecisionTreeClassifier()
}

params = {'RandomForestClassifier' :
          [{'n_estimators': [4, 6, 9]},
            {'max_features': ['log2', 'sqrt','auto']},
            {'criterion': ['entropy', 'gini']},
            {'max_depth': [2, 3, 5, 10]},
            {'min_samples_split': [2, 3, 5]},
            {'min_samples_leaf': [1,5,8]}],
           'DecisionTreeClassifier':
          [{'max_depth': [1,2,3,4,5,6,7,8,9,10]},
    {'criterion': ['gini', 'entropy']},
    {'min_samples_leaf': [1,2,3,4,5,6,7,8,9,10]}]}


```

In [17]:

```
class Result:
     def __init__(self,name, score, parameters):
         self.name = name
         self.score = score
         self.parameters = parameters
     def __repr__(self):
         return repr((self.name, self.score, self.parameters))


results = []
for key,value in classifiers.items():
    gd_sr = GridSearchCV(estimator=value,
                         param_grid=params[key],
                         scoring='accuracy',
                         cv=10,
                         n_jobs=1,
                         iid=True)
    gd_sr.fit(X_train, y_train)
    result=Result(key,gd_sr.best_score_,
                  gd_sr.best_estimator_)
    results.append(result)



results=sorted(results,
               key=lambda result: result.score,
               reverse=True)

print ('Le meilleur resultat : \n')
print ('Classifier : ',results[0].name,
       ' score %0.4f' %results[0].score,
       ' avec ',results[0].parameters,'\n')

print ('Tous les résultats : \n')
```

```
for result in results:
    print ('Classifier : ',result.name,
           ' score %0.4f' %result.score,
           ' avec ',result.parameters,'\n')
```

```
Le meilleur resultat :

Classifier :  DecisionTreeClassifier  score 0.7826  avec  DecisionTr
eeClassifier(class_weight=None, criterion='gini', max_depth=1,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=None,
            splitter='best')

Tous les résultats :

Classifier :  DecisionTreeClassifier  score 0.7826  avec  DecisionTr
eeClassifier(class_weight=None, criterion='gini', max_depth=1,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=None,
            splitter='best')
```

```
Classifier :  RandomForestClassifier  score 0.6957  avec  RandomFore
stClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None
,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=No
ne,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

Conclusion : il faut toujours faire attention aux paramètres passés dans un classifieur. Ils peuvent tout changer !

Sauvegarde du modèle

Sauvegarder le meilleur modèle appris et recharger le pour le tester avec y_test.

In [18]:

```
clf=gd_sr.best_estimator_
import pickle
filename = 'pkl_Titanicbestmodel.sav'
pickle.dump(clf, open(filename, 'wb'))
```

In [19]:

```
clf_loaded = pickle.load(open(filename, 'rb'))
print ('Modèle chargé',clf_loaded,'\n')
result = clf_loaded.predict(X_test)

print('\n accuracy:\n')
print (accuracy_score(result, y_test),'\n')

conf = confusion_matrix(y_test, result)
print ('\n matrice de confusion \n',conf)
print ('\n',classification_report(y_test, result))
```

```
Modèle chargé DecisionTreeClassifier(class_weight=None, criterion='g
ini', max_depth=1,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=None,
            splitter='best')
```

```
 accuracy:

0.8165137614678899


 matrice de confusion
 [[62 12]
 [ 8 27]]

              precision    recall  f1-score   support

           0       0.89      0.84      0.86        74
           1       0.69      0.77      0.73        35

   micro avg       0.82      0.82      0.82       109
   macro avg       0.79      0.80      0.80       109
weighted avg       0.82      0.82      0.82       109
```