Projet de Machine learning

Attali Josh & Brossat Hugo

Introduction

Dans le cadre du Master 2 Business Intelligence et Analytics à l'Université Lumière Lyon 2, nous avons été amené à réaliser un projet en Machine Learning. Cet enseignement nous a été dispensé par Monsieur Guillaume Metzler au cours du 1er semestre.

Ce projet doit s'effectuer en binôme et porte sur les techniques d'apprentissage dans un contexte de classification binaire. Le choix du langage de programmation est laissé à l'appréciation des étudiants. Nous avons donc choisi de travailler en utilisant le langage Python.

En ce qui concerne les jeux de données étudiés, nous avions à disposition un large éventail de datasets se portant sur des sujets diverses. Nous avons décidé de sélectionner et baser notre étude plus particulièrement sur les jeux de données « winequality-red.csv » et vehicle. Nous comparerons tout de même les résultats de l'ensemble des jeux de données dans la dernière partie.

Préparation des data

Cette partie permet de mettre en forme les différents jeux de données afin que ces derniers soient prêts à être employés.

In [ ]:
```
# Some required libraries

import os
import csv
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedKFo
ld
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
from sklearn import metrics

# A first function to dowlnload the datasets

def loadCsv(path):
    data = []
    with open(path, 'r') as csvfile:
        reader = csv.reader(csvfile, delimiter=',')
        for row in reader:
            data.append(np.array(row))
    data = np.array(data)
    (n, d) = data.shape
    return data, n, d

# Encode Categorical variables

def oneHotEncodeColumns(data, columnsCategories):
    dataCategories = data[:, columnsCategories]
    dataEncoded = OneHotEncoder(sparse=False).fit_transform(dataCat
egories)
    columnsNumerical = []
    for i in range(data.shape[1]):
        if i not in columnsCategories:
            columnsNumerical.append(i)
    dataNumerical = data[:, columnsNumerical]
    return np.hstack((dataNumerical, dataEncoded)).astype(float)
```

In [ ]:
```
# Another function to prepare the data

def data_recovery(dataset):
    if dataset in ['abalone8', 'abalone17', 'abalone20']:
        data = pd.read_csv("/datasets/abalone.data", header=None)
        data = pd.get_dummies(data, dtype=float)
        if dataset in ['abalone8']:
            y = np.array([1 if elt == 8 else 0 for elt in data[8]])
        elif dataset in ['abalone17']:
            y = np.array([1 if elt == 17 else 0 for elt in data[8]]
)
        elif dataset in ['abalone20']:
            y = np.array([1 if elt == 20 else 0 for elt in data[8]]
)
        X = np.array(data.drop([8], axis=1))
```

```
    elif dataset in ['autompg']:
        data = pd.read_csv("/datasets/auto-mpg.data", header=None,
sep=r'\s+')
        data = data.replace('?', np.nan)
        data = data.dropna()
        data = data.drop([8], axis=1)
        data = data.astype(float)
        y = np.array([1 if elt in [2, 3] else 0 for elt in data[7]]
)
        X = np.array(data.drop([7], axis=1))
    elif dataset in ['australian']:
        data, n, d = loadCsv('/datasets/australian.data')
        X = data[:, np.arange(d-1)].astype(float)
        y = data[:, d-1].astype(int)
        y[y != 1] = 0
    elif dataset in ['balance']:
        data = pd.read_csv("/datasets/balance-scale.data", header=N
one)
        y = np.array([1 if elt in ['L'] else 0 for elt in data[0]])
        X = np.array(data.drop([0], axis=1))
    elif dataset in ['bankmarketing']:
        data, n, d = loadCsv('/datasets/bankmarketing.csv')
        X = data[:, np.arange(0, d-1)]
        X = oneHotEncodeColumns(X, [1, 2, 3, 4, 6, 7, 8, 10, 15])
        y = data[:, d-1]
        y[y == "no"] = "0"
        y[y == "yes"] = "1"
        y = y.astype(int)
    elif dataset in ['bupa']:
        data, n, d = loadCsv('/datasets/bupa.dat')
        X = data[:, np.arange(d-1)].astype(float)
        y = data[:, d-1].astype(int)
        y[y != 1] = 0
    elif dataset in ['german']:
        data = pd.read_csv("/datasets/german.data-numeric", header=
None,
                           sep=r'\s+')
        y = np.array([1 if elt == 2 else 0 for elt in data[24]])
        X = np.array(data.drop([24], axis=1))
    elif dataset in ['glass']:
        data = pd.read_csv("/datasets/glass.data", header=None, ind
ex_col=0)
        y = np.array([1 if elt == 1 else 0 for elt in data[10]])
        X = np.array(data.drop([10], axis=1))
    elif dataset in ['hayes']:
        data = pd.read_csv("/datasets/hayes-roth.data", header=None
)
        y = np.array([1 if elt in [3] else 0 for elt in data[5]])
        X = np.array(data.drop([0, 5], axis=1))
    elif dataset in ['heart']:
        data, n, d = loadCsv('/datasets/heart.data')
        X = data[:, np.arange(d-1)].astype(float)
        y = data[:, d-1]
        y = y.astype(int)
```

```
        y[y != 2] = 0
        y[y == 2] = 1
    elif dataset in ['iono']:
        data = pd.read_csv("/datasets/ionosphere.data", header=None
)
        y = np.array([1 if elt in ['b'] else 0 for elt in data[34]]
)
        X = np.array(data.drop([34], axis=1))
    elif dataset in ['libras']:
        data = pd.read_csv("/datasets/movement_libras.data", header
=None)
        y = np.array([1 if elt in [1] else 0 for elt in data[90]])
        X = np.array(data.drop([90], axis=1))
    elif dataset == "newthyroid":
        data, n, d = loadCsv('/datasets/newthyroid.dat')
        X = data[:, np.arange(d-1)].astype(float)
        y = data[:, d-1].astype(int)
        y[y < 2] = 0
        y[y >= 2] = 1
    elif dataset in ['pageblocks']:
        data = pd.read_csv("/datasets/page-blocks.data", header=Non
e,
                           sep=r'\s+')
        y = np.array([1 if elt in [2, 3, 4, 5] else 0 for elt in da
ta[10]])
        X = np.array(data.drop([10], axis=1))
    elif dataset in ['pima']:
        data, n, d = loadCsv('/datasets/pima-indians-diabetes.data'
)
        X = data[:, np.arange(d-1)].astype(float)
        y = data[:, d-1]
        y[y != '1'] = '0'
        y = y.astype(int)
    elif dataset in ['satimage']:
        data, n, d = loadCsv('/datasets/satimage.data')
        X = data[:, np.arange(d-1)].astype(float)
        y = data[:, d-1]
        y = y.astype(int)
        y[y != 4] = 0
        y[y == 4] = 1
    elif dataset in ['segmentation']:
        data, n, d = loadCsv('/datasets/segmentation.data')
        X = data[:, np.arange(1, d)].astype(float)
        y = data[:, 0]
        y[y == "WINDOW"] = '1'
        y[y != '1'] = '0'
        y = y.astype(int)
    elif dataset == "sonar":
        data, n, d = loadCsv('/datasets/sonar.dat')
        X = data[:, np.arange(d-1)].astype(float)
        y = data[:, d-1]
        y[y != 'R'] = '0'
        y[y == 'R'] = '1'
```

```
        y = y.astype(int)
    elif dataset == "spambase":
        data, n, d = loadCsv('/datasets/spambase.dat')
        X = data[:, np.arange(d-1)].astype(float)
        y = data[:, d-1].astype(int)
        y[y != 1] = 0
    elif dataset == "splice":
        data, n, d = loadCsv('/datasets/splice.data')
        X = data[:, np.arange(1, d)].astype(float)
        y = data[:, 0].astype(int)
        y[y == 1] = 2
        y[y == -1] = 1
        y[y == 2] = 0
    elif dataset in ['vehicle']:
        data, n, d = loadCsv('/datasets/vehicle.data')
        X = data[:, np.arange(d-1)].astype(float)
        y = data[:, d-1]
        y[y != "van"] = '0'
        y[y == "van"] = '1'
        y = y.astype(int)
    elif dataset in ['wdbc']:
        data, n, d = loadCsv('/datasets/wdbc.dat')
        X = data[:, np.arange(d-1)].astype(float)
        y = data[:, d-1]
        y[y != 'M'] = '0'
        y[y == 'M'] = '1'
        y = y.astype(int)
    elif dataset in ['wine']:
        data = pd.read_csv("/datasets/wine.data", header=None)
        y = np.array([1 if elt == 1 else 0 for elt in data[0]])
        X = np.array(data.drop([0], axis=1))
    elif dataset in ['wine4']:
        data = pd.read_csv("/datasets/winequality-red.csv", sep=';'
)
        y = np.array([1 if elt in [4] else 0 for elt in data.qualit
y])
        X = np.array(data.drop(["quality"], axis=1))
    elif dataset in ['yeast3', 'yeast6']:
        data = pd.read_csv("/datasets/yeast.data", header=None, sep
=r'\s+')
        data = data.drop([0], axis=1)
        if dataset == 'yeast3':
            y = np.array([1 if elt == 'ME3' else 0 for elt in data[
9]])
        elif dataset == 'yeast6':
            y = np.array([1 if elt == 'EXC' else 0 for elt in data[
9]])
        X = np.array(data.drop([9], axis=1))
    return X, y
```

2.1 Approches non-paramétriques

Description des données

Dans cette partie, nous décrivons brièvement le jeu de données "Vehicle" sur lequel nous allons tout d'abord appliquer le modèle des k plus proches voisins.

In [ ]:
```
data = pd.read_csv("/datasets/vehicle.data", sep=';')
data.info
```

Out[ ]:
```
<bound method DataFrame.info of     95,48,83,178,72,10,162,42,20,1
59,176,379,184,70,6,16,187,197,van
0    91,41,84,141,57,9,149,45,19,143,170,330,158,72...
1    104,50,106,209,66,10,207,32,23,158,223,635,220...
2    93,41,82,159,63,9,144,46,19,143,160,309,127,63...
3    85,44,70,205,103,52,149,45,19,144,241,325,188,...
4    107,57,106,172,50,6,255,26,28,169,280,957,264,...
..                                                 ...
840  93,39,87,183,64,8,169,40,20,134,200,422,149,72...
841  89,46,84,163,66,11,159,43,20,159,173,368,176,7...
842  106,54,101,222,67,12,222,30,25,173,228,721,200...
843  86,36,78,146,58,7,135,50,18,124,155,270,148,66...
844  85,36,66,123,55,5,120,56,17,128,140,212,131,73...

[845 rows x 1 columns]>
```

In [ ]:
```
data.head
```

Out[ ]:
```
<bound method NDFrame.head of     95,48,83,178,72,10,162,42,20,159
,176,379,184,70,6,16,187,197,van
0    91,41,84,141,57,9,149,45,19,143,170,330,158,72...
1    104,50,106,209,66,10,207,32,23,158,223,635,220...
2    93,41,82,159,63,9,144,46,19,143,160,309,127,63...
3    85,44,70,205,103,52,149,45,19,144,241,325,188,...
4    107,57,106,172,50,6,255,26,28,169,280,957,264,...
..                                                 ...
840  93,39,87,183,64,8,169,40,20,134,200,422,149,72...
841  89,46,84,163,66,11,159,43,20,159,173,368,176,7...
842  106,54,101,222,67,12,222,30,25,173,228,721,200...
843  86,36,78,146,58,7,135,50,18,124,155,270,148,66...
844  85,36,66,123,55,5,120,56,17,128,140,212,131,73...

[845 rows x 1 columns]>
```

In [ ]:
```
data.describe()
```

Out[ ]:

	95,48,83,178,72,10,162,42,20,159,176,379,184,70,6,16,187,197,van
count	845
unique	845
top	91,41,84,141,57,9,149,45,19,143,170,330,158,72...
freq	1

K plus proches voisins

Algorithme de type non supervisé. Il crée une partition en k clusters homogènes. Il s'agit, en outre, d'un algorithme de classification basé sur la règle de Bayes, une donnée est prédite comme appartenant à la classe $c^\star$ si $c^\star$ =argmax pk(y=c|x). La classe assignée à un nouvel exemple xi va dépendre de son positionnement dans l'espace des données par rapport aux données de l'ensemble d'entraînement. C'est un algorithme simple à implémenter et qui peut parfois fournir de très bons résultats.

Cependant, cette méthode présente rapidement quelques limites : • en grande dimension, tous les exemples seront rapidement eloignés (l'espace est très rapidement vide) • un temps de calcul qui augmente linéairement avec le nombre d'exemples

De plus, Le choix de k doit être défini en avance sans aide La solution est fortement dépendante de k et peut beaucoup changer son interprétation

Pour cela, nous allons tester l'algorithme pour différentes valeurs de k. Ainsi nous pourrons voir quelles sont les valeur de k qui maximisent le score de classification sur le jeu de donnée 'vehicle'.

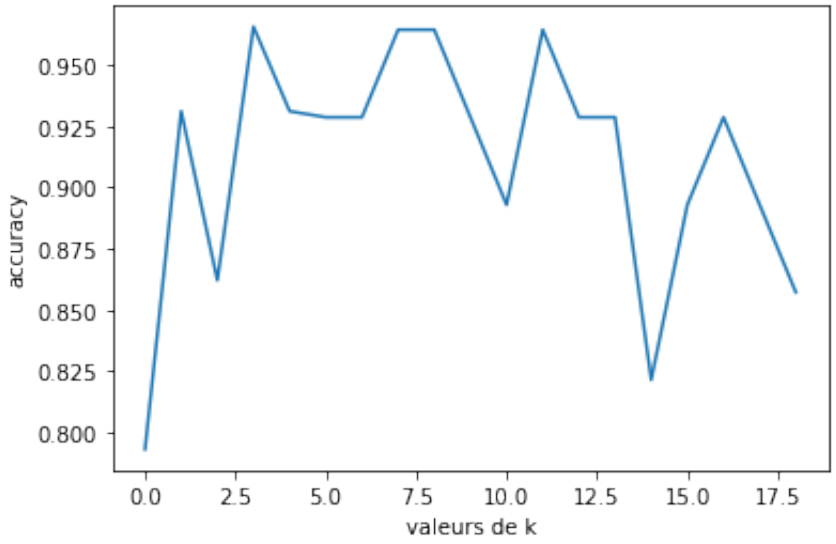
In [ ]:
```
#Partition train et test
X, y = data_recovery('vehicle')
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.33, random_state=42)
X_train


from sklearn.neighbors import KNeighborsClassifier

#K plus proches voisins

n_groupe=20
kmax=20

score={}
scoreliste=[]
res2=[]

skf=StratifiedKFold(n_groupe)
split_train_valid= list(skf.split(X_train,y_train))


for k in range(1,kmax):
    ilearn, ivalid = split_train_valid[k]
    xlearn = X_train[ilearn]
    ylearn = y_train[ilearn]
    xvalid = X_train[ivalid]
    yvalid = y_train[ivalid]

    KNN = KNeighborsClassifier(k)
    fit= KNN.fit(xlearn, ylearn)

    predict= KNN.predict(xvalid)
    score[k]=accuracy_score(predict,yvalid)
    scoreliste.append(accuracy_score(predict,yvalid))

    print("k: ", k, "score: " ,score[k])
    res= metrics.confusion_matrix(yvalid, predict)
    print(res)
```

```
k:  1 score:  0.7931034482758621
[[18  4]
 [ 2  5]]
k:  2 score:  0.9310344827586207
[[22  0]
 [ 2  5]]
k:  3 score:  0.8620689655172413
[[19  3]
 [ 1  6]]
k:  4 score:  0.9655172413793104
[[22  0]
 [ 1  6]]
k:  5 score:  0.9310344827586207
```

```
[[20  2]
 [ 0  7]]
k:  6 score:  0.9285714285714286
[[22  0]
 [ 2  4]]
k:  7 score:  0.9285714285714286
[[21  1]
 [ 1  5]]
k:  8 score:  0.9642857142857143
[[22  0]
 [ 1  5]]
k:  9 score:  0.9642857142857143
[[21  1]
 [ 0  6]]
k:  10 score:  0.9285714285714286
[[20  1]
 [ 1  6]]
k:  11 score:  0.8928571428571429
[[19  2]
 [ 1  6]]
k:  12 score:  0.9642857142857143
[[20  1]
 [ 0  7]]
k:  13 score:  0.9285714285714286
[[19  2]
 [ 0  7]]
k:  14 score:  0.9285714285714286
[[20  1]
 [ 1  6]]
k:  15 score:  0.8214285714285714
[[18  3]
 [ 2  5]]
k:  16 score:  0.8928571428571429
[[21  0]
 [ 3  4]]
k:  17 score:  0.9285714285714286
[[21  0]
 [ 2  5]]
k:  18 score:  0.8928571428571429
[[20  1]
 [ 2  5]]
k:  19 score:  0.8571428571428571
[[18  3]
 [ 1  6]]
```

Les scores de prédiction sont globalement très bons. Nous remarquons sur le jeux de données 'vehicle', que c'est pour k=3 que le score de prédiction est maximisé (pour les valeurs de k allant jusqu'a 20) :

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(scoreliste)
plt.ylabel('accuracy')
plt.xlabel('valeurs de k')
plt.show()
```



SMOTE

Même si nous nous apercevons que les scores sont globalement bons, les données semblent déséquilibrés. Une classe est sur représenter par rapport à l'autre. Afin de contrer cela nous allons utilisés la méthode SMOTE. Celle- ci permet d'augmenter le nombre d'individus de classe minoritaire est pour qu'ils aient plus d'importance lors de la modélisation. Le SMOTE est une méthode de suréchantillonnage des observations minoritaires. Pour éviter de réaliser un simple clonage des individus minoritaires, le SMOTE se base sur un principe simple : générer de nouveaux individus minoritaires qui ressemblent aux autres, sans être strictement identiques. Cela permet de densifier de façon plus homogène la population d'individus minoritaires.

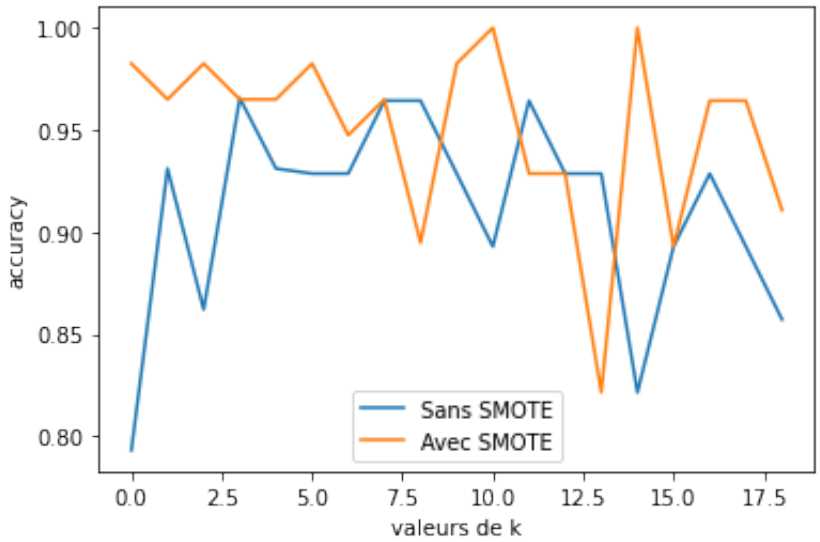
In [ ]:
```
# Importation du package
from imblearn.over_sampling import SMOTE

# Définition de l'instance SMOTE
sm = SMOTE(k_neighbors=3, sampling_strategy=0.75)

# Application du SMOTE aux données
X_train, y_train = sm.fit_resample(X, y)#le smote est appliqué aux
données d'apprentissage uniquement


from sklearn.neighbors import KNeighborsClassifier

#Reproduction desK plus proches voisins

score={}
scorelisteSMOTE=[]


skf=StratifiedKFold(n_groupe)
split_train_valid= list(skf.split(X_train,y_train))


for k in range(1,kmax):
    ilearn, ivalid = split_train_valid[k]
    xlearn = X_train[ilearn]
    ylearn = y_train[ilearn]
    xvalid = X_train[ivalid]
    yvalid = y_train[ivalid]

    KNN = KNeighborsClassifier(k)
    fit= KNN.fit(xlearn, ylearn)

    predict= KNN.predict(xvalid)
    score[k]=accuracy_score(predict,yvalid)
    scorelisteSMOTE.append(accuracy_score(predict,yvalid))

    print("k: ", k, "score: " ,score[k])
    res= metrics.confusion_matrix(yvalid, predict)
    print(res)
```

```
k:  1 score:  0.9824561403508771
[[31  1]
 [ 0 25]]
k:  2 score:  0.9649122807017544
[[30  2]
 [ 0 25]]
k:  3 score:  0.9824561403508771
[[31  1]
 [ 0 25]]
k:  4 score:  0.9649122807017544
[[30  2]
```

```
 [ 0 25]]
k:  5 score:  0.9649122807017544
[[31  2]
 [ 0 24]]
k:  6 score:  0.9824561403508771
[[32  1]
 [ 0 24]]
k:  7 score:  0.9473684210526315
[[30  3]
 [ 0 24]]
k:  8 score:  0.9649122807017544
[[31  2]
 [ 0 24]]
k:  9 score:  0.8947368421052632
[[27  6]
 [ 0 24]]
k:  10 score:  0.9824561403508771
[[33  0]
 [ 1 23]]
k:  11 score:  1.0
[[33  0]
 [ 0 24]]
k:  12 score:  0.9285714285714286
[[29  3]
 [ 1 23]]
k:  13 score:  0.9285714285714286
[[28  4]
 [ 0 24]]
k:  14 score:  0.8214285714285714
[[22 10]
 [ 0 24]]
k:  15 score:  1.0
[[32  0]
 [ 0 24]]
k:  16 score:  0.8928571428571429
[[30  2]
 [ 4 20]]
k:  17 score:  0.9642857142857143
[[30  2]
 [ 0 24]]
k:  18 score:  0.9642857142857143
[[30  2]
 [ 0 24]]
k:  19 score:  0.9107142857142857
[[28  4]
 [ 1 23]]
```

Nous remarquons que l'utilisation de SMOTE améliore globalemnt les prédictions :

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(scoreliste, label='Sans SMOTE')
plt.plot(scorelisteSMOTE, label='Avec SMOTE')
plt.ylabel('accuracy')
plt.xlabel('valeurs de k')
plt.legend()
plt.show()
```

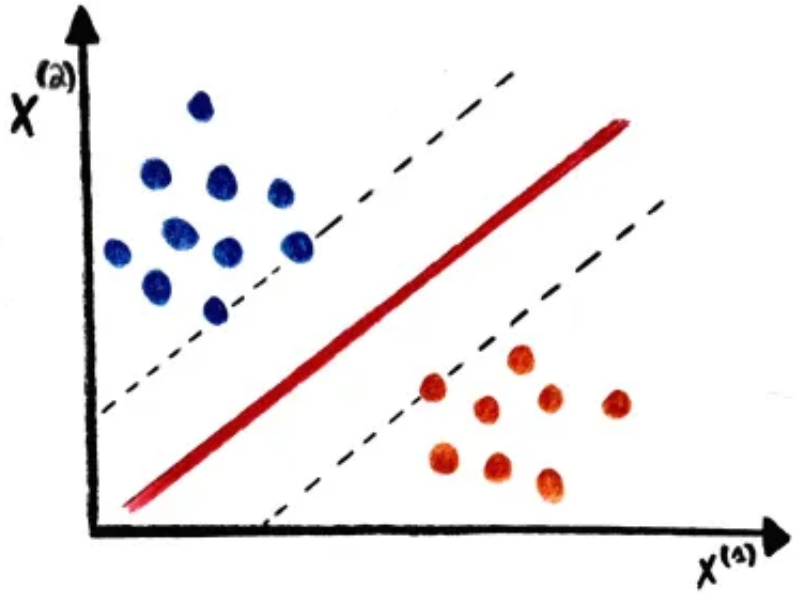


2.2 approches paramétriques linéaires

SVM

Dans cette partie, nous nous intéressons au SVM (Support Vector Machines). Un Support Vector Machines (SVM) est un modèle de machine learning très puissant et polyvalent, capable d'effectuer une classification linéaire, non linéaire ou une régression. C'est l'un des modèles les plus populaires de l'apprentissage automatique.

Le SVM est un modèle d'apprentissage automatique supervisé qui est principalement utilisé pour les classifications (mais il peut aussi être utilisé pour la régression). L'intuition derrière les Support Vector Machines est de séparer des données en les délimitant (créer des frontières) afin de créer des groupes.



En d’autres termes, les SVM visent à résoudre les problèmes de classification en trouvant de bonnes frontières de décision entre deux ensembles de points appartenant à deux catégories différentes. Une frontière de décision peut être considérée comme une ligne ou une surface séparant les données d’apprentissage en deux espaces correspondant à deux catégories. Pour classer de nouveaux points de données, il suffit de vérifier de quel côté de la frontière de décision ils se trouvent.

Les SVM procèdent à la recherche de ces frontières en deux étapes. Tout d'abord, les données sont mises en correspondance avec une nouvelle représentation à haute dimension où la frontière de décision peut être exprimée sous la forme d’un hyperplan.

Une bonne limite de décision est calculée en essayant de maximiser la distance entre l’hyperplan et les points de données les plus proches de chaque classe, une étape appelée maximisation de la marge. Cela permet à la frontière de bien s’adapter à de nouveaux échantillons en dehors de l’ensemble de données d’apprentissage.

Cette technique utilisée par les Support Vector Machines est appelée "kernel trick". Elle permet de transformer les données, puis, sur la base de ces transformations, de trouver une limite optimale entre les résultats possibles. En d’autres termes, il effectue des transformations de données extrêmement complexes, puis détermine comment séparer les données en fonction des labels définis auparavant.

In [ ]:
```
# Import du data set et des libs
%matplotlib inline
from sklearn import datasets
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
# Chargement du jeu de donnees
wine_dataset = pd.read_csv("/datasets/winequality-red.csv", sep=';'
)

print(wine_dataset)
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0               7.4             0.700         0.00             1.9
0.076
1               7.8             0.880         0.00             2.6
0.098
2               7.8             0.760         0.04             2.3
0.092
3              11.2             0.280         0.56             1.9
0.075
4               7.4             0.700         0.00             1.9
0.076
...             ...               ...          ...             ...
...
1594            6.2             0.600         0.08             2.0
0.090
1595            5.9             0.550         0.10             2.2
0.062
1596            6.3             0.510         0.13             2.3
0.076
1597            5.9             0.645         0.12             2.0
0.075
1598            6.0             0.310         0.47             3.6
0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  su
lphates  \
0                    11.0                  34.0  0.99780  3.51
0.56
1                    25.0                  67.0  0.99680  3.20
0.68
2                    15.0                  54.0  0.99700  3.26
0.65
3                    17.0                  60.0  0.99800  3.16
0.58
4                    11.0                  34.0  0.99780  3.51
0.56
...                   ...                   ...      ...   ...
...
1594                 32.0                  44.0  0.99490  3.45
```

```
0.58
1595                 39.0                 51.0  0.99512  3.52
0.76
1596                 29.0                 40.0  0.99574  3.42
0.75
1597                 32.0                 44.0  0.99547  3.57
0.71
1598                 18.0                 42.0  0.99549  3.39
0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
...       ...      ...
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6

[1599 rows x 12 columns]
```

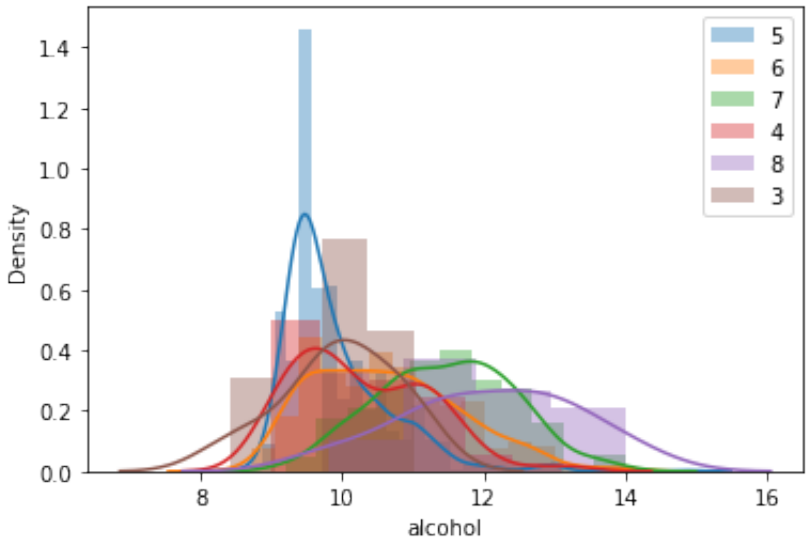
In [ ]:
```
wine_dataset.quality.value_counts()
```

Out[ ]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [ ]:
```
for i in wine_dataset.quality.unique():
    sns.distplot(wine_dataset['alcohol'][wine_dataset.quality==i],
                 kde=1,label='{}'.format(i))

plt.legend()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:26
19: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `h
istplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:26
19: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `h
istplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:26
19: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `h
istplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:26
19: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `h
istplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:26
19: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `h
istplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:26
19: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `h
istplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7ff8de38cc40>
```



On s'aperçoit que les distributions semblent plutôt normales et peuvent être classées en 6 groupes. On peut également s'amuser à analyser cette distribution en fonction d'autres caractéristiques.

Nous pouvons à présent créer notre SVM dans sklearn. Cela consiste à créer un objet SVC (support vector classificator). L'un des paramètres importants est le noyau 'kernel'. Il s'agit d'une fonction qui sert à transformer les données dans une représentation spécifique.

Les SVM utilisent différents types de fonctions noyau. Ces fonctions sont de différents types, par exemple, linéaire, non linéaire, polynomiale, fonction de base radiale (RBF) et sigmoïde.

Il faut donc avoir une attention particulière sur ce paramètre. Dans cette partie nous nous intéressons aux approches paramétriques linéaires, nous choisirons donc un kernel linéaire !

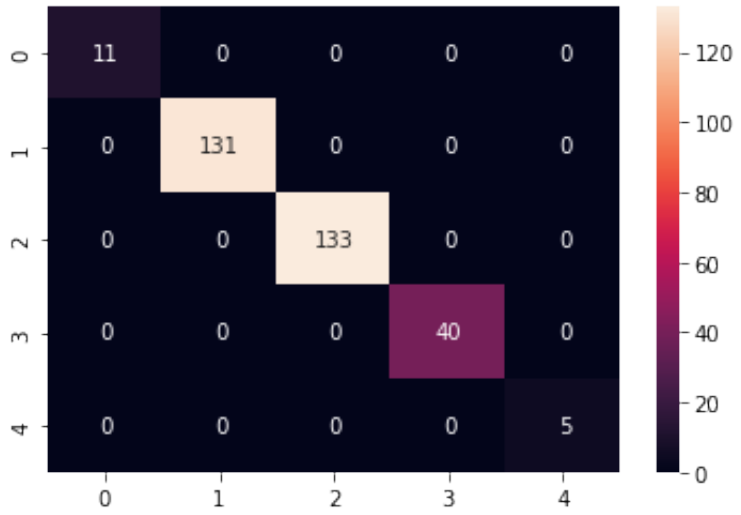
In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = \
    train_test_split(wine_dataset,wine_dataset['quality'],
                     test_size=0.2)
```

In [ ]:
```
# Fit du  Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
# Prediction sur le Test set
y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           4       1.00      1.00      1.00        11
           5       1.00      1.00      1.00       131
           6       1.00      1.00      1.00       133
           7       1.00      1.00      1.00        40
           8       1.00      1.00      1.00         5

    accuracy                           1.00       320
   macro avg       1.00      1.00      1.00       320
weighted avg       1.00      1.00      1.00       320
```

In [ ]:
```
# Matrice de confusion
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm,annot=True,fmt='2.0f')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff8db58a6a0>



Conclusion : les Support Vector Machines qui sont des modèles très intéressants dont les principaux points forts sont les suivants

- L'efficacité dans les espaces à dimension élevés
- L'efficacité dans les cas où le nombre de dimensions est supérieur au nombre d'échantillons.
- L'utilisation d'un sous-ensemble de points d'apprentissage dans la fonction de décision (appelés vecteurs de support), ce qui la rend également efficace en termes de mémoire.
- La flexibilité : différentes fonctions de noyau peuvent être spécifiées pour la fonction de décision.

Cependant, il faut faire attention au cas particulier où le nombre de variables est beaucoup plus grand que le nombre d'échantillons, car un mauvais choix des fonctions de noyau peut entrainer l'over-fitting.

Regression logistique

La régression logistique est un modèle de classification linéaire probabiliste paramétrique comme la régression linaire mais pour prédire une probabilité d'appartenance à une catégorie ou non, grâce à une fonction d'interpolation logistique logit. Utile pour prédire la probabilité d'une maladie par exemple en fonction des caractéristiques du patient ou la probabilité d'une panne en fonction des événements.

Avantages :

- La prédiction est très rapide car dépend uniquement d'une fonction linéaire
- Peu susceptible au surapprentissage
- Interpréter les différents paramètres lié à chaque feature est facile

Inconvénients :

- De fait de sa linéarité, cela empêche les interactions entre variables
- La phase d'apprentissage est longue car l'optimisation des paramètres est coûteuse.

In [ ]:
```
from sklearn.linear_model import LogisticRegression # import de cla
sse LogisticRegression qui permet d'en entrainer un
```

Pour commencer, nous allons tester sur l'ensemble des jeux de données la régression logistique pour identiier rapidement lesquels obtiennent les meilleures performances.

In [ ]:
```
datasets =['abalone8','abalone17','abalone20','autompg','australian
','balance','bupa','german','glass','hayes','heart','iono','libras'
,'newthyroid','pageblocks',
'pima','satimage','segmentation','sonar','spambase','splice','vehic
le','wdbc','wine','wine4','yeast3', 'yeast6']
```

In [ ]:
```
for dataset in datasets:
  X, y = data_recovery(dataset)
  print(dataset)

  X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
ze=0.3, random_state=0)
  model = LogisticRegression(max_iter=5000) # construction d'un obj
et de Régression logistique
  model.fit(X_train, y_train)
  print(model.__class__.__name__, model.score(X_test, y_test))
```

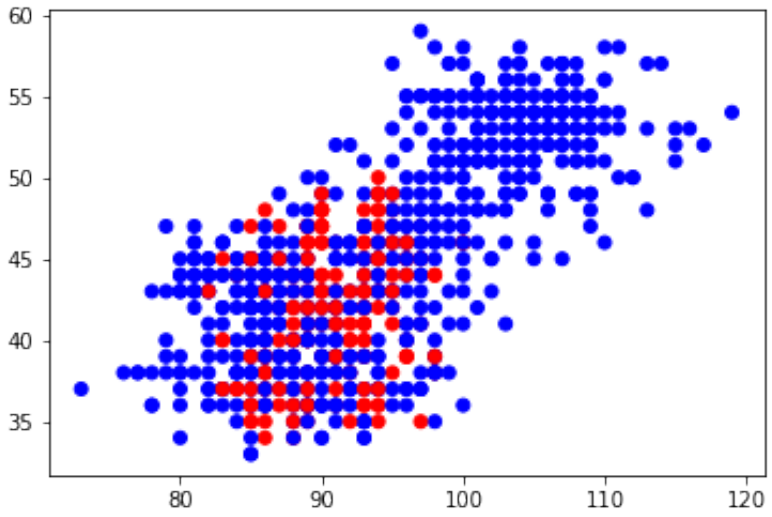
```
abalone8
LogisticRegression 0.8692185007974481
abalone17
LogisticRegression 0.9864433811802232
abalone20
LogisticRegression 0.992822966507177
autompg
LogisticRegression 0.9067796610169492
australian
LogisticRegression 0.855072463768116
balance
LogisticRegression 0.9361702127659575
bupa
LogisticRegression 0.6826923076923077
german
LogisticRegression 0.7766666666666666
glass
LogisticRegression 0.676923076923077
hayes
LogisticRegression 0.925
heart
LogisticRegression 0.8148148148148148
iono
LogisticRegression 0.8867924528301887
libras
LogisticRegression 0.9444444444444444
newthyroid
LogisticRegression 0.9230769230769231
pageblocks
LogisticRegression 0.9494518879415347
pima
LogisticRegression 0.7792207792207793
satimage
LogisticRegression 0.8933195235629208
segmentation
LogisticRegression 0.9105339105339105
sonar
LogisticRegression 0.8095238095238095
spambase
LogisticRegression 0.9239130434782609
splice
LogisticRegression 0.8562434417628542
```

```
vehicle
LogisticRegression 0.9881889763779528
wdbc
LogisticRegression 0.9590643274853801
wine
LogisticRegression 0.9814814814814815
wine4
LogisticRegression 0.96875
yeast3
LogisticRegression 0.9103139013452914
yeast6
LogisticRegression 0.9753363228699552
```

Le jeu de données "vehicle" que nous avons utilisé précedemment lors de la première partie obient de bonnes performances. Regardons de plus près :

In [ ]:
```
X, y = data_recovery('vehicle')
plt.scatter(X[:,0], X[:, 1], c=y, cmap='bwr')
```

Out[ ]: <matplotlib.collections.PathCollection at 0x7faf407cf520>



In [ ]:
```
model = LogisticRegression(max_iter=500) # construction d'un objet de Régression logistique
```

In [ ]:
```
%time model.fit(X, y) # Entrainement du modèle
# la fonction %time est utilisé pour calculer le temps pris pour en trainer le modèle
```

```
CPU times: user 243 ms, sys: 170 ms, total: 413 ms
Wall time: 233 ms

/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logis
tic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1
):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver opti
ons:
    https://scikit-learn.org/stable/modules/linear_model.html#logi
stic-regression
  n_iter_i = _check_optimize_result(
```

Out[ ]:
```
LogisticRegression(max_iter=500)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Score

In [ ]:
```
model.score(X_test, y_test)
```

Out[ ]:
```
0.9892857142857143
```

Matrices de confusion

In [ ]:
```
y_pred=model.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
print(cm)
res2 = metrics.classification_report(y_test, y_pred)
print(res2)
```

```
[[216   1]
 [  2  61]]
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       217
           1       0.98      0.97      0.98        63

    accuracy                           0.99       280
   macro avg       0.99      0.98      0.98       280
weighted avg       0.99      0.99      0.99       280
```

2.3 Approche non linéaire

Dans cette dernière partie, on va chercher à tester les approches non linéaires ou par boosting suivantes :

Arbre de décision

Il est utilisé pour représenter visuellement et explicitement les décisions et la prise de décision pour des problèmes de classification ainsi que pour des problèmes de régression. Il représente aussi l'élément de base de plusieurs modèles comme le Random Forest. Ils consistent en une série de règles (souvent binaires) qui vont permettre de successivement séparer le jeu de donn ées en deux. La nature de l'arbre d´epend de la nature de l'espace Y • quand Y ⊂ R, on parle d'arbre de régression • quand Y = {0, 1, . . . , C }, on parle d'arbre de classification

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
```

Random Forest

Random Forest est ce qu'on appelle une méthode d'ensemble ou combine des résultats afin de maximiser ses résultats. Comme dit précedemment, le random forest est composé de plusieurs arbres de décision, entrainés de manière indépendante sur des sous-ensembles du data set d'apprentissage. Chacun produit une estimation, et c'est la combinaison des résultats qui va donner la prédiction finale. C'est donc une technique de bagging. Le Bagging est une technique qui consiste à assembler un grand nombre d'algorithmes avec de faibles performances individuelles pour en créer un beaucoup plus efficace. Les algorithmes de faible performance sont appelés les « weak learners » et le résultat obtenu « strong learner ».

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

Adaboost

Adaboost est un algotithme de boosting. Il se base sur le même principe que le bagging sauf qu'il entraîne l'un après l'autre différents modèles en leur demandant de corriger les erreurs effectuer par les modèles précédents.

In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier
```

Gradient Boosting

L'algorithme de Gradient Boosting a beaucoup de points communs avec Adaboost. Tout comme Adaboost, il s'agit d'un ensemble de « weak learners », créés les uns après les autres, formant un « strong learners ». De plus, chaque « weak learners » est entraîné pour corriger les erreurs des « weak learners » précédents. Néanmoins, contrairement à Adaboost, les « apprenants faibles » ont tous autant de poids dans le système de votation, peu importe leur performance.

In [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

Test des méthodes sur les données

Nous allons, à présent tester ces différentes méthodes sur les jeux de données mis à notre disposition. On pourra avoir un aperçu des méthodes les plus efficaces sur chacuns des jeux de données. Bien-sûr ces résultats sont améliorables en jouant avec les hyper paramètres et certaines méthodes s'appliqueront mal à certains jeux de données. Mais cela pourra nous donner une idée des méthodes les plus intéressantes pour chque jeux de données. Pour cela, nous allons une nouvelle fois utiliser pour créer et évaluer les différent modèles.

In [ ]:
```
ADA =[]
DT =[]
GB =[]
RF = []

for dataset in datasets:
  X, y = data_recovery(dataset)
  print(dataset)
  #plt.scatter(X[:,0], X[:,1], c=y, alpha=0.8)#A voir

  X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
ze=0.3, random_state=0)
  modelADA = AdaBoostClassifier(n_estimators=100)
  modelDT = DecisionTreeClassifier(random_state=0)
  modelGB = GradientBoostingClassifier(n_estimators=100)
  modelRF = RandomForestClassifier(n_estimators=100)

  for model in (modelADA, modelDT, modelGB, modelRF):
    model.fit(X_train, y_train)
    print(model.__class__.__name__, model.score(X_test, y_test))

  ADA.append(modelADA.score(X_test, y_test))
  DT.append(modelDT.score(X_test, y_test))
  GB.append(modelGB.score(X_test, y_test))
  RF.append(modelRF.score(X_test, y_test))
```

```
abalone8
AdaBoostClassifier 0.8628389154704944
DecisionTreeClassifier 0.7854864433811802
GradientBoostingClassifier 0.8636363636363636
RandomForestClassifier 0.861244019138756
abalone17
AdaBoostClassifier 0.9824561403508771
DecisionTreeClassifier 0.9696969696969697
GradientBoostingClassifier 0.9840510366826156
RandomForestClassifier 0.9864433811802232
abalone20
AdaBoostClassifier 0.9912280701754386
DecisionTreeClassifier 0.9848484848484849
GradientBoostingClassifier 0.9864433811802232
RandomForestClassifier 0.992822966507177
autompg
AdaBoostClassifier 0.8559322033898306
DecisionTreeClassifier 0.8983050847457628
```

```
GradientBoostingClassifier 0.940677966101695
RandomForestClassifier 0.8813559322033898
australian
AdaBoostClassifier 0.821256038647343
DecisionTreeClassifier 0.8164251207729468
GradientBoostingClassifier 0.8454106280193237
RandomForestClassifier 0.855072463768116
balance
AdaBoostClassifier 0.9893617021276596
DecisionTreeClassifier 0.8351063829787234
GradientBoostingClassifier 0.8882978723404256
RandomForestClassifier 0.8936170212765957
bupa
AdaBoostClassifier 0.6634615384615384
DecisionTreeClassifier 0.6346153846153846
GradientBoostingClassifier 0.7307692307692307
RandomForestClassifier 0.7307692307692307
german
AdaBoostClassifier 0.77
DecisionTreeClassifier 0.6733333333333333
GradientBoostingClassifier 0.7633333333333333
RandomForestClassifier 0.7766666666666666
glass
AdaBoostClassifier 0.8153846153846154
DecisionTreeClassifier 0.7692307692307693
GradientBoostingClassifier 0.8153846153846154
RandomForestClassifier 0.7692307692307693
hayes
AdaBoostClassifier 1.0
DecisionTreeClassifier 1.0
GradientBoostingClassifier 1.0
RandomForestClassifier 1.0
heart
AdaBoostClassifier 0.7160493827160493
DecisionTreeClassifier 0.8024691358024691
GradientBoostingClassifier 0.7901234567901234
RandomForestClassifier 0.8148148148148148
iono
AdaBoostClassifier 0.9245283018867925
DecisionTreeClassifier 0.8962264150943396
GradientBoostingClassifier 0.9339622641509434
RandomForestClassifier 0.9433962264150944
libras
AdaBoostClassifier 0.9629629629629629
DecisionTreeClassifier 0.9444444444444444
GradientBoostingClassifier 0.9629629629629629
RandomForestClassifier 0.9444444444444444
newthyroid
AdaBoostClassifier 0.9846153846153847
DecisionTreeClassifier 0.9692307692307692
GradientBoostingClassifier 1.0
RandomForestClassifier 1.0
pageblocks
AdaBoostClassifier 0.9683313032886723
```

```
DecisionTreeClassifier 0.97442143727162
GradientBoostingClassifier 0.9774665042630938
RandomForestClassifier 0.9786845310596833
pima
AdaBoostClassifier 0.7532467532467533
DecisionTreeClassifier 0.7229437229437229
GradientBoostingClassifier 0.7878787878787878
RandomForestClassifier 0.7792207792207793
satimage
AdaBoostClassifier 0.9269808389435525
DecisionTreeClassifier 0.9047125841532885
GradientBoostingClassifier 0.9311237700673226
RandomForestClassifier 0.9388917659243915
segmentation
AdaBoostClassifier 0.9653679653679653
DecisionTreeClassifier 0.9725829725829725
GradientBoostingClassifier 0.9797979797979798
RandomForestClassifier 0.9797979797979798
sonar
AdaBoostClassifier 0.8253968253968254
DecisionTreeClassifier 0.746031746031746
GradientBoostingClassifier 0.8253968253968254
RandomForestClassifier 0.8888888888888888
spambase
AdaBoostClassifier 0.9427536231884058
DecisionTreeClassifier 0.9065217391304348
GradientBoostingClassifier 0.946376811594203
RandomForestClassifier 0.9514492753623188
splice
AdaBoostClassifier 0.9422875131164743
DecisionTreeClassifier 0.9359916054564533
GradientBoostingClassifier 0.9737670514165793
RandomForestClassifier 0.974816369359916
vehicle
AdaBoostClassifier 0.968503937007874
DecisionTreeClassifier 0.9015748031496063
GradientBoostingClassifier 0.9488188976377953
RandomForestClassifier 0.9606299212598425
wdbc
AdaBoostClassifier 0.9766081871345029
DecisionTreeClassifier 0.9064327485380117
GradientBoostingClassifier 0.9766081871345029
RandomForestClassifier 0.9707602339181286
wine
AdaBoostClassifier 0.9444444444444444
DecisionTreeClassifier 0.9629629629629629
GradientBoostingClassifier 0.9814814814814815
RandomForestClassifier 1.0
wine4
AdaBoostClassifier 0.9625
DecisionTreeClassifier 0.94375
GradientBoostingClassifier 0.9604166666666667
RandomForestClassifier 0.9666666666666667
```

```
yeast3
AdaBoostClassifier 0.9484304932735426
DecisionTreeClassifier 0.9417040358744395
GradientBoostingClassifier 0.9551569506726457
RandomForestClassifier 0.9551569506726457
yeast6
AdaBoostClassifier 0.968609865470852
DecisionTreeClassifier 0.968609865470852
GradientBoostingClassifier 0.9775784753363229
RandomForestClassifier 0.9820627802690582
```

In [ ]:
```
print("La moyenne de l'arbre de décision :",np.mean(DT))
print("La moyenne des Random Forrest :",np.mean(RF))
print("La moyenne de Adaboost :",np.mean(ADA))
print("La moyenne du Gradient Boosting :",np.mean(GB))
```

```
La moyenne de l'arbre de décision : 0.8802836652496923
La moyenne des Random Forrest : 0.9176631140302065
La moyenne de Adaboost : 0.904945815039587
La moyenne du Gradient Boosting : 0.9158119074331876
```

Nous remarquons que les méthodes d'ensembles (Bagging : Random Forrest 0.917 et Boosting : Adaboost 0.904, Gradiet Boosting 0.915) obtiennent de meilleures scores en moyennes que la méthode des arbres de décision.

Conclusion

Pour finir, passons en revue les algorithmes que nous avons étudié au cours de ce projet.

Dans le cas d'apprentissage supervisé non paramétrique, nous avons utilisé l'algorithme des K Nearest Neighbor associé à l'algorithme de sur-échantillonnage SMOTE. L'observation est assez nette quand à l'amélioration du modèle une fois le problème des données déséquilibrées résolu.

Dans un second temps, nous avons travaillé sur les approches paramétriques linéaires dans lesquels nous avons pu observé le fonctionnement des algorithmes des Séparateurs à Vaste Marge mais aussi de régression logistique. Dans les deux cas, les modèles selectionnés sont très performants puisqu'on constate une accuracy avoisinnant 1 ainsi qu'un score de 0.989 pour la régression logistique.

Enfin, nous avons vu dans une dernière partie l'approche non linéaire avec cette fois une utilisation des algorithmes de Bagging, tel que celui des Forêts aléatoires (Random Forest), mais aussi de Boosting avec notamment l'algorithme Adaboost et le Gradient Boosting. Ces deux méthodes permettent ainsi de combiner différents modèles pour obtenir un résultat plus performant mais aussi plus robuste.