PROGRAMME 2 – Data Mining

```
# Importation des librairies
import os
import pandas
import numpy
import scipy
import sklearn # Version de sklearn : 0.23.1
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectKBest, chi2
from sklearn import metrics

# Implémentation des parametres
#taille_train = 0.75
taille_test = 0.25
# seuil = 0.05
```

1. Importation et description des données (avant traitement)

```
""" Fonction qui importe les données à partir d'un chemin menant à un
dataframe : Importation_data(dataframe)
 Input :
   - dataframe : le chemin et le nom du dataframe que l'on veut importer
 Output :
   - data : le dataframe importé
   - chemin : le répertoire courant (c'est le répertoire où se trouve le
dataframe)
"""
def Importation_data(dataframe):
    data = pandas.read_table(dataframe, sep="\t", header=0)
    chemin = os.getcwd()
    return data, chemin

# Importation des données et définition du répertoire courant (celui où se
trouve "data_avec_etiquettes.txt")
data_brute, chemin =
Importation_data("/home/elisa/Documents/M1_Info/Semestre_1/Data_Mining/Projet
_Data_Mining/data_avec_etiquettes.txt")
print("Le répertoire courant est : " , chemin)
```

```
Le répertoire courant est :
/home/elisa/Documents/M1_Info/Semestre_1/Data_Mining/Projet_Data_Mining
```

```
data_brute.head(5)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V191	V192	V193	V194	V195	V196	V197	V198	V199	V200
0	0	0	1	5	0	8	0	1	5	1	...	9	1.0	0.0	0.11	0.0	0.0	0.0	0.0	0.0	m12
1	0	0	1	1	1	7	0	1	8	1	...	19	1.0	0.0	0.05	0.0	0.0	0.0	0.0	0.0	m12
2	1	0	0	9	0	2	0	0	3	1	...	29	1.0	0.0	0.03	0.0	0.0	0.0	0.0	0.0	m12
3	0	0	0	5	0	3	1	0	0	0	...	39	1.0	0.0	0.03	0.0	0.0	0.0	0.0	0.0	m12
4	0	1	0	9	1	4	0	1	4	1	...	49	1.0	0.0	0.02	0.0	0.0	0.0	0.0	0.0	m12

5 rows × 200 columns

```
# Dimensions
print("Le fichier (avant traitement) contient " + str(data_brute.shape[0]) +
" lignes et "
      + str(data_brute.shape[1]) + " colonnes.")
```

```
Le fichier (avant traitement) contient 494021 lignes et 200 colonnes.
```

2. Traitement des données

```
# Recodage des variables qualitatives V160, V161 et V162 en quantitatives
data = pandas.get_dummies(data_brute.iloc[:,0:199], prefix=['V160', 'V161',
'V162'])
data = data.join(data_brute["V200"])
data.head(5)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V162_m11	V162_m2	V162_m3	V162_m4	V162_m5	V162_m6	V162_m7	V162_m8	V162_m9	V200
0	0	0	1	5	0	8	0	1	5	1	...	0	0	0	0	0	0	0	0	0	m12
1	0	0	1	1	1	7	0	1	8	1	...	0	0	0	0	0	0	0	0	0	m12
2	1	0	0	9	0	2	0	0	3	1	...	0	0	0	0	0	0	0	0	0	m12
3	0	0	0	5	0	3	1	0	0	0	...	0	0	0	0	0	0	0	0	0	m12
4	0	1	0	9	1	4	0	1	4	1	...	0	0	0	0	0	0	0	0	0	m12

5 rows × 277 columns

```
# Recodage de la variable cible : « m16 » (les positifs)
data.V200 = pandas.get_dummies(data_brute.V200).m16
#data = data.join(data_brute["V200"])
data.head(5)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V162_m11	V162_m2	V162_m3	V162_m4	V162_m5	V162_m6	V162_m7	V162_m8	V162_m9	V200
0	0	0	1	5	0	8	0	1	5	1	...	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	7	0	1	8	1	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	9	0	2	0	0	3	1	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	5	0	3	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	1	0	9	1	4	0	1	4	1	...	0	0	0	0	0	0	0	0	0	0

5 rows × 277 columns

```
# Supprimer les variables sans information (c'est-à-dire nombre de modalité
<= 1)

""" Fonction qui supprime les variables qui ont un nombre de modalité <= 1
car elles n'apportent
 aucune information : suppr_var_sans_information(dataframe)
 Input :
   - dataframe : le nom du dataframe que l'on veut traiter
 Output :
   - data : le dataframe traité
"""

def suppr_var_sans_information(dataframe):
    # Parcourir les colonnes du dataframe :
    for i in dataframe.columns:
        # Si nb de modalité <= 1, on supprime la variable :
        if len(dataframe[i].unique()) <= 1 :
            dataframe.drop([i], axis='columns', inplace=True)
    return dataframe

# Appliquer la fonction au dataframe
data = suppr_var_sans_information(data)
```

3. Description des données (après traitement)

```
# Dimensions
print("Le fichier (après traitement) contient " + str(data.shape[0]) + "
lignes et "
      + str(data.shape[1]) + " colonnes.")
```

```
Le fichier (après traitement) contient 494021 lignes et 275 colonnes.
```

```
# Informations sur le dataframe
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 494021 entries, 0 to 494020
Columns: 275 entries, V1 to V200
dtypes: float64(15), int64(179), uint8(81)
memory usage: 769.4 MB
None
```

4. Subdivision en deux dataset (dataTrain et dataTest)

```
# Subdivision en deux dataset (dataTrain et dataTest)
dataTrain, dataTest = model_selection.train_test_split(data, test_size =
```

```
taille_test, random_state = 0)

# Vérification
print("Dimensions dataTrain :", dataTrain.shape)
print("Dimensions dataTest :", dataTest.shape)
```

```
Dimensions dataTrain : (370515, 275)
Dimensions dataTest : (123506, 275)
```

5. Définir la variable cible (y) et les variables explicatives (X)

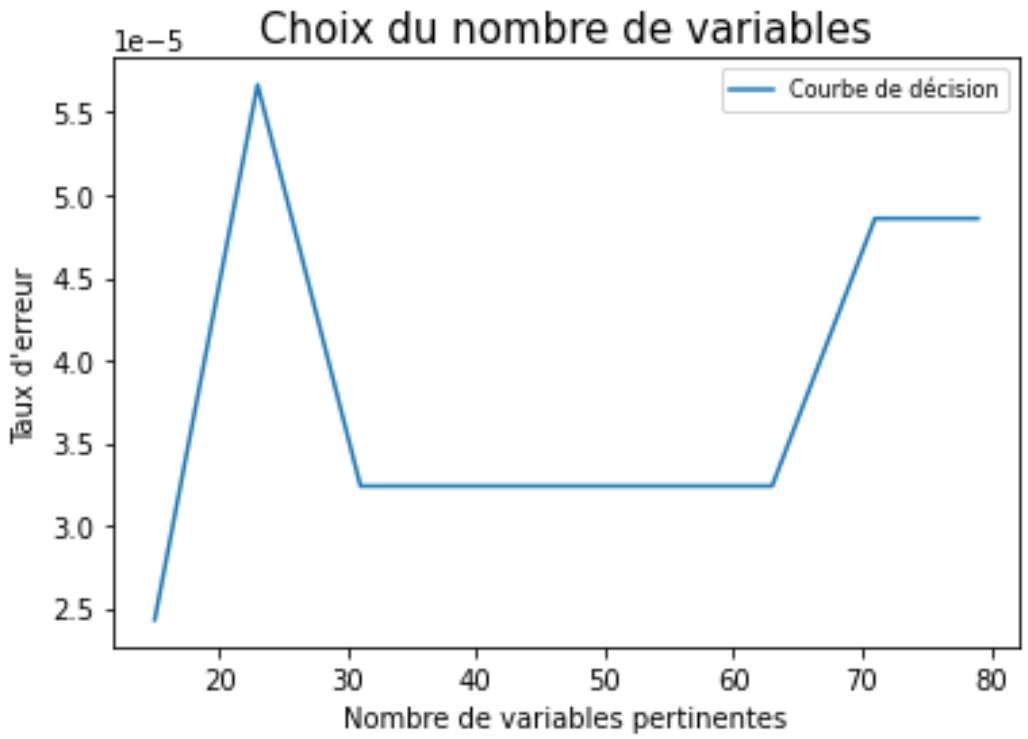
```
""" Fonction pour définir la variable cible (y) et les variables explicatives
(X) : Var_cible_expl(dataframe)
 Input :
   - dataframe : le dataframe à définir
 Output :
   - X : le dataframe des variables explicatives
   - y : la variable cible
"""
def Var_cible_expl(dataframe):
    # On récupère tous les noms de colonnes du dataframe dans une liste col
    col = [i for i in dataframe.columns]

    # On enlève "V200" de la liste col
    col.remove("V200")

    # La colonne "V200" est la variable cible, les autres sont les variables
explicatives
    X = dataframe.loc[:,col]
    y = dataframe.V200
    return X, y

# Pour l'échantillon dataTrain
XTrain, yTrain = Var_cible_expl(dataTrain)
# Pour l'échantillon dataTest
XTest, yTest = Var_cible_expl(dataTest)

# Verification
print("Dimensions XTrain :", XTrain.shape)
print("Dimensions yTrain :", yTrain.shape)

print("Dimensions XTest :", XTest.shape)
print("Dimensions XTest :", yTest.shape)
```

```
Dimensions XTrain : (370515, 274)
Dimensions yTrain : (370515,)
Dimensions XTest : (123506, 274)
Dimensions XTest : (123506,)
```

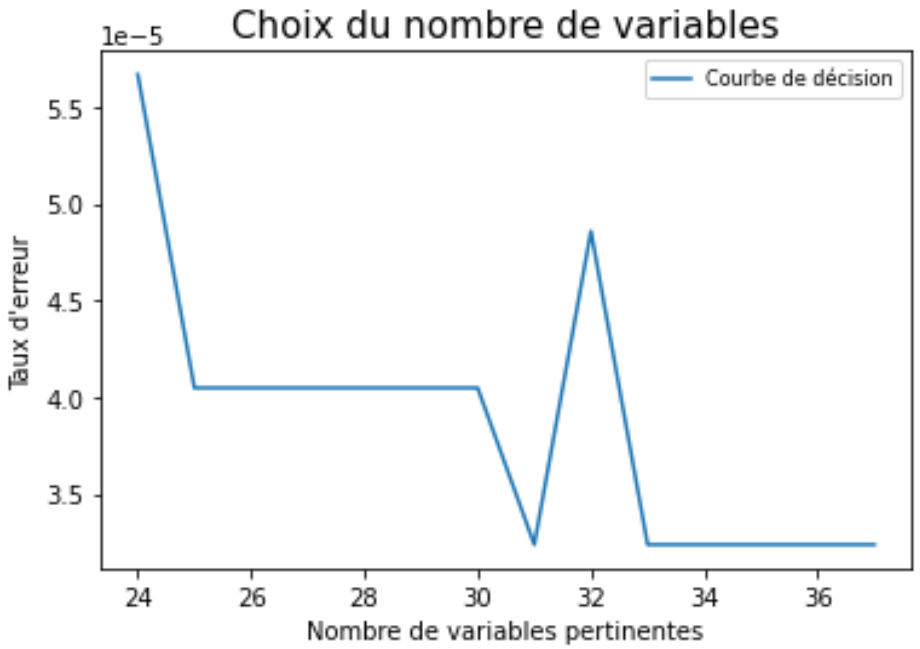
6. Choix du nombre de variables le plus optimal

```
""" Fonction de décision pour optimiser le nombre de variable : optim(from,
to, step)
 Input :
   - Nb_var : Liste qui contient les nombres de variables à tester
 Output :
   - Taux_err : Liste des taux d'erreurs
   - Nb_err : Liste des nombre d'erreurs
   - Graphique : "Choix du nombre de variables pertinentes"
"""
def optim(Nb_var):
    Taux_err = []
    Nb_err = []

    for i in Nb_var:
        # Pour l'échantillon dataTrain
        XTrain, yTrain = Var_cible_expl(dataTrain)
        # Pour l'échantillon dataTest
        XTest, yTest = Var_cible_expl(dataTest)

        # On sélectionne les k variables
        selector = SelectKBest(chi2, k = i)
        selector.fit_transform(XTrain, yTrain)
        selector.get_support()

        # Transformation des données
        XTrain = XTrain.loc[:,selector.get_support()]
        XTest = XTest.loc[:,selector.get_support()]

        # Instanciation
        tree = DecisionTreeClassifier(random_state = 0)

        # Construction du modèle sur les données d'apprentissage
        tree.fit(XTrain,yTrain)

        # Prédiction
        prédictions = tree.predict(XTest)
        #print(prédictions)

        # Matrice de confusion
        mc = pandas.crosstab(yTest, prédictions)
        mc = mc.values # Transformation en matrice numpy (plus facile à
manipuler)

        # Taux d'erreur = (FN + FP) / Total
        Tx_err = (mc[1,0] + mc[0,1]) / numpy.sum(mc)
        #print("Taux d'erreur :", Tx_err)
```

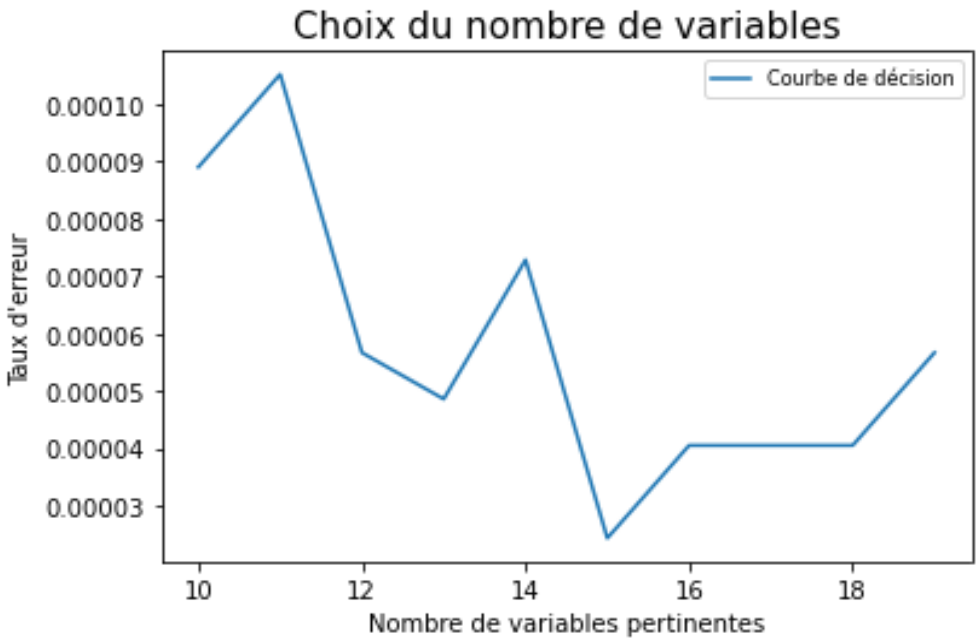
```
        Taux_err.append(Tx_err)

        # Nombre d'erreur sur 4 898 424 obs
        Nb = round(Tx_err * 4898424)
        Nb_err.append(Nb)

    #print(Taux_err)
    plt.plot(Nb_var, Taux_err, label="Courbe de décision")
    plt.xlabel("Nombre de variables pertinentes", fontsize=10)
    plt.ylabel("Taux d'erreur", fontsize=10)
    plt.title("Choix du nombre de variables", fontsize=15)
    plt.legend(loc = 'best', fontsize=8)
    plt.show()


# Choix du nombre de variables pertinentes
optim(range(15, 80, 8))
```



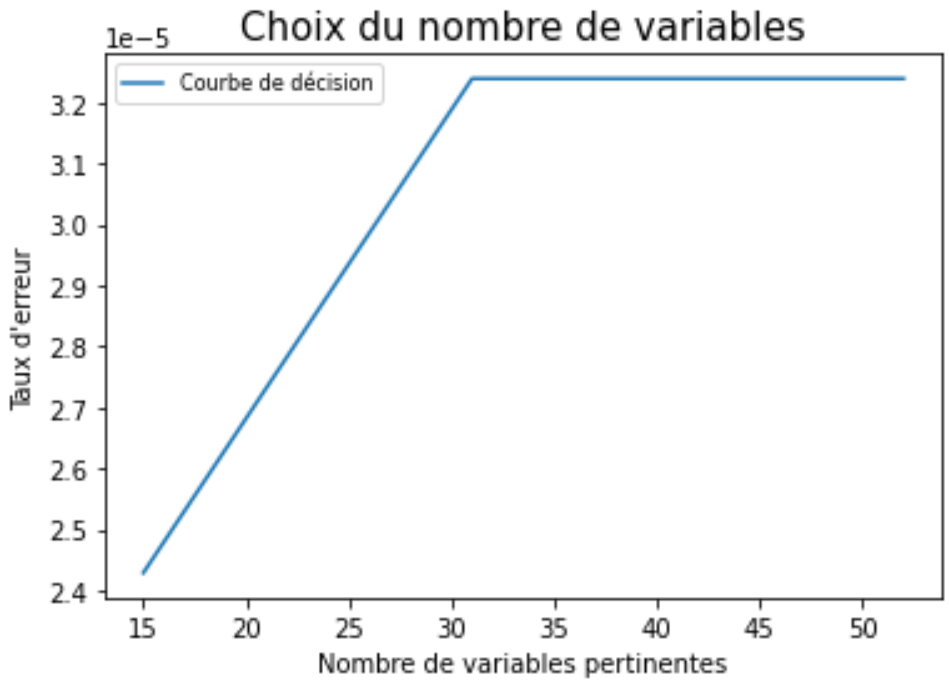
```
# Choix du nombre de variables pertinentes
optim(range(24, 38, 1))
```




```
# Choix du nombre de variables pertinentes
optim(range(10, 20, 1))
```



```
# Choix du nombre de variables pertinentes
optim([15,31,33,42,49,52])
```



7. Sélection de variables

On a pu voir (ci-dessus) que le nombre de variables optimal est 15.

```
# Pour l'échantillon dataTrain
XTrain, yTrain = Var_cible_expl(dataTrain)
# Pour l'échantillon dataTest
XTest, yTest = Var_cible_expl(dataTest)

# Vérification
print("Dimensions XTrain :", XTrain.shape)
print("Dimensions yTrain :", yTrain.shape)

print("Dimensions XTest :", XTest.shape)
print("Dimensions XTest :", XTest.shape)
```

```
Dimensions XTrain : (370515, 274)
Dimensions yTrain : (370515,)
Dimensions XTest : (123506, 274)
Dimensions XTest : (123506, 274)
```

```
# On sélectionne les k variables les plus pertinentes à l'aide d'un test du chi2
selector = SelectKBest(chi2, k = 15)
```

```
# On applique la sélection de variables sur le dataset d'entrainement
selector.fit_transform(XTrain, yTrain)
selector.get_support() # Renvoie un booléen
print("Les variables sélectionnées sont : \n",
XTrain.columns[selector.get_support()])
```

```
Les variables sélectionnées sont :
 Index(['V159', 'V163', 'V164', 'V181', 'V182', 'V185', 'V186', 'V191',
'V193',
       'V198', 'V199', 'V161_m39', 'V162_m1', 'V162_m4', 'V162_m5'],
      dtype='object')
```

8. Entrainement du modèle (Arbre de décision)

```
# Observations ponctuelles
print("Proportions d'observations positives et négatives dans dataTrain :")
print(yTrain.value_counts(normalize = True))

print("Proportions d'observations positives et négatives dans dataTest :")
print(yTest.value_counts(normalize = True))
```

```
Proportions d'observations positives et négatives dans dataTrain :
0    0.997916
1    0.002084
Name: V200, dtype: float64
Proportions d'observations positives et négatives dans dataTest :
0    0.99783
1    0.00217
Name: V200, dtype: float64
```

```
# Instanciation de l'arbre de décision (pour avoir les mêmes valeurs à chaque
fois)
tree = DecisionTreeClassifier(random_state = 0)

# Transformer les données à l'aide des variables sélectionnées
XTrain = XTrain.loc[:,selector.get_support()]
XTest = XTest.loc[:,selector.get_support()]

# Construction du modèle sur les données d'apprentissage
tree.fit(XTrain,yTrain)
```

```
DecisionTreeClassifier(random_state=0)
```

```
# Importance des variables sélectionnées
imp = {"VarName":XTrain.columns,"Importance":tree.feature_importances_}
imp_var = pandas.DataFrame(imp).sort_values(by="Importance",ascending=False)

imp_var.head(5)
```

	VarName	Importance
14	V162_m5	0.651396
1	V163	0.108413
9	V198	0.078702
11	V161_m39	0.054802
5	V185	0.033375

9. Evaluation du modèle sur le “dataTest”

```
# Probas de prédictions
probas_de_prédictions = tree.predict_proba(XTest)
print(probas_de_prédictions)
# Liste des classes
print("Liste des classes :", tree.classes_)
```

```
[[1. 0.]
 [1. 0.]
 [1. 0.]
 ...
 [1. 0.]
 [1. 0.]
 [1. 0.]]
Liste des classes : [0 1]
```

```
# Prédiction
prédictions = tree.predict(XTest)
print(prédictions)
```

```
[0 0 0 ... 0 0 0]
```

```
# Précision moyenne des scores sur les données de test
s = tree.score(XTest, yTest)
print(s)
```

```
0.9999757096821207
```

```
# Matrice de confusion
mc = pandas.crosstab(yTest, prédictions)
print(mc)
# Transformation en matrice numpy (plus facile à manipuler)
mc = mc.values
```

```
col_0       0    1
V200
0      123237    1
1           2  266
```

```
# Taux d'erreur = (FN + FP) / Total
Tx_err = (mc[1,0] + mc[0,1]) / numpy.sum(mc)
print("Taux d'erreur :", Tx_err)

# Taux de reconnaissance
Tx_reco = 1 - Tx_err
print("Taux de reconnaissance global =" , Tx_reco)
```

```
Taux d'erreur : 2.4290317879293315e-05
Taux de reconnaissance global = 0.9999757096821207
```

```
# Rappel (sensibilité) = VP / Nb total de + Observé
SE = mc[1,1]/numpy.sum(mc[1,:])
print("Rappel (sensibilité) = " + str(SE))

# Précision = VP / Nb total de + Prédit
P = mc[1,1]/numpy.sum(mc[:,1])
print("Précision = " + str(P))
```

```
Rappel (sensibilité) = 0.9925373134328358
Précision = 0.9962546816479401
```

```
# Calcul des sensibilités (rappels) et précisions par classe
print(metrics.classification_report(yTest,prédictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    123238
           1       1.00      0.99      0.99       268

    accuracy                           1.00    123506
   macro avg       1.00      1.00      1.00    123506
weighted avg       1.00      1.00      1.00    123506
```

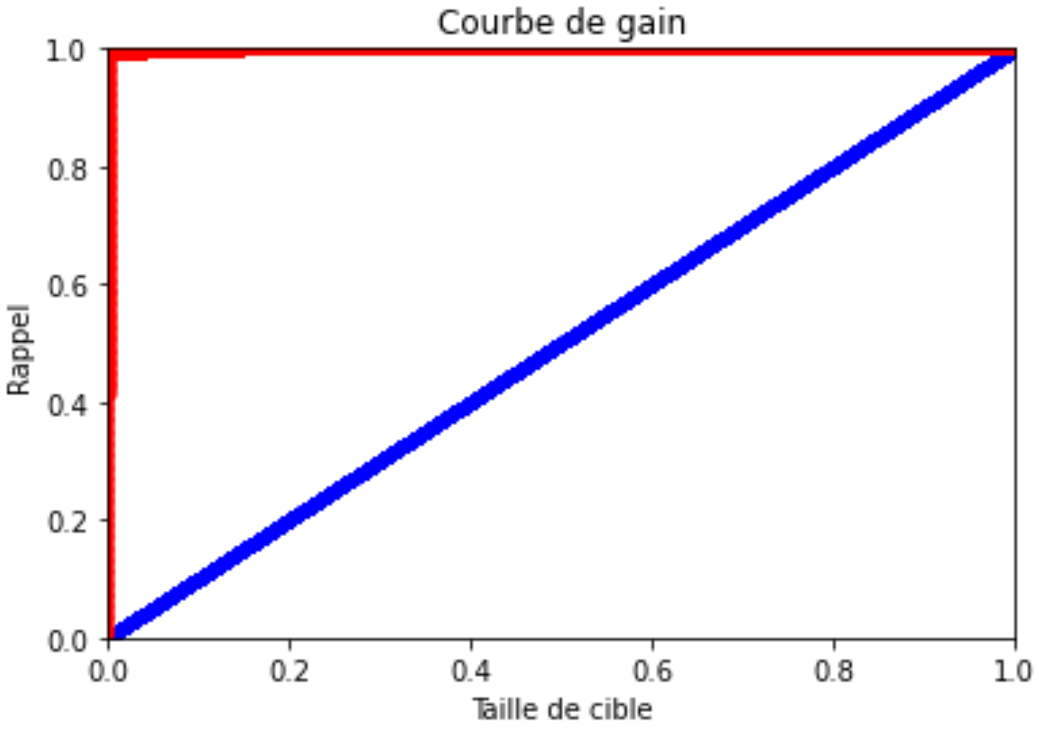
10. Construction de la courbe de gain

```
# Score de 'positif'
score = probas_de_prédictions[:,1]

# Transfert en 0/1 de yTest
pos = pandas.get_dummies(yTest).values

# Colonne de positif
pos = pos[:,1]

# Nombre total de positif
npos = numpy.sum(pos)
print("Nombre total de positif (npos) :", npos)
```

```
# Index pour tri selon le score croissant
index = numpy.argsort(score)
# Inverser pour score décroissant
index = index[::-1]
print("Index pour tri selon le score décroissant :", index)

# Tri des individus (des valeurs 0/1)
sort_pos = pos[index]
print("Tri des individus (des valeurs 0/1) :", sort_pos)

# Somme des positifs cumulée
cpos = numpy.cumsum(sort_pos)
print("Somme des positifs cumulée (cpos) : ",cpos)

# Rappel
rappel = cpos/npos
print("Rappel :", rappel)

# Nombre d'obs dans dataTest
n = yTest.shape[0]
print("Nombre d'obs dans dataTest (n) :", n)

# Taille de cible (en pourcentage)
taille = numpy.arange(start=1,stop=n+1,step=1)/ n
```

```
Nombre total de positif (npos) : 268
Index pour tri selon le score décroissant : [114840  87468  81276 ...  82303
82304      0]
Tri des individus (des valeurs 0/1) : [1 1 1 ... 0 0 0]
Somme des positifs cumulée (cpos) :  [  1   2   3 ... 268 268 268]
Rappel : [0.00373134 0.00746269 0.01119403 ... 1.         1.         1.
]
Nombre d'obs dans dataTest (n) : 123506
```

```
# Graphique
plt.title('Courbe de gain')
plt.xlabel('Taille de cible')
plt.ylabel('Rappel')
plt.xlim(0,1)
plt.ylim(0,1)
plt.scatter(taille,taille,marker='.',color='blue')
plt.scatter(taille,rappel,marker='.',color='red')
plt.show()
```



11. Prédiction sur la base de déploiement (4 898 424 obs)

```
# Nombre d'erreur sur 4 898 424 obs
Nb = round(Tx_err * 4898424)
print("Nb d'erreur sur la base de déploiement (4 898 424 obs) : ", Nb)
```

```
Nb d'erreur sur la base de déploiement (4 898 424 obs) :  119.0
```

```
# Combien d'observations positives y a-t-il parmi les 10 000 individus qui
présentent les scores
# les plus élevés de la base de déploiement (4 898 424 obs) ?

# Recherche des 10 000 / n en se basant sur la taille
numpy.argwhere(taille <= 10000/n)

# --> indice = 9999

# Rappel correspondant au 10 000 ième individu
prop_pos = rappel[9999]
print("Rappel correspondant au 10 000 ième individu : ", prop_pos)

# On multiple par le nombre de positifs
Nombre_de_pos = prop_pos * npos
print("Nb de positif parmi les 10 000 observations qui ont les scores les
plus élevés (4 898 424 obs) :",
      Nombre_de_pos)
```

```
Rappel correspondant au 10 000 ième individus :  0.996268656716418
Nb de positif parmi les 10 000 observations qui ont les scores les plus
élevés (4 898 424 obs) : 267.0
```

12. Sauvegarder “scores.txt” dans le répertoire courant

```
# Création du dataframe à enregistrer
df = pandas.DataFrame({'index' : XTest.index, 'score': [i for i in score] ,
                       'Prédictions': [j for j in prédictions] })
df.head(5)
```

	index	score	Prédictions
0	22650	0.0	0
1	5765	0.0	0
2	241826	0.0	0
3	292391	0.0	0
4	392127	0.0	0

```
# Ecriture du fichier "scores.txt" dans le répertoire courant
df.to_csv("scores.txt", sep="\t", encoding="utf-8", index=False)
```

13. Déploiement

```
deploiement, chemin = Importation_data("data_essai.txt")

# Recodage des variables qualitatives V160, V161 et V162 en quantitatives
deploiement = pandas.get_dummies(deploiement.iloc[:,:], prefix=['V160',
'V161', 'V162'])

# Supprimer les variables sans information (c'est-à-dire nombre de modalité
<= 1)
deploiement = suppr_var_sans_information(deploiement)

deploiement.head(5)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V162_m10	V162_m11	V162_m2	V162_m3	V162_m4	V162_m5	V162_m6	V162_m7	V162_m8	V162_m9
0	0	0	1	5	0	8	0	1	5	1	...	1	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	7	0	1	8	1	...	1	0	0	0	0	0	0	0	0	0
2	1	0	0	9	0	2	0	0	3	1	...	1	0	0	0	0	0	0	0	0	0
3	0	0	0	5	0	3	1	0	0	0	...	1	0	0	0	0	0	0	0	0	0
4	0	1	0	9	1	4	0	1	4	1	...	1	0	0	0	0	0	0	0	0	0

5 rows × 274 columns

```
# Sélection de variable
deploiement = deploiement.loc[:,selector.get_support()]

# Probas de prédictions
probas_de_prédictions = tree.predict_proba(deploiement)

# Prédiction
prédictions = tree.predict(deploiement)

# Score de 'positif'
score = probas_de_prédictions[:,1]

# Création du dataframe à enregistrer
df = pandas.DataFrame({'index' : deploiement.index, 'score': [i for i in
score] ,
                       'Prédictions': [j for j in prédictions] })
df.head(5)
```

	index	score	Prédictions
0	0	0.0	0
1	1	0.0	0
2	2	0.0	0
3	3	0.0	0
4	4	0.0	0

```
# Ecriture du fichier "scores.txt" dans le répertoire courant
df.to_csv("scores.txt", sep="\t", encoding="utf-8", index=False)
```