PROGRAMME 3 – 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

from sklearn import preprocessing
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
%matplotlib inline

# Implémentation des paramètres
taille_train = 0.75
taille_test = 0.25
```

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

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 494021 entries, 0 to 494020
Columns: 200 entries, V1 to V200
dtypes: float64(15), int64(181), object(4)
memory usage: 753.8+ MB
None
```

```
# 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

```
# 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), object(1), uint8(80)
```

```
memory usage: 772.7+ 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, train_size =
taille_train,
                                                       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 colonne 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)

# 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)
```

6. Choix du nombre de variables le plus optimal

Grâce au programme 1, on a pu voir que le nombre de variables optimal est 54.

7. Selection de variables

```
# 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 :", 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 = 54)
# 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', 'V165', 'V166', 'V167', 'V168', 'V169',
'V170',
       'V171', 'V172', 'V174', 'V175', 'V176', 'V180', 'V181', 'V182',
'V183',
```

```
       'V184', 'V185', 'V186', 'V187', 'V188', 'V189', 'V190', 'V191',
'V192',
       'V193', 'V194', 'V195', 'V196', 'V197', 'V198', 'V199', 'V160_m1',
       'V160_m2', 'V160_m3', 'V161_m11', 'V161_m12', 'V161_m16', 'V161_m17',
       'V161_m20', 'V161_m22', 'V161_m39', 'V161_m44', 'V161_m49',
'V161_m55',
       'V161_m9', 'V162_m10', 'V162_m11', 'V162_m2', 'V162_m3', 'V162_m5',
       'V162_m6'],
      dtype='object')
```

8. Transformation des datasets

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

9. Clustering : Méthodes centroïdes des k-means

9.a. Normalisation des données

```
x = XTrain.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
XTrain_scaled = pandas.DataFrame(x_scaled, columns = XTrain.columns)
XTrain_scaled.head(5)
```

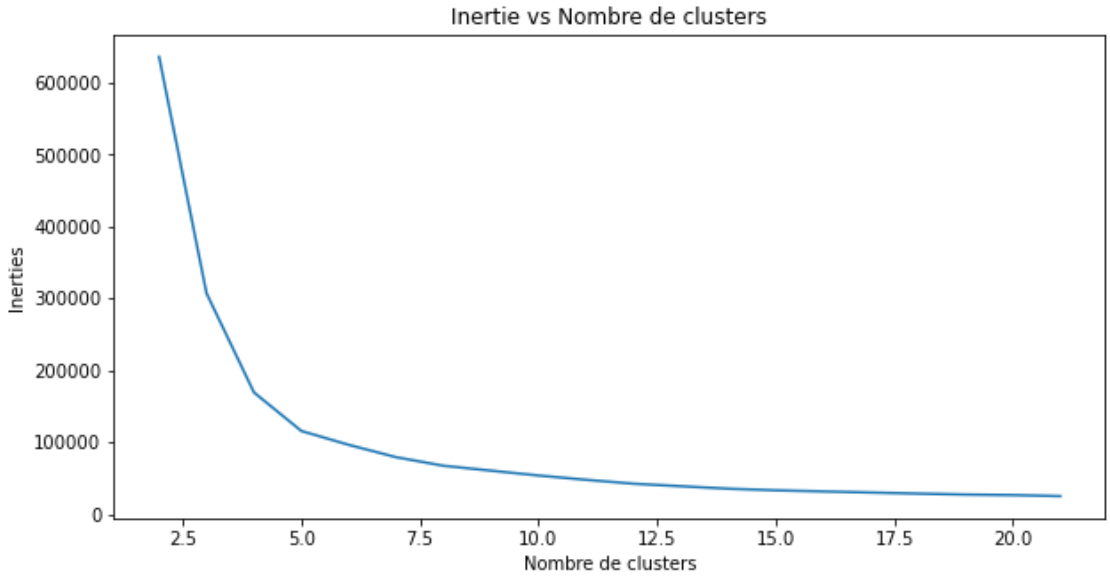
	V159	V163	V164	V165	V166	V167	V168	V169	V170	V171	...	V161_m44	V161_m49	V161_m55	V161_m9	V162_m10	V162_m11	V162_m2	V162_m3	V162_m5	V162_m6
0	0.0	0.000201	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.000043	0.000251	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.000201	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.000077	0.000149	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0

5 rows × 54 columns

9.b. Choix du nombre de k

```
def choix_nb_clusters(Nb_k):
    # On crée une liste dans laquelle on stocke les inerties
    inerties = []

    # On fait une boucle de 2 à 9 pour tester toutes ces possibilités
    for k in Nb_k:
        # Pour chaque k, on crée un modèle et on l'ajuste
        kmeans = KMeans(n_clusters = k, random_state = 0)
        kmeans.fit(XTrain_scaled)
        # On stocke l'inertie associée
```

```
        inerties.append(kmeans.inertia_)

    # On représente le graphique
    fig = plt.figure(figsize = (10,5))
    plt.plot(Nb_k, inerties)
    plt.xlabel("Nombre de clusters")
    plt.ylabel("Inerties")
    plt.title("Inertie vs Nombre de clusters")

Nb_k = range(2, 22)
choix_nb_clusters(Nb_k)
# --> L'inertie continue toujours de baisser mais on peut voir un "coude"
autour de 5 classes sur le graphique
```



9.c. Application des k-means sur nos données (k = 5)

```
# Application des k-means sur nos données
modele_km = KMeans(n_clusters = 5, random_state = 0)
modele_km.fit(XTrain_scaled)
```

```
KMeans(n_clusters=5, random_state=0)
```

```
# On stocke les classes d'appartenances dans classes
classes = modele_km.labels_
print(classes)

# On crée un DataFrame avec le nombre d'individus par classes
count = pandas.DataFrame(numpy.unique(classes, return_counts = True)[1],
                         columns = ["Nombre d'individus"])
print(count)
```

```
# On stocke le centre des classes dans un DataFrame
centre = pandas.DataFrame(modele_km.cluster_centers_, columns =
XTrain_scaled.columns)
```

```
[0 2 0 ... 0 0 3]
   Nombre d'individus
0              210925
1               65205
2               55132
3               21193
4               18060
```

```
# On affiche le DataFrame avec les 2 informations
pandas.set_option('precision', 2)
print(pandas.concat([count, pandas.DataFrame(centre, columns =
XTrain.columns)], axis = 1).T.head(10))
pandas.reset_option('precision')
```

```
                           0         1         2         3         4
Nombre d'individus  2.11e+05  6.52e+04  5.51e+04  2.12e+04  1.81e+04
V159               -2.72e-16  1.35e-06  2.88e-04  1.45e-03  1.44e-02
V163                1.82e-04  1.56e-05  1.40e-03  1.81e-06  1.84e-04
V164               -1.17e-16  2.36e-06  9.24e-04  4.09e-06  7.41e-04
V165               -2.24e-17  2.91e-04  8.91e-18  1.06e-18 -4.74e-20
V166                1.88e-04  1.82e-16  1.77e-16  1.31e-16  4.24e-02
V167                2.25e-18 -1.15e-19  2.42e-05 -9.15e-20 -8.56e-20
V168               -5.72e-16  3.58e-06  7.91e-03  7.24e-05  3.69e-06
V169               -2.69e-18  3.07e-06  2.90e-05  3.21e-04  1.55e-04
V170                6.88e-14  9.20e-05  9.97e-01  1.13e-03  5.80e-15
```

9.d. Attribution des nouvelles classes

```
# Attribution des nouvelles classes
Attribution = pandas.DataFrame(pandas.crosstab(yTrain,classes,
normalize="index"))
Attribution
```

col_0	0	1	2	3	4
V200					
m1	0.000000	0.000000	0.997612	0.002388	0.000000
m10	0.000000	0.808828	0.000000	0.191172	0.000000
m11	0.000000	0.426966	0.000000	0.000000	0.573034
m12	0.003341	0.000151	0.720412	0.055849	0.220248
m13	0.000000	0.000000	1.000000	0.000000	0.000000
m14	0.000000	0.000000	1.000000	0.000000	0.000000
m15	0.597990	0.000000	0.000000	0.000000	0.402010
m16	0.001295	0.023316	0.000000	0.970207	0.005181
m17	0.000000	0.000000	0.333333	0.000000	0.666667
m18	0.000000	0.109623	0.004184	0.763180	0.123013
m19	0.999972	0.000000	0.000000	0.000000	0.000028
m2	0.000000	0.000000	1.000000	0.000000	0.000000
m20	0.000000	0.000000	0.000000	0.000000	1.000000
m21	0.000000	0.000000	0.000000	0.000000	1.000000
m22	0.000000	0.001272	0.998728	0.000000	0.000000
m23	0.000000	0.000000	0.133333	0.000000	0.866667
m3	0.000000	0.000000	0.714286	0.000000	0.285714
m4	0.000000	0.027778	0.000000	0.944444	0.027778
m5	0.000000	0.555556	0.444444	0.000000	0.000000
m6	0.003205	0.000000	0.002137	0.069444	0.925214
m7	0.000000	1.000000	0.000000	0.000000	0.000000
m8	0.000000	0.000000	0.833333	0.000000	0.166667
m9	0.000000	0.000000	0.714286	0.000000	0.285714

```
# Création du dictionnaire de correspondance à partir du dataframe
Attribution

mon_dictionnaire = dict()

names_V200 = Attribution.index

for i in range(len(names_V200)):
    if Attribution.iloc[i,0] == max(Attribution.iloc[i,:]):
        mon_dictionnaire[names_V200[i]] = "M0"
    if Attribution.iloc[i,1] == max(Attribution.iloc[i,:]):
```

```
        mon_dictionnaire[names_V200[i]] = "M1"
    if Attribution.iloc[i,2] == max(Attribution.iloc[i,:]):
        mon_dictionnaire[names_V200[i]] = "M2"
    if Attribution.iloc[i,3] == max(Attribution.iloc[i,:]):
        mon_dictionnaire[names_V200[i]] = "M3"
    if Attribution.iloc[i,4] == max(Attribution.iloc[i,:]):
        mon_dictionnaire[names_V200[i]] = "M4"

print(mon_dictionnaire)
```

```
{'m1': 'M2', 'm10': 'M1', 'm11': 'M4', 'm12': 'M2', 'm13': 'M2', 'm14': 'M2',
'm15': 'M0', 'm16': 'M3', 'm17': 'M4', 'm18': 'M3', 'm19': 'M0', 'm2': 'M2',
'm20': 'M4', 'm21': 'M4', 'm22': 'M2', 'm23': 'M4', 'm3': 'M2', 'm4': 'M3',
'm5': 'M1', 'm6': 'M4', 'm7': 'M1', 'm8': 'M2', 'm9': 'M2'}
```

9.e. Création du fichier “classes.txt”

```
# Création du dataframe à enregistrer
Classes_init = []
Classes_final = []
for key, values in mon_dictionnaire.items():
    Classes_init.append(key)
    Classes_final.append(values)

df_classes = pandas.DataFrame({'V200' : Classes_init, 'V200_Prim':
Classes_final})
df_classes.head(25)

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

10. Regroupement des modalités de V200 à partir du fichier “classes.txt”

```
classes = pandas.read_table("classes.txt", sep="\t", header=0)
classes.head(5)
```

	V200	V200_Prim
0	m1	M2
1	m10	M1
2	m11	M4
3	m12	M2
4	m13	M2

```
# Créer le dictionnaire à partir de "classes.txt"
dic = dict()
```

```
for i in range(classes.shape[0]):
    dic[classes.V200[i]] = classes.V200_Prim[i]
print(dic)
```

```
{'m1': 'M2', 'm10': 'M1', 'm11': 'M4', 'm12': 'M2', 'm13': 'M2', 'm14': 'M2',
'm15': 'M0', 'm16': 'M3', 'm17': 'M4', 'm18': 'M3', 'm19': 'M0', 'm2': 'M2',
'm20': 'M4', 'm21': 'M4', 'm22': 'M2', 'm23': 'M4', 'm3': 'M2', 'm4': 'M3',
'm5': 'M1', 'm6': 'M4', 'm7': 'M1', 'm8': 'M2', 'm9': 'M2'}
```

```
# Recodage de la variable cible yTest
yTest = yTest.replace(dic)
yTest
```

```
22650     M2
5765      M2
241826    M0
292391    M0
392127    M1
          ..
125286    M1
316495    M0
225354    M0
104672    M2
334038    M0
Name: V200, Length: 123506, dtype: object
```

```
# Recodage de la variable cible yTrain
yTrain = yTrain.replace(dic)
yTrain
```

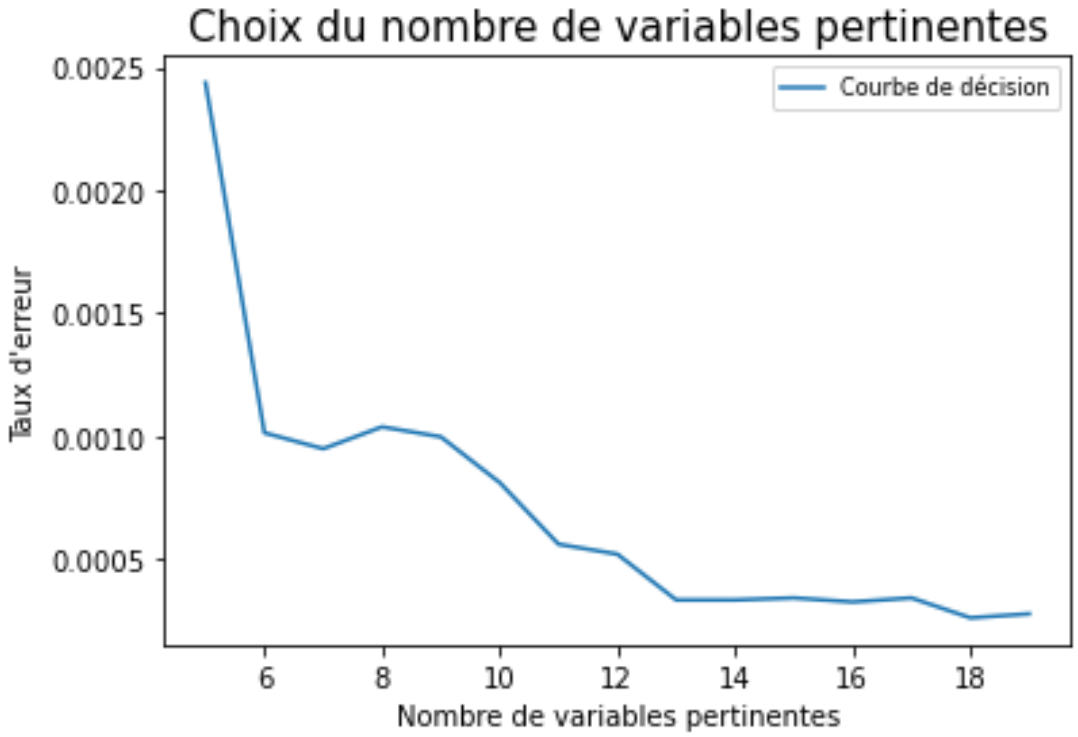
```
248606    M0
27004     M2
257508    M0
1277      M2
455412    M2
          ..
439107    M0
117952    M1
435829    M0
305711    M0
461484    M1
Name: V200, Length: 370515, dtype: object
```

11. Choix de re-sélection de variables

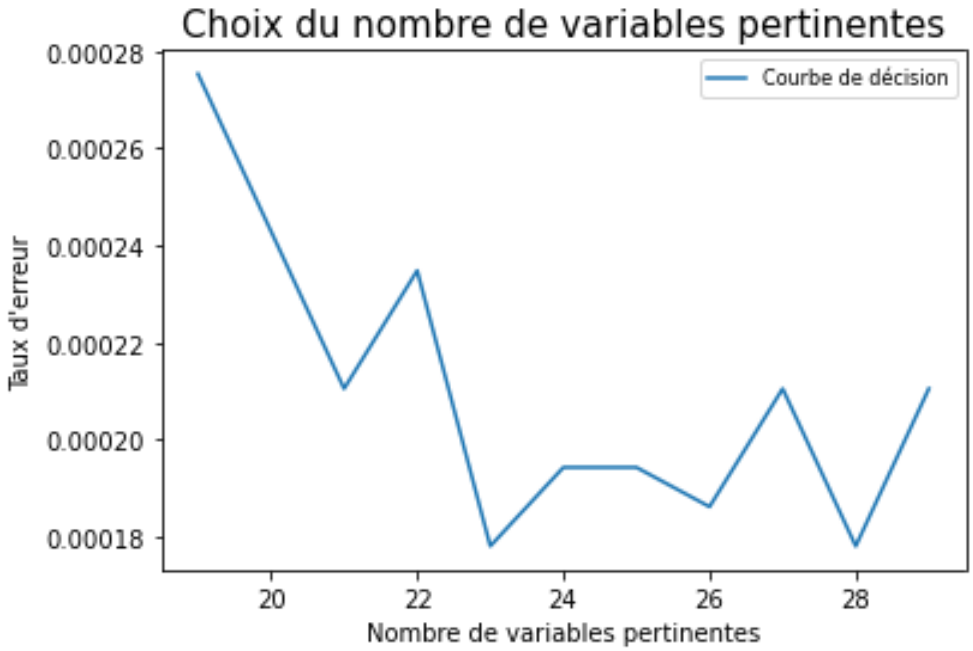
```
XTrain_pour_reselection = XTrain
XTest_pour_reselection = XTest
yTrain_pour_reselection = yTrain
yTest_pour_reselection = yTest
```

```
""" Fonction de décision pour optimiser le nombre de variable : optim(from,
to, step)
 Input :
   - from
   - to
   - step
 Output :
   - Taux_err : Liste des taux d'erreurs
   - Nb_err : Liste des nombre d'erreurs
   - Graphique : "Choix du nombre de variables pertinentes"
"""
def optim_pour_reselection(Nb_var):

    Taux_err = []
    Nb_err = []

    for i in Nb_var:
        # Pour l'échantillon dataTrain
        XTrain = XTrain_pour_reselection
        yTrain = yTrain_pour_reselection
        # Pour l'échantillon dataTest
        XTest = XTest_pour_reselection
        yTest = yTest_pour_reselection

        # 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()]

        # Instanciation - objet arbre de décision
        dtree = DecisionTreeClassifier(random_state = 0)

        # Application sur les données d'apprentissage
        dtree.fit(XTrain,yTrain)

        # Prédiction en test
        yPred = dtree.predict(XTest.loc[:,selector.get_support()])

        # Matrice de confusion
        #mc = metrics.confusion_matrix(yTest,yPred)

        # Taux de reconnaissance
        acc = metrics.accuracy_score(yTest,yPred)
        #print("Taux de reconnaissance = " + str(acc))

        # Taux d'erreur
```

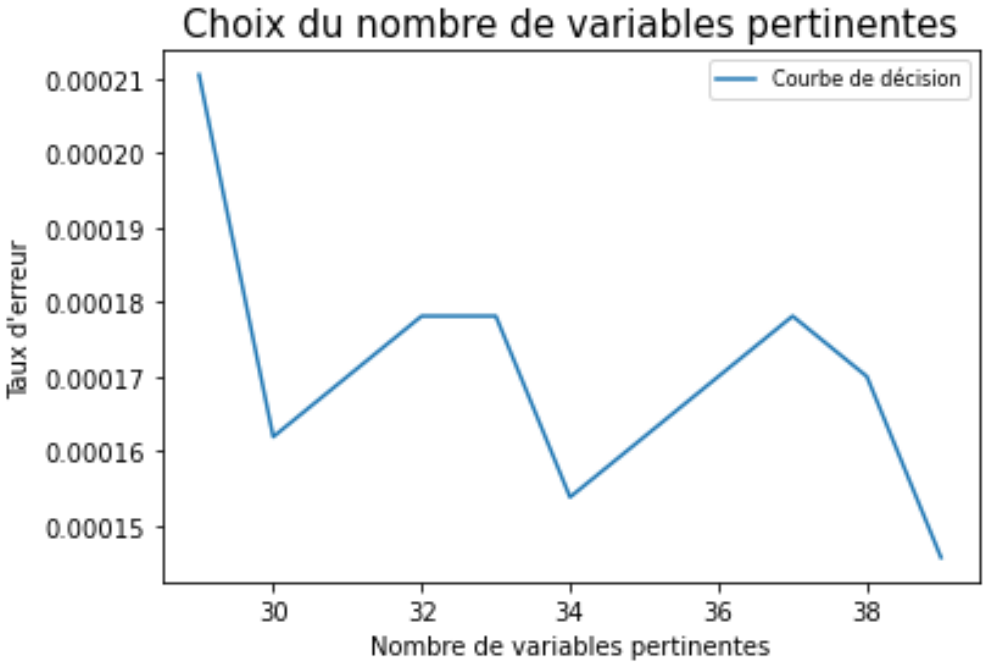
```
        Taux_erreur = 1.0 - acc
        Taux_err.append(Taux_erreur)

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

    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 pertinentes", fontsize=15)
    plt.legend(loc = 'best', fontsize=8)
    plt.show()

optim_pour_reselection(range(5, 20, 1))
```



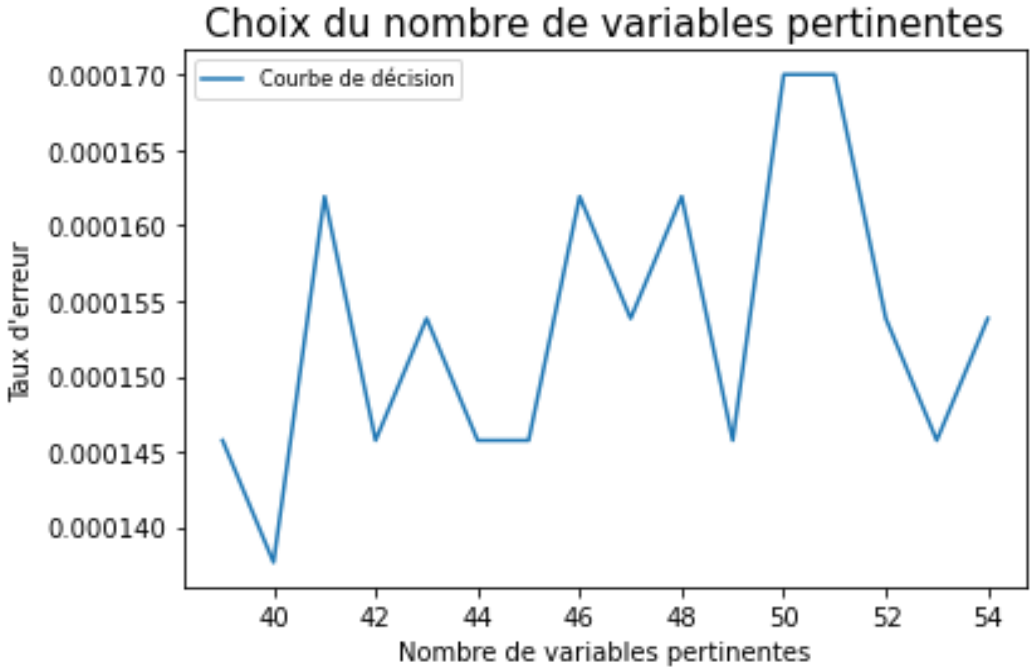
```
optim_pour_reselection(range(19, 30, 1))
```



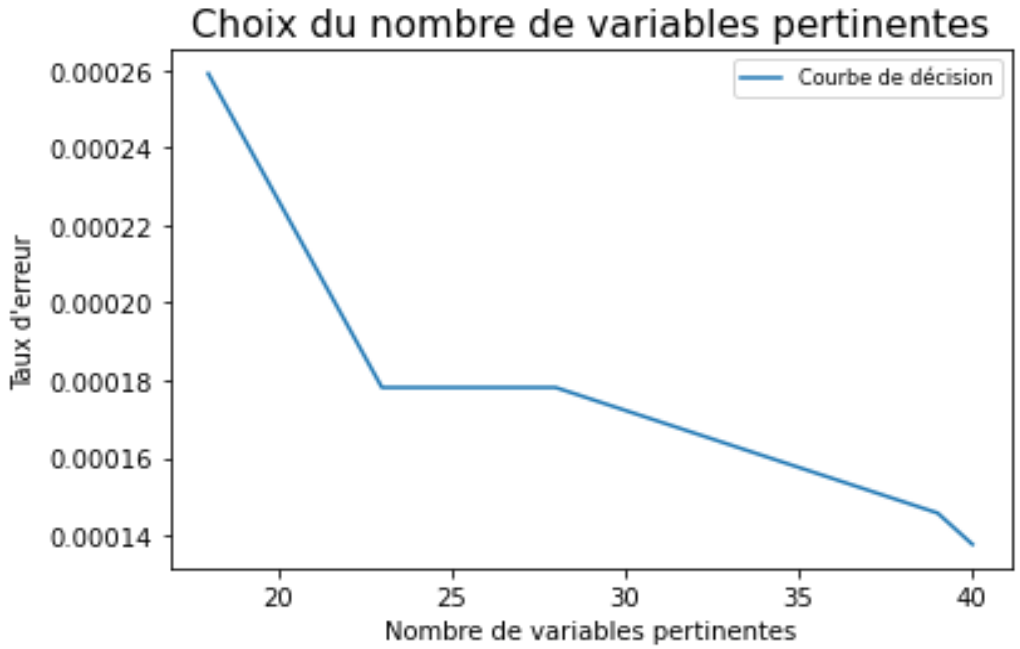

```
optim_pour_reselection(range(29, 40, 1))
```



```
optim_pour_reselection(range(39, 55, 1))
```




```
optim_pour_reselection([18,23,28,39,40])
# Choix le plus pertinent : 40
```



12. Re-sélection de variables

On a pu voir (ci-dessus) que le nombre de variables optimal est 40.

```
XTrain_pour_reselection = XTrain
XTest_pour_reselection = XTest
yTrain_pour_reselection = yTrain
yTest_pour_reselection = yTest

# On sélectionne les k variables les plus pertinentes à l'aide d'un test du
chi2
selector2 = SelectKBest(chi2, k = 40)
# On applique la sélection de variables sur le dataset d'entrainement
selector2.fit_transform(XTrain_pour_reselection, yTrain_pour_reselection)
selector2.get_support() # Renvoie un booléen
listes_variables = XTrain_pour_reselection.columns[selector2.get_support()]
print("Les variables sélectionnées sont : \n", listes_variables)
```

```
Les variables sélectionnées sont :
 Index(['V159', 'V163', 'V164', 'V166', 'V168', 'V170', 'V174', 'V181',
'V182',
       'V183', 'V184', 'V185', 'V186', 'V187', 'V188', 'V189', 'V190',
'V191',
       'V192', 'V193', 'V194', 'V195', 'V196', 'V197', 'V198', 'V199',
       'V160_m1', 'V160_m2', 'V160_m3', 'V161_m11', 'V161_m12', 'V161_m20',
       'V161_m39', 'V161_m44', 'V161_m49', 'V161_m9', 'V162_m10', 'V162_m2',
       'V162_m5', 'V162_m6'],
      dtype='object')
```

13. Transformation des dataset

```
# Transformer les données d'apprentissage à l'aide des variables
sélectionnées
XTrain = XTrain_pour_reselection.loc[:,selector2.get_support()]
XTest = XTest_pour_reselection.loc[:,selector2.get_support()]
```

14. Entrainement du modèle (Arbre de décision)

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

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

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

```
# Importance des variables sélectionnées
imp = {"VarName":XTrain.columns,"Importance":dtree.feature_importances_}
```

```
imp_var = pandas.DataFrame(imp).sort_values(by="Importance",ascending=False)

imp_var.head(5)
```

	VarName	Importance
30	V161_m12	0.614485
13	V187	0.338717
1	V163	0.013769
19	V193	0.011455
27	V160_m2	0.008603

15. Evaluation du modèle sur le "dataTest"

```
# Prédiction en test
yPred = dtree.predict(XTest)

# Matrice de confusion
mc = metrics.confusion_matrix(yTest,yPred)
print(mc)
```

```
[[70290     0     1     0     0]
 [    0 26908     2     3     0]
 [    0     0 25012     2     1]
 [    0     1     0   677     1]
 [    0     0     5     1   602]]
```

```
# Taux de reconnaissance
acc = metrics.accuracy_score(yTest,yPred)
print("Taux de reconnaissance global = " + str(acc))

# Taux d'erreur
Taux_erreur = 1.0 - acc
print("Taux d'erreur global = " + str(Taux_erreur))
```

```
Taux de reconnaissance global = 0.9998623548653507
Taux d'erreur global = 0.00013764513464931127
```

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

```
              precision    recall  f1-score   support

          M0       1.00      1.00      1.00     70291
          M1       1.00      1.00      1.00     26913
          M2       1.00      1.00      1.00     25015
```

```
           M3       0.99      1.00      0.99       679
           M4       1.00      0.99      0.99       608

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

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

```
# Nombre d'erreur sur 4 898 424 obs
Nb = round(Taux_erreur * 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) :  674.0
```

17. Sauvegarder “predictions.txt” dans le répertoire courant

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

	index	yPred
0	22650	M2
1	5765	M2
2	241826	M0
3	292391	M0
4	392127	M1

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

18. Déploiement

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

deploiement.head(5)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V190	V191	V192	V193	V194	V195	V196	V197	V198	V199
0	0	0	1	5	0	8	0	1	5	1	...	9	9	1.0	0.0	0.11	0.0	0.0	0.0	0.0	0.0
1	0	0	1	1	1	7	0	1	8	1	...	19	19	1.0	0.0	0.05	0.0	0.0	0.0	0.0	0.0
2	1	0	0	9	0	2	0	0	3	1	...	29	29	1.0	0.0	0.03	0.0	0.0	0.0	0.0	0.0
3	0	0	0	5	0	3	1	0	0	0	...	39	39	1.0	0.0	0.03	0.0	0.0	0.0	0.0	0.0
4	0	1	0	9	1	4	0	1	4	1	...	49	49	1.0	0.0	0.02	0.0	0.0	0.0	0.0	0.0

5 rows × 199 columns

```
# Recodage des variables qualitatives V160, V161 et V162 en quantitatives
deploiement = pandas.get_dummies(deploiement.iloc[:,:], prefix=['V160',
'V161', 'V162'])
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 × 276 columns

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

# Prédiction en test
yPred = dtree.predict(deploiement.loc[:,listes_variables])

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

	index	yPred
0	0	M2
1	1	M2
2	2	M2
3	3	M2
4	4	M2

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