PROGRAMME 1 – 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 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)

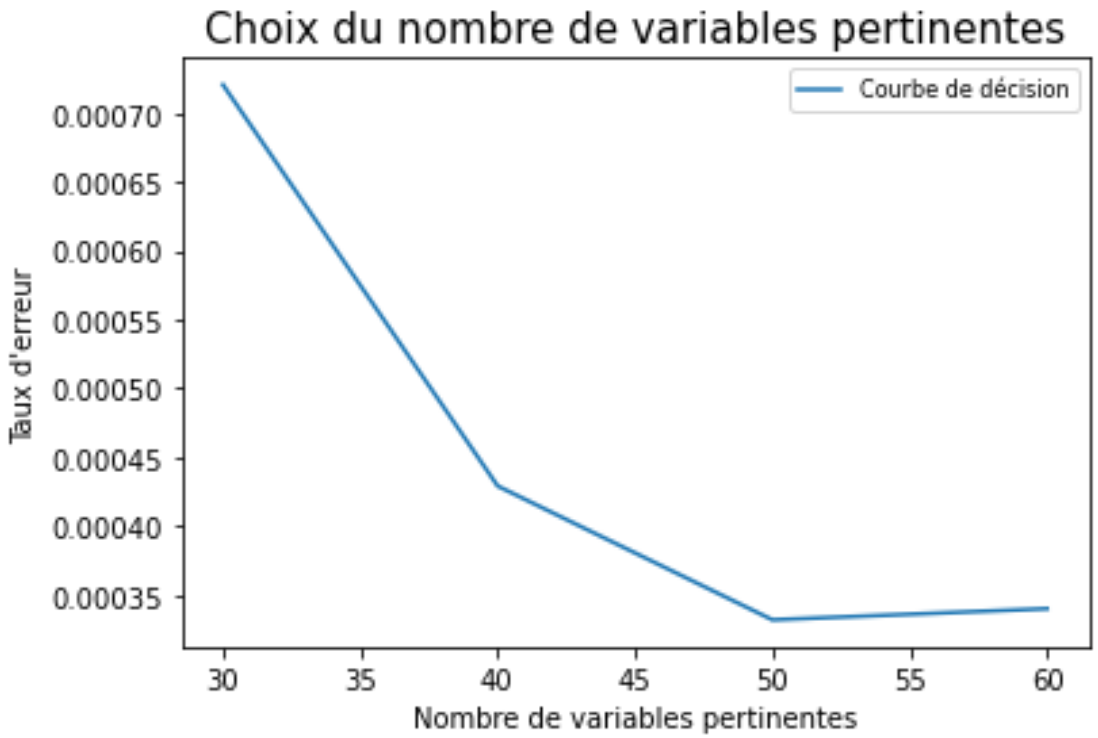
```
""" Fonction pour définir la variable cible (y) et les variables explicatives
(X) : Var_cible_expl(dataframe)
 Input :
   - dataframe : le dataframe à définir
 Output :
   - X : le dataframe des variables explicatives
   - y : la variable cible
"""
def Var_cible_expl(dataframe):
    # On récupère 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)

# 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)
```

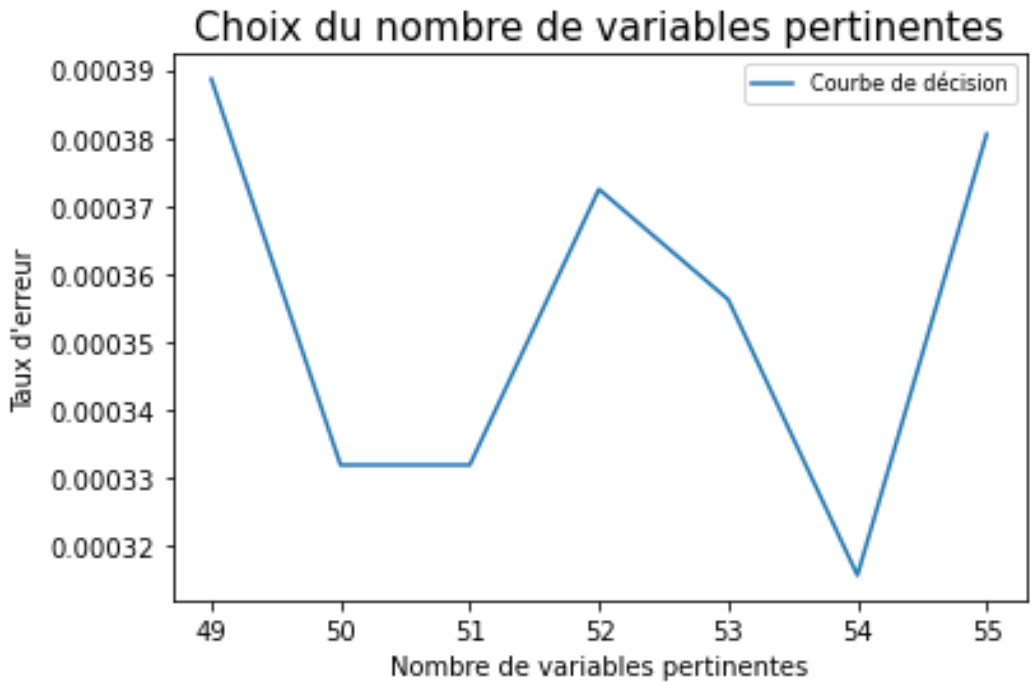
6. Choix du nombre de variables le plus optimal

```
""" 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(from_, to_, step_):
    Nb_var = range(from_, to_, step_)
    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()]

        # 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()


# Choix du nombre de variables pertinentes
optim(from_ = 30, to_ = 61, step_ = 10)
```



```
# Choix du nombre de variables pertinentes
optim(from_ = 49, to_ = 56, step_ = 1)
```



7. Sélection de variables

On a pu voir (ci-dessus) que le nombre de variables optimal est 54.

```
# 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. 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)

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

# 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
16	V182	0.604607
21	V187	0.325891
9	V171	0.013195
27	V193	0.011625
1	V163	0.007588

9. Evaluation du modèle sur le “dataTest”

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

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

# 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 = 0.9996842258675692
Taux d'erreur global = 0.0003157741324307945
```

```
# Calcul des sensibilité (rappel) et précision par classe
print(metrics.classification_report(yTest,yPred))
```

```
/home/elisa/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/elisa/anaconda3/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning:
Recall and F-score are ill-defined and being set to 0.0 in labels with no
true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))


              precision    recall  f1-score   support

          m1       1.00      1.00      1.00       528
         m10       1.00      1.00      1.00     26907
```

```
          m11       0.98      0.98      0.98        53
          m12       1.00      1.00      1.00     24242
          m13       0.00      0.00      0.00         0
          m14       1.00      1.00      1.00         1
          m15       1.00      0.98      0.99        65
          m16       0.98      0.99      0.98       268
          m17       0.50      0.25      0.33         4
          m18       0.98      1.00      0.99       394
          m19       1.00      1.00      1.00     70226
           m2       0.80      0.67      0.73         6
          m20       1.00      1.00      1.00         1
          m21       1.00      1.00      1.00       234
          m22       0.98      0.99      0.99       234
          m23       1.00      0.80      0.89         5
           m3       0.00      0.00      0.00         1
           m4       1.00      0.94      0.97        17
           m5       1.00      0.33      0.50         3
           m6       0.99      1.00      1.00       311
           m7       1.00      1.00      1.00         3
           m8       0.00      0.00      0.00         3
           m9       0.00      0.00      0.00         0

    accuracy                           1.00    123506
   macro avg       0.79      0.74      0.75    123506
weighted avg       1.00      1.00      1.00    123506
```

10. 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) :  1547.0
```

11. 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	m12
1	5765	m12
2	241826	m19
3	292391	m19
4	392127	m10

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

12. 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[:,selector.get_support()])
```