PROJET DATA MINING
CLASSEMENT ET CIBLAGE

Université Lyon 2/ICOM
M1 Informatique 2020/2021

Auteurs :
- HAMZA Assoumani Chissi
- LO Mouhamadou Mansour

28 DECEMBRE 2020

Table des matières

I. Introduction

Dans le cadre du cours de Dataminig de la formation M1 Informatique de l'université de Lyon 2, Les étudiants de cette formation étaient amenés a réalisé un projet de groupe portant sur de la prédiction et du scoring d'une base de données.

Les objectifs de ce dossier sont :

-La production d'un système de classement

-La production d'un système de scoring

-La production d'un système de classement d'une variable cible avec les modalités regroupées

Notre travail a été réalisé avec le langage python et ses différentes librairies utilisés en traitement de données comme : Pandas, sklearn, scipy, yellowbrick, numpy …

Le travail rendu comportant ce rapport possède 3 dossiers

1) Le dossier *Prédictions et Scoring*
   - get_prediction.py le premier livrable
   - get_score.py le deuxième livrable
   - get_prediction_Vprim.py le troisième livrable
   - et d'autres fichiers nécessaires pour le fonctionnement des livrables

2) Le dossier *modélisations* avec les programmes utilisés pour faire la modélisation ainsi que des fichiers nécessaires.

3) Le dossier *fichiers complémentaires* contenant nos différents tests permettant de retracer notre travail

II. Compréhension de données

1. Description de données

Nous disposons d'une base de données avec 494 021 observations et 200 variables numérotées V1 à V200 dont la variable cible est V200 qui présente 23 modalités. Les variables explicatives sont représentées sous différents formats : V160, V161 et V162 sont qualitatives et les autres quantitatives. On rajoute à cela que les variables prédictives quantitatifs n'ont pas la même précision.

2. Exploration de données

Pendant la phase d'exploration de données, certaines variables ne nous semblent pas avoir un grand impact sur la prédiction de la variable cible (colonne pleine de zéros …), mais pour en avoir une confirmation, par la suite nous effectuerons des tests statistiques qui nous permettront d'effectuer la sélection des variables, pour garder que les variables pertinentes pour le modèle finale.

III. Préparation de données

1. Le recodage des variables qualitatives

L'étape de recodage de **variables prédictives qualitatives** est primordiale dans notre cas pour pouvoir effectuer la modélisation. Les méthodes d'apprentissage supervisé telles que la régression logistique, l'arbre de décision et l'analyse discriminante ne peuvent pas être mises en œuvre directement si les variables prédictives sont catégorielles.

Il existe plusieurs techniques de recodage mais dans notre cas, nous avons choisi la technique de codage disjonctif complet ou codage 0/1. Ce dernier permet de transformer chaque modalité de la variable originelle à une variable binaire qui prend la valeur 1 lorsque la modalité est présente pour un individu, 0 sinon.

Dans notre base, seulement les colonnes V160, V161 et V162 qui ont été sélectionnés pour subir la transformation. A la fin de cette manipulation, nous sommes passés de 199 variables prédictives à 276 variables au final. Pour le traitement de cette tâche, nous avons utilisé la fonction **get_dummies()** de la librairie pandas.

2. La sélection de données

Nous avons effectué une sélection des variables dans le but de garder les attributs les plus pertinentes dans notre modèle. Il existe plusieurs techniques pour la sélection des

variables. Dans notre cas, nous avons testé en premier lieu la méthode RFE de ***scikit-learn feature_selection***. Cet algorithme permet d'éliminer au fur et à mesure les coefficients les plus faibles en valeur absolue et s'arrête quand on arrive à la moitié de variables qui est le paramètre par défaut ou à un nombre spécifié de variables réglable par le paramètre ***n_features_to_select***.

Avec l'immense volume de données que nous avions à traiter, la méthode RFE n'était pas la bonne solution puisqu'elle était trop gourmande en temps et en mémoire. Pour cette raison, nous avons opté pour d'autres techniques de sélection de la même librairie scikit-learn dont les méthodes ***SelectFwe*** et ***f_classif.*** Cette technique nous permet d'effectuer une analyse de la variance (ANOVA) en sélectionnant les attributs dont la p-values est inférieur à un alpha que nous définissons en paramètre. Dans notre cas, la valeur de alpha a été fixé à 0.001. Dans tout le cas, la valeur d'alpha comprise entre 10^{-3} à 10^{-15} donne toujours les mêmes variables prédictives.

Après la sélection de variables, 113 variables prédictives ont été sélectionnés contre 276 après recodage. Ce qui fait que pour la suite, nous avons travaillé avec 114 variables avec la variable cible comme dernière colonne.

IV. Modélisation

1. Choix de technique de modélisation

De nombreuses techniques de modélisation sont disponibles telles que la régression logistique, l'analyse discriminante et l'arbre de décision. Nous avons mené plusieurs tests pour le choix du modèle final à utiliser et nous avons conclu que selon la base que nous avions à traité la régression légistique paramétrée est l'idéal pour nos traitements.

Le tableau 1 résume les différents paramètres que nous avions testé. Les paramètres misent en jeux ici est **solver** qui représente l'algorithme à utiliser et **multi_class** qui définit la classe de la modalité cible. Le test a été réalisé sur une base de 518 012 observations avec 55 colonnes dont la colonne cible présente 7 modalités. Cette base a été subdivisé avec 20000 observations en apprentissage et le reste pour le test.

	Multi_class	Solver	Donnée centré réduit	Temps de prédiction (en s)	Taux d'erreur	Convergence
1	***multinomial***	***newton-cg***	non	1min 54	0.2829	non
2	***multinomial***	***newton-cg***	oui	31.6 s	0.2775	oui
3	***multinomial***	***saga***	oui	18.7 s	0.2809	non
4	***multinomial***	***sag***	oui	7.9 s	0.2790	non
5	***multinomial***	***sag***	non	8.1 s	0.3422	non
6	***multinomial***	***lbfgs***	oui	5.1 s	0.2777	non
7	***multinomial***	***lbfgs***	non	4.9 s	0.3820	non
8	***ovr***	***liblinear***	non	6.12 s	0.2896	oui

9	***ovr***	***liblinear***	oui	9.17 s	0.2863	oui
10	***ovr***	***newton-cg***	oui	7.82 s	0.2862	oui
11	***ovr***	***newton-cg***	non	53.8 s	0.2872	non
12	***ovr***	***lbfgs***	oui	5.77 s	0.2862	non

Tableau 1 : performance de la régression logistique

A partir de ces différents tests que nous avons réalisé sur la régression logistique, les paramètres optimaux pour la modélisation sont donc **multi_class=multinomial** et **solver=lbfgs.**

Le tableau 2 montre une étude comparative entre le modèle d'analyse discriminante et la régression logistique sur **notre base de déploiement**.

Paramètre (solver)	*Temps de prédiction*	*Taux d'erreur*
Analyse discriminante		
svd	6.75 s	0.00526
lsqr	6.41 s	0.00530
Régression logistique		
lbfgs	6.46 s	0.00087

Tableau 2 : comparaison des modèles

2. Création du modèle

Pour répondre aux deux premiers objectifs de notre travail : la prédiction et le scoring, nous avons créé en premier lieu le modèle d'apprentissage en utilisant la régression logistique de la librairie **scikit-learn.** Voici les différentes étapes que nous avons effectué pour la création du modèle :

- Importation de données en mémoire
- Recodage des variables prédictives qualitatifs en codage 0/1.
- La sélection des variables pertinents
- Sauvegarde de notre sélection pour l'appliquer à la base de test
- Construction du modèle d'apprentissage
- Sauvegarde du modèle crée pour l'appliquer à la base de test.

Ces différentes étapes sont illustrées dans le fichier python **get_modelisation.py** que nous avons fournie avec notre dossier.

Nous commençons toujours par le changement du répertoire de travail puis l'import des données

```
#modification du dossier par défaut
import os
import pandas as pd

os.chdir("D:/dossier personelle/Lyon 2/M1/Data Mining/projet")

#Lire les données d'apprentissage
```

```
import pandas
data = pandas.read_table("data_avec_etiquettes.txt",sep="\t", header=0)
```

Vérification de la dimension et du type de nos données

```
#dimension du dataset
print(data.shape)

(494021, 200)

#description
data.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
```

Nous avons 494 021 observations et 200 colonnes dont la variable cible est V200.

Nous passons à l'étape de recodage des variables prédictives qualitatives

```
import numpy
colquali = [var for var in data.columns[:-1] if data[var].dtype == numpy.object_]
#liste des variables quali
print(colquali)

['V160', 'V161', 'V162']

#recodage en 0/1 ces variables
dataquali = pandas.get_dummies(data[colquali])

#liste des variables quantitatives
colquanti = [var for var in data.columns[:-1] if data[var].dtype != numpy.object_]

#constitution du nouveau dataframe
newdata = pandas.concat([dataquali,data[colquanti]],axis=1)

#rajoutons la variable cible
newdata['V200'] = data.V200
#dimension du nouveau dataframe
print(newdata.shape)

(494021, 277)

#description
print(newdata.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 494021 entries, 0 to 494020
Columns: 277 entries, V160_m1 to V200
dtypes: float64(15), int64(181), object(1), uint8(80)
memory usage: 780.2+ MB
None
```

Après le recodage des variables notre nouveau dataframe possède 277 attributs tout en sachant que le dernier est la variable cible.

On passe en suite à l'étape de la sélection des variables

```
#separation des X
x=newdata.iloc[:,:-1]

#sélection de variables - filtrage
from sklearn.feature_selection import SelectFwe,f_classif
sel = SelectFwe(f_classif,alpha=0.001)
sel.fit(newdata[newdata.columns[:-1]],newdata.V200)
SelectFwe(alpha=0.001)
```

Nous devons sauvegarder cette sélection pour l'appliquer à la base test pendant la prédiction et le scoring

```
#sauvegarde de la selection pour l'utiliser sur la base test

#librairie pour sauvegarde du modèle selection
import pickle

#référence du fichier - ouverture en écriture binaire
f = open("selection.sav","wb")

#sauvegarde dans le fichier référencé
pickle.dump(sel, f)

#fermeture du fuchier
f.close()

#liberons un peu d''espace mémoire
del data
del colquali
del colquanti
del dataquali

#lister les variables sélectionnées
print(len(newdata.columns[:-1][sel.pvalues_ < 0.001]))

114
```

Nous pouvons constater qu'il reste 114 variables pertinentes.

```
#construire la matrice X avec les var. sél. pour l'apprentissage
newX = sel.transform(x)
newX = pd.DataFrame(newX) #conversation en dataframe
newX['V200'] = newdata.V200 #ajoute de la colone cible
print(newX.shape)

(494021, 114)
```

Avant la création du modèle, nous avons pris 70 % de données pour l'apprentissage et 30 % pour tester notre modèle

```
#sepération de données en apprentissage et de test
from sklearn.model_selection import train_test_split
data, DT = train_test_split(newX,train_size=345800,random_state=3)

#centrer et réduire les données pour la régression multinomiale
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
```

```
#centrer et réduire
ZS = std.fit_transform(data[data.columns[:-1]])
```

Construction du modèle d'apprentissage

```
#construction du modèle

from sklearn.linear_model import LogisticRegression

#régressions multinomiale avec lbfgs -- données centrées et réduite
lr = LogisticRegression(multi_class='multinomial',solver='lbfgs')

#apprentissage
lr.fit(ZS,data.V200)


LogisticRegression(multi_class='multinomial')
```

Sauvegarde du modèle pour l'appliquer sur l'échantillon de test

```
#librairie pour sauvegarde du modèle
import pickle

#référence du fichier - ouverture en écriture binaire
f = open("mon_modele.sav","wb")

#sauvegarde dans le fichier référencé
pickle.dump(lr, f)

#fichier qu'il faut fermer
f.close()
```

Nous avons mesuré le taux d'erreur sur l'échantillon test pour évaluer notre modèle.

```
#prédiction
pred = lr.predict(std.fit_transform(DT.iloc[:,:-1]))

#taux d'err
print(numpy.mean(DT.V200 != pred))

0.0008635753368281148
```

Nous pouvons conclure de l'efficacité de notre modèle grâce à un taux d'erreur qui est très faible.

V. Déploiement

1. Système de classement

Pour répondre à cet objectif, les étapes suivantes illustrent notre démarche :

- Chargement de données sans étiquettes
- Le recodage des variables prédictives qualitatives

- La sélection des variables
- Chargement du modèle déjà crée
- Faire la prédiction et l'exportation du résultat
- Taux d'erreur sur un échantillon test de 1 482 063 observations

Le fichier **get_prediction.py** contient le code source de notre programme. Ci-dessous les traces de notre programme.

Chargement de données sans étiquettes

```
#dossier de travail
import os
os.chdir("D:/dossier personelle/Lyon 2/M1/Data Mining/projet")

#lire les données d'apprentissage
import pandas
testdata = pandas.read_table("data_test1.txt",sep="\t", header=0)

#dimension
print(testdata.shape)

(1482063, 199)
```

Recodage des variables

```
#liste des variables quali
import numpy
colquali = [var for var in testdata.columns if testdata[var].dtype == numpy.object
_]

#recodage en 0/1 ces variables
dataquali = pandas.get_dummies(testdata[colquali])

#liste des variables quantitatives
colquanti = [var for var in testdata.columns if testdata[var].dtype != numpy.objec
t_]

#constitution du nouvelle data frame
newdata = pandas.concat([dataquali,testdata[colquanti]],axis=1)
print(newdata.shape) #(1482063, 276)

(1482063, 276)

#liberer l'espace
del colquali, colquanti
del testdata, dataquali
```

Sélection des variables

```
#chargeons le module contenant la séléction appliquée sur la base d'apprentisage

#librairie chargement
import pickle

#ouverture en lecture binaire
f = open("selection.sav","rb")

#et chargement
sel = pickle.load(f)
```

```
#fermeture du fichier
f.close()

#réduction de la base test aux var. sélectionnées
#en utilisant le filtre
newX=sel.transform(newdata)

#verifions la nouvelle dimension
print(newX.shape) #(1482063, 113)

(1482063, 113)

#centrer et réduire les données
from sklearn.preprocessing import StandardScaler
std = StandardScaler()

#centrer et réduire
ZT = std.fit_transform(newX)
```

On effectue la prédiction

```
#chargeons notre modele de prediction
#librairie chargement du modèle
import pickle

#ouverture en lecture binaire
f = open("mon_modele.sav","rb")

#et chargement
lr = pickle.load(f)

#fermeture du fichier
f.close()

#prediction
pred = lr.predict(ZT)
```

On génère le fichier de sortie prediction.txt

```
#exportation
import numpy as np

np.savetxt("prediction.txt",pred, fmt='%s',header="prediction")
```

Du au manque de puissance de nos machines, nous étions incapables de travailler avec un méga fichier de 4 898 424 observations. Nous avons travaillé sur un fichier de 1 482 063 observations pour le test pour l'estimation de l'erreur.

```
#calcul le taux d'erreur

#import le Ytest

ytest=pandas.read_table("cible_test1.txt",sep="\t", header=0)

print(numpy.mean(ytest["V200"] != pred)) #0.0006700120035383111

0.0006983508798208984
```

b) le nombre d'erreurs de classement estimé sur une base de 4898424 obs = taux_d'erreur_calculé_base_test * 4898424 = 0.0008635753368281148*4898424 = **4230 erreurs estimés**

2. Système de scoring

Pour répondre à cet objectif, les étapes suivantes illustrent notre démarche :

- Chargement de données sans étiquettes
- Le recodage des variables prédictives qualitatives
- La sélection des variables
- Chargement du modèle déjà crée
- Effectuer le score et l'exportation du résultat
- Traçage de la courbe de gain
- Annoncer le nombre de positifs parmi les 10 000 observations

Le fichier get_score.py contient le code source de notre programme.

Les quatre premiers points restent les mêmes que pour la partie prédiction. Nous présentons ici la suite de la partie ciblage.

```
#prediction proba
probas = lr.predict_proba(ZT)
#print(probas)

#liste des classes
print(lr.classes_)

['m1' 'm10' 'm11' 'm12' 'm13' 'm14' 'm15' 'm16' 'm17' 'm18' 'm19' 'm2'
 'm20' 'm21' 'm22' 'm23' 'm3' 'm4' 'm5' 'm6' 'm7' 'm8' 'm9']
```

La modalité positive est m16. On peut l'identifier à la 8 ème position. Mais nous pouvons la récupérer de la manière suivante

```
result = numpy.where(lr.classes_ == 'm16')
mod=result[0]
print(mod)

[7]
```

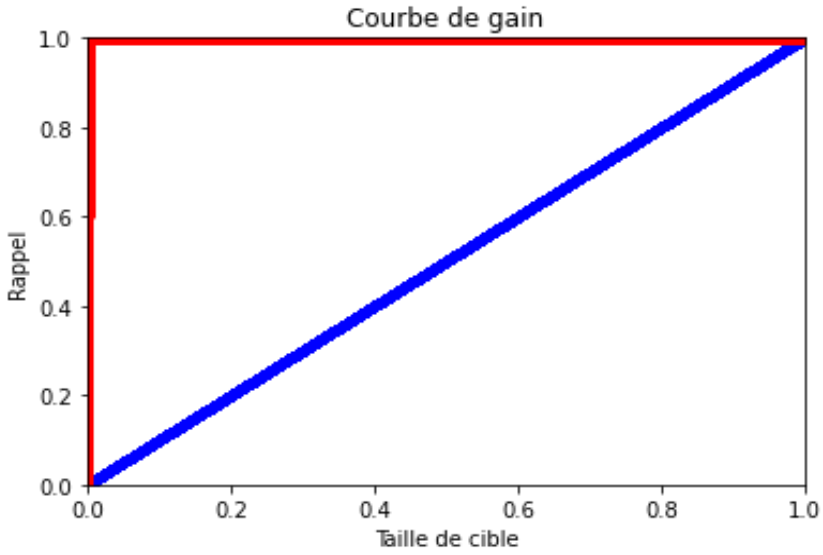
Exporter le score de la modalité positive.

```
#score de 'positif' m16
score = probas[:,7]

#exportation du score
import numpy as np
np.savetxt("score.txt",score, fmt='%s',header="score")

#import le Ytest

ytest=pandas.read_table("cible_test1.txt",sep="\t", header=0)
```

```
#transf. en 0/1 de Y_test
pos = pandas.get_dummies(ytest).values
#colonne de positif
pos = pos[:,7]

#nombre total de positif
import numpy
npos = numpy.sum(pos)
print(npos) #3120

3120

#index pour tri selon le score croissant
index = numpy.argsort(score)
#inverser pour score décroissant
index = index[::-1]
#tri des individus (des valeurs 0/1)
sort_pos = pos[index]
#somme cumulée
cpos = numpy.cumsum(sort_pos)
#rappel
rappel = cpos/npos
#nb. obs ech.test
n = ytest.shape[0] #1482063 observations
print(n)
#taille de cible
taille = numpy.arange(start=1,stop=n+1,step=1)
#passer en proportion
taille = taille / n

1482063
```

Traçage du graphe

```
#graphique
import matplotlib.pyplot as plt
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()
```




```
#vrai positif dans le 10 000 premier

val=int(10000*npos/n)

prop_pos = rappel[val]

#on multiple par le nombre de positifs
print(prop_pos * npos) #22

22.0
```

Donc on peut tirer conclusion qu'il existe 22 positifs parmi les 10 000 observations qui présentent les scores les plus élevés dans une base(dupliquée) de 1 482 063 obs.

3. Système de classement variable cible modifié V200_Prim

Il nous est demandé de regrouper les modalités de la variable cible puis de construire un modèle permettant de prédire de la manière la plus précise possible cette nouvelle variable cible.

Regroupement de modalités :

Après des recherches sur comment regrouper des modalités en vue d'une prédiction, nous sommes tombés sur le **clustering** ou partitionnement des données qui est une méthode en analyse des données. Elle vise à diviser un ensemble de données en différents « paquets » homogènes, en ce sens que les données de chaque sous-ensemble partagent des caractéristiques communes, qui correspondent le plus souvent à des critères de proximité (similarité informatique) que l'on définit en introduisant des mesures et classes de distance entre objets. (définition Wikipédia)

Le clustering est généralement utilisé pour de l'apprentissage non supervisé, Nous avions essayé deux algorithmes de clustering :

-Le CAH : classification ascendante hiérarchique

-Le K-means : partitionnement en K-moyenne

Pour le CAH, Nous n'avons pas réussi à afficher un **dendrogramme** à cause de la trop grande quantité de données a traité. Méthode testé : **scipy.cluster.hierarchy.linkage** de la librairie **scipy**.

Nous avons donc décidé d'utiliser la méthode des **k-means** qui pouvez elle supporter un très grand volume de données.

Procédure :

1) Nous avons pris les données avec étiquettes et nous avons opéré dessus du prétraitement :

-Recoder les variables qualitatives en 0/1 y compris la variable cible avec pandas.get_dummies

-Une sélection de variable avec SelectFwe de sklearn

- Nous avons ensuite centrée et réduit les données avec StandardScaler de sklearn(base apprentissage)

2)Une fois le prétraitement fait, nous avons séparé les données en deux : une base d'apprentissage et une base de test.

3)Nous avons importé les librairies **k-means** et **KElbowVisualizer** de respectivement sklearn et yellowbrick

Nombre de groupes

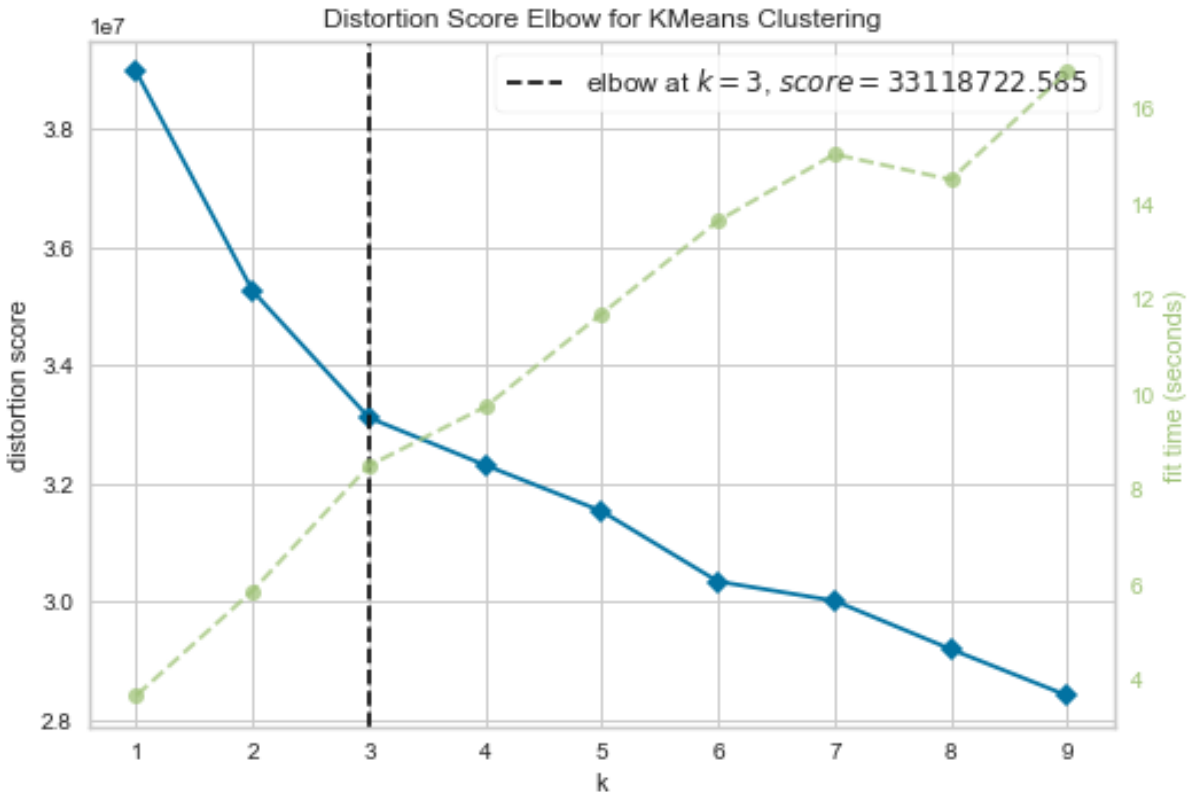
Grace a KElbowVisualizer , bous avons pu observer le K-optimal grâce a deux algorithmes :

-La methode de la « silhouette »

-La methode du « elbow ou coude »

Nous avons testé les deux méthodes et nous avons finalement gardé la méthode de calcul du k-optimal par l'elbow

En l'appliquant a nos données nous obtenons

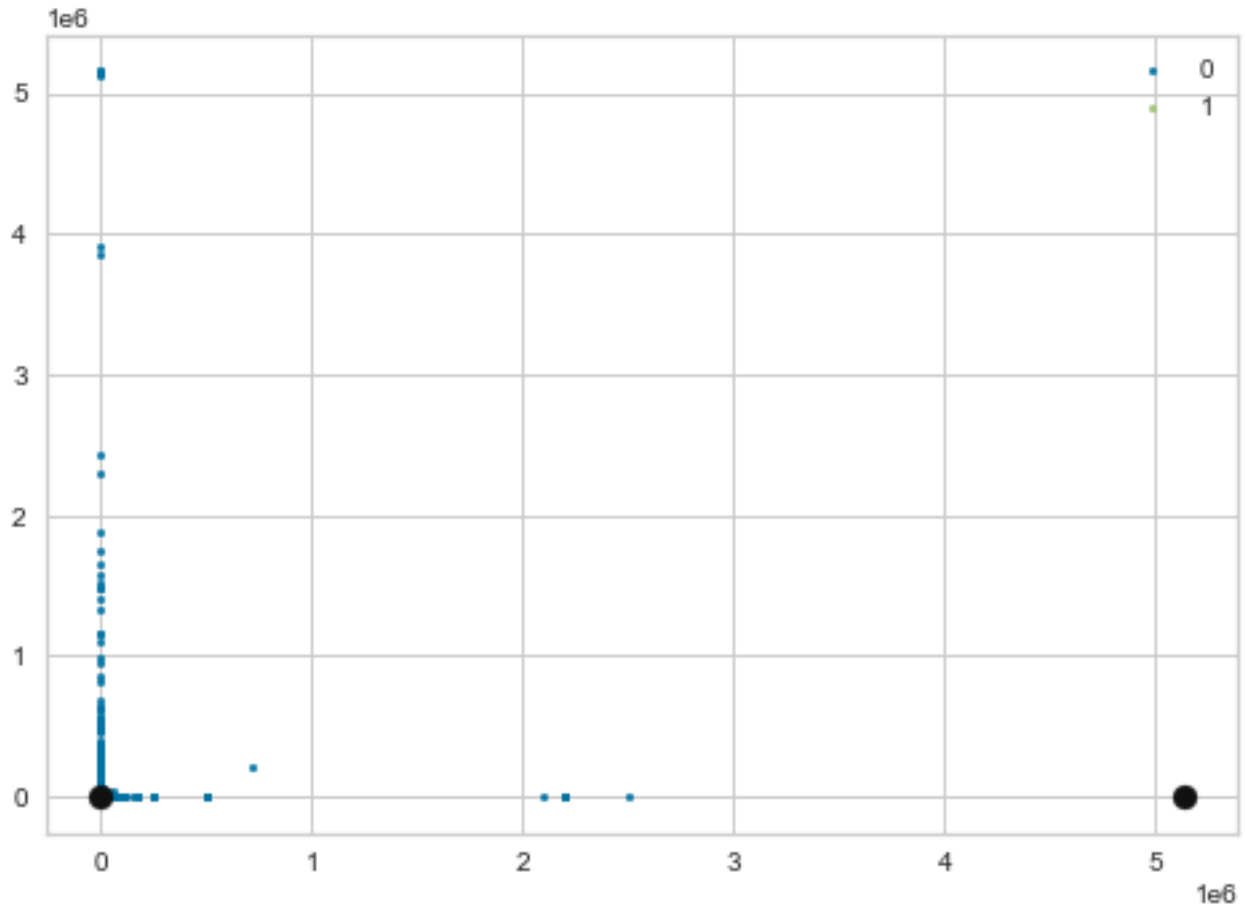


Conclusion : Le k-optimal de notre algorithme k-mean est 3 donc nous aurons 3 clusters

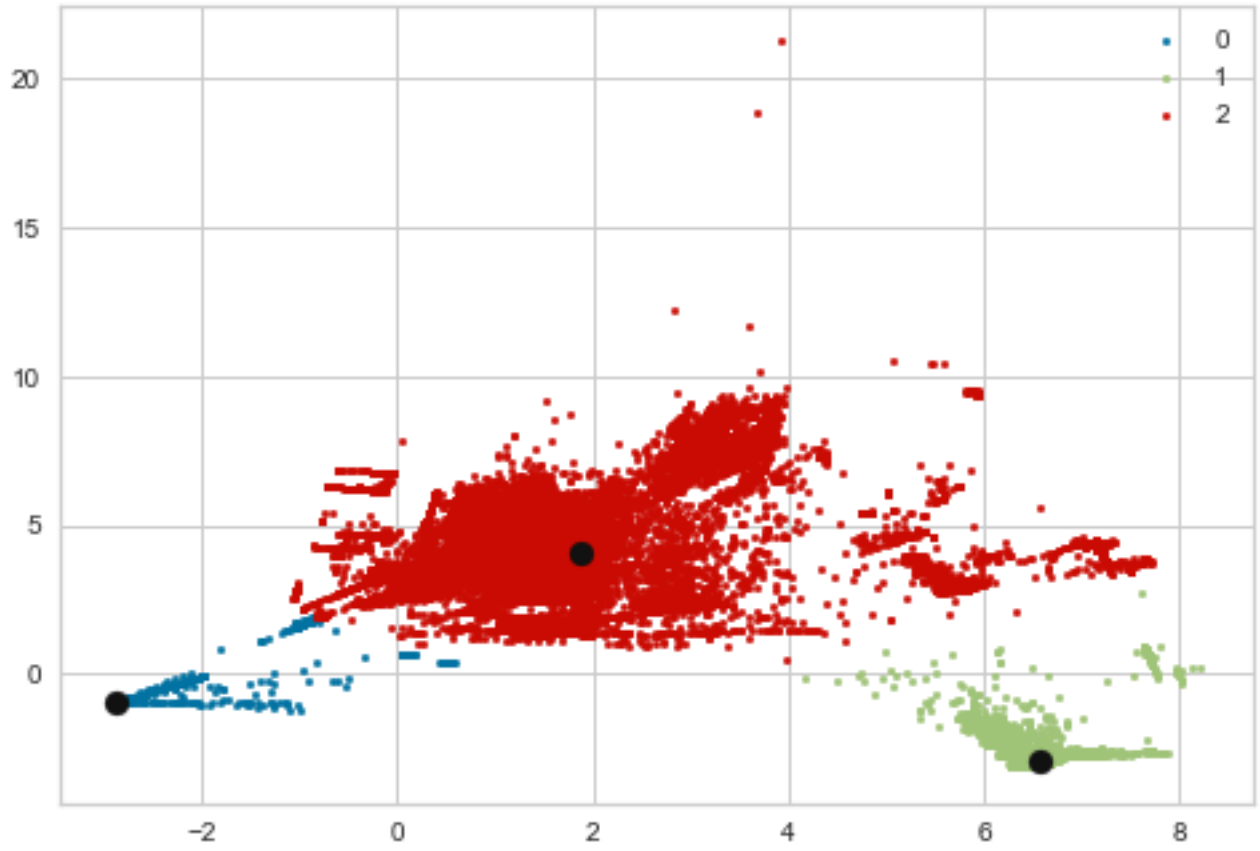
Essayons de visualiser les données sur un plan factorielle :

Pour cela nous réduisons d'abord la dimension de nos données avec une **PCA (Principal component analysis)** de sklearn

Avec la méthode du K-means de paramètre 3 clusters , nous récupérons les centroïdes et les positions des points



Résultat sans centrer et réduire les données



Résultat en centrant et réduisant les données

Nous obtenons 3 clusters déterminons maintenant le nombre de groupe et leurs constitutions.

Pour trouver les groupes nous entrainons un modèle avec la méthode fit(X) de k-means

X contient toutes les lignes et colonnes de la base apprentissage y compris la variable cible

Puis Nous prédisons grâce à la méthode Predict(X) de k-means le cluster d'appartenance de chaque modalité de V200 avec X : la base test

Après plusieurs tests avec différents K : 5,8,2,3…, Nous remarquons que deux groupes de modalités sont toujours prédit dans les mêmes clusters peu importe le nombre de clusters hormis 2 ou nous obtenons qu'un seul cluster toujours prédit peu importe la modalité

1) **(m4,m5,m6,m7,m10,m11,m12,m13,m16,m17,m18,m19,m20,m21,m22)**

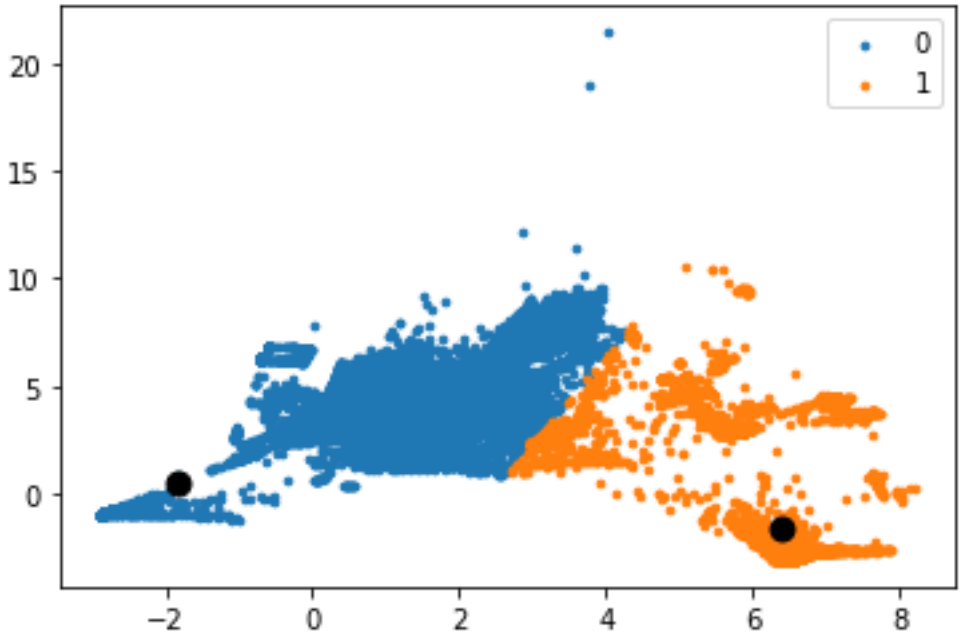
2) **(m1, m2,m9,m14,m23,m3,m8,m15)**

Pour K = 3 deux clusters étaient toujours prédits :

Le premier ensemble était dans le cluster 0 et

Le deuxième ensemble dans le cluster 2

Pour K = 2 un cluster était toujours prédit :



Le cluster 0 était toujours prédit

Le cluster 1 jamais prédit

Conclusion finale

Nous pouvons séparer les modalités en deux groupes comme ci-dessous car ils sont a chaque fois prédit dans les mêmes clusters indépendamment du nombre de clusters.

1) **(m4,m5,m6,m7,m10,m11,m12,m13,m16,m17,m18,m19,m20,m21,m22)**

2) **(m1, m2,m9,m14,m23,m3,m8,m15)**

Prédictions

Pour la prédiction nous pouvons le faire de deux manières

1) Nous pouvons réappliquer notre modèle en transformant la variable cible de V200 a V200_Prim (2 modalités) et refaire un apprentissage puis les prédictions, ou bien nous pouvons utiliser un autre modèle d'apprentissage supervisé.

2) La deuxième manière serait d'enlever les variables cibles binarisées de notre base d'apprentissage pour le k-mean, entrainer ce modèle du k-mean à prédire les groupes d'appartenance (**méthode fit(X))** puis faire les prédictions sur DB avec la (**méthode predict(X)**)

Nous avons choisi la première solution pour notre projet en réappliquant le même modèle qu'a la question 1 (*Régression logistique multinomial avec solver = lbfgs*) sur notre base avec la nouvelle variable cible.

Paramètre (solver)	*Temps de prédiction*	*Taux d'erreur(base_test)*
Analyse discriminante		
svd	5.85 s	0.00246
lsqr	5.41 s	0.00246
Régression logistique		
lbfgs	6.76 s	0.000317

Tableau 3 : comparaison des modèles

Le fichier **get_modelisation_Vprim.py** contient le code source permettant de créer le modèle.

Le fichier **get_prediction_Vprim.py** contient le code source permettant de prédire le groupe d'appartenance.

En sortie du programme **get_prediction_Vprim.py** nous avons deux fichiers :

-sorties1.txt : Les prédictions c.-à-d. groupe1 ou groupe2

-sorties2.txt : les modalités présentes dans chaque groupe

VI. Conclusion

Dans ce travail, nous avons eu à créer et expérimenter des modèles dans le but de répondre de la manière la plus précise possible aux problèmes donnés.

Différentes manières de méthodes, algorithmes, visualisation, approches ont été entrepris dans ce but-là.

Nous avons finalement opté pour l'utilisation d'une régression logistique paramétrée pour le classement et la méthode du k-means pour le regroupement des modalités de la variable cible car étant les plus rapides et performants des méthodes utilisées avec nos données.

Ce travail nous a permis d'améliorer nos connaissance et compétences dans les domaines de l'apprentissage supervisé et non supervisé et de la programmation plus généralement.

Les principales difficultés rencontrées étaient : le volume de données qui rendait certaines tâches et calculs très lents (plusieurs heures pour certains tests), le fait de ne pas forcement avoir d'indicateur de bonnes performances sur la base de déploiement (une navigation a l'aveugle).

Ce projet a été pour nous une véritable nouvelle source de connaissance dans son ensemble et un grand défi dans sa réalisation.