

Projet Data Mining

Classement et ciblage

DORONZO Franck, DUDOIT Romain, COUSTANCE Nathan
M1 INFORMATIQUE - UNIVERSITÉ LUMIÈRE LYON 2

28 décembre 2020

Table des matières

1	Compréhension des données et outils utilisés	2
2	Préparation des données	2
3	Systèmes de classement	4
3.1	Modélisation	4
3.1.1	Analyse discriminante	4
3.1.2	Régression logistique	5
3.1.3	Arbre de décision	6
3.2	Évaluation	7
3.2.1	Analyse discriminante	7
3.2.2	Régression logistique	10
3.2.3	Arbre de décision	11
3.3	Déploiement	13
4	Système de scoring	15
4.1	Modélisation	15
4.1.1	Analyse discriminante	15
4.1.2	Régression Logistique	15
4.2	Évaluation	16
4.2.1	Analyse discriminante	16
4.2.2	Régression Logistique	16
4.3	Déploiement	18
5	Regroupement des modalités	20

1 Compréhension des données et outils utilisés

Nous disposons d'un fichier de données "data_avec_etiquettes.txt" de 200 variables pour 494021 observations pour lequel nous ne connaissons ni la signification des variables ni du contexte dans lequel les données ont été obtenues. Nous cherchons dans le cadre de ce projet à prédire, sur un gros volume de données, les valeurs de la variable V200 (variable cible) à partir des données des 199 premières variables (variables explicatives) de ce fichier.

Nous avons choisit de travailler avec le langage Python et utilisé les IDE Spyder et Jupyter Notebook. Au cours de notre travail, nous avons réfléchi à l'utilisation d'autres outils en raison des problèmes de capacités en mémoire vive que nous avons rencontrés sur nos machines en voulant tester nos modèles sur un fichier de taille supérieure à 2go. Nous avons néanmoins réussi à dépasser ce problème en optimisant l'importation des données.

2 Préparation des données

Avant de commencer à créer des modèles de prédictions, nous allons procéder à une sélection de variables afin de prédire au mieux la variable cible. Pour faire cela, différentes méthodes existent et nous avons choisi d'appliquer l'élimination progressive (Backward Elimination) qui consiste à calculer la p-value de chaque variable afin de déterminer sa pertinence dans la prédiction de la variable cible.

Une fois le calcul effectué on va chercher si p-value dépasse les 5% (0.05) et si c'est le cas on va supprimer la variable qui a la plus grande p-value, recalculer toutes les nouvelles valeurs et recommencer l'opération de suppression.

Une fois que toutes nos variables restantes ont une p-value inférieure ou égale à 5% alors on garde ces dernières.

Ci-dessous le code python utilisé pour la Backward Elimination :

```
data = pd.read_csv("./data/data_avec_etiquettes.txt", sep="\t")

encoder = OrdinalEncoder()
encoder.fit(data[["V160", "V161", "V162", "V200"]])
data[["V160", "V161", "V162", "V200"]] =
    encoder.transform(data[["V160", "V161", "V162", "V200"]])

X = data.iloc[:,0:199]
y = data.iloc[:,199]

cols = list(X.columns)
pmax = 1
while (len(cols) > 0):
```

```

p = []
X_1 = X[cols]
X_1 = sm.add_constant(X_1)
model = sm.OLS(y.astype(float), X_1.astype(float)).fit()
p = pd.Series(model.pvalues.values[1:], index=cols)
pmax = max(p)
feature_with_p_max = p.idxmax()
if(pmax>0.05):
    cols.remove(feature_with_p_max)
else:
    break

selected_features_BE = cols
print(selected_features_BE)

```

Avant d'arriver à ce choix, nous avons essayé d'autres méthodes comme SelectKBest de scikit-learn qui va sélectionner les K meilleurs scores parmi nos variables ou encore Extra-TreesClassifier qui utilise des arbres de décision afin de produire un modèle prédictif ainsi qu'une valeur d'importance associée à chaque variable, ce qui nous permet de sélectionner encore une fois les K meilleures. Si notre choix s'est porté sur la méthode Backward Elimination c'est parce qu'elle nous a fourni une sélection de variable plus satisfaisante que les autres méthodes pour les calculs de prédiction que nous avons effectué. Nous nous retrouvons donc finalement avec 41 variables pour prédire notre variable cible :

```

# ['V14', 'V84', 'V96', 'V159', 'V160', 'V161', 'V162', 'V163', 'V164',
#  'V165', 'V166', 'V167', 'V168', 'V169', 'V170', 'V171', 'V172',
#  'V173', 'V174', 'V175', 'V176', 'V177', 'V180', 'V181', 'V182',
#  'V183', 'V184', 'V185', 'V186', 'V187', 'V188', 'V189', 'V190',
#  'V191', 'V192', 'V193', 'V194', 'V195', 'V197', 'V198', 'V199']

```

Après avoir sélectionné nos variables, nous avons séparé nos données en échantillons d'entraînement et de test avec une proportion respective de 70%/30%. Nous avons nommé ces variables X_train, y_train, X_test, y_test avec X les variables explicatives qui permettront de prédire la variable cible y.

Les variables V160, V161, V162 avaient également besoin d'un encodage pour les transformer en variables quantitatives puisque bien souvent les modèles ne peuvent traiter les variables qualitatives. Nous avons utilisé l'encoder LabelEncoder de scikit-learn pour coder nos variables explicatives catégorielles. Nous aurions normalement dû utiliser l'encoder OneHotEncoder car il n'y a pas nécessairement d'ordre de grandeur à notre connaissance

entre les différentes modalités. Le problème se serait notamment posé pour la variable V161 qui comporte un nombre de modalité conséquent et qui donc aurait induit un nombre trop important de variable à ajouter sur le jeu de données qui comporte déjà beaucoup de variables. Le LabelEncoder n'a manifestement pas posé de problèmes pour faire de bonnes prédictions et nous l'avons donc gardé tel quel (excepté lorsque nous avons travaillé avec un arbre de décision avec scikit-learn où nous avons voulu utiliser les deux (DecisionTreeClassifier n'acceptant malheureusement pas les variables catégorielles)).

3 Systèmes de classement

Nous cherchons dans un premier temps à produire un système de classement qui permet de prédire le plus précisément possible la variable cible V200. Nous avons exploré différents modèles qui utilisent soit un système à base de règle, soit un système qui s'exprime à l'aide de combinaisons linéaires. Nous allons pour chacun d'entre eux expliquer leur mise en œuvre et les évaluer afin de choisir un modèle définitif à appliquer.

3.1 Modélisation

3.1.1 Analyse discriminante

Nous pouvons entraîner notre modèle par le biais de la fonction «LinearDiscriminantAnalysis» qui est intégrée dans la librairie sklearn. Pour terminer, la librairie pickle nous a permis d'enregistrer le modèle dans un fichier .sav qui sera à nouveau utilisé lors du déploiement. Afin d'évaluer notre modèle nous allons donc appliquer notre prédiction sur les variables _test afin d'évaluer la pertinence de ce dernier.

```
import numpy as np
import pandas
import os
os.chdir("chemin d'accès")

cols=['V14', 'V84', 'V96', 'V159', 'V160', 'V161', 'V162', 'V163',
      'V164', 'V165', 'V166', 'V167', 'V168', 'V169', 'V170', 'V171',
      'V172', 'V173', 'V174', 'V175', 'V176', 'V177', 'V180', 'V181',
      'V182', 'V183', 'V184', 'V185', 'V186', 'V187', 'V188', 'V189',
      'V190', 'V191', 'V192', 'V193', 'V194', 'V195', 'V197', 'V198',
      'V199', 'V200']
df =pandas.read_table("data_avec_etiquettes.txt",delimiter='\t
↵',usecols=cols)

from sklearn import preprocessing
le = preprocessing.LabelEncoder()
X=pandas.get_dummies(X, columns=['V160','V162'],drop_first=True)
X[["V161"]]=X[["V161"]].apply(LabelEncoder().fit_transform)
```

```

X = df.iloc[:,0:41]
y = df.iloc[:,41]
y=y.to_numpy().reshape(-1)

from sklearn import model_selection
X_train,X_test,y_train,y_test=model_selection.train_test_split(X,y,test_
    ↪size=0.3,random_state=42,stratify=y)

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(store_covariance = True)
lda.fit(X_train,y_train)

import pickle

f=open("LDA.sav","wb")
pickle.dump(lda,f)
f.close()

```

3.1.2 Régression logistique

La régression logistique étant un modèle long à exécuter, la sélection des variables a été très utile pour cette partie afin d'effectuer un maximum de tests. Ce modèle s'entraîne sur `X_train` et `y_train` puis nous retourne sa prédiction grâce à la fonction `predict()`.

```

from sklearn.linear_model import LogisticRegression

startTime = datetime.now()

# Encodage des variables qualitatives
encoder = OrdinalEncoder()
encoder.fit(data[["V160", "V161", "V162"]])
data[["V160", "V161", "V162"]] =
    encoder.transform(data[["V160", "V161", "V162"]])

# Séparation des données et de la variable cible
X = data.iloc[:,0:41]
y = data.iloc[:,41]

# Séparation en données de train et de test

```

```

X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Déclaration et entraînement de notre modèle
regressor = LogisticRegression(solver='liblinear')
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

```

Ici on a la variable `y_pred` qui contient les valeurs prédites par notre modèle à partir de `X_test` que l'on va pouvoir comparer avec `y_test`.

3.1.3 Arbre de décision

Après avoir travaillé avec des classificateurs linéaires nous avons voulu faire de même avec un arbre de décision. Son avantage par rapport aux autres méthodes est qu'il permet, par sa simplicité, de bien comprendre et de visualiser les règles d'affectation des objets à une classe d'une variable qualitative. Ils sont également rapides à entraîner et nécessitent peu de pré-traitements.

Pour sa mise en oeuvre, nous avons utilisé la classe `DecisionTreeClassifier` de la librairie `scikit-learn` qui prend en entrée deux tableaux, le premier étant celui des données des variables explicatives de l'échantillon d'apprentissage et le second un tableau d'une dimension qui contient les labels des classes de la variable cible. Comme pour les modèles précédents nous avons utilisés les mêmes variables explicatives déterminées à l'étape de préparation des données.

```

import pickle
import pandas as pd
import numpy as np
from sklearn import model_selection
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

# Variables pertinentes
cols = ['V14', 'V84', 'V96', 'V159', 'V160', 'V161', 'V162', 'V163',
        'V164', 'V165', 'V166', 'V167', 'V168', 'V169', 'V170',
        'V171', 'V172', 'V173', 'V174', 'V175', 'V176', 'V177',
        'V180', 'V181', 'V182', 'V183', 'V184', 'V185', 'V186',
        'V187', 'V188', 'V189', 'V190', 'V191', 'V192', 'V193',
        'V194', 'V195', 'V197', 'V198', 'V199', 'V200']

# variables explicatives
X = pd.read_csv('data_avec_etiquettes.txt', delimiter='\t', usecols=cols)

```

```
[: -1])

# variable cible
y = pd.read_csv('data_avec_etiquettes.txt', delimiter='\t', usecols=[cols
[-1]])

y=y.to_numpy().reshape(-1)

# recodage des variables qualitatives
X=pd.get_dummies(X, columns=['V160', 'V162'], drop_first=True)
X[["V161"]]=X[["V161"]].apply(LabelEncoder().fit_transform)

# Division apprentissage/test
X_train,X_test,y_train,y_test=model_selection.train_test_split(X,y,
test_size=0.3,random_state=42,stratify=y)

# Apprentissage sur le train set
dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train) # met un peu de temps

# Enregistrement du modèle
f=open("dtree.sav", "rb")
dtree=pickle.load(f)
f.close()
```

3.2 Évaluation

3.2.1 Analyse discriminante

Pour commencer regardons l'évaluation globale du modèle après préparation des données.

MANOVA

Stat	Value	p-value
Wilks' Lambda	0,0000	-
Bartlett -- C(902)	12177509,6091	0,0000
Rao -- F(902, 9586820)	27224,3922	0,0000

On constate que le modèle est significatif à 5%, on peut donc dire que l'analyse discriminante est viable dans ce contexte.

Attribute	Wilks L.	Partial L.	F(22,4939 58)	p-value
V14	0	0,999944	1,264	0,18214
V84	0	0,999931	1,54243	0,0499
V96	0	0,999947	1,18611	0,247659
V159	0	0,983296	381,42568	0
V160	0	0,632243	13060,02	0
V161	0	0,94974	1188,1925	0
V162	0	0,629176	13233,162	0
V163	0	0,996617	76,20529	0
V164	0	0,418008	31260,749	0
V165	0	0,047142	453819,35	0
V166	0	0,005561	4015155,1	0
V167	0	0,894757	2640,9196	0
V168	0	0,799445	5632,6509	0
V169	0	0,477688	24550,122	0
V170	0	0,712182	9073,8889	0
V171	0	0,895521	2619,5141	0
V172	0	0,619385	13797,254	0
V173	0	0,913534	2125,1516	0
V174	0	0,896837	2582,7226	0
V175	0	0,980007	458,06205	0
V176	0	0,908473	2262,0513	0
V177	0	0,990074	225,09655	0
V180	0	0,917347	2022,9832	0
V181	0	0,850302	3952,8477	0
V182	0	0,414731	31685,208	0
V183	0	0,952304	1124,5261	0
V184	0	0,954299	1075,2544	0
V185	0	0,952301	1124,6013	0
V186	0	0,959882	938,39029	0
V187	0	0,370187	38199,43	0
V188	0	0,55436	18049,254	0
V189	0	0,957747	990,53679	0
V190	0	0,925238	1814,2419	0
V191	0	0,881123	3029,2174	0
V192	0	0,883731	2953,9935	0
V193	0	0,744945	7687,3487	0
V194	0	0,844863	4122,8286	0
V195	0	0,479915	24332,016	0
V197	0	0,988225	267,52793	0

Grâce à l'outil TANAGRA, nous sommes en capacité de mesurer l'importance des variables au sein de notre analyse discriminante. Premièrement en regardant les p-values : plus celles-ci sont petites et plus les variables sont importantes. Mais aussi avec la colonne Partial L. : on peut en déduire le R carré partiel en faisant (1 - Partial L.) Ainsi plus le R carré partiel d'une variable est grand et plus cette même variable est importante dans le modèle.

```
dfypred=pandas.DataFrame({'y_test':y_test,'prediction':ypred})
dfypred[dfypred['y_test']!=dfypred['prediction']]
```



```

from sklearn import metrics
#matrice de confusion
mc = metrics.confusion_matrix(y_test,ypred)
print(mc)

#calcul du taux d'erreur
print("taux d'erreur "+str(1.0-metrics.accuracy_score(y_test,ypred)))

#calcul des sensibilité et précision par classe
print(metrics.classification_report(y_test,ypred))

```

Ainsi nous avons comme mesure :

```

In [22]: dfypred[dfypred['y_test']!=dfypred['prediction']]
Out[22]:
   y_test prediction
232    m18      m10
273     m6       m1
344    m12      m19
887    m12       m1
1599   m12       m9
...     ...      ...
147204 m16      m18
147446 m11       m6
147791 m12      m16
147804 m18      m12
148075 m12       m1
[697 rows x 2 columns]

```

Le nombre d'erreur pour la partie test de notre analyse discriminante est de 697 pour un dataframe qui correspond à 30% du nombre de ligne totale soit 148206 lignes sur notre base test.

```

In [26]: print("taux d'erreur "+str(1.0-metrics.accuracy_score(y_test,ypred)))
taux d'erreur 0.004702881780212831

```

Ainsi le taux d'erreur de notre analyse discriminante après la préparation de nos variables est de 0.0048

	precision	recall	f1-score	support
m1	0.91	0.96	0.93	661
m10	1.00	1.00	1.00	32160
m11	0.76	0.32	0.45	69
m12	0.99	0.99	0.99	29184
m13	0.50	1.00	0.67	1
m14	0.10	1.00	0.18	1
m15	1.00	1.00	1.00	79
m16	0.80	0.79	0.79	312
m17	0.00	0.00	0.00	3
m18	0.89	0.88	0.89	477
m19	1.00	1.00	1.00	84237
m2	0.00	0.00	0.00	9
m20	0.00	0.00	0.00	1
m21	1.00	1.00	1.00	294
m22	0.76	0.88	0.82	306
m23	0.80	0.67	0.73	6
m3	1.00	0.50	0.67	2
m4	0.80	1.00	0.89	16
m5	0.17	0.50	0.25	4
m6	0.84	0.98	0.91	374
m7	1.00	1.00	1.00	6
m8	0.00	0.00	0.00	3
m9	0.00	0.00	0.00	2
accuracy			1.00	148207
macro avg	0.62	0.67	0.62	148207
weighted avg	1.00	1.00	1.00	148207

Grâce à la fonction classification report nous pouvons constater les performance globale de notre modèle. Mais aussi celle-ci nous donne accès à la précision, la sensibilité de chaque modalité de la variable cible. Ce qui va nous permettre dans un second temps de définir le modèle définitif.

3.2.2 Régression logistique

Afin de savoir si la régression logistique est fiable ou non, on exécute ce code :

```
print('Test Score: ', regressor.score(X_test, y_test))
print('Temps d\'exécution: ', datetime.now() - startTime)
```

Qui nous retourne :

```
# Test Score: 0.9883676209625726
# Temps d'exécution: 0:05:07.484262
```

On a donc une précision de 98,84% pour un temps d'exécution de 5m07s. Une telle précision insinue une erreur toutes les 100 valeurs et cela reste un taux d'erreur élevé par rapport au

taux des autres modèles.

3.2.3 Arbre de décision

Nous repartons du fichier de la section modélisation de l'arbre de décision. Procédons à l'évaluation du modèle :

```
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, classification_report
```

On peut comparer le nombre d'observation de chaque modalité entre la variable cible et la prédiction.

```
pred=dtree.predict(X_test)df=pd.DataFrame({'y_test':pd.Series(y_test).
    ↳value_counts(),'pred':pd.Series(pred).value_counts()})
df=df.fillna(0)
df=df.astype(int)
df
```

	y_test	pred
m1	661	661
m10	32160	32162
m11	69	71
m12	29184	29181
m13	1	2
m14	1	1
m15	79	79
m16	312	307
m17	3	0
m18	477	477
m19	84237	84236
m2	9	12
m20	1	0
m21	294	294
m22	306	310
m23	6	6
m3	2	2
m4	16	20
m5	4	5
m6	374	372
m7	6	6
m8	3	1
m9	2	2

On constate par exemple que les modalités m17 et m20 n'ont jamais été prédites. On peut voir aussi que sur les 23 modalités, 9 ont été parfaitement prédites.

```
print(df[df['y_test']==df['pred']].shape[0])
```

9

On peut afficher le nombre de mauvaises affectations des modalités à la variable cible, soit 79 :

```
df=pd.DataFrame({'y_test':y_test,'prediction':pred})
df[df['y_test']!=df['prediction']].shape[0]
```

79

Ci dessous l'obtention du taux de succès et du taux d'erreur :

```
# Taux de succès
print("Taux de précision : ",accuracy_score(y_test,pred))

#Calcul du taux d'erreur
print("Taux d'erreur : "+str(1.0-accuracy_score(y_test,pred)))
```

Taux de précision : 0.999466961749445

Taux d'erreur : 0.0005330382505549514

Ci dessous les variables les plus importantes classées selon leur indice de Gini :

```
#Importance des variables
imp = {"VarName":X_train.columns,"Importance":dtree.feature_importances_}
print(pd.DataFrame(imp).sort_values(by="Importance",ascending=False))
```

	VarName	Importance
22	V182	0.605098
27	V187	0.325979
13	V171	0.012993
33	V193	0.011618
5	V163	0.007944
35	V195	0.007213
36	V197	0.006502
8	V166	0.006251

L'arbre étant très grand, il ne convient pas à l'affichage. On peut néanmoins afficher les règles qui ont été utilisées pour faire les prédictions dont en voici une partie :

```
from sklearn.tree.export import export_text
r=export_text(dtree,feature_names = list(X_train.columns))
print(r)
```

```

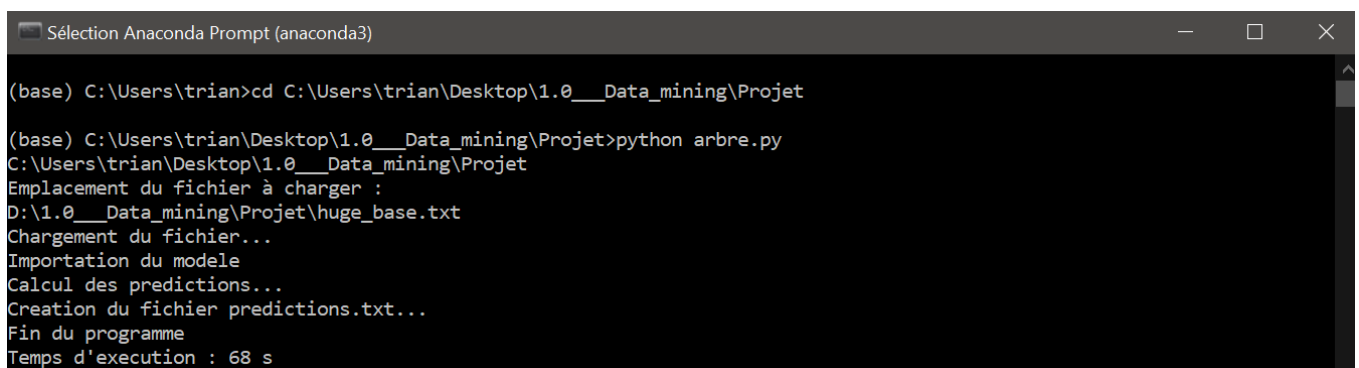
|--- V182 <= 316.50
|   |--- V187 <= 0.33
|   |   |--- V193 <= 0.16
|   |   |   |--- V163 <= 0.50
|   |   |   |   |--- V194 <= 0.06
|   |   |   |   |   |--- V188 <= 0.93
|   |   |   |   |   |   |--- V194 <= 0.05
|   |   |   |   |   |   |   |--- class: m10
|   |   |   |   |   |   |   |--- V194 > 0.05
|   |   |   |   |   |   |   |--- V187 <= 0.09
|   |   |   |   |   |   |   |   |--- class: m16
|   |   |   |   |   |   |   |   |--- V187 > 0.09
|   |   |   |   |   |   |   |   |--- class: m10
|   |   |   |   |   |--- V188 > 0.93
|   |   |   |   |   |--- V162_m2 <= 0.50
|   |   |   |   |   |   |--- class: m10
|   |   |   |   |   |   |--- V162_m2 > 0.50
|   |   |   |   |   |   |--- class: m18
|   |   |   |   |--- V194 > 0.06

```

3.3 Déploiement

Dans la section évaluation, nous avons donc évalué nos modèles d'analyse discriminante, de régression logistique et d'arbre de décision. Nous avons pour chacun d'entre eux obtenus de très bonnes prédictions après avoir fait un choix judicieux des variables à utiliser.

C'est le modèle de l'arbre de décision qui a obtenu le meilleur taux de succès. De plus, son temps d'exécution est relativement court comparé aux autres. Nous avons testé son déploiement sur une base test bien plus conséquente de 4898424 observations que nous avons obtenu en dupliquant notre échantillon d'apprentissage autant de fois que nécessaire. Nous avons exécuter notre programme en ligne de commande dont voici une capture avec les différentes étapes et le temps d'exécution de ce dernier qui à été de 69 secondes, avec en sortie un fichier prediction.txt.



```

Sélection Anaconda Prompt (anaconda3)

(base) C:\Users\trian>cd C:\Users\trian\Desktop\1.0__Data_mining\Projet

(base) C:\Users\trian\Desktop\1.0__Data_mining\Projet>python arbre.py
C:\Users\trian\Desktop\1.0__Data_mining\Projet
Emplacement du fichier à charger :
D:\1.0__Data_mining\Projet\huge_base.txt
Chargement du fichier...
Importation du modele
Calcul des predictions...
Creation du fichier predictions.txt...
Fin du programme
Temps d'execution : 68 s

```

Ci-dessous le code en question :

```

import os
import time
import numpy as np
import pandas as pd
import pickle
from sklearn.preprocessing import LabelEncoder

os.chdir(os.path.dirname(__file__))

file_path = input('Chemin absolu du fichier à charger : \n')

start_time = time.time()

# Variables explicatives pertinentes obtenus en amont
#après une sélection de variables
cols = ['V14', 'V84', 'V96', 'V159', 'V160', 'V161', 'V162', 'V163',
        'V164', 'V165', 'V166', 'V167', 'V168', 'V169', 'V170',
        'V171', 'V172', 'V173', 'V174', 'V175', 'V176', 'V177',
        'V180', 'V181', 'V182', 'V183', 'V184', 'V185', 'V186',
        'V187', 'V188', 'V189', 'V190', 'V191', 'V192', 'V193',
        'V194', 'V195', 'V197', 'V198', 'V199', 'V200']

print("Chargement du fichier...")
#data = pd.read_csv(path,delimiter='\t',usecols=cols,nrows=4898424)

# recodage des variables qualitatives
data=pd.get_dummies(pd.read_csv(file_path,delimiter='\t',
usecols=cols,nrows=4898424),columns=['V160', 'V162'],drop_first=True)
data[["V161"]]=data[["V161"]].apply(LabelEncoder().fit_transform)

print("Importation du modele")
f=open("dtree.sav","rb")
dtree=pickle.load(f)
f.close()

print("Calcul des predictions...")
pred=dtree.predict(data)

print("Creation du fichier predictions.txt...")

np.savetxt('predictions.txt',pred,delimiter="\t",fmt='%s')

```

```

print("Fin du programme")
end_time = (time.time() - start_time)
print("Temps d'execution : %d s" %int(end_time))

```

4 Système de scoring

4.1 Modélisation

4.1.1 Analyse discriminante

Ici nous n'avons pas de fichier de modélisation a proprement parler, nous nous sommes servi du fichier LDA.sav qui à déjà été fait dans la première partie , pour la réalisation du scoring et qui contient notre modèle qui nous aidera à la prédiction du score

```

#import modèle
import pickle
f=open("LDA.sav","rb")
lda=pickle.load(f)
f.close()

```

4.1.2 Régression Logistique

La partie scoring de la régression logistique est similaire à la partie classement puisqu'on entraîne le modèle sur les mêmes variables :

```

startTime = datetime.now()

encoder = OrdinalEncoder()
encoder.fit(data[["V160", "V161", "V162"]])
data[["V160", "V161", "V162"]] =
    encoder.transform(data[["V160", "V161", "V162"]])

X = data.iloc[:,0:41]
y = data.iloc[:,41]

X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

```

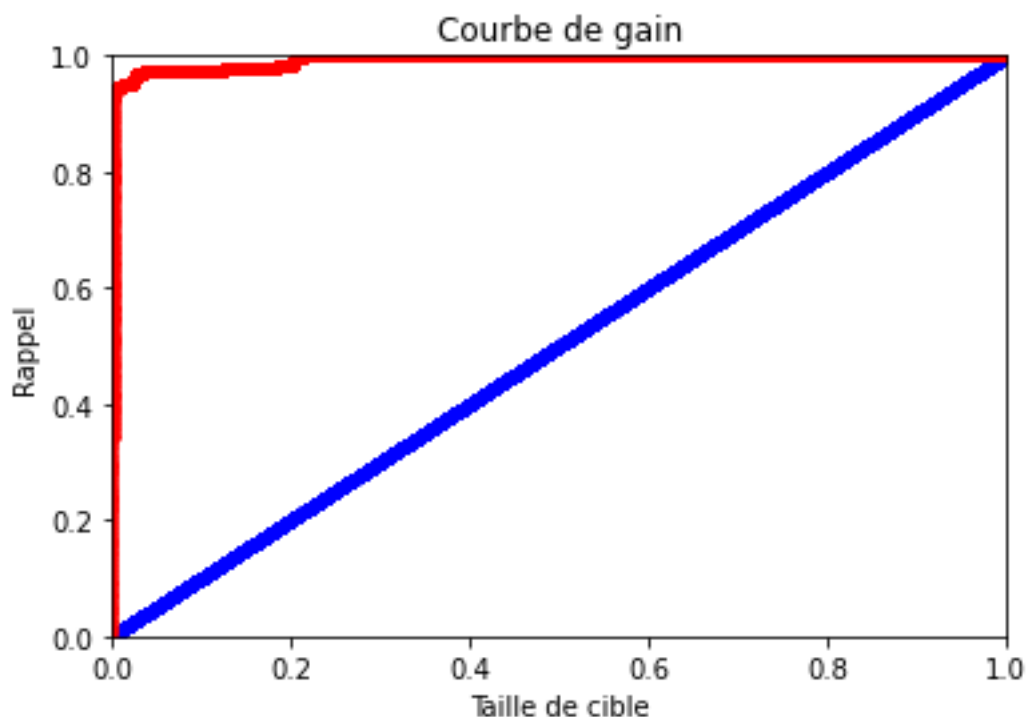
```
lr = LogisticRegression(solver='liblinear')
modele_all = lr.fit(X_train,y_train)
probas = lr.predict_proba(X_test)
```

Ici le modèle va calculer la probabilité pour chaque modalité de V200, or ce qui nous intéresse c'est la valeur m16.

4.2 Évaluation

4.2.1 Analyse discriminante

Au vu du taux d'erreur extrêmement faible de notre analyse discriminante, mais aussi du très bon score trouvé pour la modalité "m16", nous pouvons constater ci-dessous que la courbe de gain bénéficie d'un très bon taux de réussite de valeur positive sur celle-ci.



4.2.2 Régression Logistique

La valeur m16 étant la 8eme valeur de la liste des classes du modèle, on peut exécuter ce code afin d'afficher notre courbe de scoring :

```
score = probas[:,7] # m16 est la 8eme valeur
pos = pd.get_dummies(y_test)
```



```

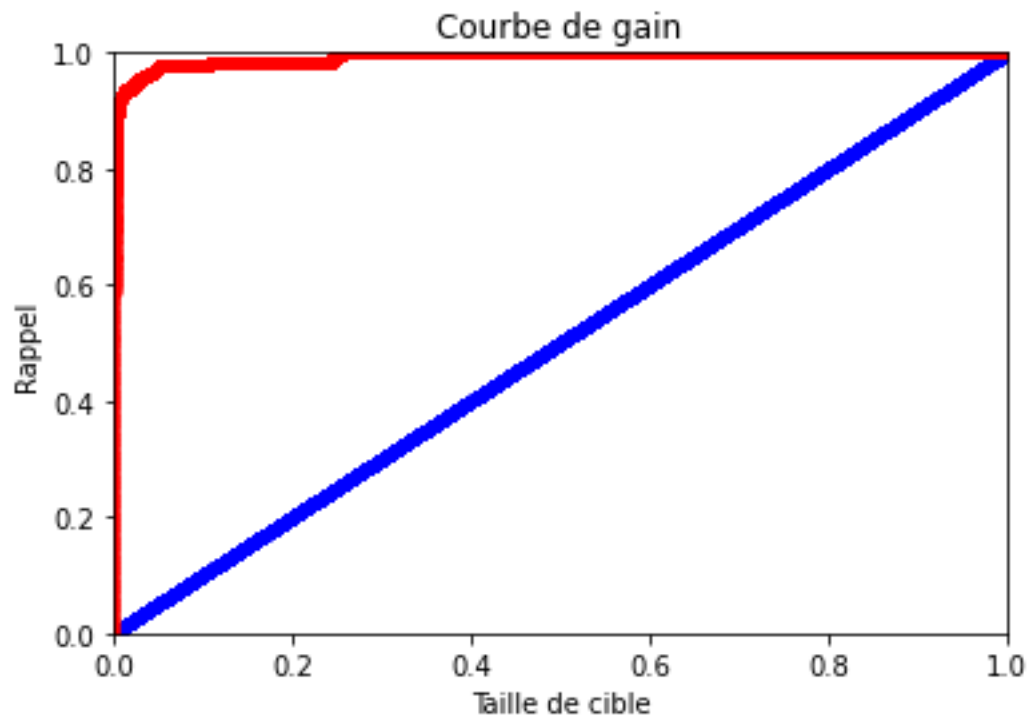
pos = pos['V200_m16']

npos = np.sum(pos)
index = np.argsort(score)
index = index[::-1]
sort_pos = pos[index]
cpos = np.cumsum(sort_pos)
rappel = cpos/npos
n = y_test.shape[0]
taille = np.arange(start=1, stop=148208, step=1)
taille = taille / n

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()
print('Temps d\'exécution: ', datetime.now() - startTime)

```



```
# Temps d'exécution: 0:05:09.570487
```

On a donc un scoring très performant en 5m09s, ce qui nous permet de cibler de manière très efficace les individus ayant comme valeur m16 pour la variable V200.

4.3 Déploiement

Les résultats graphiques étant similaires on a préféré choisir l'analyse discriminante étant donnée que ce modèle est plus adapté lorsqu'il s'agit d'analyser une variable qui a plus de deux modalités tandis qu'il est préférable d'appliquer la régression logistique pour une variable dichotomique.

On va donc, à partir de notre fichier LDA.sav, récupérer notre modèle entraîné puis l'appliquer sur les valeurs auxquelles on souhaite effectuer le ciblage. En sortie on aura un fichier Score.csv qui contient le score d'appartenance à la classe cible pour chaque valeur du fichier.

```
import numpy
import os
import pandas
os.chdir(os.path.dirname(__file__))
path=input("chemin absolu du fichier à charger : ")

cols=['V14', 'V84', 'V96', 'V159', 'V160', 'V161', 'V162', 'V163',
      'V164', 'V165', 'V166', 'V167', 'V168', 'V169', 'V170', 'V171',
      'V172', 'V173', 'V174', 'V175', 'V176', 'V177', 'V180', 'V181',
      'V182', 'V183', 'V184', 'V185', 'V186', 'V187', 'V188', 'V189',
      'V190', 'V191', 'V192', 'V193', 'V194', 'V195', 'V197', 'V198',
      'V199', 'V200']
df =pandas.read_table(path,delimiter='\t',usecols=cols)

#encodage variable qualitative
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df[["V160", "V161", "V162"]]=df[["V160", "V161", "V162"]].apply(le.fit_
    ↪transform)

#import modèle
import pickle
f=open("LDA.sav", "rb")
lda=pickle.load(f)
f.close()
```

```

X = df.iloc[:,0:41]
y = df.iloc[:,41]
y=y.to_numpy().reshape(-1)

from sklearn import model_selection
XTrain,XTest,yTrain,yTest=model_selection.train_test_split(X,y,test_size=0.
    ↪3,random_state=42,stratify=y)

#prediction
probas = lda.predict_proba(XTest)
print(probas)

classes=lda.classes_
print(lda.classes_)
score = probas[:,7]

#Insertion du scoring dans un DataFrame
dfScore= pandas.DataFrame(score,columns=['Score'])
dfScore.info()

#exportation dans un fichier excel
dfScore.to_csv("Score.csv",index=False)

#yTest en binaire
pos = pandas.get_dummies(yTest).values
#colonne positif
pos = pos[:,7]

#nb positifs(nb de m16)
npos = numpy.sum(pos)
print(npos)

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

#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()

```

5 Regroupement des modalités

Nous n'avons pas traité cette partie. Nous avons parcouru différents articles traitant du sujet qui faisaient référence à la classification ascendante hiérarchique mais nous ne sommes pas parvenu à la mettre en application.