Projet Data Mining

BOURDIN Yvan & FRINTZ Elisa

Master 1 Informatique - Semestre 1

Année 2020-2021

Sommaire

Introduction 2
Nettoyage des données 2
Programme 1 3
1. Introduction 3
2. Choix et entraînement du modèle 3
3. Description et évaluation du modèle définitif 3
4. Estimation des performances sur la base de déploiement 5
5. Utilisation de l'interface graphique 5
Programme 2 7
1. Introduction 7
2. Choix et entraînement du modèle 7
3. Description et évaluation du modèle définitif 8
4. Estimation des performances sur la base de déploiement 9
5. Utilisation de l'interface graphique 10
Programme 3 12
1. Introduction 12
2. Choix et entraînement du modèle + Regroupement des modalités de « V200 » 12
3. Description et évaluation du modèle définitif 13
4. Estimation des performances sur la base de déploiement 14
5. Utilisation de l'interface graphique 14
Test de nos 3 programmes avec un grand nombre de données 16
Perspectives d'évolution 16
Conclusion 16

Introduction

La fouille de données est une composante essentielle du master Informatique de Lyon 2. Elle est d'autant plus importante pour les étudiants voulant s'orienter dans le domaine de la Data Science.
Ce projet de data mining nous permet de nous confronter à la réalité des différentes étapes d'un projet de machine Learning que l'on peut retrouver en entreprise. En effet, du nettoyage des données au déploiement de l'interface graphique, en passant par la programmation des algorithmes et l'optimisation des paramètres, il y a beaucoup d'étapes à réaliser avant d'obtenir un produit fini. Le choix du langage étant libre, nous avons choisi python puisque c'est l'un des langages les plus utilisés dans le domaine de la data science.
Dans le cadre de ce projet, nous disposons d'un grand jeu de données contenant 199 variables prédictives et une variable cible « V200 » à partir duquel, nous allons créer trois produits :

1) Le premier est un programme dont l'objectif est de prédire le plus précisément possible la variable cible « V200 ».
2) Le second est un système de « scoring » permettant de cibler la modalité « m16 » (les positifs) de la variable cible « V200 ».
3) Le dernier est une variante du premier programme dont l'objectif est de prédire le plus précisément possible la variable cible « V200_Prim » modifiée via le regroupement des modalités de « V200 ». Ces choix de regroupement devront être argumentés.

Tout d'abord, nous observerons et nettoierons les données mises à notre disposition. Puis, pour chacun des trois programmes, nous décrirons leur objectif, le choix et l'entrainement du modèle, la description et l'évaluation du modèle définitif, l'estimation des performances sur la base de déploiement, et enfin, nous expliquerons comment utiliser l'interface mise à disposition de l'utilisateur pour accéder à ce programme. Enfin, nous expliquerons les différentes difficultés rencontrées et nous conclurons par une synthèse globale de ce projet.

Nettoyage des données

Pour chacun des trois programmes, nous avons commencé par nettoyer les données. Ce nettoyage comprend 2 étapes :

- Le recodage des variables qualitatives V160, V161 et V162 en variables quantitatives (0/1).
- La suppression des variables sans informations, c'est-à-dire les variables qui n'ont qu'une seule modalité.
- La vérification pour s'assurer que les données ne contiennent pas de valeurs manquantes

Programme 1

1. Introduction

L'objectif de ce programme 1 est de prédire le plus précisément possible la variable cible « V200 » d'une base de déploiement contenant 199 variables descriptives (V1...V199). Cette prédiction doit se faire à l'aide d'une interface graphique liée à un modèle prédictif de machine Learning préalablement entraîné.

Au final, l'utilisateur doit pouvoir entrer une base de déploiement (~2GB) dans le programme. Et celui-ci doit retourner un fichier « predictions.txt » dans le répertoire courant de l'utilisateur contenant pour chaque observation une prédiction sur la variable « V200 ».

2. Choix et entraînement du modèle

Nous avons choisis d'utiliser un Arbre de Décision car la variable cible « V200 » à prédire est qualitative et elle possède 23 modalités. De plus, nous avons tout de même essayé une Analyse Discriminante Linéaire mais nous nous sommes aperçus que les performances étaient nettement moins bonnes qu'avec un Arbre de Décision.

Pour la sélection de variables, nous avons utilisé la méthode SelectKBest à partir du test du chi2. Cela fonctionne de la manière suivante : On fixe k, le nombre de variables, que l'on souhaite et l'algorithme fait un test du chi2 entre chaque variable descriptive et la variable cible afin de retourner les k variables qui expliquent le mieux la variable cible.

Afin de sélectionner le k le plus optimal, on a fait tourner l'algorithme pour plusieurs k afin de définir celui qui minimise le taux d'erreur. C'est de cette manière qu'on a pu fixer le nombre de variables optimal k = 54.

La description exacte de l'entraînement de notre modèle se situe dans le Jupyter Notebook (Projet-Data-Mining-Programme-1.pdf). Afin de fixer les résultats obtenus, nous avons fixé l'option random_state = 0 à chaque fois.

3. Description et évaluation du modèle définitif

Le modèle définitif est un arbre de décision qui se construit sur les 54 variables les plus pertinentes choisies à l'aide de la méthode de sélection de variables SelectKBest à partir du test du chi2.

Voici les 54 variables sélectionnées (triées par ordre d'importance) :

	VarName	Importance
16	V182	0.604607
21	V187	0.325891
9	V171	0.013195
27	V193	0.011625
1	V163	0.007588
29	V195	0.007202
30	V196	0.006331
4	V166	0.006282
52	V162_m5	0.004716
40	V161_m17	0.003393
6	V168	0.001755
28	V194	0.001633
37	V161_m11	0.001011
53	V162_m6	0.000565
15	V181	0.000552
44	V161_m44	0.000504
38	V161_m12	0.000434
34	V160_m1	0.000340
19	V185	0.000304
7	V169	0.000255
32	V198	0.000236
41	V161_m20	0.000224
25	V191	0.000170
14	V180	0.000142
2	V164	0.000139
36	V160_m3	0.000118
0	V159	0.000104
24	V190	0.000103
17	V183	0.000094
49	V162_m11	0.000076
12	V175	0.000073
42	V161_m22	0.000061
26	V192	0.000042
8	V170	0.000039
3	V165	0.000035
31	V197	0.000030
43	V161_m39	0.000021
35	V160_m2	0.000016
10	V172	0.000016
18	V184	0.000015
46	V161_m55	0.000014
33	V199	0.000014
5	V167	0.000009
23	V189	0.000008
22	V188	0.000007
20	V186	0.000006
11	V174	0.000003
13	V176	0.000000
45	V161_m49	0.000000
47	V161_m9	0.000000
48	V162_m10	0.000000
50	V162_m2	0.000000
51	V162_m3	0.000000
39	V161_m16	0.000000

Figure 1 - Description des 54 variables sélectionnées - Programme 1

Nous avons évalué le modèle sur l'échantillon test qui représente 25 % du dataset initial. A l'aide de la matrice de confusion, nous avons pu calculer :

Taux de reconnaissance global = 0.9996842258675692

Taux d'erreur global = 0.00031577413243077945

Sensibilité (rappel) et précision par classe :

	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

Figure 2 - Sensibilité et précision par classe - Programme 1

4. Estimation des performances sur la base de déploiement

Afin d'estimer le nombre d'erreur sur la base de déploiement (4 898 424 obs), nous avons effectué le calcul suivant :

Nombre d'erreur estimé = Taux d'erreur global x 4 898 424 = 1547

Le nombre d'erreur annoncé sur la base de déploiement (4 898 424 obs) à l'aide de ce modèle est donc de : 1547 erreurs de prédiction.

5. Utilisation de l'interface graphique

Voici l'interface graphique qui correspond au programme 1 :

Figure 3 - Interface - Programme 1

- **Etape 1 :** Placer le fichier « data_avec_etiquettes.txt » dans le même dossier que la base de déploiement.
- **Etape 2 :** Vérifier que toutes les librairies utilisées sont correctement installées.
- **Etape 3 :** Exécuter « Programme1_BOURDIN_FRINTZ.py » à l'aide de l'IDE Spyder. L'interface va s'ouvrir.
- **Etape 4 :** Importer la base de déploiement (fichier.txt) à l'aide de la barre de recherche. Le répertoire courant devient celui où se trouve la base de déploiement sélectionnée.
- **Etape 5 :** Cliquer sur le bouton « Valider » qui exécute le programme.
- **Etape 6 :** Attendre la fin de l'exécution, le fichier 'predictions.txt' se trouvera dans le répertoire courant de l'utilisateur.

Programme 2

1. Introduction

L'objectif de ce programme 2 est de produire le score d'appartenance à la classe « m16 » de la variable cible « V200 » d'une base de déploiement contenant 199 variables descriptives (V1...V199). Cette prédiction doit se faire à l'aide d'une interface graphique liée à un modèle de scoring de machine learning préalablement entraîné.
Au final, l'utilisateur doit pouvoir entrer une base de déploiement (~2GB) dans le programme. Et celui-ci doit retourner un fichier «scores.txt » dans le répertoire courant de l'utilisateur contenant pour chaque observation, son score d'appartenance à « m16 » et sa prédiction.

2. Choix et entraînement du modèle

La première étape de ce programme est de recoder la variable cible « V200 » en variable binaire (0/1) :

- 1 correspond à la présence de la modalité « m16 » (modalité positive)
- 0 correspondent à son absence.

Pour l'entrainement du modèle de scoring, nous avons choisis d'utiliser un Arbre de Décision car il s'agit de la méthode la plus efficace sur nos données. En effet, nous avons également testé la méthode de l'Analyse Discriminante Linéaire mais les performances étaient nettement moins bonnes qu'avec un Arbre de Décision.

Pour la sélection de variables, nous avons utilisé la méthode SelectKBest à partir du test du chi2. Cela fonctionne de la manière suivante : On fixe k, le nombre de variables, que l'on souhaite et l'algorithme fait un test du chi2 entre chaque variable descriptive et la variable cible afin de retourner les k variables qui expliquent le mieux la variable cible.

Afin de sélectionner le k le plus optimal, on a fait tourner l'algorithme pour plusieurs k afin de définir celui qui minimise le taux d'erreur. C'est de cette manière qu'on a pu fixer le nombre de variables optimal k = 15.

Une fois la sélection de variable effectuée, nous avons pu prédire les probabilités d'appartenance à chacune des modalités 0 ou 1. Le score correspond alors la probabilité d'appartenance à la modalité positive. La prédiction correspond à la modalité qui a la plus grande probabilité d'appartenance.

La description exacte de l'entraînement de notre modèle se situe dans le Jupyter Notebook (Projet-Data-Mining-Programme-2.pdf). Afin de fixer les résultats obtenus, nous avons fixé l'option random_state = 0 à chaque fois.

3. Description et évaluation du modèle définitif

Le modèle définitif de scoring est basé sur un arbre de décision qui se construit sur les 15 variables les plus pertinentes choisies à l'aide de la méthode de sélection de variables SelectKBest à partir du test du chi2.

Voici les 15 variables sélectionnées (triées par ordre d'importance) :

	VarName	Importance
14	V162_m5	0.651396
1	V163	0.108413
9	V198	0.078702
11	V161_m39	0.054802
5	V185	0.033375
3	V181	0.030438
8	V193	0.024693
13	V162_m4	0.009089
7	V191	0.006495
12	V162_m1	0.001731
4	V182	0.000866
0	V159	0.000000
2	V164	0.000000
6	V186	0.000000
10	V199	0.000000

Figure 4 - Description des 15 variables sélectionnées - Programme 2

Nous avons évalué le modèle sur l'échantillon test qui représente 25 % du dataset initial.

Précision moyenne des scores sur les données de test = 0.9999757096821207

A l'aide de la matrice de confusion, nous avons pu calculer :

Taux d'erreur = 2.4290317879293315e-05
Taux de reconnaissance global = 0.9999757096821207
Rappel (sensibilité) = 0.9925373134328358
Précision = 0.9962546816479401

Sensibilité (rappel) et précision par classe :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	123238
1	1.00	0.99	0.99	268
accuracy			1.00	123506
macro avg	1.00	1.00	1.00	123506
weighted avg	1.00	1.00	1.00	123506

Figure 5 - Sensibilité et précision par classe - Programme 2

4. Estimation des performances sur la base de déploiement

Afin d'estimer le nombre d'erreur sur la base de déploiement (4 898 424 obs), nous avons effectué le calcul suivant :

Nombre d'erreur estimé = Taux d'erreur global x 4 898 424 = 119

Le nombre d'erreur annoncé sur la base de déploiement (4 898 424 obs) à l'aide de ce modèle de scoring est donc de : 119 erreurs de prédiction.

Voici les estimations ponctuelles sur la base d'entraînement :

Proportions d'observations positives et négatives :

0 0.997916

1 0.002084

Afin de prédire le nombre d'observations positives parmi les 10 000 individus qui présentent les scores les plus élevés de la base de déploiement (4 898 424 obs), nous utilisons la courbe de gain estimée à partir de ce modèle :

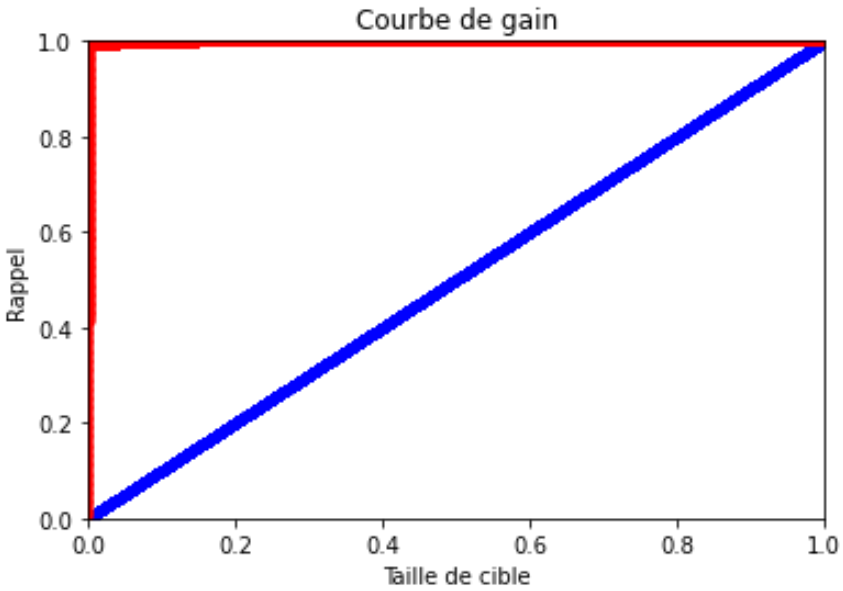

Figure 6 - Courbe de gain - Programme 2

Nous recherchons le rappel correspondant au 10 000 ème individu, puis on le multiplie par le nombre total de positifs prédis (268).

Rappel correspondant au 10 000 ème individu : Rappel[9999] = 0.996268656716418
Nombre total de positifs prédis : npos = 268
Le nombre de positifs parmi les 10 000 observations qui ont les scores les plus élevés (4 898 424 obs) est donc estimé à : 267

5. Utilisation de l'interface graphique

Voici l'interface graphique qui correspond au programme 2 :

Figure 7 - Interface - Programme 2

- **Etape 1 :** Placer le fichier « data_avec_etiquettes.txt » dans le même dossier que la base de déploiement.
- **Etape 2 :** Vérifier que toutes les librairies utilisées sont correctement installées.
- **Etape 3 :** Exécuter « Programme2_BOURDIN_FRINTZ.py » à l'aide de l'IDE Spyder. L'interface va s'ouvrir.
- **Etape 4 :** Importer la base de déploiement (fichier.txt) à l'aide de la barre de recherche. Le répertoire courant devient celui où se trouve la base de déploiement sélectionnée.
- **Etape 5 :** Cliquer sur le bouton « Valider » qui exécute le programme.
- **Etape 6 :** Attendre la fin de l'exécution, le fichier 'scores.txt' se trouvera dans le répertoire courant de l'utilisateur.

Programme 3

1. Introduction

L'objectif de ce programme 3 est de prédire le plus précisément possible la variable cible « V200_Prim » modifiée via le regroupement des modalités de « V200 » d'une base de déploiement contenant 199 variables descriptives (V1...V199). Cette prédiction doit se faire à l'aide d'une interface graphique liée à un modèle prédictif de machine Learning préalablement entraîné.
Au final, l'utilisateur doit pouvoir entrer une base de déploiement (~2GB) et un fichier « classes.txt » contenant les regroupements de la variable cible dans le programme. Et celui-ci doit retourner un fichier « sorties.txt » dans le répertoire courant de l'utilisateur contenant pour chaque observation une prédiction sur la variable « V200_Prim ».
Ces choix de regroupement des modalités de « V200 », présents dans le fichier « classes.txt », seront argumentés à l'aide d'un algorithme de clustering.

2. Choix et entraînement du modèle + Regroupement des modalités de « V200 »

Nous avons choisis d'utiliser un Arbre de Décision puisque la variable cible « V200_Prim » à prédire est qualitative et possède 5 modalités. De plus, nous avons tout de même essayé une Analyse Discriminante Linéaire mais nous nous sommes aperçus que les performances étaient nettement moins bonnes qu'avec un Arbre de Décision.

Pour la sélection de variables, nous avons utilisé la méthode SelectKBest à partir du test du chi2. Cela fonctionne de la manière suivante : On fixe k, le nombre de variables que l'on souhaite et l'algorithme fait un test du chi2 entre chaque variable descriptive et la variable cible afin de retourner les k variables qui expliquent le mieux la variable cible.

Afin de sélectionner le k, le nombre de variables que l'on souhaite, le plus optimal, nous avons d'abord fixé k = 54 grâce au programme 1.

Puis nous avons réalisé la méthode de clustering des k-means. Celle-ci fonctionne de la manière suivante : On fixe k, le nombre de classe souhaité, et l'algorithme converge vers les meilleurs k groupes de variables. (Attention les données doivent être normalisées !)

Afin de sélectionner k, le nombre de clusters le plus optimal, on fait tourner l'algorithme pour plusieurs k et on affiche un graphique qui contient l'inertie en fonction du nombre de clusters. C'est de cette manière que l'on fixe le nombre de clusters à 5.

Afin d'attribuer les anciennes classes de « V200 » aux nouvelles classes de « V200_Prim », on affiche pour chaque ancienne classe de « V200 », sa proportion dans chacune des nouvelles classes de « V200_Prim » à l'aide d'un tableau croisé. On attribue à chaque classe de « V200 » la valeur de la classe de « V200_Prim » dans laquelle elle est la plus présente. C'est de cette manière que l'on a créé le fichier "classes.txt" qui contient pour chaque classe de « V200 » sa correspondance en classe de « V200_Prim ».

Afin de sélectionner le k, le nombre de variables que l'on souhaite, le plus optimal sur la nouvelle variable cible, nous avons fait tourner l'algorithme SelectKBest à partir du test du chi2 pour plusieurs k afin de définir celui qui minimise le taux d'erreur. C'est de cette manière qu'on a pu fixer k = 40.

La description exacte de l'entraînement de notre modèle se situe dans le Jupyter Notebook (Projet-Data-Mining-Programme-3.pdf). Afin de fixer les résultats obtenus, nous avons fixé l'option random_state = 0 à chaque fois.

3. Description et évaluation du modèle définitif

Le modèle définitif est un arbre de décision qui se construit sur les 40 variables les plus pertinentes choisies à l'aide de la méthode de sélection de variables SelectKBest à partir du test du chi2.

Voici les 40 variables sélectionnées (triées par ordre d'importance) :

	VarName	Importance
30	V161_m12	0.614485
13	V187	0.338717
1	V163	0.013769
19	V193	0.011455
27	V160_m2	0.008603
16	V190	0.006344
39	V162_m6	0.002789
25	V199	0.000726
35	V161_m9	0.000505
23	V197	0.000492
24	V198	0.000460
10	V184	0.000310
38	V162_m5	0.000308
20	V194	0.000280
17	V191	0.000202
2	V164	0.000148
33	V161_m44	0.000106
5	V170	0.000069
18	V192	0.000041
9	V183	0.000026
9	V183	0.000026
22	V196	0.000022
4	V168	0.000019
6	V174	0.000019
11	V185	0.000018
0	V159	0.000016
26	V160_m1	0.000013
7	V181	0.000013
34	V161_m49	0.000008
3	V166	0.000008
31	V161_m20	0.000008
14	V188	0.000007
29	V161_m11	0.000007
21	V195	0.000007
15	V189	0.000000
28	V160_m3	0.000000
8	V182	0.000000
32	V161_m39	0.000000
36	V162_m10	0.000000
37	V162_m2	0.000000
12	V186	0.000000

Figure 8 -Description des 40 variables sélectionnées - Programme 3

Nous avons évalué le modèle sur l'échantillon test qui représente 25 % du dataset initial.

A l'aide de la matrice de confusion, nous avons pu calculer :
Taux de reconnaissance global = 0.9998623548653507
Taux d'erreur global = 0.00013764513464931127

Sensibilité (rappel) et précision par classe :

	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

Figure 9 - Sensibilité et précision par classe - Programme 3

4. Estimation des performances sur la base de déploiement

Afin d'estimer le nombre d'erreur sur la base de déploiement (4 898 424 obs), nous avons effectués le calcul suivant :

Nombre d'erreur estimé = Taux d'erreur global x 4 898 424 = 674

Le nombre d'erreur annoncé sur la base de déploiement (4 898 424 obs) à l'aide de ce modèle est donc de : 674 erreurs de prédiction.

→ On remarque que le regroupement des modalités de « V200 » nous a permis d'améliorer les performances du modèle par rapport au programme 1 (dont le nombre d'erreur estimé était de 1547).

5. Utilisation de l'interface graphique

Voici l'interface graphique qui correspond au programme 2 :

Figure 10 - Interface - Programme 3

- **Etape 1 :** Placer les fichiers « data_avec_etiquettes.txt » et « classes.txt » dans le même dossier que la base de déploiement.
- **Etape 2 :** Vérifier que toutes les librairies utilisées sont correctement installées.
- **Etape 3 :** Exécuter « Programme3_BOURDIN_FRINTZ.py » à l'aide de l'IDE Spyder. L'interface va s'ouvrir.
- **Etape 4 :** Importer la base de déploiement (fichier.txt) et le fichier « classes .txt » à l'aide des deux barres de recherche. Le répertoire courant devient celui où se trouve la base de déploiement sélectionnée.
- **Etape 5 :** Cliquer sur le bouton « Valider » qui exécute le programme.
- **Etape 6 :** Attendre la fin de l'exécution, le fichier 'sorties.txt' se trouvera dans le répertoire courant de l'utilisateur.

Test de nos 3 programmes avec un grand nombre de données

Afin de tester notre algorithme, nous avons créé un fichier "essai2fois.txt" qui contient 2 fois la base "data_avec_etiquettes.txt" sans la variable cible, seulement 2 fois puisque nous n'avons pas réussi à la dupliquer 10 fois sans faire planter le noyau. Le fichier "essai2fois.txt" est donc de taille 437,7 Mo (437 705 903 octets).

Nous avons quand même pu créer les fichiers suivant de la même manière :

- "essai4fois.txt" : 875,4 Mo (875 410 919 octets)
- "essai6fois.txt" : 1,3 Go (1 313 115 935 octets)

Résultat : Cela fonctionne sans erreur pour "essai2fois.txt" mais le noyau plante pour "essai4fois.txt" et "essai6fois.txt".

Remarques :

- Peut-être que nos ordinateurs ne sont pas assez puissants pour prendre en charge ces volumes de données.
- Peut-être que certains calculs ne sont pas assez optimisés pour permettre à l'ordinateur de les supporter.

Perspectives d'évolution

Nous avons pu voir que l'Arbre de décision est une méthode de Machine Learning efficace sur ce type de données. Pourtant, nous avons pour perspective d'évolution de mettre en place un algorithme de classification RandomForest qui est une méthode de Machine Learning basée sur de nombreux arbres de décision qui réduit la variance des prévisions d'un arbre seul, améliorant ainsi ses performances.
Pour entraîner le modèle, on choisit un dataset d'apprentissage qui contient déjà les réponses. L'algorithme va donc réaliser de nombreux arbres de décision aléatoirement et faire la moyenne des réponses obtenues. Le nombre d'arbre réalisé est défini par l'utilisateur lors de la construction du modèle.
Une autre perspective d'évolution serait de récupérer le modèle construit afin de le l'appliquer directement sur la base de déploiement sans avoir à le ré-entrainer sur le fichier « data_avec_etiquettes.txt ». En effet, nous nous sommes aperçus trop tard de cette possibilité, nous n'avons donc pas eu le temps de l'implémenter.

Conclusion

Pour conclure, nous pouvons dire que les trois programmes sont fonctionnels et liés à une interface qui permet de faciliter leur utilisation. Le premier programme nous a permis de nous familiariser avec la prédiction, le second d'approfondir les méthodes de scoring et le dernier de découvrir des méthodes de clustering. Les principales

difficultés de ce projet ont été de choisir les algorithmes à utiliser et de travailler avec un très grand nombre de données, ce qui nous donne de très longs temps de calcul, parfois difficiles à gérer.

	Objectif	Méthode	Nombre de variable	Taux d'erreur	Annonces sur la base de déploiement (4 898 424 obs)	Fichier de sortie
Prog. 1	Prédiction	Arbre de décision	54	0.0003157741324307795	Nb d'erreur = 1547	« predictions.txt »
Prog. 2	Scoring	Arbre de décision	15	2.4290317879293315e-05	Nb positif parmi les 10 000 obs qui ont les scores les plus élevés = 267 Nb d'erreur = 119	« scores.txt »
Prog. 3	Clustering	Arbre de décision + k-means	40	0.0001376451346493117	Nb d'erreur = 674	« sortie.txt »

Figure 11 – Récapitulatif des trois programmes