Projet DataMining

Pierre Le Galèze et Clément Malvy

Université Lyon 2 / ICOM M1 Informatique 2020/2021

1. Table des matières

2. La base de données

2.1. Les données collectées

La base de données « data_avec_etiquettes.txt » nous a été fournie par Mr. Rakotomalala, elle contient 200 variables, nommées de V1 à V200 pour un total de 494 021 observations. La variable V200 est la variable cible, comportant 23 modalités allant de m1 à m23. Les variables de V1 à V199 sont celles pouvant expliquer la variable cible, certaines sont quantitatives et d'autres qualitatives (V160, V161, 162).

2.2. Exploration des données

Durant l'exploration de données, nous avons regardé quelles variables n'avaient qu'une seule modalité, que ce soit des 0 ou bien une valeur constante. Nous nous sommes aussi intéressés à la corrélation entre toutes les variables, à savoir quelles variables étaient corrélées à plus de 0.8.

Nous avons aussi étudié la variable cible, V200:

m19	280790
m10	107201
m12	97278
m1	2203
m18	1589
m6	1247
m16	1040
m22	1020
m21	979
m15	264
m11	231
m4	53
m2	30
m7	21
m23	20
m5	12
m17	10
m8	9
m3	8
m9	7
m14	4
m13	3
m20	2

Comme on peut le voir, 3 modalités ressortent particulièrement : m1 est présent 280 790 fois, cela représente plus de 56% de l'effectif du dataframe, mais ce n'est pas la seule, m12 est présent 107 201 fois et m10, 97 278 fois. Si on ne prenait que les lignes correspondant à ces trois modalités, on aurait alors environ 98% du dataset.

3. La classification et le scoring

3.1. Préparation des données

Afin d'améliorer les performances de notre futur modèle le plus possible, nous avons supprimé les variables ayant une seule modalité (V178 et V179) et celles fortement corrélés entre elles (supérieur à 0.8.). Au total 16 variables ont été supprimées pour le scoring et 17 pour la classification.

Étant donné que les variables n'étaient pas à la même échelle, nous les avons standardisées (action de transformer les variables pour qu'elles aient toutes une moyenne de 0 et une même variance).

Il nous a aussi fallu les changer les variables qualitatives : deux choix se sont offert à nous : la méthode get_dummies (module de pandas) ou LabelEncoder (module de sklearn).
La méthode get_dummies consiste à transformer la colonne en un nombre N de colonnes. Chaque nouvelle colonne correspond à une modalité de la colonne initiale avec pour chaque ligne, un 1 si la modalité apparaît et un 0 si ce n'est pas le cas.
La méthode LabelEncoder consiste simplement à transformer chaque modalité en un nombre dédié. Tous les m10 seront par exemple transformés en 1, les m11, en 2, etc...
A noter que LabelEncoder suit l'ordre d'apparition des variables : si la variable apparaît en 1er alors elle aura comme chiffre un 1, en 2ème, un 2,etc...

Toutes les dummy variables (créées via get_dummies) d'une variable n'ont pas tous forcément une grosse importance. Etant donné que nous préférons prendre les variables importantes, on ne peut pas prendre juste une dummy variable sans les autres. Il faudrait alors faire d'autres changements, ce qui complique les choses. Nous avons donc choisi d'utiliser LabelEncoder pour pouvoir mieux sélectionner les variables (que nous verrons plus tard).

Afin de mettre en œuvre notre modèle, nous devons séparer la variable cible des variables explicatives. On crée alors deux variables : X et y avec X qui contiendra les variables explicatives et y, la variable cible.
La dernière étape pour pouvoir utiliser notre modèle est de séparer notre base de données en un jeu de données train-test avec l'aide du module model_selection de sklearn. On choisit en paramètre test_size = 30% (pour entraîner notre modèle sur 70% des données et le tester sur 30%) et stratify = y (pour faire en sorte de toujours avoir la même distribution

des modalités de V200). Le modèle sera entraîné sur le jeu de données train et nous le validerons sur le jeu de données test.

3.2. Choix du modèle

Lors des TD, nous avons étudié deux modèles en particulier : les arbres de décision et l'Analyse Discriminante Linéaire (LDA). Les résultats obtenus pour les deux modèles sont très élevés avec des scores au-delà de 99% : les deux modèles sont donc très bien pour notre étude. Le score est le ratio entre les données bien classées (vrai positif et vrai négatif) et le nombre total de données. C'est cela que nous utiliserons par la suite pour évaluer rapidement notre modèle avant de l'évaluer plus précisément dans les parties "Évaluation" .

Pour le scoring, nous avons donc choisi le modèle de LDA pour éviter les problèmes d'ex-aequo et donc pris les arbres pour la classification.

3.3. Le modèle de classification

3.3.1. Présentation du modèle : les arbres de décision

Un arbre de décision (ou Decision Tree en anglais) est une méthode visuelle permettant de représenter un ensemble de règles sous la forme d'un arbre. Cette méthode fait partie de la famille des modèles d'apprentissage supervisé, c'est-à-dire que nous avons sous la main la variable que nous voulons prédire, ce qui permet au modèle de s'ajuster. Il existe deux types d'arbres : les arbres de régression, permettant de prédire une quantité et ceux de classification, permettant de prédire à quelle classe une variable appartient. Dans notre cas, nous nous intéresserons aux arbres de classification.

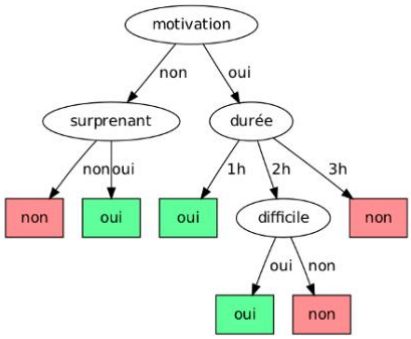
Un arbre est constitué :

- de nœuds, le premier étant appelé la racine, pris à partir des variables du dataset.
- de branches, indiquant une règle pour chaque modalité des variables nœuds (dans le cas d'une variable qualitative) ou une règle en fonction d'une limite (dans le cas d'une variable quantitative).
- de feuilles, représentant la décision finale du modèle et dépend du nombre de modalités de la variable cible.

<u>Exemple :</u>

Ici, on pourrait imaginer que nous ayons à notre disposition un dataset avec tous les devoirs que nous avons fait dans notre vie et si à ce moment-là, en fonction de diverses conditions (motivation/surprenant/durée/difficile/météo/température/...), nous l'avons fini à temps.

En l'utilisant, nous aimerions maintenant savoir, pour ce projet-ci de DataMining, quelles sont les principales conditions qui nous permettraient de le rendre à la date prévue.



En observant les feuilles de l'arbre, on comprend que nous pouvons le finir à temps dans 3 trois cas : si le devoir est surprenant même sans être motivé, si on est motivé et que l'on estime qu'il nous prendrait 1h à finir ou bien on peut potentiellement le finir en 2h et qu'il est considéré comme difficile.

Le choix de chaque nœud est en fonction d'une mesure appelée impureté (impureté car les variables ne prédisent pas parfaitement cette cible). On peut mesurer l'impureté par différents moyens comme par exemple via l'impureté Gini. Tout d'abord, pour chaque feuille d'un potentiel nœud, on calcule l'impureté Gini, ensuite, en fonction des effectifs de chaque feuille, on pondère les résultats obtenus pour pouvoir les combiner et ainsi obtenir l'impureté Gini du potentiel nœud. On choisit finalement comme nœud, la variable ayant l'impureté la plus basse et on continue comme cela pour chaque étape de construction de l'arbre en considérant les effectifs et l'impureté des nœuds précédents. On s'arrête si, parmi toutes les variables restantes, aucune de permet de baisser l'impureté.

La formule de l'impureté, exemple d'une variable cible ayant des modalité oui/non :

$$1 - (\text{probabilité de oui})^2 - (\text{probabilité de non})^2$$

Nous avons utilisé ce modèle avec sklearn via le module DecisionTreeClassifier.

3.3.2. Quelles sont les variables intégrées dans le modèle définitif ?

Dans un premier temps, nous avons testé notre modèle sur les 199 variables : il en résulte un score de 99.94%. Notre but désormais est de sélectionner moins de variable tout en gardant un score proche de 99.94%, pour cela, le module DecisionTreeClassifier sous sklearn a un paramètre ".feature_importances_" permettant d'obtenir l'importance des variables pour le modèle. Il nous suffit dès lors de les ranger dans un DataFrame et de les trier par ordre décroissant. Afin de sélectionner le moins de variables possibles, nous avons testé différents seuils et choisi celui donnant un score proche de celui inital.

Extrait des variables triées en fonction de leur importance :

	Variable	Importance
159	V160	5.984329e-01
176	V187	3.268749e-01
161	V163	3.624848e-02
180	V193	1.459127e-02
164	V166	8.040120e-03
160	V162	7.770119e-03
179	V190	1.958366e-03
166	V168	1.689249e-03
162	V164	1.224902e-03
175	V185	8.733340e-04
177	V188	6.640496e-04

Une variable a particulièrement attiré notre attention : V160. Celle-ci a une importance de 60% et permet à elle seule d'obtenir un score allant jusqu'à 82%. Pourtant, elle n'a que 3 modalités, ce qui nous paraissait assez étonnant. Pour pouvoir comparer V160 avec V200, nous les avons regroupés dans un dataset puis trié en fonction de V160, on remarque alors que chaque modalité de V160 est largement représentée par les modalités les plus présentes de V200. Soit 0 et m19, 1 et m10 (avec aussi du m12), 2 et m12. Étant donné que ces trois variables forment plus de 90% de la variable cible, cela n'est pas étonnant qu'à elle seule, V160 permettent un score de 82%.

Au final, avec un seuil de 0.01, on ne garde que 4 variables (V160, V187, V163 et 193) et on obtient un score de 99.77%.

3.3.3. Évaluation du modèle définitif

Étant donné qu'il y a trop de modalités, la matrice de confusion n'est pas très lisible et de ce fait nous ne nous y attarderons pas.

Pour obtenir plus de précision quant aux prédictions, on peut regarder le rapport de classification de sklearn :

	precision	recall	f1-score	support
m1	1.00	1.00	1.00	664
m10	1.00	1.00	1.00	32157
m11	0.97	0.89	0.93	75
m12	1.00	1.00	1.00	29216
m13	0.00	0.00	0.00	0
m14	0.00	0.00	0.00	0
m15	0.97	1.00	0.99	77
m16	0.96	0.99	0.97	304
m17	0.00	0.00	0.00	0
m18	0.97	0.99	0.98	472
m19	1.00	1.00	1.00	84239
m2	0.44	0.80	0.57	5
m20	0.00	0.00	0.00	0
m21	1.00	1.00	1.00	294
m22	0.95	0.99	0.97	296
m23	0.83	0.83	0.83	6
m3	0.00	0.00	0.00	0
m4	1.00	0.67	0.80	24
m5	0.00	0.00	0.00	0
m6	0.99	1.00	0.99	372
m7	0.83	0.83	0.83	6
m8	0.00	0.00	0.00	0
m9	0.00	0.00	0.00	0
accuracy			1.00	148207
macro avg	0.61	0.61	0.60	148207
weighted avg	1.00	1.00	1.00	148207

Légende :

- La précision est le ratio de valeur bien classé sur toutes les valeurs.
- Le recall (ou rappel) est la proportion d'éléments prédits comme appartenant à la classe sélectionnée divisée par la proportion d'éléments faussement prédit comme appartenant à la classe.
- Le f1-score est une moyenne harmonique entre rappel et précision.
- Le support est le nombre d'éléments de chaque classe des prédictions du modèle.

On remarque que le modèle ne prédit pas bien certaines classes (comme la m11) et que d'autres ne sont pas du tout prédites. Malgré tout, et du fait de l'écrasante majorité des classes m19,m12 et m10, très bien prédites, on peut valider le modèle car cela n'impacte en rien ses performances.

3.4. Le modèle de scoring

Afin d'appliquer le scoring, nous transformons la variable V200 pour cibler la modalité “m16” soit tout ce qui n’est pas m16 = 0 et m16 = 1.

3.4.1. Présentation du modèle : l’Analyse Discriminante Linéaire

L’Analyse Discriminante Linéaire (LDA) a pour but de prédire l’appartenance d’un individus à une classe en fonction de ses caractéristiques. Pour cela on passe par la création d’un modèle d’apprentissage basé sur l’égalité des matrices variance & co-variance. Ce dernier cherche à estimer la probabilité conditionnelle : $P(Y = y_k / X)$.
Chaque affectation est soumise à des règles :

$$D(Y_1,X)\ a_{1,0} + a_{1,1}X_1 + a_{1,2}X_2 + \ldots.\ a_{1,J}X_J$$

$$D(Y_2,X)\ a_{2,0} + a_{2,1}X_1 + a_{2,2}X_2 + \ldots.\ a_{2,J}X_J$$

$$Y_{k*} = \text{argmax}\ d(Y_k , X)$$

Ces règles d’affectation ont pour but de minimiser l’erreur d’affection et d’affecter l’individu a la classe maximisant sa probabilité d’appartenance.

Nous utilisons ce modèle avec sklearn avec le module LinearDiscriminantAnalysis.

3.4.2. Quelles sont les variables intégrées dans le modèle définitif ?

De la même manière que pour la classification, nous avons dans un premier temps testé notre modèle sans nous occuper de la sélection de variable et nous avons obtenu un score de 99.85%.
Malheureusement, le module LDA sous sklearn n’a pas de fonction permettant de récupérer directement l’importance des variables. Nous avons donc dû la calculer à partir de la matrice de covariance. Ce calcul ayant déjà été fait durant nos TD, nous l’avons récupéré et adapté à notre code. Il en résulte un tableau de données triée de manière décroissante nous donnant la p-value de chaque variable, plus la p-value est faible et plus la variable à d’importance (on considère aussi qu'au-dessus de 0.05, la variable est inutile. Désormais, il ne nous reste plus qu’à tester, pour un nombre différent de variables, le score.

Extrait des variables triés en fonction de la p-value :

	var	F	pvalue
21	V165	89.327619	1.110223e-16
40	V193	9985.277076	1.110223e-16
39	V191	743.897053	1.110223e-16
38	V190	8686.166172	1.110223e-16
37	V189	1525.504507	1.110223e-16
36	V188	2718.013980	1.110223e-16
35	V187	36364.724530	1.110223e-16
34	V185	72000.338670	1.110223e-16
33	V181	5263.967504	1.110223e-16
32	V180	77.191679	1.110223e-16
26	V170	13036.177600	1.110223e-16

Après quelques tests, nous nous sommes arrêtés à un seuil p-value de 0.00001, notamment à cause d'un nombre important de variables ayant une p-value très petites allant jusqu'à un ordre de $1e^{-16}$. Nous nous sommes donc arrêtés là, ce qui nous donne un total de 17 variables et un score final de 99.85%, égal au score initial.

3.4.3. Évaluation du modèle définitif

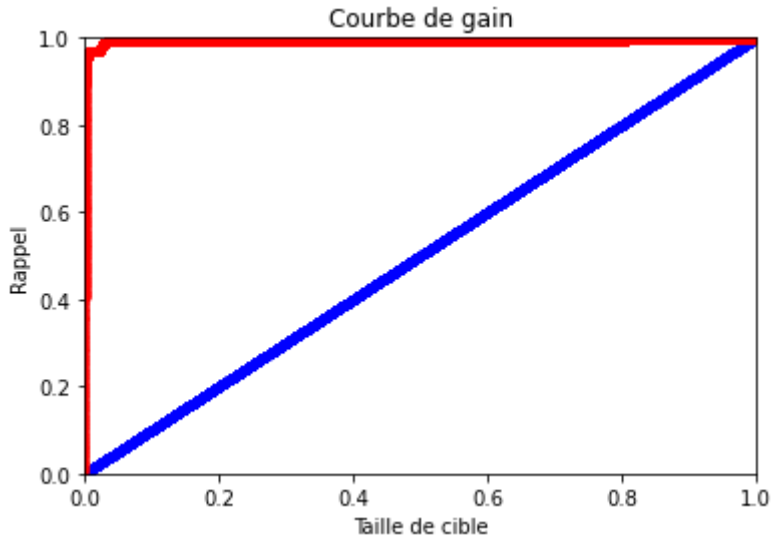
Étant donné que nous n'avons que deux modalités, on peut s'intéresser à la matrice de confusion qui nous indique qu'au total, on a 209 valeurs mal classées : 30 valeurs faussement considérées comme positives (appartenant à la classe 1) et 179 variables faussement considérées comme négatives (appartenant à la classe 0).

```
[[147716     30]
 [   179    282]]
```

Pour obtenir plus de précision quant aux prédictions, on peut regarder le rapport de classification de sklearn :

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	147746
1.0	0.90	0.61	0.73	461
accuracy			1.00	148207
macro avg	0.95	0.81	0.86	148207
weighted avg	1.00	1.00	1.00	148207

On remarque que le modèle prédit parfaitement la non-présence de la modalité m16 mais à quelques difficultés pour prédire sa présence (précision et recall plus faible). Cela peut s'expliquer par le fait que la classe 0 soit bien plus présente que la classe 1.
Malgré tout, au final, on obtient une accuracy de près de 100% (il est marqué 1.00 mais la valeur est arrondie), ce qui permet de valider notre modèle.
On obtient une courbe de gain tel que ci-dessous :



Le modèle est très précis et permet donc de prédire presque parfaitement la modalité m16.

4. Le regroupement des données

4.1. Préparation des données

La partie préparation des données se divise en deux : celle faites pour le clustering et celle faite pour la classification.

Pour le clustering, notre but était de regrouper les variables de chaque modalité de V200 dans un dataframe afin d'appliquer une méthode de clustering dessus. On regroupe donc chaque modalité et on calcule la moyenne pour chaque variable, ce faisant, on obtient le barycentre de chaque modalité.
Étant donné que standardiser transforme les variables afin que leur moyenne soit de 0, nous avons standardisé après que les données aient été regroupé.
Avant d'avoir regroupé, nous avons aussi supprimé les variables ayant une seule modalité, celle ayant une corrélation supérieure à 0.8. et transformé les variables qualitatives avec LabelEncoder.

Pour la classification, nous avons gardé la transformation faites avant le regroupement et ré-appliquer la standardisation sur la nouvelle base de données regroupée.

4.2. Le modèle de clustering

4.2.1. Présentation du modèle : le CAH

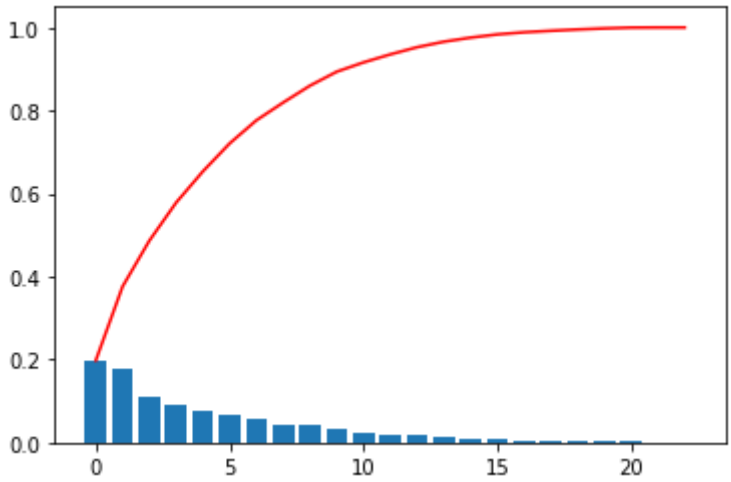
La CAH ou classification ascendante hiérarchique a pour objectif de classifier via la création de classes (ensemble d'individu possédant des traits de caractères communs).
Son fonctionnement est assez simple, l'algorithme calcule la distance entre chaque groupe d'individus, on peut aussi ajouter une pondération en fonction de l'effectif du groupe avec le paramètre « ward ». Après avoir calculé la distance, il regroupe pas à pas les individus ayant les plus petites distances entre eux. Cette distance est calculée par le critère du saut minimum, c'est-à-dire, la distance minimal entre les individus les plus proches de chacun des groupes. À chaque itération la matrice des distances est recalculée, le regroupement est de nouveau effectué et ainsi de suite jusqu'à ce que tous les groupes soient reliés. Nous verrons plus tard à quoi ce dernier ressemble-t-il et comment l'interpréter.

Nous utilisons ce modèle avec scipy avec le module dendrogram.

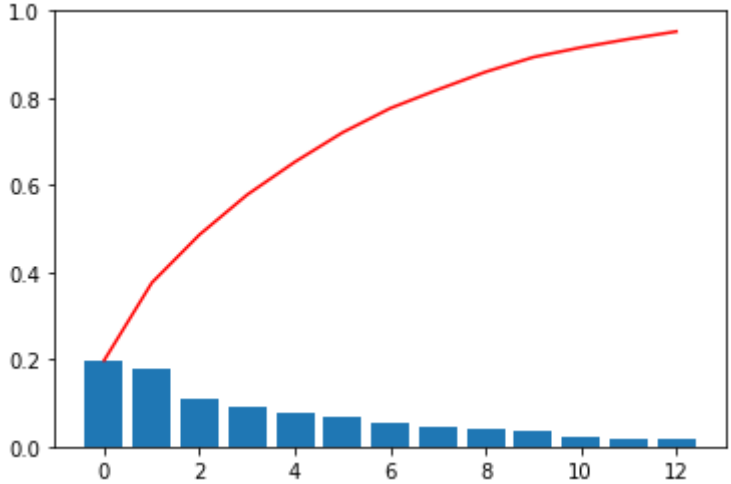
4.2.2. Les variables pris en compte lors du clustering

Afin de réduire le nombre de dimensions du jeu de données, nous avons utilisé l'Analyse en Composantes Principales (ACP). Son objectif est de transformer des variables corrélées en nouvelles variables décorrélées les unes des autres. Ces nouvelles variables sont nommées « composantes principales ». Cela permet une visualisation simplifiée et une accélération des calculs.

Un diagramme de pareto permet de s'assurer que les composantes retenues seront bien les plus importantes. Ce dernier est représenté par un histogramme trié par ordre décroissant en fonction de l'importance. Une ligne rouge représente l'importance relative de la variabilité. On a donc :



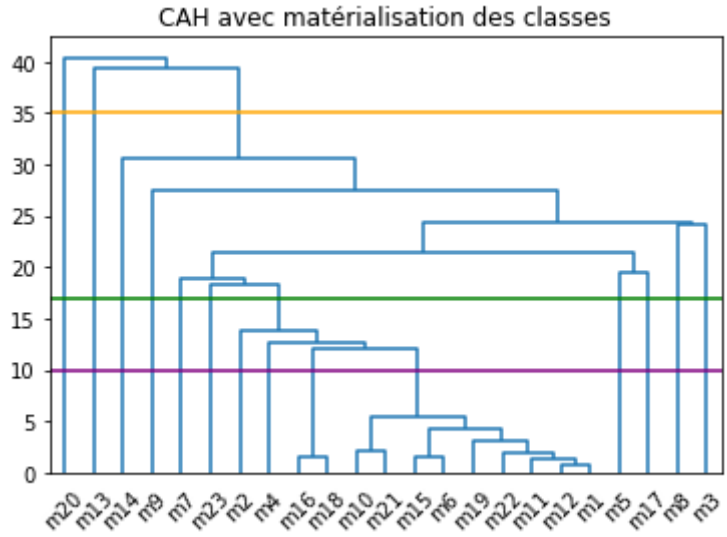
En regardant à partir de quel moment la courbe commence à s'aplatir (la méthode du coude), on remarque qu'à partir d'environ 13 composantes, on obtient la quasi-totalité de la variabilité, soit :



C'est donc à partir de ces 13 composantes que nous avons fait le clustering.

4.2.3. Comment les clusters ont-ils été choisis ?

La méthode CAH étant une méthode visuelle, nous pouvons afficher le dendrogramme créé :

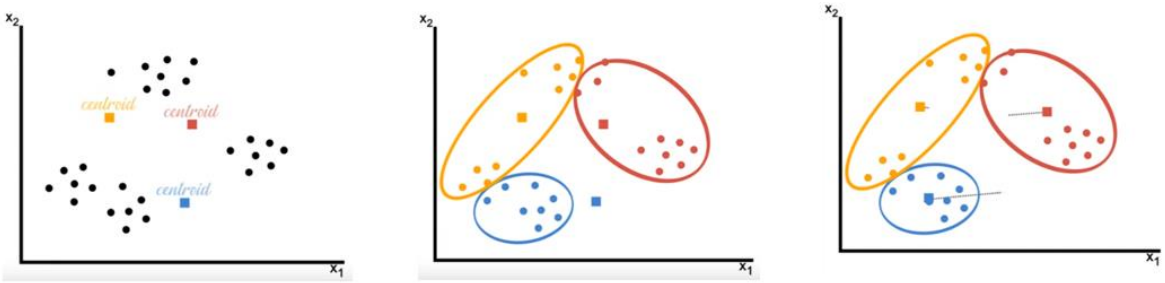


La hauteur des branches indique le niveau de proximité des regroupements. Pour compter les clusters, il suffit de sélectionner une hauteur, de tracer une ligne horizontale et de compter son nombre de croisement avec les lignes du dendrogramme. Un souci ici est que si on se réfère uniquement à la hauteur des branches, nous ne sommes pas sûr de notre choix, soit entre 3 (en jaune), 11 (en vert) ou 14 clusters (en violet).

Pour choisir entre les trois, nous nous appuierons sur le modèle KMeans avec la méthode du coude.

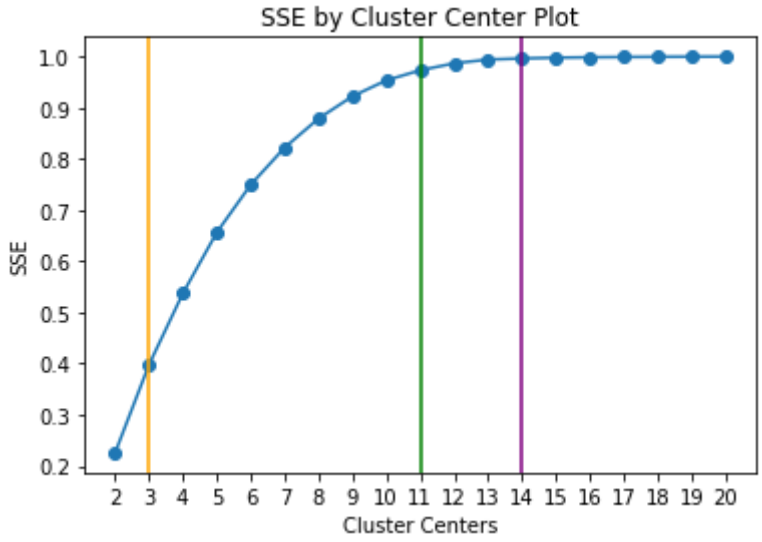
Le principe du modèle KMeans est de placer dans un premier temps K centroids des K clusters au hasard dans le dataset. Puis chaque point du dataset est affecté au centroïd le plus proche. Une fois que chaque point est relié à un centroïd, ces derniers sont déplacés au milieu de leur groupe (ou cluster), soit la moyenne des points du groupe. On continue de faire cela jusqu'à ce que les centroïdes ne bougent plus.

De manière visuelle, on a :

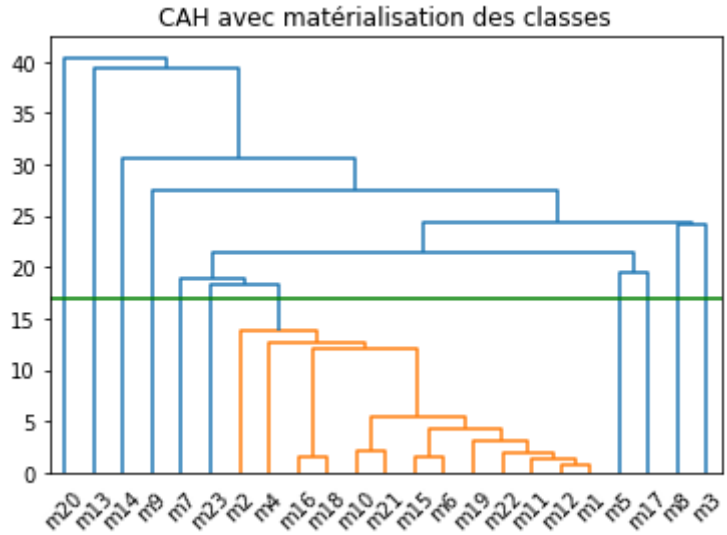


Pour utiliser la méthode du coude, il faut tester le modèle KMeans avec différents clusters et indiquer sur un graphique leur SSE. La SSE est la somme du carré de la distance entre le

centroid d'un cluster et ses membres : plus la SSE est importante et moins les clusters sont compactés (ici représenté en pourcentage du total de SSE).



Comme dit précédemment, la méthode du coude consiste simplement à choisir un nombre de clusters à partir du moment où la courbe s'aplatit. Après cela, la SSE ne change que très peu. Si l'on reprend les clusters trouvés via la CAH, on se rend bien compte que 3 et 14 ne sont pas un bon choix. Nous sommes donc partis sur 11 clusters :



Maintenant que nous avons déterminé le nombre de clusters, nous n'avons plus qu'à regrouper les classes par cluster et créer un nouveau data frame avec les classes correspondantes. C'est sur ce dernier que nous avons appliqué la classification.

4.3. Le modèle de classification

4.3.1. Choix du modèle : les arbres de classification

Entre les arbres et la LDA, sans aucune modification de la base de données, nous obtenons un meilleur score avec les arbres de classification soit 99. 98%.

4.3.2. Quelles sont les variables intégrées dans le modèle définitif ?

Avec toutes les variables, le modèle obtient un score de 99.98%. Comme on peut le voir ci-dessous, il n'y a que 2 variables ayant une vraie importance, si on les choisit, on obtient le même score que celui initial.

Variables ayant une importance supérieur à 0 :

	Variable	Importance
163	V165	0.425484
162	V164	0.318430
170	V172	0.047509
178	V189	0.044780
158	V159	0.040300
8	V9	0.032761
161	V163	0.016649
35	V36	0.015270
80	V81	0.013453
181	V195	0.009783
180	V193	0.007407
3	V4	0.006658
76	V77	0.006222
108	V109	0.004549
169	V171	0.002905
40	V41	0.002162

On se retrouve donc avec seulement 2 variables, V165 et V164, pour un score de 99.98%. Pour essayer de comprendre pourquoi prendre seulement ces deux variables expliquent toujours aussi bien la variable cible regroupée, nous avons regardé leur distribution et il s'avère qu'elles ont comme point communs d'avoir une modalité (0) fortement présente (la quasi-totalité de la distribution). Étant donné que nous avons nous même un cluster représentant la quasi-totalité du dataset, on peut en conclure que le regroupement que nous avons effectué a suivi cette logique.

4.3.3. Évaluation du modèle définitif

Même problème que précédemment quant à la matrice de confusion.

Néanmoins, pour obtenir plus de précision quant aux prédictions, on peut regarder le rapport de classification de sklearn :

	precision	recall	f1-score	support
1	1.00	1.00	1.00	148196
2	0.67	1.00	0.80	4
3	1.00	0.86	0.92	7
4	0.00	0.00	0.00	0
5	0.00	0.00	0.00	0
6	0.00	0.00	0.00	0
7	0.00	0.00	0.00	0
8	0.00	0.00	0.00	0
9	0.00	0.00	0.00	0
10	0.00	0.00	0.00	0
11	0.00	0.00	0.00	0
accuracy			1.00	148207
macro avg	0.24	0.26	0.25	148207
weighted avg	1.00	1.00	1.00	148207

Même remarque que lors de l'évaluation de la classification, malgré que certaines classes ne soient pas présentes et grâce à l'écrasante majorité des classes m10,m12 et m19, cela n'affecte pas les résultats du modèle. On a effectivement la classe 1 prédite presque parfaitement (les résultats sont arrondis). On valide donc notre modèle.

5. Déploiement

Afin de voir comment réagissent nos modèles sur un nombre de données élevé, nous avons dupliqué la base de données jusqu'à arriver à 4 898 424 observations. Pour atteindre ce nombre, nous avons dupliqué 9 fois la base de données initiale, puis, étant donné qu'il manquait encore 452 235 individus, nous avons fait le choix de simplement rajouter les 452 235 premiers individus du dataset. La création se fait en un peu moins de 8 minutes.

Une fois la base de données de déploiement créée, nous avons créé 3 applications : une pour la classification, une autre pour le scoring et une dernière pour la classification des données regroupées.
Pour les deux premières, nous avons sélectionné les colonnes en fonction du modèle utilisé, puis, nous avons standardisé les variables numériques et transformé celles qualitatives via LabelEncoder. Pour finir, il ne restait plus qu'à importer les modèles correspondants via le package pickle et à les appliquer.
L'application classification s'exécute en en environ 1 minute et fournit un taux d'erreur de 0.006, ce qui est très bien. Quant à l'application scoring, il nous était demandé de prendre en compte les 10 000 scores les plus élevés, on obtient alors 7956 positives contre 2044 négative en moins de 2 minutes.
Pour la classification des données regroupées, il nous a d'abord fallu rajouter la colonne V200_Prim en utilisant le fichier "clustering.csv". Cette colonne représente le numéro de cluster de chaque modalité de V200. Ensuite, nous avons juste standardisé les données, vu

que les variables V165 et V164 sont quantitatives. L'application des données regroupées s'exécute en moins de 1 minute 30.

6. Conclusion

Pour conclure, nous n'avons trouvé que des résultats quasi-parfaits, cela était assez déroutant car nous nous attendions à pouvoir améliorer significativement le modèle que ce soit via la manipulation des hyper-paramètres, la sélection de variables ou bien le choix du modèle.

Malgré tout, cela nous a permis de nous poser des questions sur notre travail, et donc de creuser encore plus. Cela nous a aussi montrer que des fois, nous allons trop loin au lieu de faire les choses simplement, notamment pour la CAH avec le regroupement de données ou nous avons commencé à essayer de comprendre et tester toute sorte d'algorithme de clustering alors qu'un simple regroupement via le barycentre suffisait.

7. Annexes

7.1. Sources

http://eric.univ-lyon2.fr/~ricco/cours
https://sites.google.com/view/aide-python/statistiques/analyses-en-composantes-principales
https://stackoverflow.com/
https://scaron.info/doc/intro-arbres-decision/

7.2. Packages

- sklearn
- pandas
- time
- os
- scipy
- numpy
- pickle
- matplotlib
- collections