SOD322 : Recherche opérationnelle et données massives

Projet RODM - Classifcation associative

Réalisé par :
Yassine Filali
Mohamed El Khames Boumaiza

Classe: 3éme année
Parcours: Science de l'optimisation des données

Encadré par :
M.Zacharie Ales

Année universitaire 2020/2021

Introduction

Lors de ce projet, on mettra en application la méthode des règles ordonnées pour la classification vue en cours sur une base de données relative à un ensemble des individus et dont la variable cible est leurs revenus. Cette dernière dépend d'un ensemble de donnée à l'instar de l'âge, l'occupation, les heures de travail par semaines et plein d'autres.
Pour former une idée plus claire sur ce qu'on va faire, l'histogramme suivant nous renseigne sur la répartition des individus sur les deux classes.

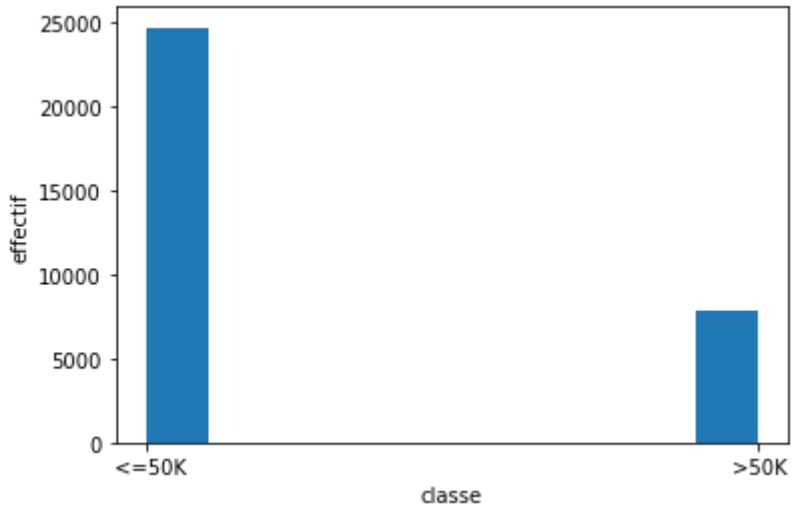

Figure 1: Histogramme de la répartition des classes

Pour ce faire, on va entamer par une représentation binaire des caractéristiques étant donné que la méthode ORC nécessite que chaque observation soit représente par un vecteur de caractéristiques binaires. Dans un second temps, les règles seront génères et classé selon les principes de la méthodes avant d'arriver à la partie de l'évaluation. Cette dernière sera en terme de précession et de rappel sur l'ensemble des données de test aussi bien que sur celles de test d'entraînement

Analyse exploratoire

Comme nous l'avons évoqué, nous utilisons ensuite ces données pour construire un classifieur capable de prédire si un individu a gagné plus ou moins de 50k $
L'ensemble de données sur le revenu du recensement compte 32 561 entrées. Chaque entrée,à propos d'un individu, contient les informations suivantes:

- **âge** : l'âge d'un individu : Entier supérieur à 0
- **workclass** : un terme général pour représenter le statut d'emploi d'un individu
- **fnwgt** : poids final. En d'autres termes, il s'agit du nombre de personnes que le recensement croit l'entrée représente

- **marital status** : l'état matrimonial d'un individu.
- **education** :le plus haut niveau d'éducation atteint par un individu
- **education num** : le plus haut niveau d'éducation atteint sous forme numérique
- **occupation** : le type général d'occupation d'un individu
- **relationship** : représente ce que cet individu est par rapport aux autres.
- **sex** :le sexe biologique de l'individu
- **capital gain** : gains en capital de l'individu
- **target income** : si un particulier gagne ou non plus de 50 000 $ par année

L'ensemble de données d'origine contient une distribution de 23,93% d'entrées étiquetées avec $>$ 50k et 76,07% entrées étiquetées avec $<=$ 50k.
L'ensemble de données est en suite devisé en ensembles d'entraînement et de test.

Pour obtenir des informations sur les variables les plus utiles , nous examinons la distribution des entrées des deux classes. Nous faisons cela en espérant identifier celles qui fournissent peu d'informations afin de simplifier notre modèle en termes de complexité et de durée d'exécution.

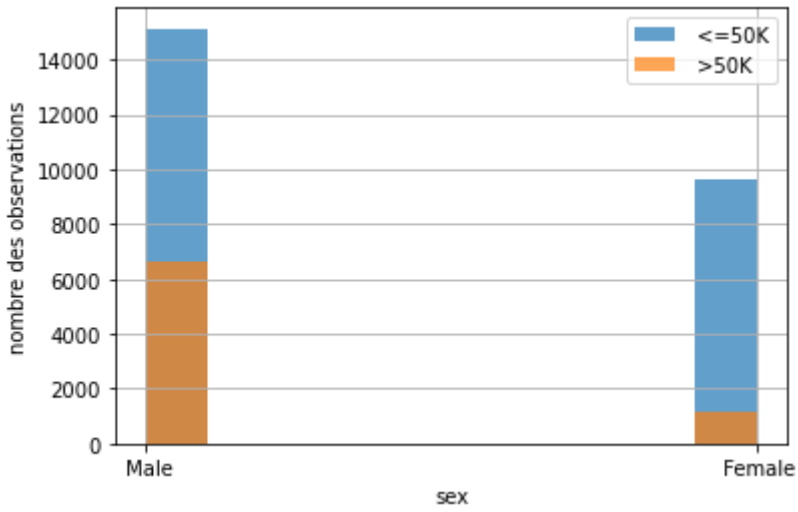

Figure 2: **Répartition du sexe des individus en fonction de la classe** On peut observer que les hommes sont plus susceptibles d'avoir un revenu supérieur à 50k que les femmes, c'est à dire que la proportion des hommes ayant un revenu élevé parmi tous les hommes, est supérieure à des femmes ayant un revenu élevé parmi toutes les femmes

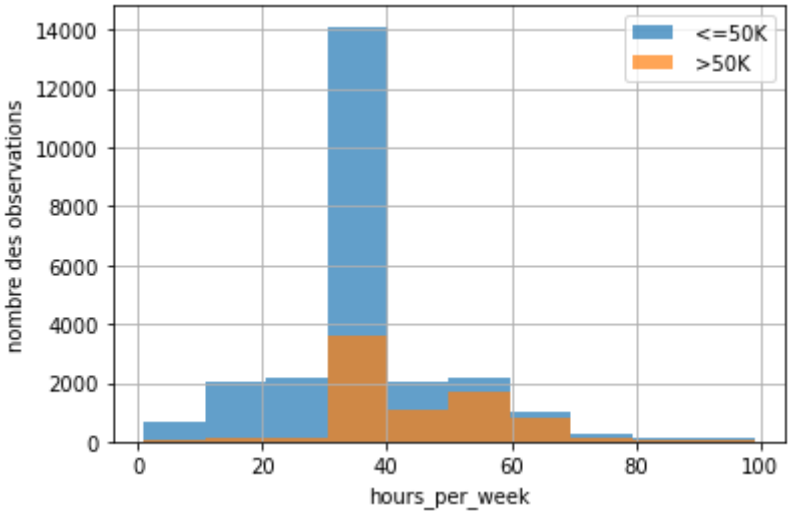

Figure 3: **Répartition des heures de travail par semaine en fonction de la classe** On peut observer que les individus dont le revenu est élevé sont moins susceptibles de travailler moins de 30 heures par semaine.

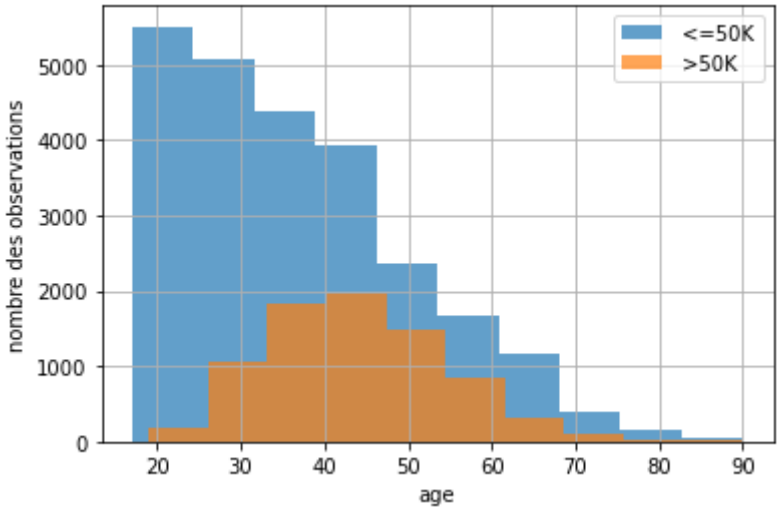

Figure 4: **Histogramme de la distribution de la variable âge**: En regardant le graphique, nous pouvons voir qu'il y a un écart significatif entre le rapport $>$ 50k à $<=$ 50k entre les groupes d'âge. L'information la plus intéressante à noter est celle des groupes d'âge entre 17 et 20 et entre 70 et90 où il y a presque aucune chance d'avoir un revenu supérieur à 50K

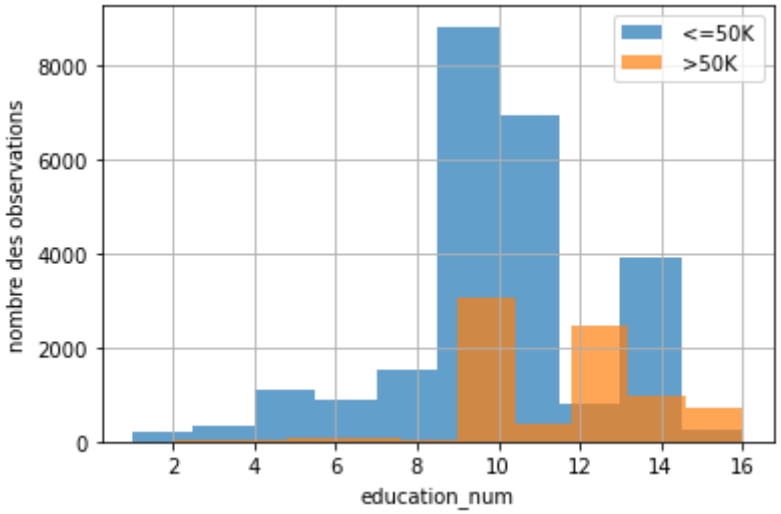

Figure 5: **Histogramme de la distribution de la variable education num** La plupart les individus de l'ensemble de données ont au plus un diplôme d'études secondaires alors qu'une petite partie seulement avoir un doctorat. En gros, un niveau d'éducation plus élevé est corrélé à un pourcentage plus élevé d'individus ayant un revenu supérieur à 50k

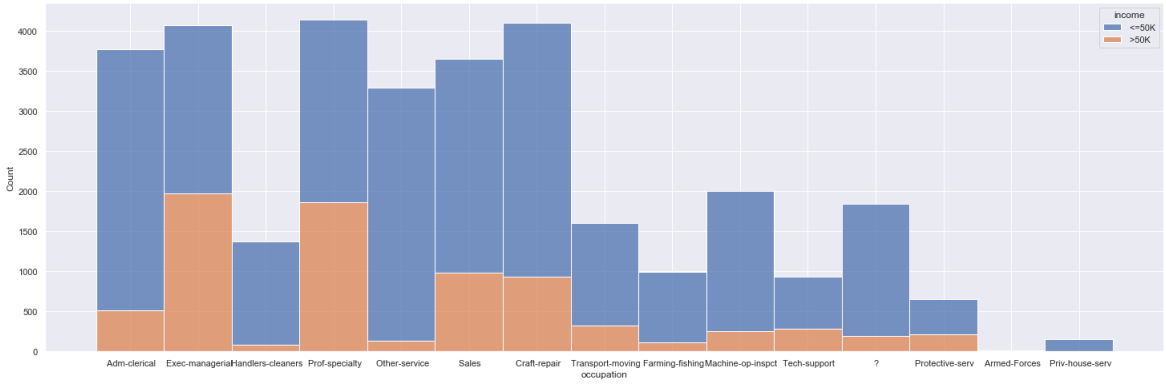

Figure 6: **Répartition des postes occupées en fonction de la classe** On remarque que pour la classe de personnes gagnant plus de 50k, il y a des postes qui sont plus répandus que d'autres, comme exec-managerial ou Prof-speciality

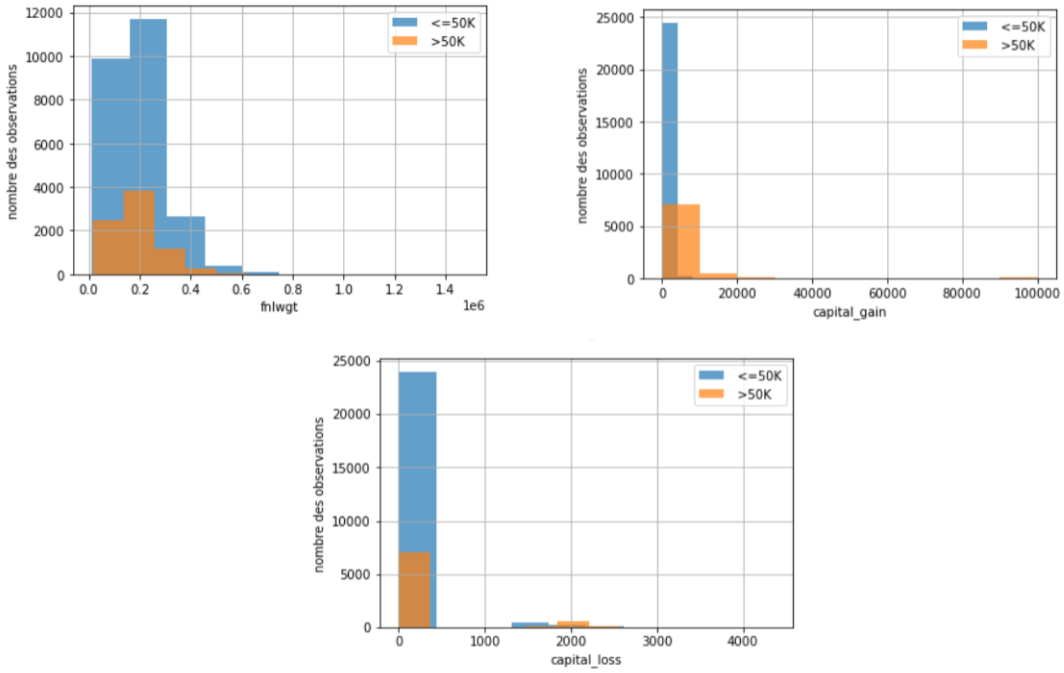

Figure 7: **Histogramme de la distribution de des variables fnlwgt, capital loss et capital gain**

Importance des variables On peut obtenir l'importance des différentes variables en ajustant un classifieur basé sur le arbres de décision comme les forêts aléatoires basée sur l'indice de diversité de Gini. Ainsi, plus une variable a une importance élevée, plus elle contribue à l'amélioration de a performance du classifieur (forêt aléatoire). Ainsi, on peut en déduire quelles variables sont les plus significatives dans notre problème.

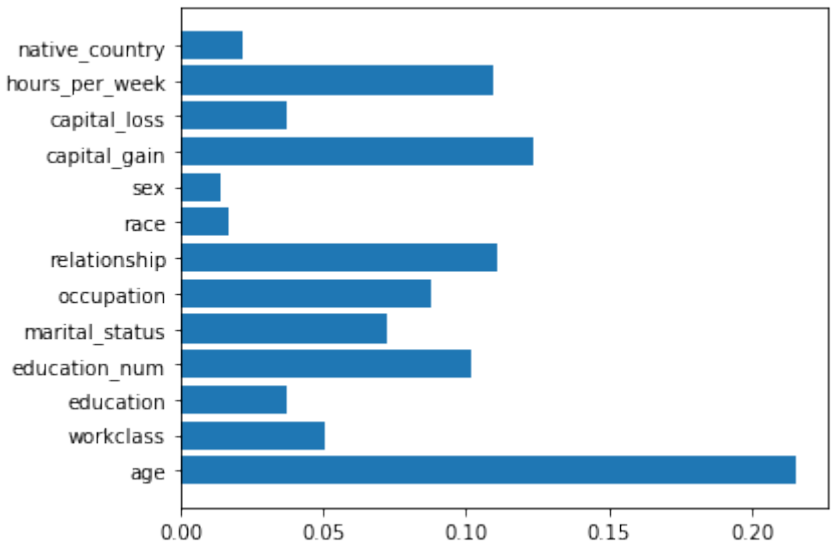

Figure 8: Importance des différentes variables dans un forêt aléatoire.

Binarisation des données

Afin de pouvoir utiliser les données dans notre classifier, il est nécessaire de binariser, c'est à dire, transformer les features en des features binaires ne pouvant avoir que la valeur 0 ou 1, c'est à dire, un one hot encoding en quelque sortes. Au vu du temps de calcul pris par l'algorithme, il est tout à fait raisonnable de ne pas tout considérer. Du coup, on considère un nombre réduit de variables pour effectuer la binarisation.

En tenant compte de l'analyse exploratoire réalisée sur les données, et en tenant compte de différentes combinaisons de choix, du temps d'éxecution et de la performance finale, on se propose d'effectuer une binarisation des variables selon le schéma suivant :

- On transforme la variable âge en 4 variables binaires, suivant l'appartenance de l'individu à une certaine tranche d'âge. Les tranches d'âge considérées sont $[0, 20]$, $[20, 30]$, $[30, 45]$ et $[45, +\infty]$
- En observant les histogrammes pour capital_gain, on peut voir que les individus ayant un salaire supérieur à 50 milles, dont plus susceptibles d'avoir un capital_gain non nul. Ainsi on transforme cette variable en une variable binaires, suivant si le gain de capital est supérieur ou non à un certain seuil fixé à 5000.
- On transforme la varible sex en une variable binaire, qui prend 1 si l'individu est une femme, et 0 si c'est un homme.
- On transforme la variable hours_per_week en 5 variables binaires suivant l'appartenance de de la valeur à un certain intervalle. Les intervalles considérés sont $[0, 20]$, $[20, 30]$, $[30, 40]$, $[40, 50]$ et $[50, +\infty]$
- On transforme la variable education_num en 3 variables binaires suivant l'appartenance de de la valeur à un certain intervalle. Les intervalles considérés sont $[0, 9]$, $[9, 12]$ et $[12, +\infty]$,

En procédant ainsi, on obtient un total de 14 variables binaires que l'on peut utiliser pour ajuster notre classifieur.

Génération des règles de classification

La Machine utilisée pour réaliser tous les tests comporte 16 Gb de RAM et un processeur disposant de 8 coeurs (i7-9750H CPU @ 2.6 GHz). Le solveur utilisé est CPLEX et sa version est 12.10.0.

Pour nos données binarisées de la façon décrite à la partie précédente, le modèle a généré 53 règles :

- 14 règles associées à la classe 1 qui représente les individus gagnant plus de 50K.
- 39 règles associées à la classe 0 qui représente les individus gagnant moins de 50K.

Le temps mis pour générer ces règles est de 77.78 secondes.

Classement des règles

Le classifieur utilise ainsi ces règles, ordonnées afin d'effectuer la classification. On aboutit au final à un total de 3 règles:

- 2 règles associées à la classe 0 qui représente les individus gagnant moins de 50K.
- 1 règle associée à la classe 1 qui représente les individus gagnant plus de 50K.

Les règles ordonnées qui sont obtenues sont présentées dans le tableau suivant:

Classe	Règle													
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Le temps de classification des règles est plafonné à 300 secondes en raison du paramètre iterlim qui plafonne le temps d'éxécution du solveur dans cette étape.
La classification des observation se fait en prenant la première règle vérifiée par l'individu parmi les règles classées, et on lui associe la classe associée à cette règle là.

Résultats

Les résultats obtenus par notre classifieur sont:

	Train			Train		
	Precision	Recall	Size	Precision	Recall	Size
Class 0	0.81	0.99	787	0.79	0.99	23933
Class 1	0.88	0.18	230	0.91	0.18	7611
Average	0.84	0.59	-	0.85	0.59	-
Weighted Average	0.82	0.81	-	0.82	0.8	-

Table 1: Performances du classifieur

On peut noter que la métrique de rappel est faible pour la classe 1, que ce soit sur les données d'entraînement ou de test, ceci est principalement dû au fait que nos données ne sont pas équilibrées, que le nombre d'observations appartenant à la classe 0 est 3 fois plus grand que ceux appartenant à la classe 1.

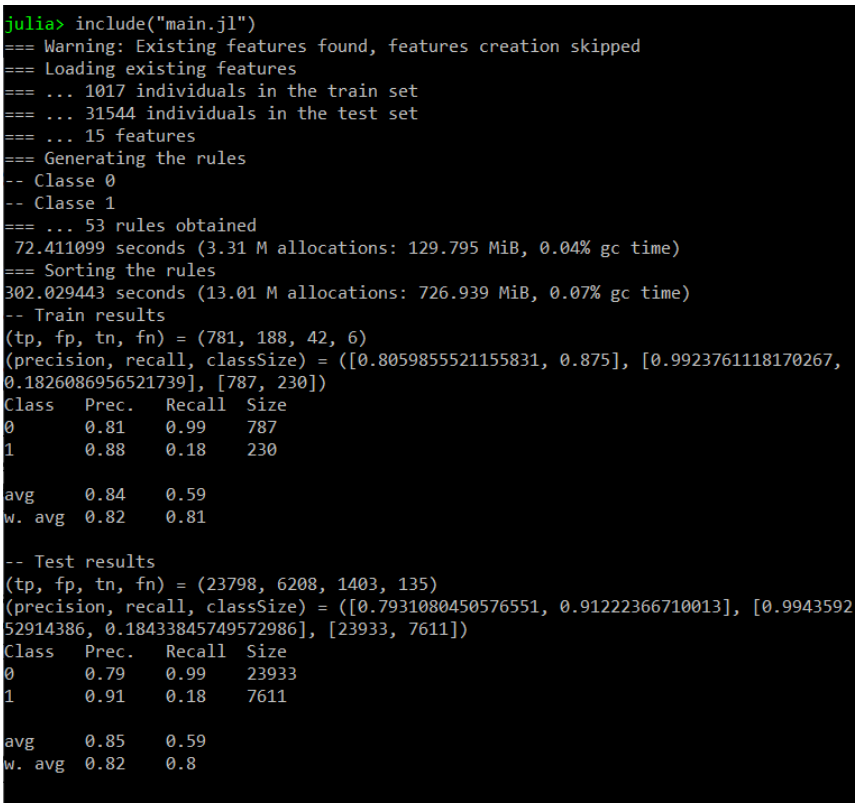

Figure 9: Résultats compilés pour la binarisation choisie

Reproduction du code

Pour exécuter le code, il est nécessaire de fixer le chemin sur l'invite de commande de Julia sur le dossier **src** qui contient les codes main.jl et function.jl. Pour exécuter le code, il suffit de taper la commande include("main.jl").
Si l'on veut refaire tourner le code pour d'autres variables après l'avoir fait tourné une fois, il est nécessaire de supprimer les fichier adult_rules.csv et adult_ordered_rules.csv dans le dossier res, ainsi que les fichier adult_train.csv et adult_test.csv dans le dossier data.
Si l'on veut modifier iterlim, il suffit de la changer dans main.jl.
On peut tester les différentes binarisations utilisée (ou bien des combinaison) en décommentant certaines parties dans la fonction CreateFeatures du fichier Functions.jl.

Questions d'ouverture

Question 1

On va considérer deux autres binarisations, avec plus de variables totales :

- La première consiste à rajouter la contribution de la variable catégorielle workclass à la binarisation choisie précédemment. Ainsi nous introduisons en plus 9 variables binaires associées à la variable workclass, ce qui nous amène à un total de 23 variable binaire. Les réultats obtenus pour cette binarisation sont les suivant :

	Train			Train		
	Precision	Recall	Size	Precision	Recall	Size
Class 0	0.8	0.99	773	0.79	0.99	23947
Class 1	0.91	0.2	244	0.91	0.19	7597
Average	0.85	0.6	-	0.85	0.59	-
Weighted Average	0.82	0.8	-	0.82	0.8	-

Table 2: Performances du classifieur avec 23 variables

On génère au total 59 règle, et on en utilise 9 pour faire la classification.

```
julia> include("main.jl")
=== Creating the features
=== ... 1017 individuals in the train set
=== ... 31544 individuals in the test set
=== ... 24 features
=== Generating the rules
-- Classe 0
-- Classe 1
=== ... 59 rules obtained
103.308550 seconds (43.50 M allocations: 2.134 GiB, 0.82% gc time)
=== Sorting the rules
306.538123 seconds (25.38 M allocations: 1.307 GiB, 0.18% gc time)
-- Train results
(tp, fp, tn, fn) = (768, 194, 50, 5)
(precision, recall, classSize) = ([0.7983367983367984, 0.9090909090909091],
 [0.9935316946959897, 0.20491803278688525], [773, 244])
Class   Prec.   Recall  Size
0       0.8     0.99    773
1       0.91    0.2     244

avg     0.85    0.6
w. avg  0.82    0.8

-- Test results
(tp, fp, tn, fn) = (23808, 6190, 1407, 139)
(precision, recall, classSize) = ([0.793652910194013, 0.9100905562742562],
[0.9941955150958366, 0.18520468606028695], [23947, 7597])
Class   Prec.   Recall  Size
0       0.79    0.99    23947
1       0.91    0.19    7597

avg     0.85    0.59
w. avg  0.82    0.8
```

Figure 10: Résultats compilés pour une binarisation avec 23 variables

- La seconde consiste à rajouter à la binarisation précédente (celle avec 23 variables binaires donc) la contribution de la variable relationship, ce qui rajoute encore variable binaire, qui nous fait un total de 29 variable binaire. Les résultats obtenus sont :

	Train			Train		
	Precision	Recall	Size	Precision	Recall	Size
Class 0	0.82	0.96	763	0.82	0.96	23957
Class 1	0.77	0.37	254	0.75	0.34	7587
Average	0.79	0.66	-	0.78	0.65	-
Weighted Average	0.81	0.81	-	0.8	0.81	-

Table 3: Performances du classifieur avec 29 variables

On génère au total 54 règle, et on en utilise 7 pour faire la classification.

```
julia> include("main.jl")
=== Creating the features
=== ... 1017 individuals in the train set
=== ... 31544 individuals in the test set
=== ... 30 features
=== Generating the rules
-- Classe 0
-- Classe 1
=== ... 54 rules obtained
116.947385 seconds (4.82 M allocations: 209.088 MiB, 0.04% gc time)
=== Sorting the rules
305.533206 seconds (15.01 M allocations: 779.352 MiB, 0.07% gc time)
-- Train results
(tp, fp, tn, fn) = (735, 161, 93, 28)
(precision, recall, classSize) = ([0.8203125, 0.768595041322314], [0.9633027
52293578, 0.3661417322834646], [763, 254])
Class   Prec.   Recall  Size
0       0.82    0.96    763
1       0.77    0.37    254

avg     0.79    0.66
w. avg  0.81    0.81

-- Test results
(tp, fp, tn, fn) = (23088, 5033, 2554, 869)
(precision, recall, classSize) = ([0.8210234344440098, 0.7461291264972246],
[0.9637266769628918, 0.33662844339000925], [23957, 7587])
Class   Prec.   Recall  Size
0       0.82    0.96    23957
1       0.75    0.34    7587

avg     0.78    0.65
w. avg  0.8     0.81
```

Figure 11: Résultats compilés pour une binarisation avec 29 variables

On remarque que la première binarisation donne le meilleur résultat en terme de précision et de rappel sur l'ensemble des binarisation mentionnées. Et on peut voir que la seconde donne une performance moins bonne, ce qui confirme que le fait de choisir beaucoup de variables, en plus d'augmenter le temps de calcul (On passe de 29 seconde pour 14 variables pour générer les règles, à 100 secondes pour 23 variable, puis 117 secondes pour 29 variables), n'assure pas forcément une meilleure performance.
On remarque aussi que plus le nombre de variables augmente, plus le nombre de règles classées diminie, ce qui est dû principalemet au fait que la résolution par le solveur dure plus de temps si l'on a plus de variables, et comme le temps d'exécution de l'étape de classement est limité par le paramètre iterlim, on en a donc moins.

Question 6

On considère le modèle avec la binarisation 14 variable initialement introduite, et on fait varier le paramètre iterlim pour contrôler le temps de résolution pour l'étape de classement des règles de décision. On fait varier le paramètre iterlim vers 800, puis 1000, puis 1500.

- Pour iterlim = 800, on obtient les performances suivantes.

	Train			Train		
	Precision	Recall	Size	Precision	Recall	Size
Class 0	0.83	0.95	775	0.83	0.95	23945
Class 1	0.69	0.39	242	0.73	0.4	7599
Average	0.76	0.67	-	0.78	0.68	-
Weighted Average	0.8	0.81	-	0.81	0.82	-

Table 4: Performances du classifieur pour iterlim = 800

Ainsi, on génère au total 50 règles et on utilise 17 règles classées dans le classifieur.

```
julia> include("main.jl")
=== Creating the features
=== ... 1017 individuals in the train set
=== ... 31544 individuals in the test set
=== ... 15 features
=== Generating the rules
-- Classe 0
-- Classe 1
=== ... 50 rules obtained
 55.385364 seconds (3.06 M allocations: 123.470 MiB, 0.06% gc time)
=== Sorting the rules
802.360735 seconds (12.36 M allocations: 694.489 MiB, 0.02% gc time)
-- Train results
(tp, fp, tn, fn) = (733, 147, 95, 42)
(precision, recall, classSize) = ([0.8329545454545455, 0.6934306569343066], [0.9458064516129032,
0.3925619834710744], [775, 242])
Class   Prec.   Recall  Size
0       0.83    0.95    775
1       0.69    0.39    242

avg     0.76    0.67
w. avg  0.8     0.81

-- Test results
(tp, fp, tn, fn) = (22841, 4574, 3025, 1104)
(precision, recall, classSize) = ([0.8331570308225424, 0.7326229111164931], [0.9538943411985801,
0.3980786945650743], [23945, 7599])
Class   Prec.   Recall  Size
0       0.83    0.95    23945
1       0.73    0.4     7599

avg     0.78    0.68
w. avg  0.81    0.82
```

Figure 12: Résultats compilés pour iterlim = 800

- Pour iterlim = 1000, on obtient les performances suivantes.

	Train			Train		
	Precision	Recall	Size	Precision	Recall	Size
Class 0	0.87	0.92	782	0.86	0.91	23938
Class 1	0.69	0.56	235	0.65	0.52	7606
Average	0.78	0.74	-	0.75	0.72	-
Weighted Average	0.83	0.84	-	0.81	0.82	-

Table 5: Performances du classifieur pour iterlim = 1000

Ainsi, on génère au total 41 règles et on utilise 11 règles classées dans le classifieur.

```
julia> include("main.jl")
=== Creating the features
=== ... 1017 individuals in the train set
=== ... 31544 individuals in the test set
=== ... 15 features
=== Generating the rules
-- Classe 0
-- Classe 1
=== ... 41 rules obtained
 57.879319 seconds (42.05 M allocations: 2.079 GiB, 1.41% gc time)
=== Sorting the rules
1005.724360 seconds (20.24 M allocations: 1.063 GiB, 0.03% gc time)
-- Train results
(tp, fp, tn, fn) = (723, 104, 131, 59)
(precision, recall, classSize) = ([0.8742442563482467, 0.6894736842105263], [0.9245524296675192,
0.5574468085106383], [782, 235])
Class   Prec.   Recall  Size
0       0.87    0.92    782
1       0.69    0.56    235

avg     0.78    0.74
w. avg  0.83    0.84

-- Test results
(tp, fp, tn, fn) = (21794, 3639, 3967, 2144)
(precision, recall, classSize) = ([0.8569181771713915, 0.6491572574046801], [0.9104352911688529,
0.5215619247962135], [23938, 7606])
Class   Prec.   Recall  Size
0       0.86    0.91    23938
1       0.65    0.52    7606

avg     0.75    0.72
w. avg  0.81    0.82
```

Figure 13: Résultats compilés pour iterlim = 1000

- Pour iterlim = 1500, on obtient les performances suivantes.

	Train			Train		
	Precision	Recall	Size	Precision	Recall	Size
Class 0	0.85	0.95	796	0.83	0.95	23924
Class 1	0.71	0.41	221	0.69	0.38	7620
Average	0.78	0.68	-	0.76	0.66	-
Weighted Average	0.82	0.84	-	0.79	0.81	-

Table 6: Performances du classifieur pour iterlim = 1500

Ainsi, on génère au total 46 règles et on utilise 12 règles classées dans le classifieur.

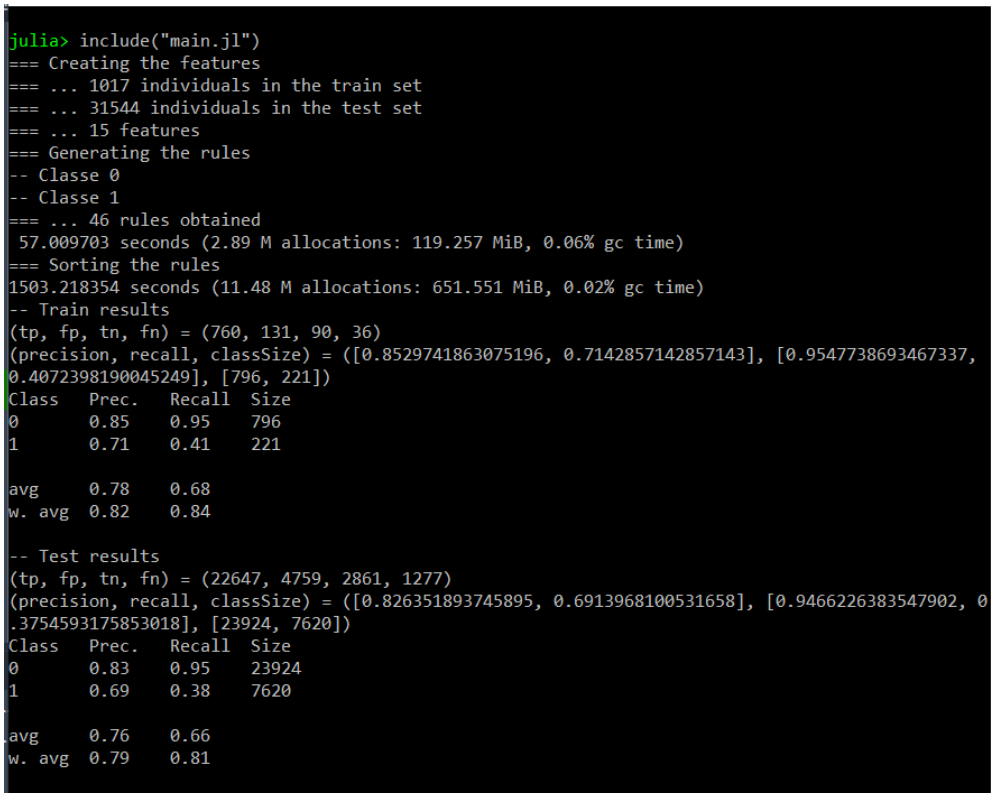

Figure 14: Résultats compilés pour iterlim = 1500

On remarque que dans le cas de notre classifieur avec 14 variable, il n'y a pas vraiment de changement net dans les résultat, juste que le résultat avec iterlim = 1000 est le meilleur de tous les essais faits sur ce modèle la, même si la différence est minime.
Cependant, dans certains cas où l'on utilise énormément de variable, 300 secondes peut ne pas être suffisant pour classifier les règles, du coup le modèle effectue des prédiction en se basant seulement sur la règle nulle, ce qui engendre des NaNs. Dans ce cas, il faut augmenter le paramètre iterlim pour espérer avoir une solution.