Projet Analyse de données

Marilyne Mafo

Léonard Pannetier

MAIN4

Nous allons présenter ici notre projet d'analyse de données sur une base de données rassemblant de nonbreuses caractéristiques sur 20052 plats de cuisines du monde entier. Cette base publique est disponible sur Kaggle avec ce lien https://www.kaggle.com/code/upsylend/pr-diction-sur-des-recettes-de-cuisine/input.

La base possède notamment une variable rating (entre 0.0 et 5.0) représentant si un plat est apprécié/goutu, le but ce cette analyse est de prédire la valeur de la variable rating en fonction des autres variables ainsi que faire de la classification des plats afin d'en regrouper certains.

La base de données ayant beaucoup de variables, nous avons décidé de ne garder que les principales, de plus énormément de variables n'ont que 2 modalitées (comme la variable disant si oui ou non il y a du bacon dans le plat) qui vaut 1 que très rarement.

Nous ne garderons que les variables : - rating (appréciation du plat) - calories (énergie apporté par le plat) - protein (quantité de protéines) - fat (quantité de graisses) - sodium (quantité de sel) - alcoholic (0 ou 1, présence d'alcool) - bake (0 ou 1, plat rotie)

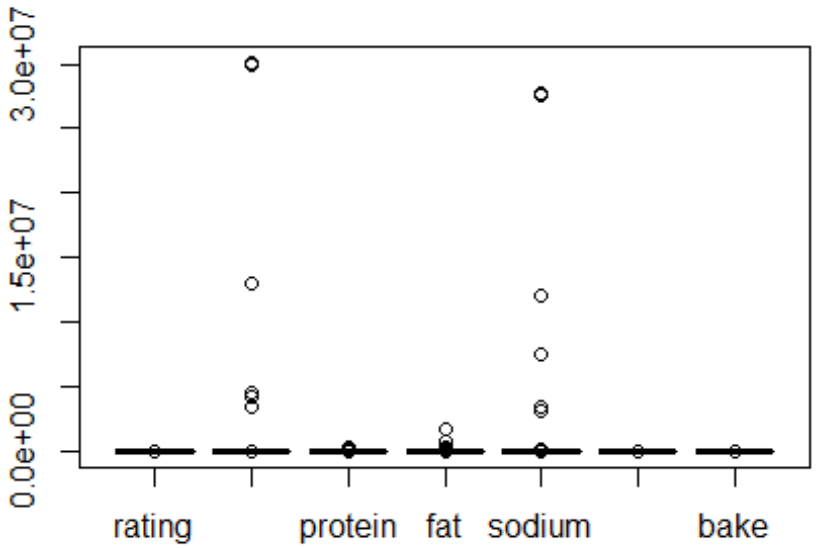
En supprimant les lignes avec au moins une valeur manquante sur ces variables nous n'avons plus que 15864 plats disponibles pour notre études.

```
library(tidyr)
rm(list=ls())
data = read.csv("epi_r.csv", sep =",")
data = data[c(2,3,4,5,6,15,39)]
data = data[data$calories != "",]
data = data[data$protein != "",]
data = data[data$fat != "",]
data = data[data$sodium != "",]
data = drop_na(data)
head(data)
```

```
##   rating calories protein fat sodium alcoholic bake
## 1  2.500      426      30   7    559         0    0
## 2  4.375      403      18  23   1439         0    1
## 3  3.750      165       6   7    165         0    0
## 5  3.125      547      20  32    452         0    1
## 6  4.375      948      19  79   1042         0    0
## 9  4.375      170       7  10   1272         0    0
```

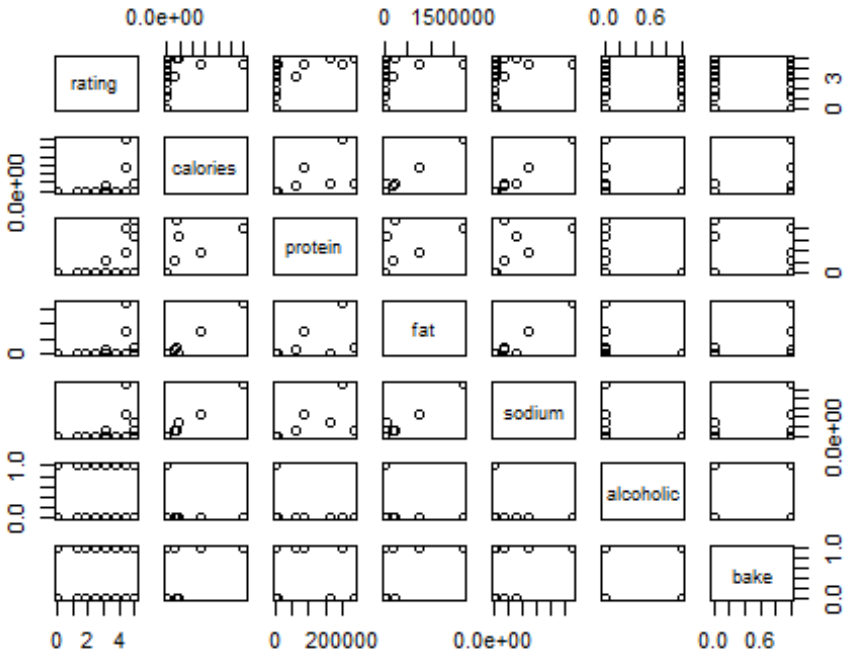
Partie 1 : Statistiques descriptives

```
boxplot(data)
```



On observe des grandes différences entre toutes nos variables, il faudra en prendre compte lors de l'analyse.

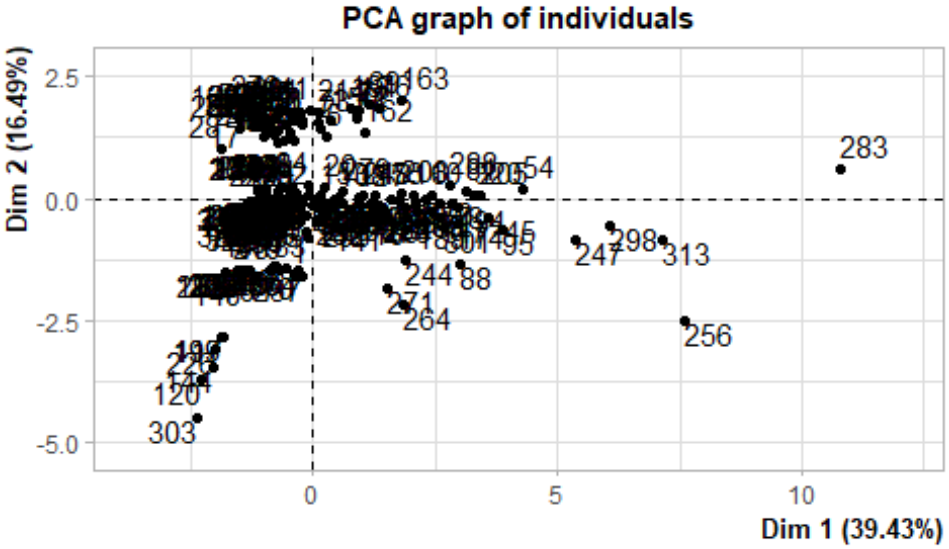
```
pairs(data)
```

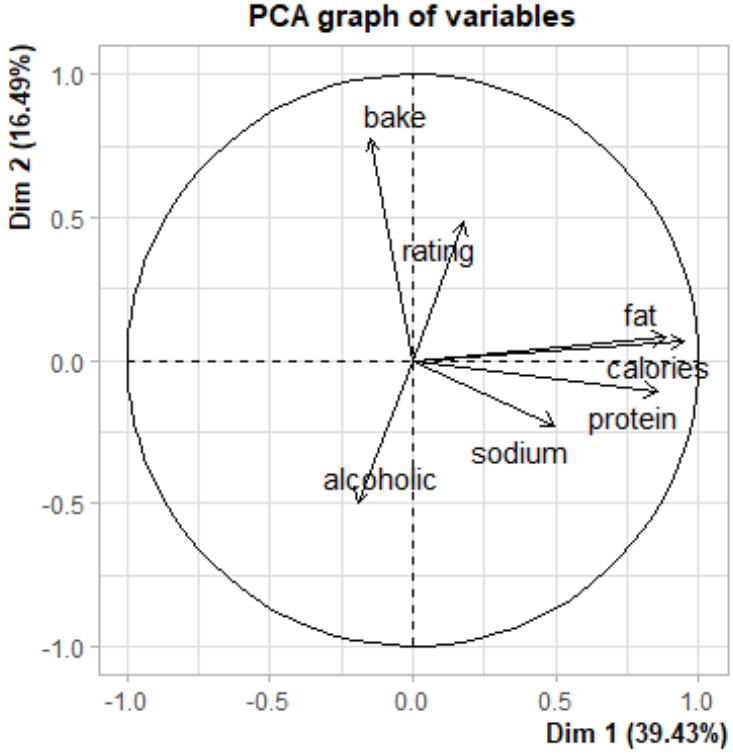


Ce diagramme permet de faire une analyse bivariée des données, si des données sont corrélées, on va observer une relation linéaire entre les deux variables.

Ainsi les variables calories et fat ont l'air d'être corrélées, ainsi que les variables calories et sodium. Nous allons vérifier cela en faisant une analyse des composantes principales (ACP).

```
tmpData = data[c(1:250),]
library(FactoMineR)
res <- PCA(tmpData)
```





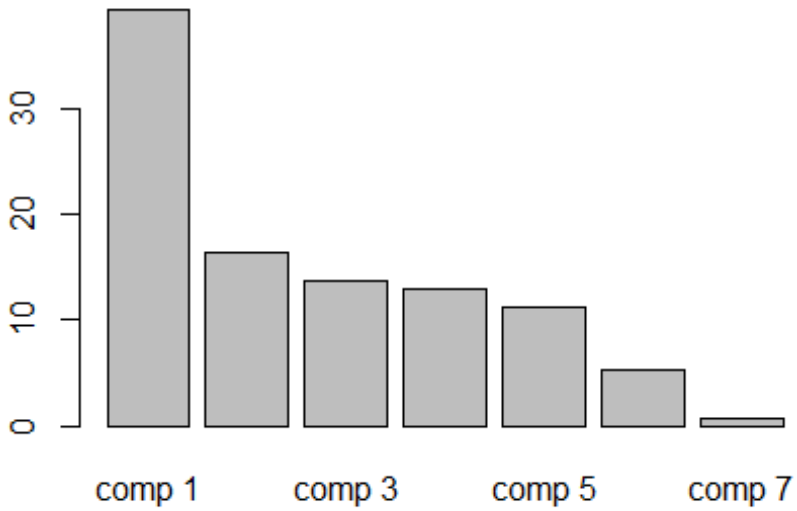

```
res

## **Results for the Principal Component Analysis (PCA)**
## The analysis was performed on 250 individuals, described by 7 variables
```

```
## *The results are available in the following objects:
##
##    name               description
## 1  "$eig"             "eigenvalues"
## 2  "$var"             "results for the variables"
## 3  "$var$coord"       "coord. for the variables"
## 4  "$var$cor"         "correlations variables - dimensions"
## 5  "$var$cos2"        "cos2 for the variables"
## 6  "$var$contrib"     "contributions of the variables"
## 7  "$ind"             "results for the individuals"
## 8  "$ind$coord"       "coord. for the individuals"
## 9  "$ind$cos2"        "cos2 for the individuals"
## 10 "$ind$contrib"     "contributions of the individuals"
## 11 "$call"            "summary statistics"
## 12 "$call$centre"     "mean of the variables"
## 13 "$call$ecart.type" "standard error of the variables"
## 14 "$call$row.w"      "weights for the individuals"
## 15 "$call$col.w"      "weights for the variables"
```

Ici on peut interpréter le graphe des variables en disant que l'axe 1 est fortement corrélé à fat, protein et calories. Cet axe représente alors à droite les recettes les plus caloriques, salées et protéinées (c'est à dire les plats assez lourds) et à gauche celles qui ne le sont pas(les plats légers). L'axe 2 quant à lui est corrélé positivement avec les variable bake et rating et négativement avec alcoholic. On peut dire alors que cet axe distingue les recettes les mieux notées en haut (c'est à dire les plats appréciés) contre celles mal notées et alcoolisées vers le bas.

Sur le graphe des individus on peut dire que la recette 283 (Ribs première braisées avec pois chiches et raisins secs) est très à droite, ainsi elle est très calorique et grasse . De plus le plat 303 (Café épicé à l'eau de vie) et en bas donc ce plat est alcoolisé et peu apprécié (il a un rating de 0).

```
barplot(res$eig[,2])
```




```
summary(res)

##
## Call:
## PCA(X = tmpData)
##
##
## Eigenvalues
##                        Dim.1   Dim.2   Dim.3   Dim.4   Dim.5   Dim.6
Dim.7
## Variance               2.760   1.154   0.967   0.913   0.790   0.376
0.041
## % of var.             39.425  16.491  13.808  13.038  11.290   5.364
0.584
## Cumulative % of var.  39.425  55.917  69.725  82.763  94.052  99.416
100.000
##
## Individuals (the 10 first)
##                Dist    Dim.1    ctr   cos2    Dim.2    ctr   cos2    Dim.3
## 1          |  1.309 | -0.085  0.001  0.004 | -0.804  0.224  0.377 | -0.670
## 2          |  2.043 |  0.164  0.004  0.006 |  1.439  0.718  0.496 |  0.170
## 3          |  1.247 | -0.998  0.144  0.640 | -0.275  0.026  0.049 | -0.049
## 5          |  1.936 |  0.276  0.011  0.020 |  1.259  0.549  0.423 | -0.402
## 6          |  2.920 |  2.413  0.844  0.683 |  0.082  0.002  0.001 |  0.251
## 9          |  1.390 | -0.559  0.045  0.162 | -0.271  0.025  0.038 |  0.197
## 10         |  1.518 |  1.213  0.213  0.638 | -0.443  0.068  0.085 | -0.121
## 11         |  1.239 | -0.962  0.134  0.603 | -0.231  0.019  0.035 | -0.029
```

```
## 13        |  2.403 |  0.932  0.126  0.150 |  1.794  1.115  0.558 |  0.253
## 14        |  1.198 | -0.726  0.076  0.367 | -0.073  0.002  0.004 |  0.281
##             ctr   cos2
## 1          0.185  0.262 |
## 2          0.012  0.007 |
## 3          0.001  0.002 |
## 5          0.067  0.043 |
## 6          0.026  0.007 |
## 9          0.016  0.020 |
## 10         0.006  0.006 |
## 11         0.000  0.001 |
## 13         0.026  0.011 |
## 14         0.033  0.055 |
##
## Variables
##              Dim.1    ctr   cos2    Dim.2    ctr   cos2    Dim.3    ctr
cos2
## rating    |  0.179  1.157  0.032 |  0.482 20.155  0.233 |  0.700 50.710
0.490
## calories  |  0.950 32.717  0.903 |  0.069  0.409  0.005 |  0.039  0.159
0.002
## protein   |  0.857 26.608  0.734 | -0.108  1.004  0.012 |  0.035  0.130
0.001
## fat       |  0.885 28.369  0.783 |  0.080  0.556  0.006 | -0.025  0.064
0.001
## sodium    |  0.498  8.982  0.248 | -0.226  4.434  0.051 | -0.081  0.671
0.006
## alcoholic | -0.195  1.380  0.038 | -0.496 21.338  0.246 |  0.683 48.232
0.466
## bake      | -0.147  0.788  0.022 |  0.776 52.105  0.601 | -0.018  0.033
0.000
##
## rating    |
## calories  |
## protein   |
## fat       |
## sodium    |
## alcoholic |
## bake      |
```

Observons le pourcentage d'inertie expliqué par les différents axes trouvés par l'ACP. Ainsi en voulant conservé 80% de l'inertie il faut conserver 4 axes.

```
res$var$contrib

##                Dim.1      Dim.2       Dim.3       Dim.4       Dim.5
## rating     1.1573034 20.1545623 50.71037772 21.31009668  6.25164090
## calories  32.7166938  0.4090521  0.15938687  6.55459396  0.20447743
## protein   26.6075245  1.0041028  0.12952180  0.07841764  0.06786703
## fat       28.3693610  0.5557865  0.06415940 10.29396840  1.59924198
```

```
## sodium     8.9816561  4.4336784  0.67137026 34.78935857 44.60136016
## alcoholic  1.3799447 21.3379957 48.23174606 16.85624631 12.07074773
## bake       0.7875165 52.1048221  0.03343789 10.11731844 35.20466478
```

En regardant la table des contributions, on remarque que le sodium contribue peu aux dimensions 1 à 3, sinon toutes les variables ont un poids important dans le calcule des 4 premiers axes.

Analyse factorielles des correspondances (AFC) :

Nous commencons par vérifier avec un test du Ki-deux que nos variables sont bien liées..

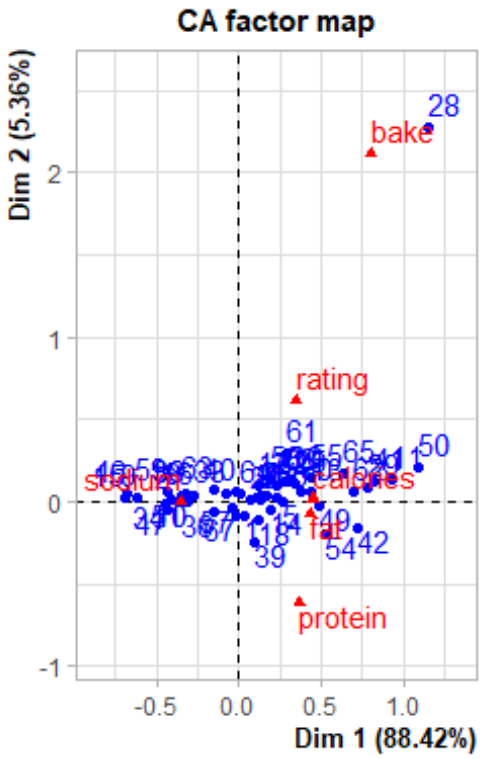
```
chisq.test(data[c(1:5000),])
```

```
## Warning in chisq.test(data[c(1:5000), ]): Chi-squared approximation may be
## incorrect

##
##  Pearson's Chi-squared test
##
## data:  data[c(1:5000), ]
## X-squared = 3512890, df = 29994, p-value < 2.2e-16
```

Nous remarquons que la p-value est inférieur à 5% ainsi nos variable sont bien liées, nous allons pousuivre avec l'analyse factorielles des correspondances.

```
data2 = data[c(1:50),]
res <- CA(data2)
```

```
## Warning in CA(data2): The columns alcoholic sum at 0. They were suppressed
from
## the analysis
```

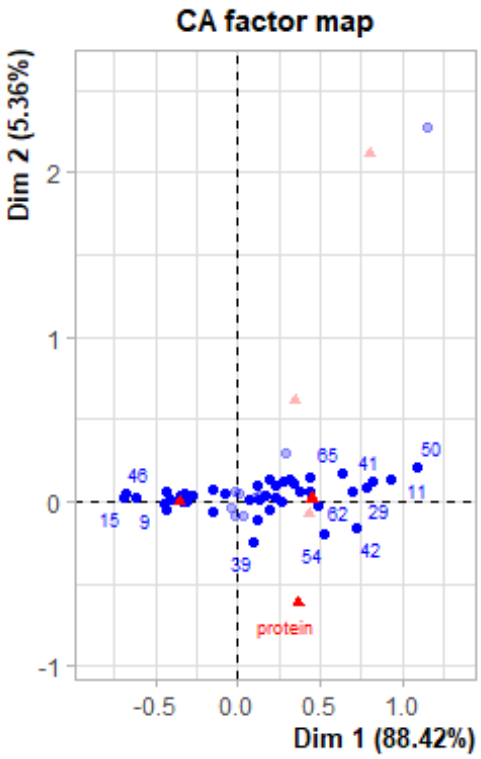



```
res$col$contrib
```

```
##                Dim 1       Dim 2        Dim 3        Dim 4       Dim 5
## rating    0.3440818 17.07757462 20.099581800 2.629837e+00 59.41444216
## calories 51.5077472  1.22137640  0.620662231 6.781431e+00  0.44820754
## protein   1.3428385 61.87579412 34.148379403 9.245111e-01  0.17818214
## fat       2.9001006  1.42414705 14.285020611 7.851057e+01  0.50785633
## sodium   43.7430601  0.01996782  0.004683564 8.233862e-04  0.02812173
## bake      0.1621718 18.38113999 30.841672391 1.115282e+01 39.42319011
```

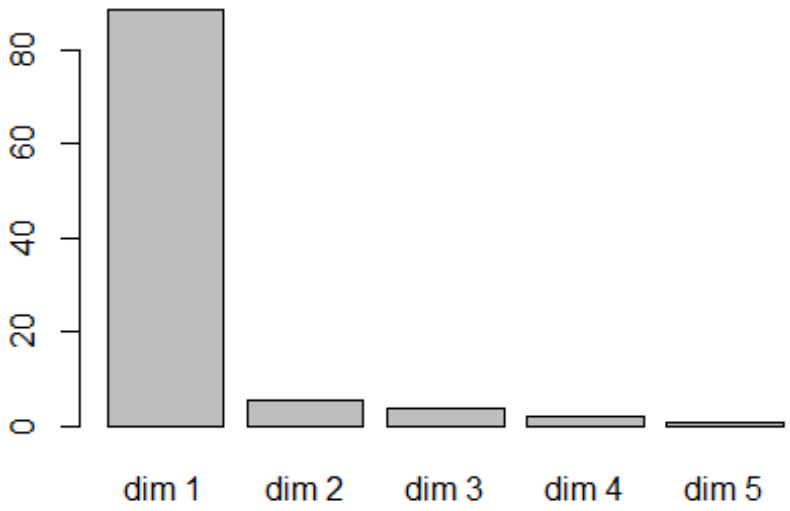
```
plot(res, selectRow="cos2 0.6",selectCol="cos2 0.6", cex=0.6)
```

```
## Warning: ggrepel: 31 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



Nous avons tracé ici les points les mieux représentés. Les points bleus (donc les plats) proche d'un triangle rouge contient beaucoup de ce composant, ainsi les points autour du triangle fat sont les plats les plus gras. De plus la table des contributions nous indique quelles variables contribuent aux axes trouvés par l'AFC.

```
barplot(res$eig[,2])
```



Nous remarquons aussi que la dimension 1 d'écrit 88% de l'inertie. Cette même dimension étant en majorité reliée aux variables calories et sodium.

Parties 2 : Méthodes de classifications

Nous allons premièrement utiliser la méthode de classification supervisée de l'analyse discriminante linéaire (LDA) pour expliquer l'appartenance d'un plat à une certaine classe de rating en fonction de ses caractéristiques.

```
library(MASS)
res = lda(rating ~., data=data)
res
```

```
## Call:
## lda(rating ~ ., data = data)
##
## Prior probabilities of groups:
##           0        1.25       1.875         2.5       3.125        3.75
## 0.081694402 0.007753404 0.005105900 0.025529501 0.073436712 0.260716087
##       4.375           5
## 0.413010590 0.132753404
##
## Group means:
##         calories   protein       fat     sodium  alcoholic       bake
## 0       362.3866  11.23071  17.57793   575.8858 0.19984568 0.09645062
## 1.25    385.9837  14.74797  19.39024   420.6504 0.08943089 0.23577236
## 1.875   369.9506  10.25926  23.58025   647.0370 0.02469136 0.30864198
```

```
## 2.5     383.4123  14.00247  20.66914   500.4222 0.03703704 0.24197531
## 3.125  6206.1528 118.53562 345.78627  6426.1914 0.01630901 0.24463519
## 3.75    509.3641  22.24154  29.06165   737.9296 0.01499033 0.25193424
## 4.375 11719.2184  99.33410 670.76862 10980.9071 0.01098901 0.27197802
## 5      6611.4155 326.56980 261.74549  7433.4482 0.07122507 0.16096866
##
## Coefficients of linear discriminants:
##                      LD1           LD2           LD3           LD4
LD5
## calories   1.112811e-06 -1.016917e-04 -0.0003419657 -5.156158e-05
2.006500e-05
## protein   -1.409915e-05  4.282215e-04  0.0006721152  2.921598e-04 -
2.458951e-04
## fat       -5.937384e-06  8.974170e-04  0.0032152692  4.725043e-04 -
1.077485e-04
## sodium    -6.921825e-07  5.128918e-05  0.0001667369  2.661289e-05 -
1.109488e-05
## alcoholic -5.048371e+00 -1.758708e+00  0.4371888704  8.791247e-01 -
9.554010e-01
## bake       7.889166e-01 -1.698382e+00  0.3030835699  8.810738e-01 -
1.148122e+00
##                      LD6
## calories  -1.448747e-06
## protein    2.127693e-04
## fat        3.052500e-04
## sodium    -1.911664e-05
## alcoholic  7.072581e-03
## bake      -7.716419e-03
##
## Proportion of trace:
##    LD1    LD2    LD3    LD4    LD5    LD6
## 0.9602 0.0309 0.0060 0.0023 0.0005 0.0000
```

La LDA a trouvé 8 classes différentes pour nos plats donc les valeurs moyennes pour les autres variables (calories, fat, …) nous ont été affiché. Ainsi un plat dans la 1ère classe, c'est à dire dans la classe des plats les moins bien notés, possède en moyenne 11 de protéine. On remarque aussi que ces recettes sont les moins caloriques et les moins grasses, ce qui est tout le contraire pour un plat de la 8ème classe.

Nous pouvons maintenant prédire avec la LDA le rating d'un plat inventé de pur pièce.

```
#prediction
newdata=
data.frame(calories=175,protein=125,fat=51,sodium=140,alcoholic=0,bake=1)

K=nlevels(data$rating)
pred.afd = predict(res, newdata) #par défaut, classe un individi selon la
règle du MAP.
pred.afd
```

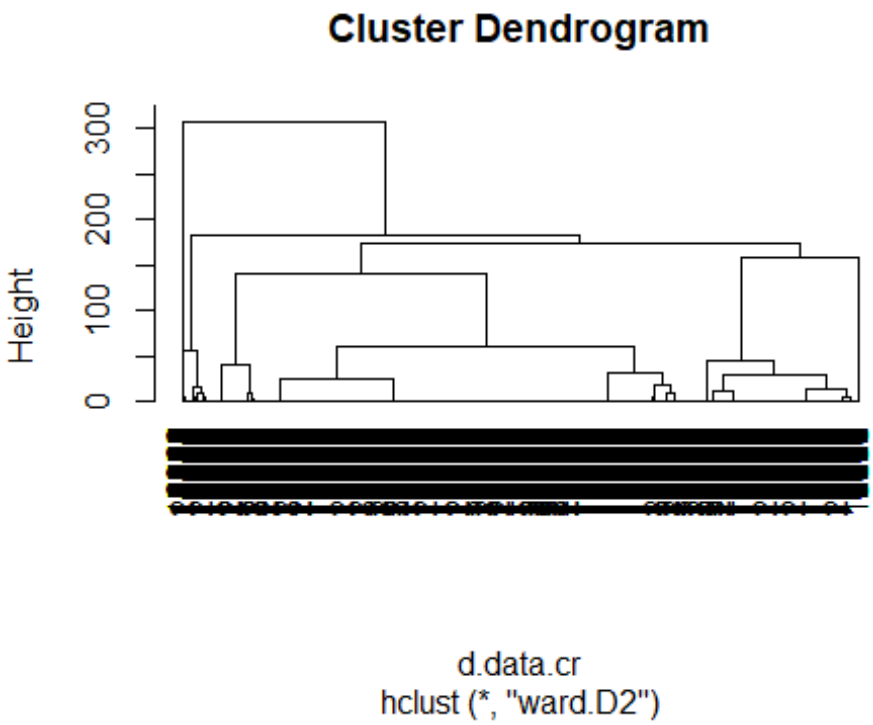
```
## $class
## [1] 4.375
## Levels: 0 1.25 1.875 2.5 3.125 3.75 4.375 5
##
## $posterior
##            0        1.25       1.875        2.5      3.125      3.75
4.375
## 1 0.02743808 0.007272354 0.006943122 0.02618978 0.07756065 0.2807491
0.4812417
##            5
## 1 0.09260517
##
## $x
##         LD1       LD2       LD3      LD4        LD5         LD6
## 1 0.7901952 -1.174739 0.3732165 0.664599 -0.8732716 0.03453469
```

On remarque ainsi que notre plat appartient à la 7ème classe avec une probabilité à postériori de 48.1% .

Nous pouvons aussi faire de la classification non supervisée avec des méthodes comme CAH et Kmeans.

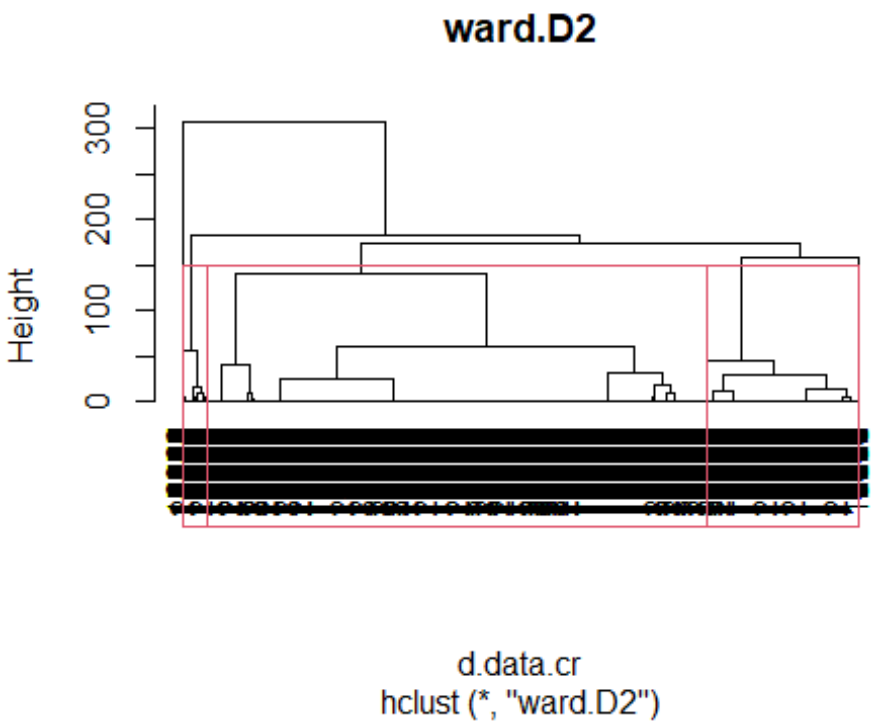
Commencons pas la classification ascendante hiérarchique (CAH).

```
data.cr <- scale(data,center=TRUE, scale=TRUE)
d.data.cr <- dist(data.cr)
#On utilise la mesure de Ward :
cah.ward <- hclust(d.data.cr, method="ward.D2")
#On affichage le dendrogramme :
plot(cah.ward, hang=-1)
```



Le dendrogramme nous permet visuellement de choisir le nombre de classes pour notre classification en élagant notre arbre. Ici nous choisissons K = 5 classes.

```
plot(cah.ward, hang =-1,main="ward.D2")
K=5
rect.hclust(cah.ward,K)
```



Nous voyons bien que un élagage en 5 classes permet d'avoir des groupes les plus homogènes et réparties possibles pour notre jeu de données.

```
groupes.cah <- cutree(cah.ward, K)
table(groupes.cah)

## groupes.cah
##     1     2     3     4     5
## 11726  3542   590     4     2
```

Cepandant il n'y a que 4 et 2 plats dans les groupes 4 et 5.

```
Means_groupes <- matrix(NA, nrow=K, ncol=dim(data)[2])
colnames(Means_groupes)=colnames(data)
rownames(Means_groupes) =1:K
for (i in 1:K) Means_groupes[i,]<- colMeans(data[groupes.cah==i,])
round(Means_groupes)

##   rating calories protein     fat   sodium alcoholic bake
## 1      4      506      25      30      751         0    0
## 2      4     2448      49     136     2606         0    1
## 3      2      225       1       2      165         1    0
## 4      5  6473970  181659  308640  6454126         0    0
## 5      4 30054568  200589 1719521 27623054         0    1
```

Cette table montre les caractéristiques moyennes des plats de chacunes des 5 classes. On remarque tout comme la classification LDA précédente, que les plats avec le rating le plus

faibles (classe 3) sont bien les plats peu gras, peu caloriques et avec peu de sodium et souvent alcoolisés. Au contraire la classe avec le rating le plus haut possède des plats gras, caloriques et salés.

Nous pouvons continuer aussi avec la méthodes des Kmeans, pour cela nous devons fixer à l'avance le nombre de classes. Pour cela nous allons nous aider de l'analyse de la CAH ou nous avions pris 5 classes, nous allons donc faire de même pour la méthode des Kmeans.

```
K=5
kmeans.result <- kmeans(data.cr,centers=K)
kmeans.result$size

## [1] 4239 1120 6313 3602  590
```

Nous remarquons que la répartition n'est pas du tout la même que celle avec la méthode CAH, celle-ci est beaucoup plus équilibré. Cependant si on relance une méthode des Kmeans.

```
kmeans.result <- kmeans(data.cr,centers=K)
kmeans.result$size

## [1]    2 1627 6316 4322 3597
```

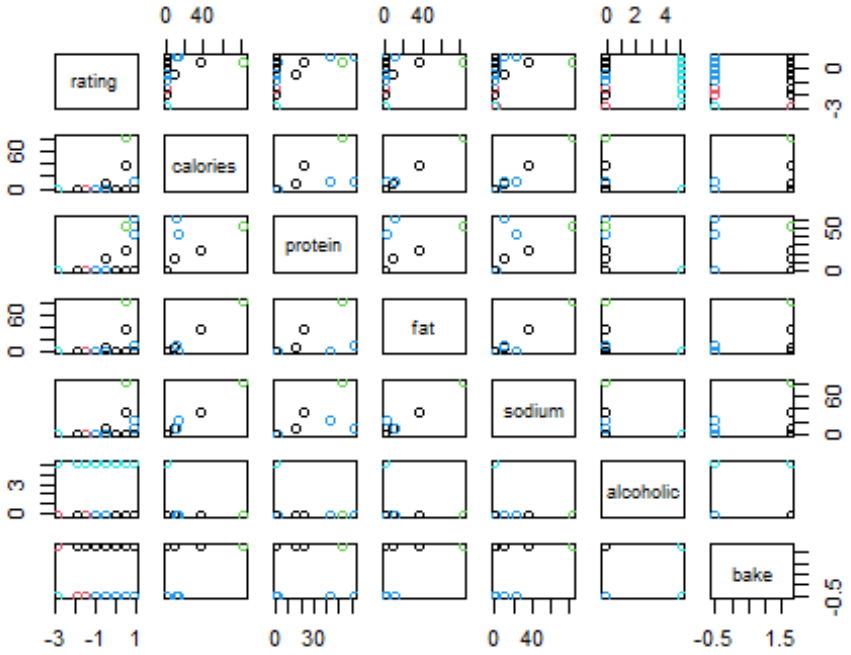
On voit que le résultat a changé comparé au précédement, car c'est dépendant de l'initialisation. Il faut donc faire une bonne initialisation avec une CAH ou une stabilisation en lançant plusieurs fois Kmeans. Ici nous allons lancé 1000 méthodes des Kmeans pour que les classes soit stabilisées.

```
kmeans.result <- kmeans(data.cr,centers=K,nstart=1000)
kmeans.result$size

## [1]  3597  1174     2 10501   590
```

Nous retrouvons environ les même effectifs que la méthode LDA.

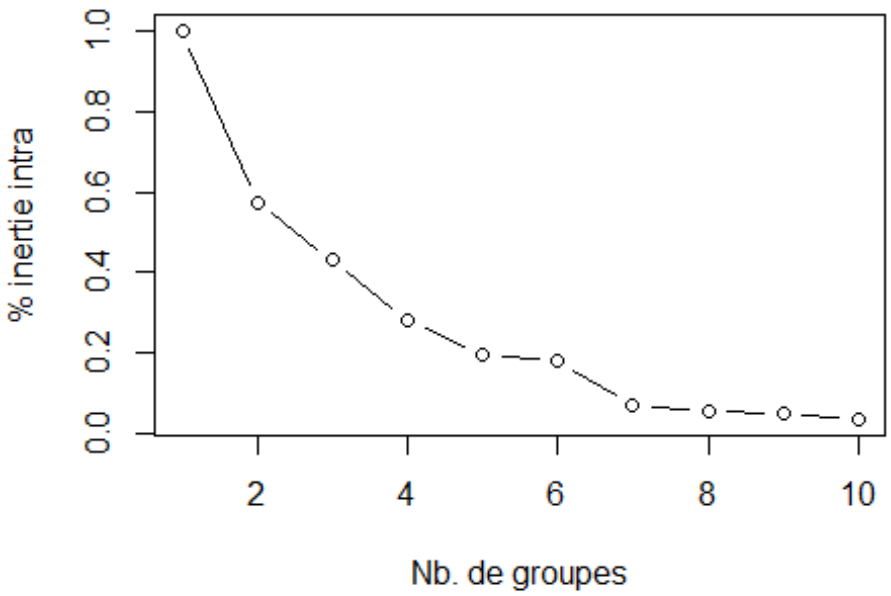
Nous pouvons maitenant visualiser les dépendances des variables en mettant en couleur les classe trouvées. Cependant avec 5 classes l'analyse devient compliquée.

```
pairs(data.cr, col=kmeans.result$cluster)
```



Nous avons également une autre façon plus graphique pour choisir le nombre de classes K.

```
inertie.intra <- rep(0,times=10)
for (k in 1:10)
{
  kmeans.result <- kmeans(data.cr,centers=k,nstart=100)
  inertie.intra[k] <- kmeans.result$tot.withinss/kmeans.result$totss
}
# graphique
plot(1:10,inertie.intra,type="b",xlab="Nb. de groupes",ylab="% inertie
intra")
```



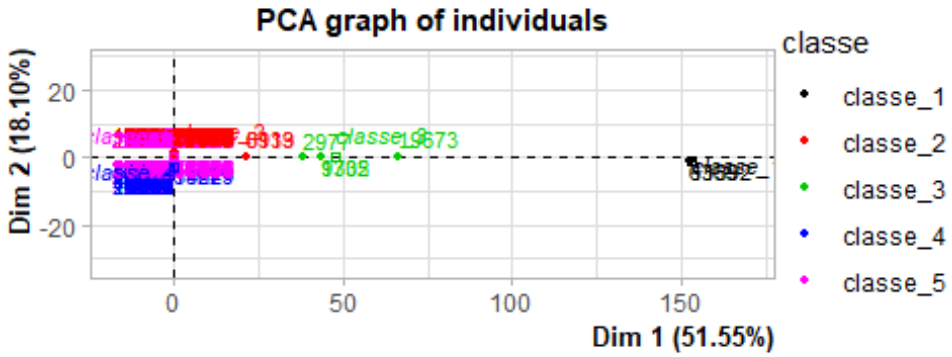
Nous choisissons de sorte que prendre un K plus grand ne diminue plus assez l'inertie. Nous pouvons voir ici que K = 5 est un bon choix car à partir de K = 5, l'ajout de classe de diminue plus l'inertie de manière significative.

Nous pouvons maintenant interprété avec l'ACP

```
K=5
kmeans.result <- kmeans(data.cr,centers=K,nstart=1000)

data.Avecclasse = cbind.data.frame(data, classe =
factor(kmeans.result$cluster))
head(data.Avecclasse)
```

```
##   rating calories protein fat sodium alcoholic bake classe
## 1  2.500      426      30   7    559         0    0      5
## 2  4.375      403      18  23   1439         0    1      2
## 3  3.750      165       6   7    165         0    0      5
## 5  3.125      547      20  32    452         0    1      2
## 6  4.375      948      19  79   1042         0    0      5
## 9  4.375      170       7  10   1272         0    0      5
```

```
res=PCA(data.Avecclasse,scale.unit=TRUE, quali.sup = 8, graph=FALSE)
plot(res, choix="ind", habillage=8, cex=0.7)
```



Nous pouvons bien observer ici les 2 classes (verte et rose) avec 4 et 2 plats qui sont bien éloignés de tout le reste de la base de données. De plus on remarque bien la disposition des 3 autres classes autour de l'origine, cela correspond bien aux observations précédentes, la classification à l'air de d'avoir bien fonctionné.

Partie 3 : Modèles de Prédiction

On réouvre toute les données pour cette partie car il y a quelques convertions à faire.

```
library(tidyr)
rm(list=ls())
data = read.csv("epi_r.csv", sep =",")
data = data[c(2,3,4,5,6,15,39)]
data = data[data$calories != "",]
data = data[data$protein != "",]
data = data[data$fat != "",]
data = data[data$sodium != "",]
data = drop_na(data)
#on convertie rating en variable qualitative à 5 modalités (entre 0 et 1).
data$rating = as.factor(((data$rating > 1) + (data$rating > 2) + (data$rating
> 3) + (data$rating > 4)) / 4)
head(data)

##   rating calories protein fat sodium alcoholic bake
## 1    0.5      426      30   7    559         0    0
## 2      1      403      18  23   1439         0    1
## 3   0.75      165       6   7    165         0    0
## 5   0.75      547      20  32    452         0    1
```

```
## 6      1      948      19  79   1042         0    0
## 9      1      170       7  10   1272         0    0
```

Le but de cette partie est de prédire la variable rating de notre jeu de données à partir des 6 autres variables. Pour cela nous allons entrainer puis tester différents modèles pour voir lequel est le plus adapté.

Pour cela commençons par couper notre jeu de données en 2, un pour l'apprentissage des modèles et un autre pour leurs évaluations.

```
seed = 1
set.seed(seed)
n = nrow(data)
p = ncol(data)-1
test.ratio = 0.2 # ratio of test/train samples
n.test = round(n*test.ratio)
tr = sample(1:n,n.test)
data.test = data[tr,]
data.train = data[-tr,]
head(data.train)
```

```
##   rating calories protein fat sodium alcoholic bake
## 1    0.5      426      30   7    559         0    0
## 2      1      403      18  23   1439         0    1
## 3   0.75      165       6   7    165         0    0
## 5   0.75      547      20  32    452         0    1
## 6      1      948      19  79   1042         0    0
## 9      1      170       7  10   1272         0    0
```

```
head(data.test)
```

```
##       rating calories protein fat sodium alcoholic bake
## 1279    0.75      636      31  41   1774         0    0
## 10100   0.75      905      37  52     69         0    0
## 5997       1      452      27  33    220         0    0
## 13062      0       67       2   3    562         0    0
## 16679   0.75      271      12  16    265         0    0
## 12253      1     1148      75  45   1779         0    0
```

Commençons par les modèles LDA et QDA.

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
res_lda = lda(rating~., data.train)
pred_lda <- predict(res_lda, data.test, type = "class")

## Table confusion et accuracy :
table(data.test$rating, pred_lda$class)

##
##           0 0.25  0.5 0.75    1
##   0      58    0    0    0  217
##   0.25    2    0    0    0   44
##   0.5     4    0    0    0   84
##   0.75   14    0    0    0 1035
##   1      45    0    1    0 1669

accuracy_lda = mean(data.test$rating == pred_lda$class)
accuracy_lda

## [1] 0.5442799

ROC_lda <- roc(data.test$rating,as.numeric(pred_lda$class))

## Warning in roc.default(data.test$rating, as.numeric(pred_lda$class)):
## 'response' has more than two levels. Consider setting 'levels' explicitly
or
## using 'multiclass.roc' instead

## Setting levels: control = 0, case = 0.25

## Setting direction: controls < cases

plot(ROC_lda, print.auc=TRUE,  print.auc.y = 0.5)
ROC_lda$auc

## Area under the curve: 0.5837

#lda version stepwise

library(klaR)
stepwise_lda= stepclass(rating~., data=data.train, method="lda",
direction="backward")

##  `stepwise classification', using 10-fold cross-validated correctness rate
of method lda'.

## 12691 observations of 6 variables in 5 classes; direction: backward

## stop criterion: improvement less than 5%.

## correctness rate: 0.54897;  starting variables (6): calories, protein,
fat, sodium, alcoholic, bake
##
##  hr.elapsed min.elapsed sec.elapsed
##        0.00        0.00        1.41
```

```
stepwise_lda
```

```
## method      : lda
## final model : rating ~ calories + protein + fat + sodium + alcoholic +
bake
## <environment: 0x000001ae0d838a98>
##
## correctness rate = 0.549
```

```
res_stepwise_lda = lda(stepwise_lda$formula, data=data.train)

pred_stepwise_lda <- predict(res_lda, data.test, type = "class")

## Table confusion et accuracy :
table(data.test$rating, pred_stepwise_lda$class)
```

```
##
##           0 0.25  0.5 0.75    1
##   0      58    0    0    0  217
##   0.25    2    0    0    0   44
##   0.5     4    0    0    0   84
##   0.75   14    0    0    0 1035
##   1      45    0    1    0 1669
```

```
accuracy_stepwise_lda = mean(data.test$rating == pred_stepwise_lda$class)
accuracy_stepwise_lda
```

```
## [1] 0.5442799
```

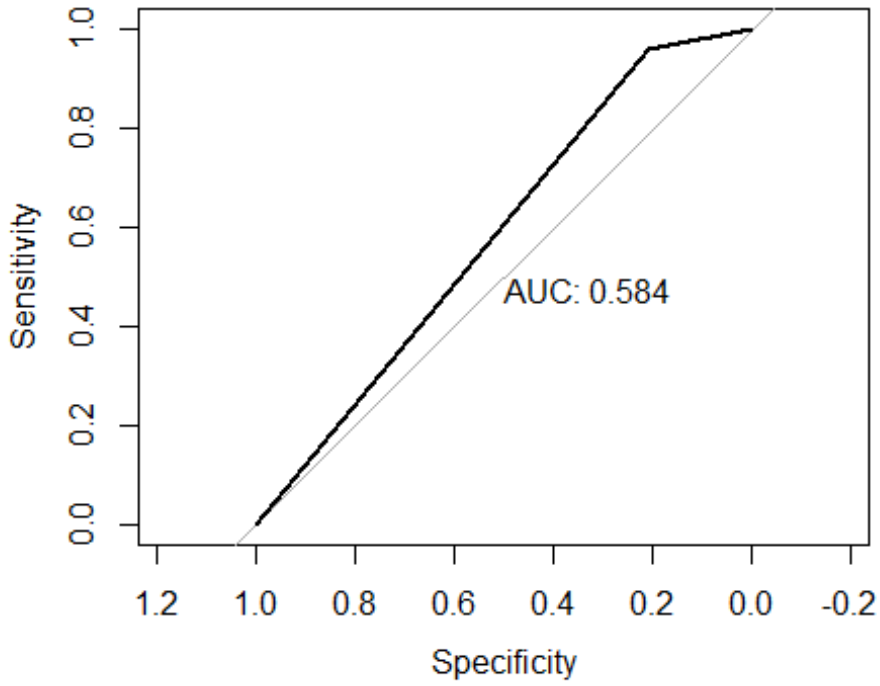
```
ROC_stepwise_lda <- roc(data.test$rating,as.numeric(pred_stepwise_lda$class))
```

```
## Warning in roc.default(data.test$rating,
as.numeric(pred_stepwise_lda$class)):
## 'response' has more than two levels. Consider setting 'levels' explicitly
or
## using 'multiclass.roc' instead
```

```
## Setting levels: control = 0, case = 0.25
```

```
## Setting direction: controls < cases
```

```
plot(ROC_stepwise_lda, print.auc=TRUE,  print.auc.y = 0.5)
```




```
ROC_stepwise_lda$auc
```

```
## Area under the curve: 0.5837
```

```
accuracy_lda = max(accuracy_lda, accuracy_stepwise_lda)

#QDA

res_qda=qda(rating~., data.train)

#prediction:
pred_qda = predict(res_qda, data.test, type = "class")

## Table confusion et accuracy :
table(data.test$rating, pred_qda$class)
```

```
##
##            0 0.25  0.5 0.75    1
##   0       70    5  194    3    3
##   0.25     4    2   39    1    0
##   0.5      7    2   78    1    0
##   0.75   106   27  883   24    9
##   1      193   44 1390   57   31
```

```
accuracy_qda = mean(data.test$rating == pred_qda$class)
accuracy_qda
```

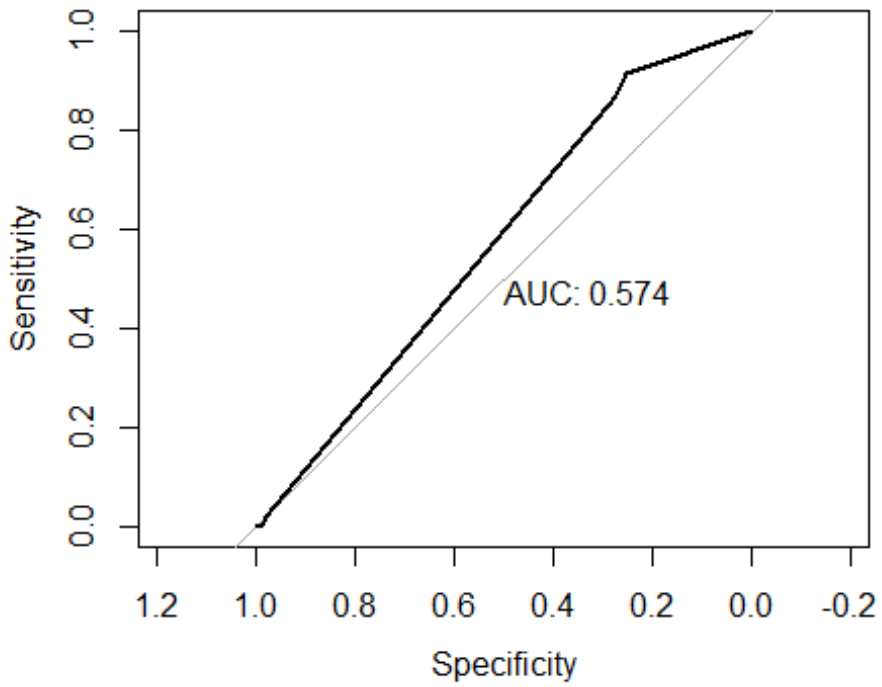
```
## [1] 0.06460763
```

```
ROC_qda <- roc(data.test$rating, as.numeric(pred_qda$class))
```

```
## Warning in roc.default(data.test$rating, as.numeric(pred_qda$class)):
## 'response' has more than two levels. Consider setting 'levels' explicitly
or
## using 'multiclass.roc' instead
```

```
## Setting levels: control = 0, case = 0.25
## Setting direction: controls < cases
```

```
plot(ROC_qda, print.auc=TRUE,  print.auc.y = 0.5)
```




```
ROC_qda$auc
```

```
## Area under the curve: 0.5742
```

Nous avons effectué la LDA de deux manières différentes mais comme le montre leurs graphiques leurs résultats sont équivalents. On cherche à avoir l'aire en dessous de la courbe ROC le plus proche de 1 possible, pour l'instant ca n'est pas trop le cas, nous allons voir si d'autres modèles font mieux.

Mise en place du modèle CART

```
library(rpart)
library(rpart.plot)
arbre = rpart(rating~.,data.train)
```

```r
cp.opt <- arbre$cptable[which.min(arbre$cptable[, "xerror"]), "CP"]
arbre.opt <- prune(arbre,cp=cp.opt)

## prédiction :
pred_arbre = predict(arbre.opt, data.test, type = "class")
## Table confusion et accuracy :
table(data.test$rating, pred_arbre)
```

```
##       pred_arbre
##           0 0.25  0.5 0.75    1
##   0       0    0    0    0  275
##   0.25    0    0    0    0   46
##   0.5     0    0    0    0   88
##   0.75    0    0    0    0 1049
##   1       0    0    0    0 1715
```

```r
accuracy_cart = mean(data.test$rating == pred_arbre)
accuracy_cart
```

```
## [1] 0.540498
```

```r
## aire sous courbe ROC
pred_cart = predict(arbre.opt, data.test, type="prob")[,2]
ROC_cart <- roc(data.test$rating, pred_cart)
```

```
## Warning in roc.default(data.test$rating, pred_cart): 'response' has more
than
## two levels. Consider setting 'levels' explicitly or using 'multiclass.roc'
## instead

## Setting levels: control = 0, case = 0.25

## Setting direction: controls < cases
```

```r
ROC_cart$auc
```

```
## Area under the curve: 0.5
```

On a une aire ROC de 0.5 se qui est très proche des résultats des méthodes LDA et QDA.

Essayons à présent le modèle random forest.

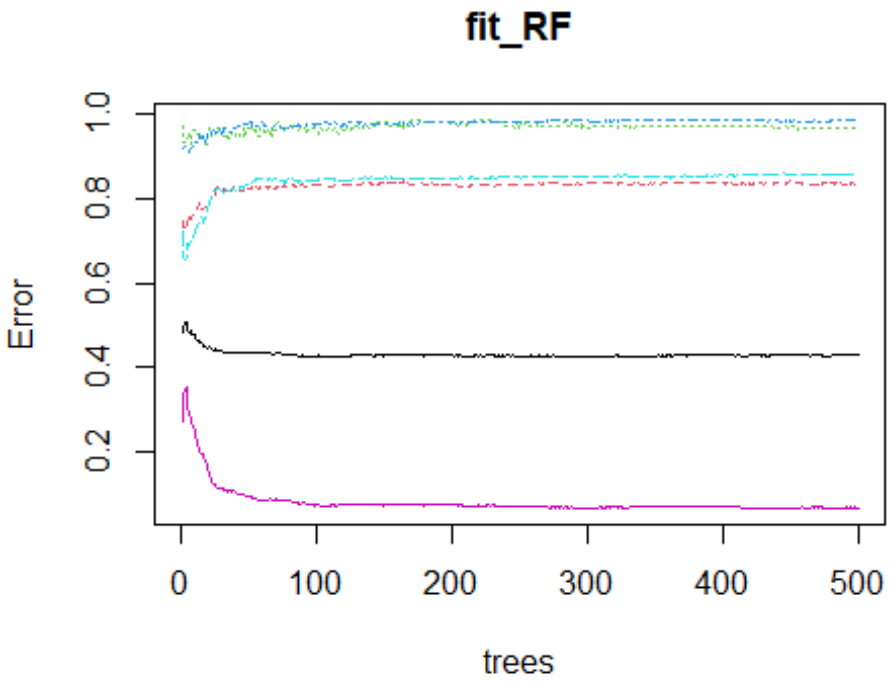
```r
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.
```

```r
fit_RF <- randomForest(rating~.,data.train)
fit_RF
```

```
##
## Call:
```

```
##  randomForest(formula = rating ~ ., data = data.train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 42.87%
## Confusion matrix:
##        0 0.25 0.5 0.75    1 class.error
## 0    165    0   0   46  810  0.83839373
## 0.25   6    5   0   19  128  0.96835443
## 0.5    9    0   4   36  268  0.98738170
## 0.75  29    0   1  600 3622  0.85888993
## 1     80    0   0  387 6476  0.06726199

plot(fit_RF)
```




```
## prédiction :
pred_rf = predict(fit_RF, data.test, type="class")

## Table confusion et accuracy :
table(data.test$rating, pred_rf)

##       pred_rf
##           0 0.25  0.5 0.75    1
##   0      57    0    0    9  209
##   0.25    1    2    0    4   39
##   0.5     1    0    3    4   80
```

```
##   0.75    8    0    1  147  893
##   1      21    0    0   63 1631
```

```
accuracy_RF = mean(data.test$rating == pred_rf)
accuracy_RF
```

```
## [1] 0.5798928
```

```
## aire sous courbe ROC
pred_RF = predict(fit_RF, data.test, type="prob")[,2]
ROC_RF <- roc(data.test$rating, pred_RF)
```

```
## Warning in roc.default(data.test$rating, pred_RF): 'response' has more
than two
## levels. Consider setting 'levels' explicitly or using 'multiclass.roc'
instead
```

```
## Setting levels: control = 0, case = 0.25
```

```
## Setting direction: controls < cases
```

```
ROC_RF$auc
```

```
## Area under the curve: 0.6358
```

Nous pouvons remarquer dans la courbe des erreurs d'apprentisage (courbe en noir) du modèle qui est bien décroissante et se stabilise vers 150 arbres. Cependant les autres courbes sont les erreurs de classification pour chaques classe, seule la classe rose a une erreur décroissante, le modèle n'a pas l'air stable. De plus on à une aire ROC de 0.58 ce qui est un peu mieux que les modèles précédents.

Passons au modèle de régression logistique

```
### Modèle
logit.train <- glm(rating ~ ., family = binomial , data=data.train)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
# régression logistique Lasso
library(glmnet)
```

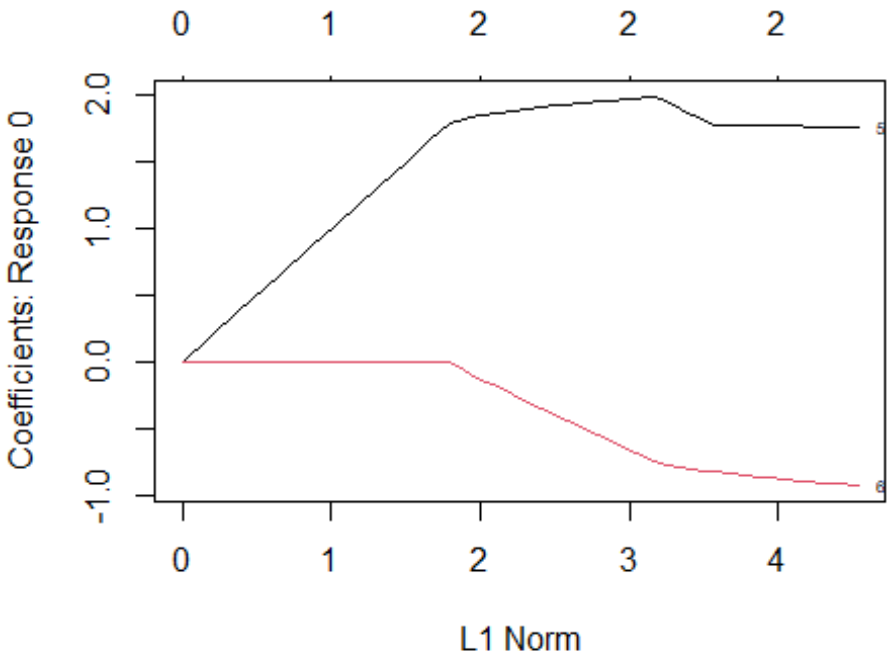
```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

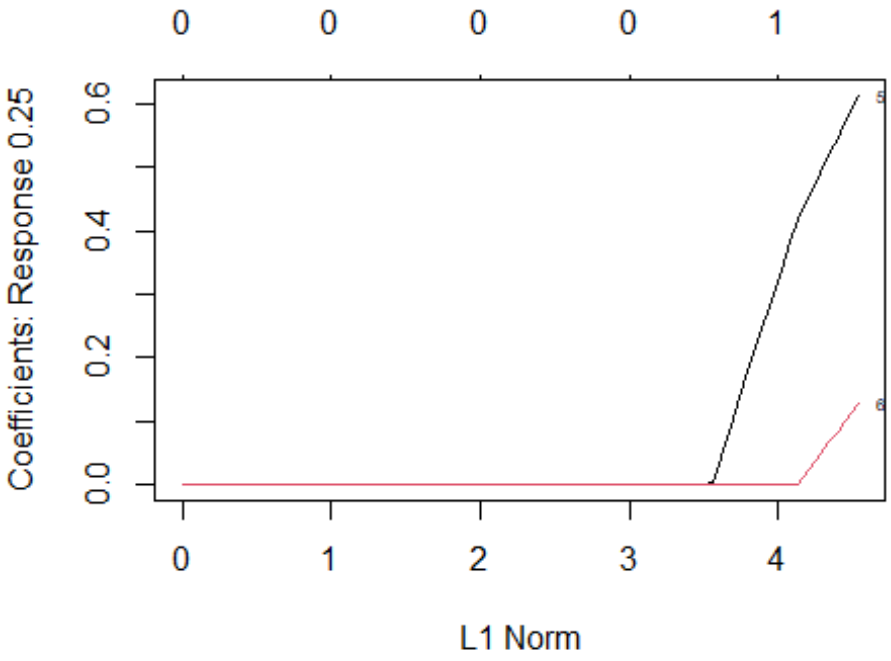
```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

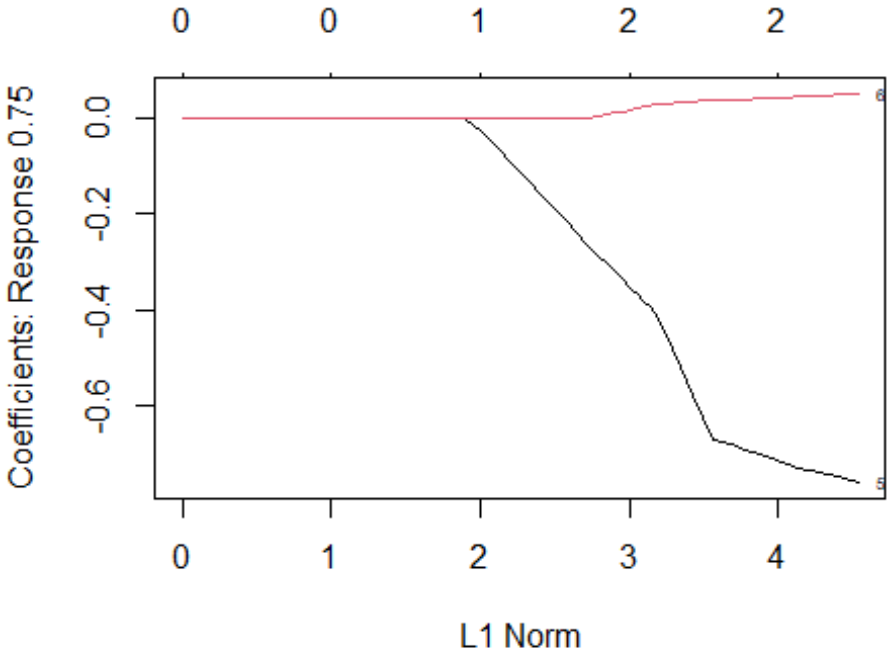
```
## Loaded glmnet 4.1-7
```

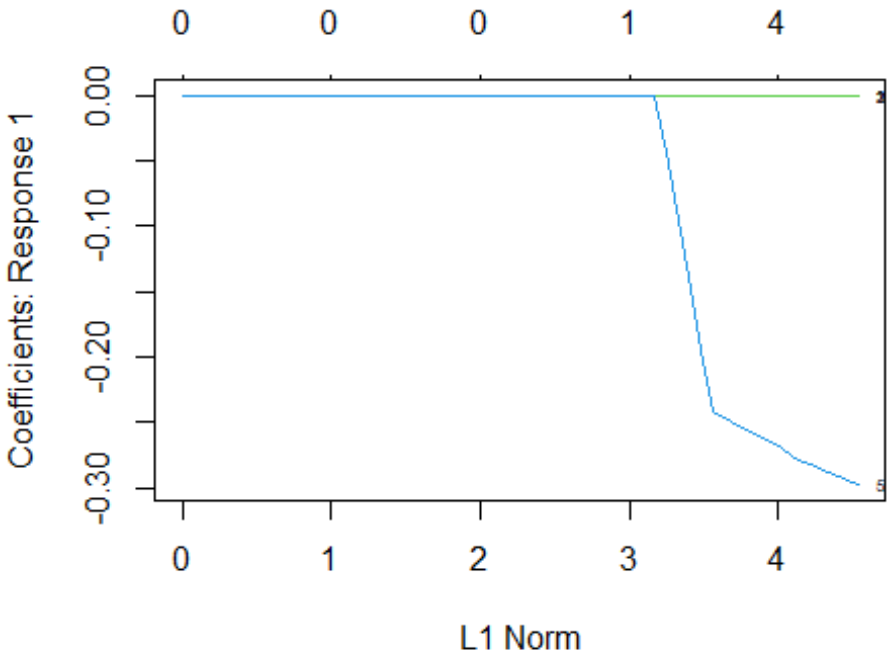
```
res_Lasso <- glmnet(as.matrix(data.train[,-1]),data.train$rating,
family='multinomial')
plot(res_Lasso, label = TRUE)  # en abscisse : norme des coefficients
```



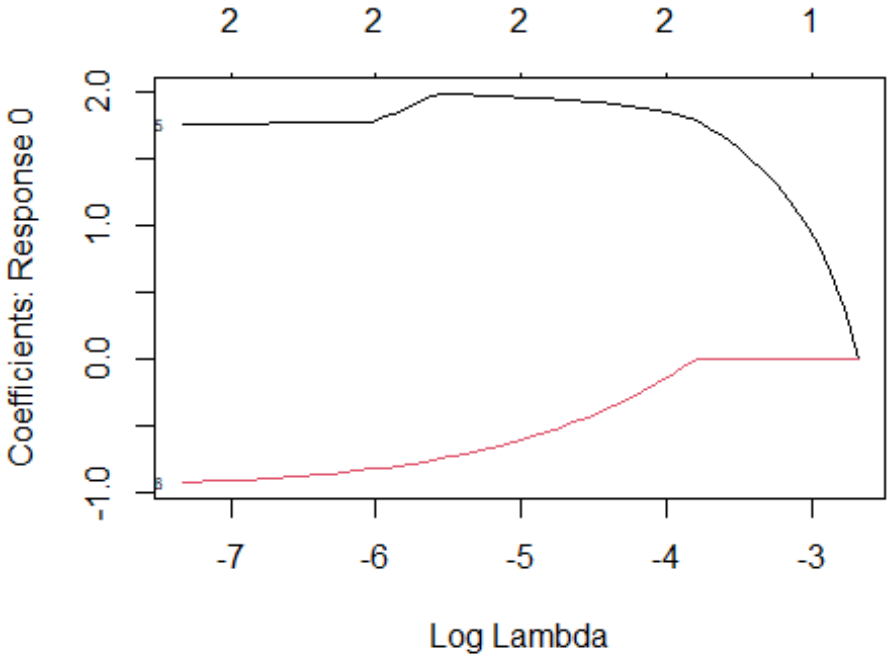

```
## Warning in plotCoef(beta[[i]], norm, x$lambda, dfmat[i, ], x$dev.ratio, :
No
## plot produced since all coefficients zero
```


0 0 0 0 1
Coefficients: Response 0.25
0.0 0.2 0.4 0.6
0 1 2 3 4
L1 Norm

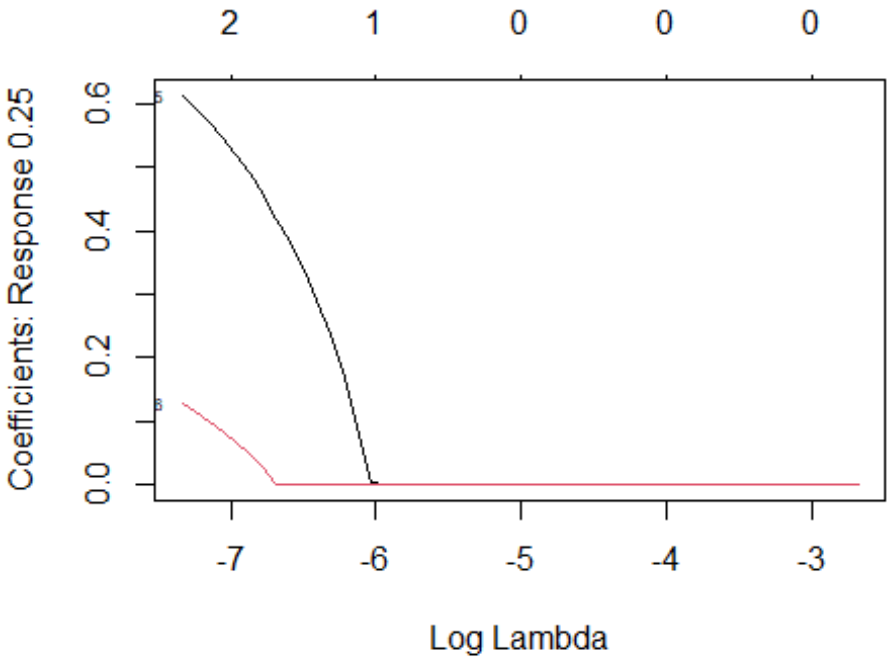

0 0 1 2 2
Coefficients: Response 0.75
-0.6 -0.4 -0.2 0.0
0 1 2 3 4
L1 Norm

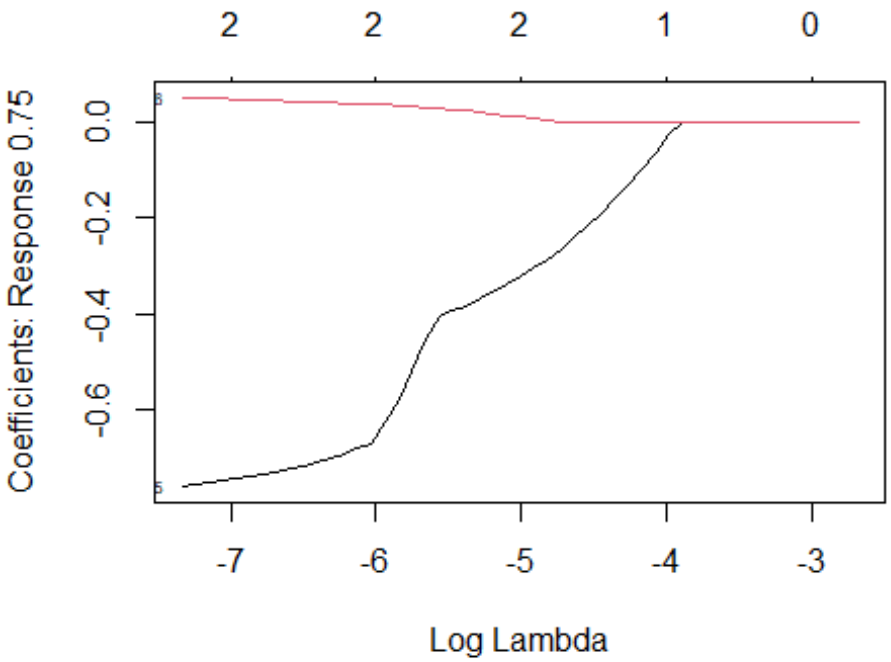


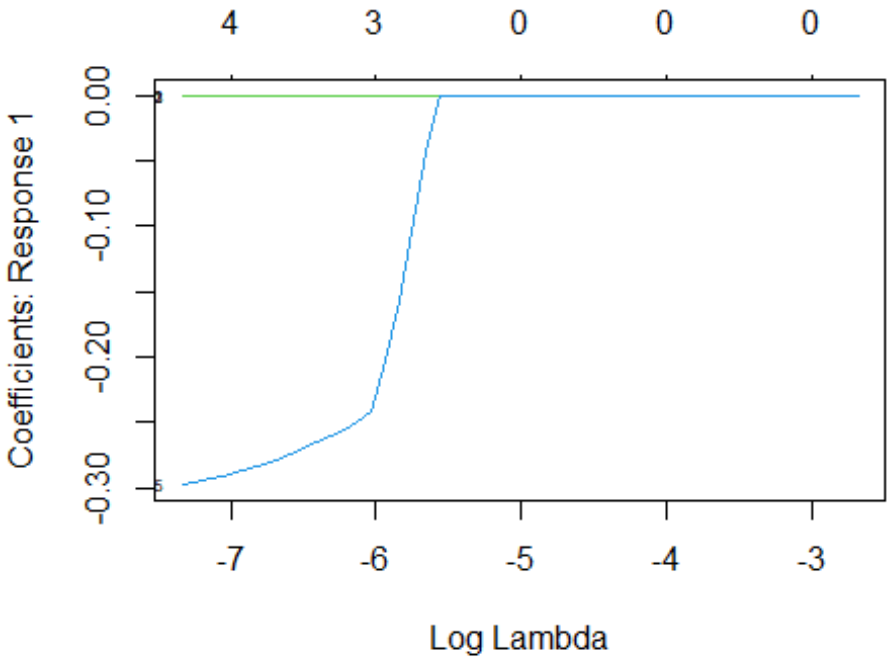

```r
plot(res_Lasso, xvar = "lambda", label = TRUE) # en abscisse : log(lambda)
```



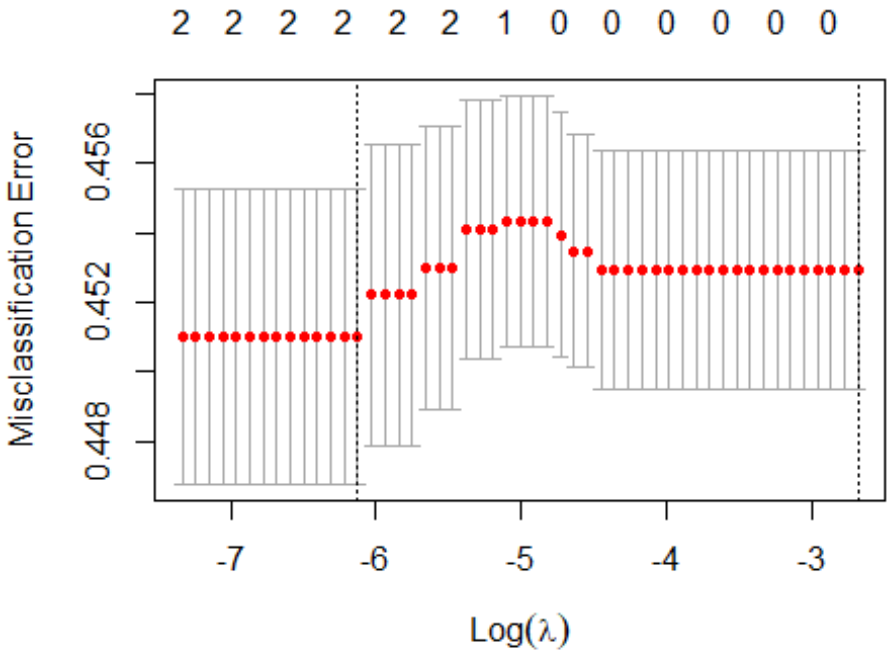
```
## Warning in plotCoef(beta[[i]], norm, x$lambda, dfmat[i, ], x$dev.ratio, :
No
## plot produced since all coefficients zero
```


2
1
0
0
0
Coefficients: Response 0.25
0.0
0.2
0.4
0.6
-7
-6
-5
-4
-3
Log Lambda


2
2
2
1
0
Coefficients: Response 0.75
-0.6
-0.4
-0.2
0.0
-7
-6
-5
-4
-3
Log Lambda




```
#sum(coef(res_Lasso, s=exp(-7))!=0)


cvLasso <- cv.glmnet(as.matrix(data.train[,-
1]),data.train$rating,family="multinomial", type.measure = "class")

## Warning: from glmnet C++ code (error code -84); Convergence for 84th
lambda
## value not reached after maxit=100000 iterations; solutions for larger
lambdas
## returned

plot(cvLasso)
```




```
cvLasso$lambda.min
```

```
## [1] 0.002188346
```

```
coef(res_Lasso, s=cvLasso$lambda.min)
```

```
## $`0`
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) -0.08559467
## calories     .
## protein      .
## fat          .
## sodium       .
## alcoholic    1.77903107
## bake        -0.83625022
##
## $`0.25`
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) -1.92745068
## calories     .
## protein      .
## fat          .
## sodium       .
## alcoholic    0.09367759
## bake         .
```

```
##
## $`0.5`
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                      1
## (Intercept) -1.227873
## calories     .
## protein      .
## fat          .
## sodium       .
## alcoholic    .
## bake         .
##
## $`0.75`
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)  1.37516365
## calories     .
## protein      .
## fat          .
## sodium       .
## alcoholic   -0.68279486
## bake         0.03994453
##
## $`1`
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  1.865754e+00
## calories     2.255873e-09
## protein      3.267468e-06
## fat          3.612021e-07
## sodium       .
## alcoholic   -2.487966e-01
## bake         .
```

```
#prédiction
class_logit_lasso=predict(cvLasso, newx = as.matrix(data.test[,-1]), s =
'lambda.min', type = "class")

#Table de confusion et accuracy
accuracy_logit_lasso = mean(data.test$rating == class_logit_lasso)
#accuracy_logit_lasso

#courbe ROC
pred_logit_lasso_tmp = predict(cvLasso, newx = as.matrix(data.test[,-1]), s =
'lambda.min', type = "response")

pred_logit_lasso = 1:n.test
for(i in 1:n.test)
{
  pred_logit_lasso[i] = pred_logit_lasso_tmp[i,2,] * 0.25 +
```

```
pred_logit_lasso_tmp[i,3,] * 0.5 + pred_logit_lasso_tmp[i,4,] * 0.75 +
pred_logit_lasso_tmp[i,5,] * 1
}

#pred_logit_lasso
ROC_logit_lasso = roc(data.test$rating, pred_logit_lasso)

## Warning in roc.default(data.test$rating, pred_logit_lasso): 'response' has
more
## than two levels. Consider setting 'levels' explicitly or using
'multiclass.roc'
## instead

## Setting levels: control = 0, case = 0.25

## Setting direction: controls < cases

ROC_logit_lasso$auc

## Area under the curve: 0.6701
```

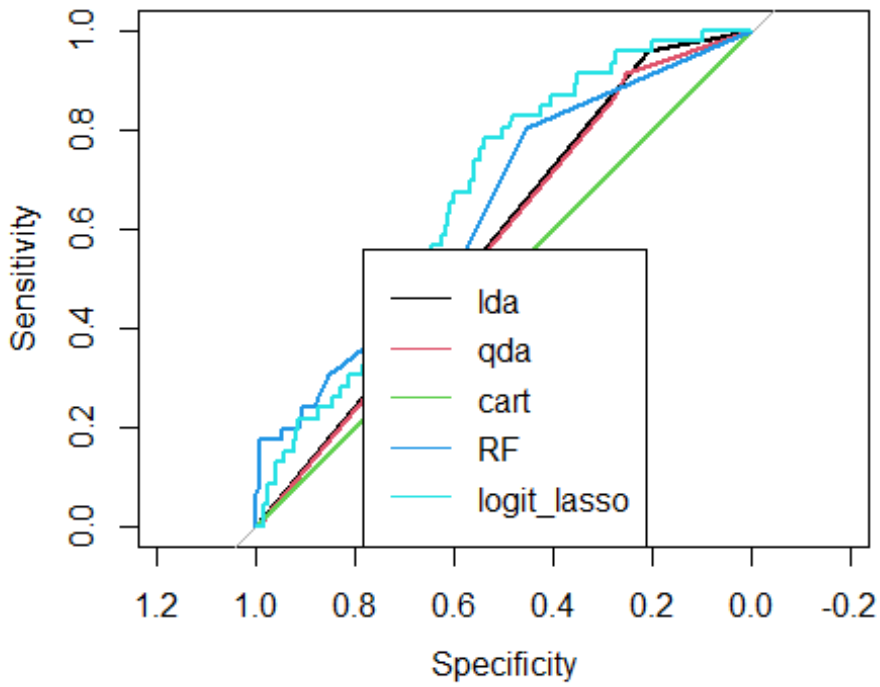
Les graphiques des coefficients en fonction de log(lambda) correspond à un chemin de régularisation du Lasso. Le chemin cherche à augmenter lamba pour réduire le nombre de coefficients non nuls. A la fin, les coefficients non nuls sont les variables significatives.

On choisit lambda par cross-validation, c'est à dire la valeur de lambda qui minimise l'erreur de classification.

Nous rassemblons tous ces graphiques pour les comparer.

```
result=matrix(NA, ncol=5, nrow=2)
rownames(result)=c('accuracy', 'AUC')
colnames(result)=c('lda', 'qda', 'cart', 'RF', 'logit_lasso')
result[1,]= c(accuracy_lda, accuracy_qda, accuracy_cart, accuracy_RF,
accuracy_logit_lasso)
result[2,]=c(ROC_lda$auc, ROC_qda$auc, ROC_cart$auc, ROC_RF$auc,
ROC_logit_lasso$auc)
result

##                lda        qda     cart        RF logit_lasso
## accuracy 0.5442799 0.06460763 0.540498 0.5798928   0.5442799
## AUC      0.5837154 0.57418972 0.500000 0.6358103   0.6701186

apply(result,1, which.max )

## accuracy      AUC
##        4        5

plot(ROC_lda, xlim=c(1,0))
plot(ROC_qda, add=TRUE, col=2)
plot(ROC_cart, add=TRUE, col=3)
plot(ROC_RF, add=TRUE, col=4)
```

```
plot(ROC_logit_lasso, add=TRUE, col=5)
legend('bottom', col=1:5, paste(c('lda', 'qda', 'cart', 'RF',
'logit_lasso')), lwd=1)
```



Nous remarquons donc que la méthode avec la meilleure précision est le modèle Random forest mais nous avons vu qu'il était instable. Au niveau de l'aire sous la courbe ROC la régression logistique semble plus performante, c'est le modèle que nous choisirons pour prédire le rating d'un plat, bien que de nombreuses améliorations semblent faisable, notamment en mettant des termes de pénalité supplémentaire sur les modèles faisant une mauvaise prédiction sur une classe avec moins d'éléments. De plus , mieux structurer le jeu de données aurait également permi d'avoir de meilleures prédictions.