Importance des variables dans les modèles prédictifs

Tutoriel Tanagra

1. Introduction

En baguenaudant sur le web, je suis tombé (ça va, ça va, je ne me suis pas fait bobo) sur l'excellent ouvrage libre, accessible gratuitement, de Christoph Molnar : "Interpretable Machine Learning" (13 fév. 2019). Ah bon ? A une époque où il est de bon ton de se shooter au millième de Log Loss en trifouillant des images de chats et de chiens dans des concours de type Kaggle, il y a des gens pour s'intéresser à l'interprétation des modèles ? C'est limite iconoclaste, voire réactionnaire, ça…

Je plaisante, je plaisante. Bien sûr que l'interprétation est importante en machine learning, ne serait-ce que parce que dans de nombreux domaines, il faut expliquer – justifier – la prise de décision induite par le modèle prédictif. Il est heureux que ce genre d'ouvrages vienne nous le rappeler. J'ai pris beaucoup de plaisir à le lire et surtout j'ai (re)découvert des approches intéressantes notamment dans le chapitre 5 intitulé "Model-Agnostic Methods". Rien à voir avec la religion, je vous rassure tout suite, il s'agit en réalité de méthodes pouvant s'appliquer à tous types de classifieurs.

Je me suis intéressé en particulier à la "permutation feature importance", une technique destinée à mesurer l'importance des variables. Son énorme avantage est qu'elle s'applique à n'importe quel type de modèle prédictif. Je suis toujours un peu effaré de voir la tête de mes étudiants quand, après avoir développé un SVM performant sur un problème quelconque en travaux dirigés, je leur demande innocemment de m'identifier les variables pertinentes. Ils restent cois souvent, tout simplement parce qu'ayant utilisé un SVM, ils ne se sont pas posé la question. La situation est identique si nous partons sur un K plus proche-voisins, ou encore un naive bayes. Dans toutes ces configurations, des techniques performantes mais qui ne donnent pas d'emblée une indication sur l'impact des variables, la question de l'identification des variables pertinentes reste centrale, ne serait-ce que pour comprendre le mécanisme d'affectation sous-jacent au modèle.

Dans ce tutoriel, nous étudierons la "permutation feature importance" en la programmant nous-même sur un jeu de données bien connu (Breast Cancer Wisconsin) pour lequel nous avons rajouté des variables générées aléatoirement pour corser l'affaire. Nous utilisons une

régression logistique parce qu'elle propose intrinsèquement un procédé permettant d'évaluer l'influence des variables. Ce sera l'occasion d'étalonner la technique agnostique et étudier sa capacité à identifier la solution adéquate. Dans un deuxième temps, nous utiliserons le package "iml" développé par l'auteur de l'ouvrage et qui fournit des outils clés en main. Nous pourrons ainsi comparer nos résultats.

2. Principe de la "permutation feature importance"

L'idée de la "permutation feature importance" consiste à opposer les performances du modèle en prédiction avec et sans la variable à évaluer. On peut se baser sur le taux d'erreur, mais tout autre indicateur peut convenir (Log Loss, etc.). Pour neutraliser la variable, les auteurs (Fisher et al., 2018) préconisent de mélanger aléatoirement les valeurs à l'intérieur du vecteur et, par conséquent, de casser le lien qu'elle peut entretenir avec la classe à prédire (et les autres variables par la même occasion).

L'idée n'est pas nouvelle. J'avais moi-même implémenté une solution similaire dans le composant MULTILAYER PERCPTRON de TANAGRA. Pour neutraliser la variable en prédiction, je remplaçais ses valeurs par sa moyenne (''Réseaux de neurones avec SIPINA, TANAGRA et WEKA'', Avril 2008 ; page 7). Mais à la différence de l'approche étudiée dans ce tutoriel, je ne disposais que d'une solution unique. Ici, en réitérant l'opération plusieurs fois, nous pouvons disposer de résultats plus stables (en prenant la médiane du ratio mesuré par exemple) avec de surcroit une indication sur la variabilité des résultats.

Je retranscris ici la description de l'algorithme repris sur le site du livre :

— début —

```
Input: Trained model f, feature matrix X, target vector y, error measure L(y,f).
  1. Estimate the original model error eorig = L(y, f(X)) (e.g. mean squared error)
  2. For each feature j = 1,.,p do:
       ○ Generate feature matrix Xperm by permuting feature j in the data X. This breaks the association between feature j and true outcome y.
       ○ Estimate error eperm = L(Y,f(Xperm)) based on the predictions of the permuted data.
```

```
        ○ Calculate permutation feature importance FIj= eperm/eorig.
          Alternatively, the difference can be used: FIj = eperm –
          eorig
  3. Sort features by descending FI.
```

— fin —

Plusieurs remarques :

- Comme indiqué plus haut, il est possible de répéter plusieurs fois l'opération et proposer par exemple un intervalle de variation des résultats (ce que fait le package ''iml'' élaboré par l'auteur).
- Cela est d'autant plus faisable que la construction du modèle n'est à réaliser qu'une seule fois, sur les données d'apprentissage. Seule la prédiction sera sollicitée dans la phase d'évaluation des variables. Multiplier les répétitions n'est pas (trop) préjudiciable en termes de temps de calcul.
- La démarche peut s'appliquer sur les données d'apprentissage qui ont servi à la construction du modèle, mais aussi sur un échantillon à part. L'auteur engage une discussion très intéressante à ce sujet, parce qu'apparemment nous n'évaluons pas exactement les mêmes propriétés des variables dans les deux cas. Nous reviendrons sur ce sujet plus loin.
- En neutralisant individuellement les variables, nous ne tenons pas compte des effets possibles d'interaction.
- Enfin, la technique est très facile à programmer. Je vais m'attacher à le montrer dans ce tutoriel.

3. Importation et pré-traitement des données

Importation

Nous utilisons la base bien connue de "Breast Cancer Wisconsin". Il s'agit d'identifier le caractère malin (malignant) ou bénin (begnin) de cellules à partir de leur caractéristiques, 9 variables explicatives (CLUMP, …, MITOSES), prenant toutes des valeurs entières comprises entre 1 et 10. Pour rajouter de la difficulté à la tâche, nous avons rajouté 9 autres variables (RND1, …, RND9) générées aléatoirement mais définies dans le même domaine de valeurs.

Nous chargeons les données et affichons les caractéristiques de la base.

```
#changement de répertoire
setwd("… votre dossier de travail …")

#charger les données et afficher les caractéristiques
library(xlsx)
D <- read.xlsx("breast_rnd.xlsx",header = T,sheetIndex = 1)
print(str(D))

## 'data.frame':    699 obs. of  19 variables:
##  $ classe    : Factor w/ 2 levels "begnin","malignant": 1 1 1 2 1 1 1 1 1 1 ...
##  $ clump     : num  4 1 2 10 4 1 1 5 3 1 ...
##  $ ucellsize : num  2 1 1 6 1 1 1 1 1 1 ...
##  $ ucellshape: num  2 1 1 6 1 1 1 1 1 1 ...
##  $ mgadhesion: num  1 1 1 2 1 1 1 1 1 1 ...
##  $ sepics    : num  2 2 2 4 2 2 2 2 2 2 ...
##  $ bnuclei   : num  1 1 1 10 1 1 1 1 1 4 ...
##  $ bchromatin: num  2 2 2 9 2 1 2 2 2 2 ...
##  $ normnucl  : num  1 1 1 7 1 1 1 1 1 1 ...
##  $ mitoses   : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ rnd1      : num  10 1 5 1 4 9 2 1 1 3 ...
##  $ rnd2      : num  2 7 3 2 10 1 3 1 3 4 ...
##  $ rnd3      : num  8 1 2 7 10 5 3 2 6 8 ...
##  $ rnd4      : num  4 1 4 4 1 4 10 1 10 5 ...
##  $ rnd5      : num  1 3 4 10 1 4 10 7 5 3 ...
##  $ rnd6      : num  6 7 6 10 1 7 6 2 3 3 ...
##  $ rnd7      : num  8 3 2 5 7 10 6 7 10 3 ...
##  $ rnd8      : num  4 6 9 4 9 10 3 1 1 2 ...
##  $ rnd9      : num  4 1 2 2 3 9 10 9 10 2 ...
## NULL
```

Nous disposons de 699 observations. "classe" est la variable cible. Les variables RND1 à RND9 ne devraient jouer aucun rôle dans la modélisation.

Partition en échantillons d'apprentissage et de test

Nous subdivisions les données en échantillons d'apprentissage (499 obs.) et de test (200). Nous isolons les variables explicatives dans des structures spécifiques que nous utiliserons par la suite.

```
#partition des données en TRAIN et TEST
set.seed(100)
id <- sample(1:nrow(D),499,replace = FALSE)
DTrain <- D[id,]
DTest <- D[-id,]

#descripteurs X : TRAIN et TEST
XTrain <- DTrain[2:ncol(DTrain)]
XTest <- DTest[2:ncol(DTest)]
```

Nous sommes maintenant parés pour réaliser notre étude.

4. Importance des variables dans la régression logistique

Régression logistique

Nous réalisons la régression avec la fonction `glm()` du package "stats". Nous disposons des coefficients du modèle, la z-value (rapport entre le coefficient et son écart-type) permet de statuer sur la significativité des coefficients, en d'autres termes de savoir si la variable joue un rôle effectif dans la régression.

```
#régression logistique
modele <- glm(classe ~ ., data = DTrain, family = binomial)
print(summary(modele))

##
## Call:
## glm(formula = classe ~ ., family = binomial, data = DTrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1671  -0.1050  -0.0419   0.0102   2.3135
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -14.130185   2.653084  -5.326 1.00e-07 ***
## clump         0.697817   0.187991   3.712 0.000206 ***
## ucellsize    -0.020904   0.299900  -0.070 0.944431
## ucellshape    0.258290   0.350653   0.737 0.461367
## mgadhesion    0.255348   0.154761   1.650 0.098953 .
## sepics        0.087488   0.215135   0.407 0.684255
## bnuclei       0.475709   0.115575   4.116 3.85e-05 ***
## bchromatin    0.533563   0.209891   2.542 0.011019 *
## normnucl      0.090654   0.158067   0.574 0.566295
## mitoses       0.723860   0.438907   1.649 0.099100 .
## rnd1          0.067459   0.117773   0.573 0.566785
## rnd2          0.079342   0.120610   0.658 0.510643
## rnd3         -0.146935   0.130803  -1.123 0.261298
## rnd4          0.084521   0.127514   0.663 0.507434
## rnd5          0.269660   0.145387   1.855 0.063629 .
## rnd6          0.038936   0.119606   0.326 0.744779
## rnd7          0.138853   0.128502   1.081 0.279897
## rnd8          0.002517   0.121791   0.021 0.983511
## rnd9         -0.020374   0.115469  -0.176 0.859942
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 630.52  on 498  degrees of freedom
## Residual deviance:  73.80  on 480  degrees of freedom
## AIC: 111.8
##
## Number of Fisher Scoring iterations: 8
```

Au risque alpha = 5%, les variables pertinentes sont CLUMP, BNUCLEI et BCHROMATIN. On remarque que RND5 flirte avec le seuil (p-value = 6.3629%). Nous savons que c'est un pur artefact statistique. Cette variable a été générée complètement aléatoirement et n'a aucune relation avec le problème traité.

Coefficients standardisés

On pourrait se baser sur la valeur absolue de la z-value pour mesurer la contribution des variables dans le modèle. Je préfère pour ma part passer par les coefficients standardisés parce que les résultats sont interprétables ("Pratique de la régression logistique", mai 2017 ; section 5.3.2).

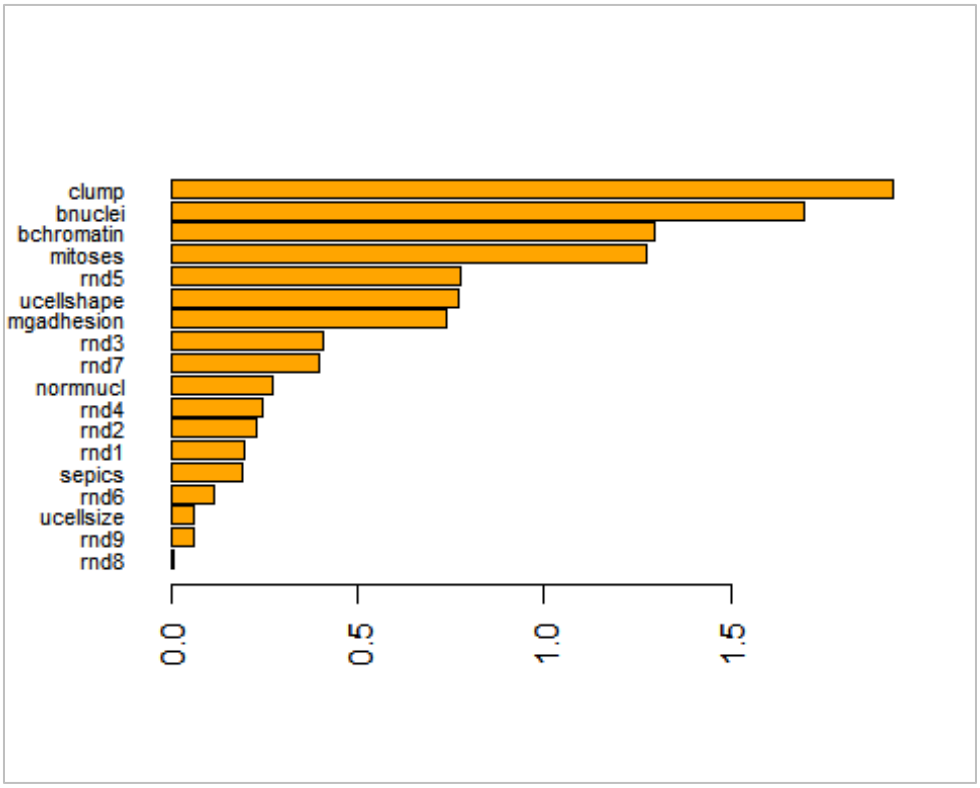
Une solution simple consiste à multiplier les coefficients estimés par les écarts-type des variables. Les coefficients standardisés correspondent aux paramètres estimés si nous avions lancé la régression sur les données centrées et réduites.

```
#coefficients standardisés
cstd <- modele$coefficients[2:ncol(DTrain)]*sapply(XTrain,sd)
print(cstd)

##        clump    ucellsize   ucellshape   mgadhesion       sepics
##  1.936983831 -0.062520894  0.772127481  0.735858225  0.190389842
##      bnuclei   bchromatin     normnucl      mitoses         rnd1
##  1.698063309  1.294888069  0.269940564  1.272851280  0.196563727
##         rnd2         rnd3         rnd4         rnd5         rnd6
##  0.228214197 -0.409797224  0.244113625  0.775483119  0.112677282
##         rnd7         rnd8         rnd9
##  0.396985631  0.007348765 -0.060140314
```

Une augmentation d'un écart-type de CLUMP entraîne un accroissement de 1.93 du LOGIT. Il est dès lors possible de comparer le poids relatif des variables dans la régression.

Résumons la situation des variables dans un graphique autrement plus sympathique à lire, surtout lorsque nous trions les variables selon la valeur absolue des coefficients standardisés.

```
#pour inscrire le nom des variables horizontalement le long de l'axe des ordonnées
par(las=2)

#représentation graphique
barplot(sort(abs(cstd)),horiz=TRUE,cex.names=0.7,col="orange")
```



CLUMP et BNUCLEI sont les variables qui contribuent le plus fortement, puis viennent BCHROMATIN et MITOSES. RND5 vient s'immiscer parmi les variables originelles de la base. UCELLSIZE est absolument loin du compte visiblement.

Nous disposons d'une référence maintenant pour juger du bien-fondé des solutions mises en avant par la mesure "permutation importance" que nous étudions à partir de la section suivante. Les résultats ne seront pas forcément identiques, mais il y aura problème si nous n'identifions pas au moins le groupe des 4 premières variables les plus influentes dans la régression.

5. Programmation de la "permutation importance"

L'importance des variables peut être estimée en modélisation (sur l'échantillon d'apprentissage) ou en prédiction (sur l'échantillon test). Dans les deux cas, les principales étapes sont les mêmes : calculer le taux d'erreur de référence, calculer ensuite le même indicateur en neutralisant tour à tour chaque variable prédictive, former le ratio entre les deux valeurs.

Travail sur l'échantillon d'apprentissage

La première étape consiste à élaborer une fonction permettant de mesure l'erreur sur un ensemble de données. La fonction `err_pred()` prend en entrée : la matrice des explicatives X, le vecteur représentant la variable cible y, le modèle (déjà entraîné) à utiliser.

```
#fonction pour le calcul de l'erreur en prédiction
err_pred <- function(X,y,modele){
  p <- factor(ifelse(predict(modele,newdata=X,type="response") > 0.5, "malignant", "begnin"))
  err <- mean(as.numeric(y != p))
  return(err)
}
```

Nous pouvons maintenant calculer l'erreur en resubstitution.

```
#erreur de référence en train
err_ref_train <- err_pred(XTrain,DTrain$classe,modele)
print(err_ref_train)

## [1] 0.02204409
```

Sans altération des variables, le taux d'erreur du modèle sur les données d'apprentissage est de 2.20%. Ce sera notre valeur de référence. Effectuer la prédiction sur les mêmes données en corrompant les variables ne peut que détériorer (l'augmenter) le taux d'erreur, faiblement si la variable joue un rôle négligeable, fortement si elle est importante dans le processus d'affectation.

Nous écrivons une fonction pour mesurer le ratio entre l'erreur de référence (err_orig) et l'erreur après neutralisation de chaque variable.

```
#calculer le ratio pour un ensemble de variables
#permutation feature importance
calc_ratio <- function(X,y,err_orig,modele){
  #traiter une colonne j
  calc_for_j <- function(j){
    Z <- X
    Z[,j] <- sample(X[,j])
    err <- err_pred(Z,y,modele)
    #renvoyer le ratio err mesurée / erreur de référence
    return(err/err_orig)
  }
  #pour l'ensemble des colonnes
  ratios <- sapply(1:ncol(XTest),calc_for_j)
  return(ratios)
}
```

Nous effectuons un essai pour situer le comportement de la fonction.

```
#un essai sur le train set
set.seed(1) #pour que le résultat soit reproductible
cr <- calc_ratio(XTrain,DTrain$classe,err_ref_train,modele)
names(cr) <- colnames(XTrain)
print(cr)

##      clump  ucellsize ucellshape mgadhesion     sepics    bnuclei
##  3.1818182  1.0000000  1.3636364  1.1818182  0.9090909  3.0000000
## bchromatin   normnucl    mitoses       rnd1       rnd2       rnd3
##  2.1818182  1.0000000  2.0000000  1.0909091  1.0909091  1.0000000
##       rnd4       rnd5       rnd6       rnd7       rnd8       rnd9
##  1.0000000  1.1818182  1.0000000  1.4545455  1.0000000  1.0000000
```

C'est le résultat d'une seule expérimentation, les résultats sont forcément très instables. Nous notons néanmoins que les variables CLUMP, BNUCLEI, BCHROMATIN et MITOSES se distinguent.

Pour CLUMP par exemple, le taux d'erreur en prédiction sur l'échantillon d'apprentissage serait 3.18 fois (3.1818182 x 0.02204409 = 0.07014) plus élevé si nous la désactivons (en mélangeant aléatoirement ses valeurs) dans la prédiction. La dégradation est forte, CLUMP joue donc un rôle important dans le processus prédictif.

Nous écrivons maintenant une fonction qui permet de réitérer l'opération "n_rep" fois.

```
#fonction pour repliquer n_rep fois l'expérimentation
feature_importance <- function(modele, X, y, err_orig, n_rep){
  #répeter les calculs n_rep fois
  few_ratios <- replicate(n_rep,calc_ratio(X,y,err_orig,modele))
  #nommer les lignes et les colonnes de la matrice
  rownames(few_ratios) <- colnames(X)
  colnames(few_ratios) <- 1:n_rep
  #
  return(few_ratios)
}
```

Il ne reste plus qu'à appeler la fonction, nous demandons 200 réplications. Comme l'apprentissage du modèle a déjà été effectué, seule la prédiction est sollicitée ici, le temps de calcul reste raisonnable.

```
#répliquer 200 fois
set.seed(1)
results_train <- feature_importance(modele,XTrain,DTrain$classe, err_ref_train,n_rep=200)
```

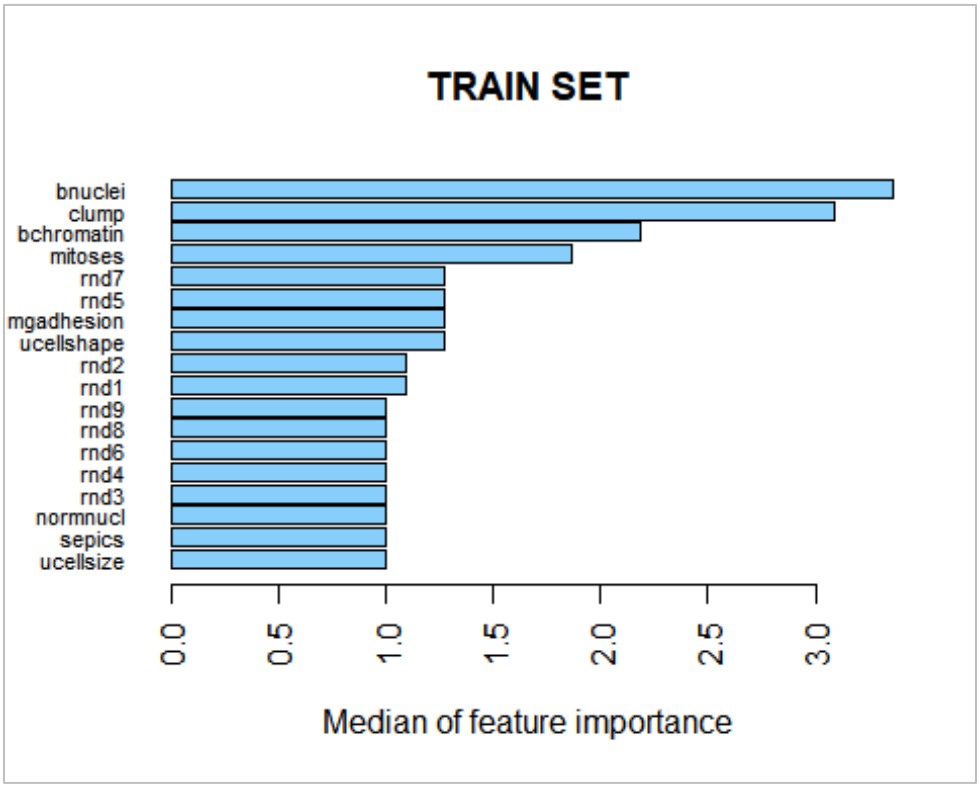
La structure produite est une matrice : en ligne, les variables ; en colonne, les répétitions. Nous affichons la médiane des ratios pour chaque variable.

```
#médianes
med_train <- apply(results_train,1,median)
print(med_train)
```

```
##      clump  ucellsize ucellshape mgadhesion     sepics    bnuclei
##   3.090909   1.000000   1.272727   1.272727   1.000000   3.363636
## bchromatin   normnucl    mitoses       rnd1       rnd2       rnd3
##   2.181818   1.000000   1.863636   1.090909   1.090909   1.000000
##       rnd4       rnd5       rnd6       rnd7       rnd8       rnd9
##   1.000000   1.272727   1.000000   1.272727   1.000000   1.000000
```

De nouveau, une représentation graphique est plus sympathique.

```
#pour inscrire le nom des variables horizontalement le long de l'axe des ordonnées
par(las=2)

#représentation graphique
barplot(sort(med_train),horiz=TRUE,cex.names=0.7,col="lightskyblue",main="TRAIN SET",xlab="Median of feature importance")
```



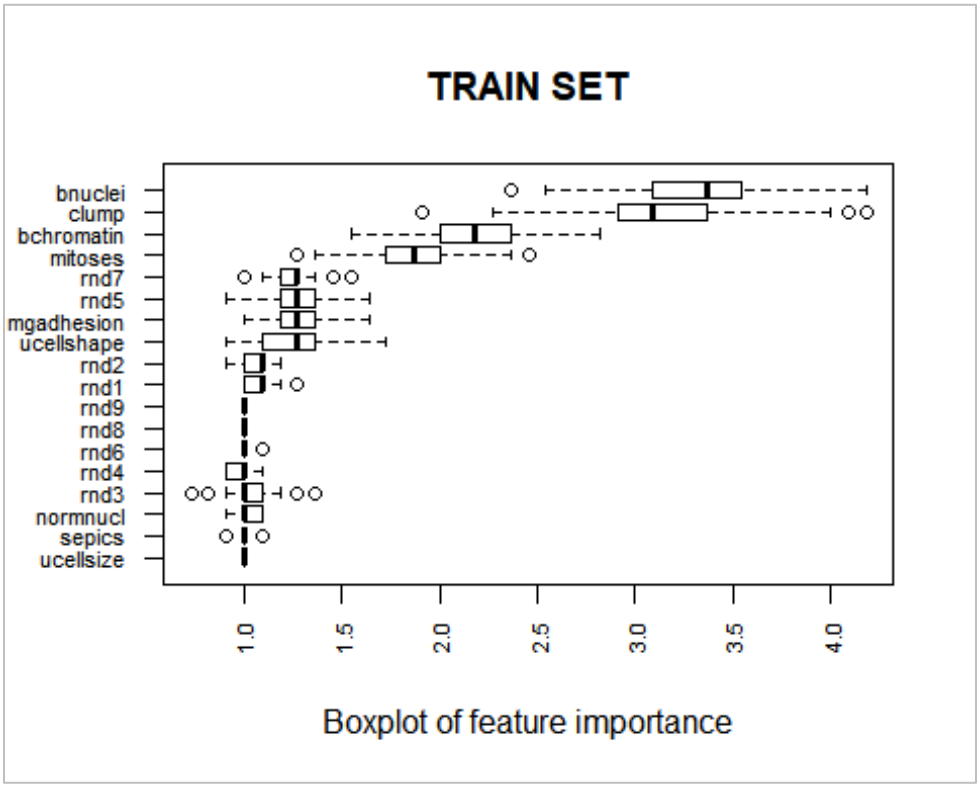
Nous avons bien le binôme (BNUCLEI, CLUMP) mis en avant par les coefficients standardisés (mais pas dans le même ordre), suivi de (BCHROMATIN, MITOSES). On peut considérer que les variables suivantes jouent un rôle anecdotique.

Les résultats sont assez conformes de ce que l'on pouvait attendre. Il était très important ici de travailler sur une base connue dont on connaît depuis longtemps les contours. Le fait d'utiliser la régression logistique qui propose une procédure intrinsèque d'évaluation des contributions des variables permettait également d'étalonner l'approche agnostique qui, à

aucun moment, n'exploite les spécificités de la méthode. Nous aurions pu utiliser un SVM ou K-plus proches voisins, rien n'est à modifier dans la procédure.

Puisque nous disposons de plusieurs valeurs pour chaque variable, afficher les résultats sous forme de *boxplot* permet de se rendre compte de leur variabilité.

```
#pour inscrire le nom des variables horizontalement le long de l'axe des ordonnées
par(las=2)

#si on passe sur des box plot
index <- order(med_train) #trier les variables selon la médiane des ratios
boxplot(t(results_train[index,]),horizontal=TRUE,pars=list(cex.axis=0.7),main="TRAIN SET",xlab="Boxplot of feature importance")
```



On se rend compte que la distribution pour RND7 est très dissymétrique.

Enfin, avec les 1er et 3ème quartiles, nous avons une sorte d'intervalle de confiance à 50% des ratios.

Travail sur l'échantillon test

Nous pouvons réaliser la même démarche sur l'échantillon test. Les fonctions ayant déjà été écrites, il ne reste plus qu'à les appeler.

Le taux d'erreur de référence est de 3%.

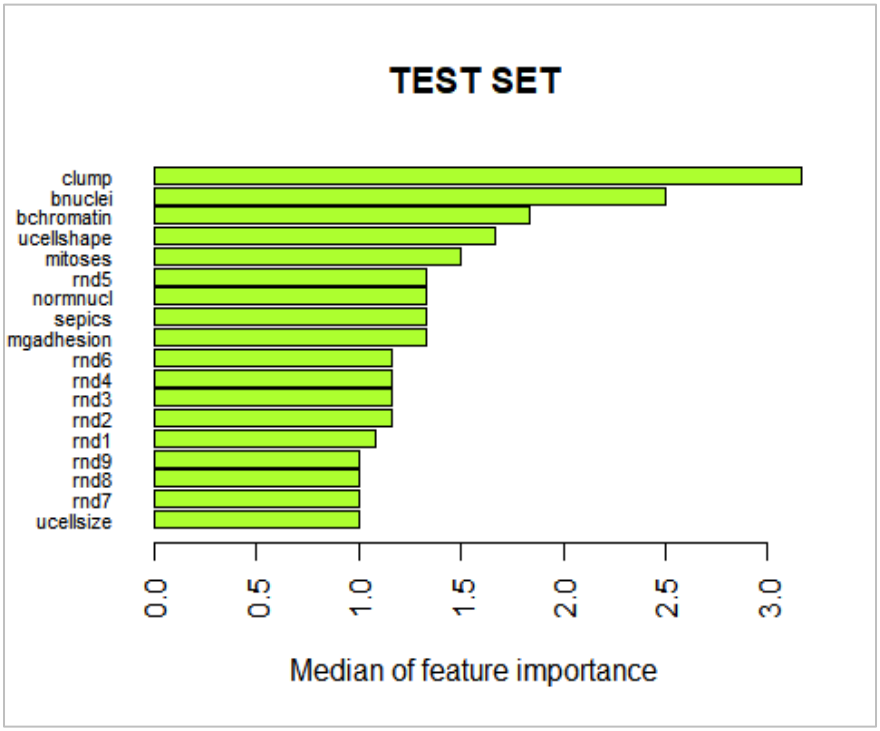
```
#calcul de l'erreur en test
err_ref_test <- err_pred(XTest,DTest$classe,modele)
print(err_ref_test)

## [1] 0.03
```

Nous calculons les ratios en répétant 200 fois l'opération.

```
#utilisation de l'échantillon test
set.seed(1)
results_test <- feature_importance(modele,XTest,DTest$classe, err_ref_test,n_rep=200)
```

Les résultats mettent de nouveau en avant les variables CLUMP, BNUCLEI et BCHROMATIN.

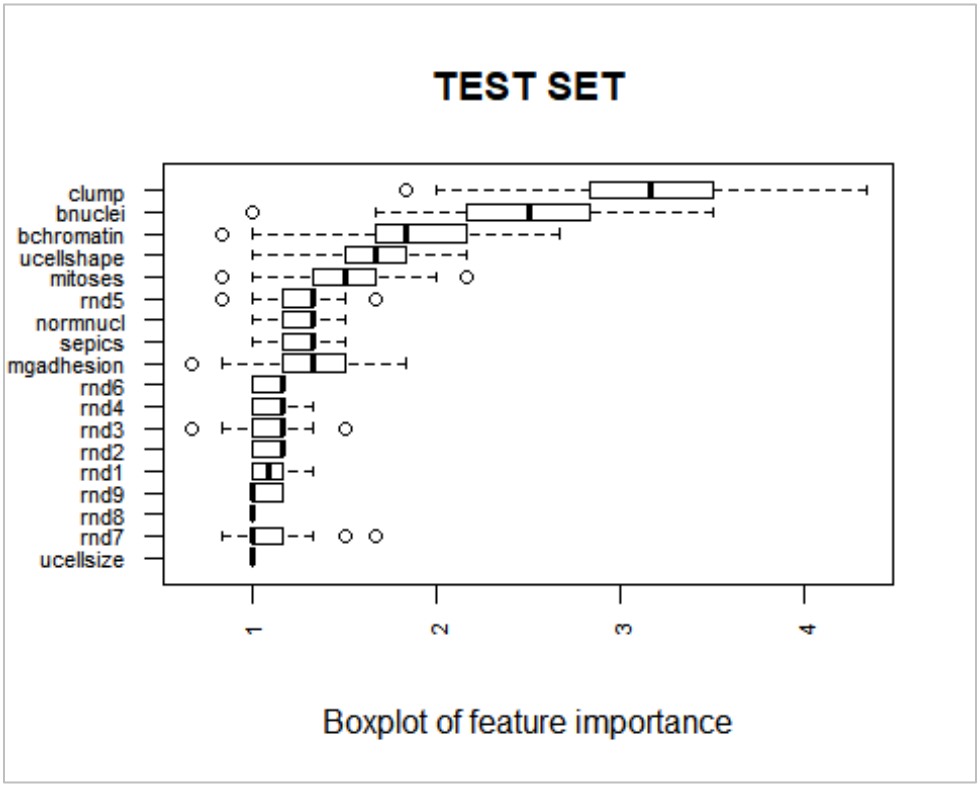
```
#médianes
med_test <- apply(results_test,1,median)

#pour inscrire le nom des variables horizontalement le long de l'axe des ordonnées
par(las=2)

#représentation graphique
barplot(sort(med_test),horiz=TRUE,cex.names=0.7,col="greenyellow",main="TEST SET",xlab="Median of feature importance")
```



Surprise ici, UCELLSHAPE s'interpose devant MITOSES. Une nouvelle facette de l'influence des variables est mise en avant en utilisant l'échantillon test.

Nous disposons d'une vue plus détaillée toujours en passant par les *boxplot*.

```
#pour inscrire le nom des variables horizontalement le long de l'axe des ordonnées
par(las=2)

#si on passe sur des box plot
index <- order(med_test)
boxplot(t(results_test[index,]),horizontal=TRUE,pars=list(cex.axis=0.7),main="TEST SET",xlab="Boxplot of feature importance")
```



Echantillon d'apprentissage ou de test ?

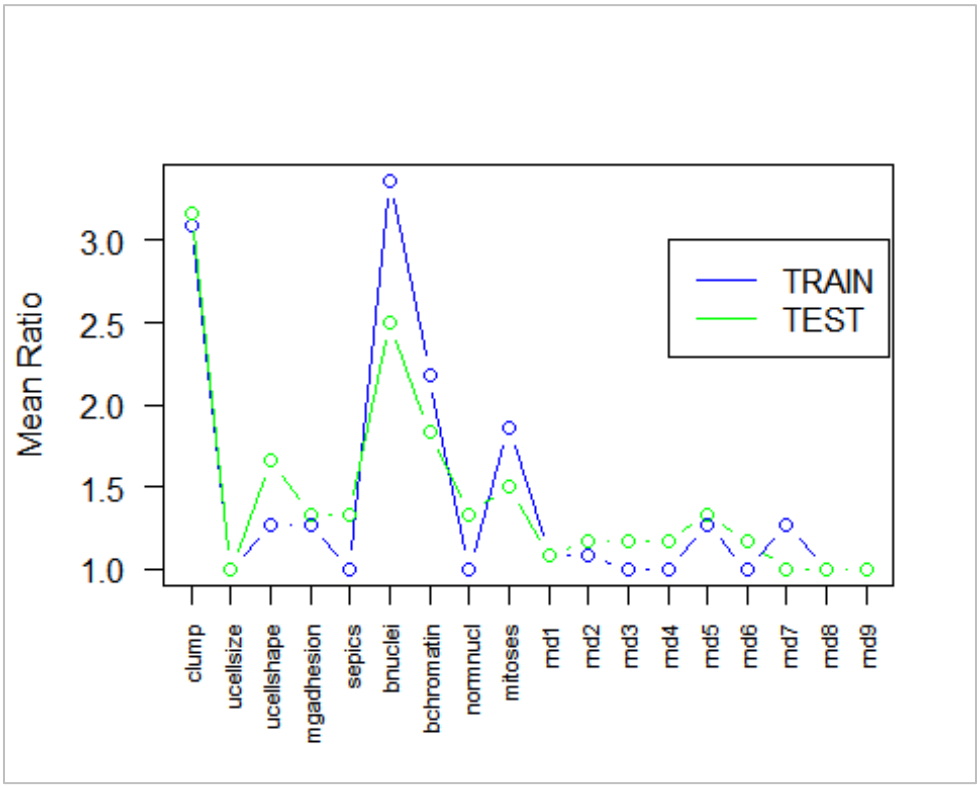
La question du choix entre les échantillons d'apprentissage et de test pour calculer l'importance des variables est discuté dans l'ouvrage (Section 5.5.2). L'auteur ne tranche pas vraiment. Il dit en substance – je schématise – que l'échantillon d'apprentissage permet de mettre en avant le rôle des variables dans la modélisation (il le dit un peu différemment, mais c'est l'idée que j'ai retenu), alors que l'échantillon test éclaire sur l'influence de la variable sur les performances en généralisation (lorsque le modèle est déployé dans la population). Nous nous rappelons que UCELLSHAPE n'était pas si mal placé que cela lorsque nous étudions les coefficients standardisés de la régression logistique.

La nuance entre ces deux alternatives reste néanmoins plus que subtile. Je ne suis pas totalement convaincu par l'argumentation de l'auteur et je me garderai de trancher moi-même. Simplement, si on suit son raisonnement, il serait plus approprié d'utiliser un critère de modélisation quand on travaille sur l'échantillon d'apprentissage, la déviance par

exemple ; et un critère de performance prédictive si on travaille sur l'échantillon test, le taux d'erreur est tout indiqué pour cela (ou encore l'AUC, etc.).

On notera quand-même dans le graphique ci-dessous que les ratios en apprentissage et en test restent cohérents globalement.

```
#pour inscrire le nom des variables dans le bon sens
par(las=2)

#comparaison de feature importance en TRAIN et TEST
plot(med_train,type="b",col="blue",xlab="",ylab ="Mean Ratio",xaxt="n")
axis(side=1,at=1:length(med_train),labels=names(med_train),cex.axis=0.7)
lines(med_test,type="b",col="green")
legend(13,3,legend=c("TRAIN","TEST"),col=c("blue","green"),lty=1)
```



6. Utilisation du package "iml"

Christopher Molinar a développé le package "iml" implémentant plusieurs outils en relation avec l'interprétation des modèles. La classe Predictor ne prend en entrée que des modèles d'un certain type, provenant de packages spécifiques. Il recommande en particulier "caret" et "mlr" qui, tous deux, proposent une série d'outils clés en main, avec des prototypes de fonctions standardisés, facilitant la pratique de la data science.

Ayant déjà étudié "caret" par le passé ("Machine learning avec caret", avril 2018), j'utilise "mlr" dans ce tutoriel.

Modélisation via le package "mlr"

Le package "mlr" fournit une interface standardisée à de nombreux algorithmes de machine learning avec pour objectif de faciliter la tâche du data scientist. Nous pouvons accéder à l'algorithme glm() de "stats" dans cette section, mais en se conformant à la grammaire de "mlr". Les différentes étapes rappellent un peu celles de "scikit-learn" pour Python ("Machine Learning avec Scikit-Learn", septembre 2015).

Dans un premier temps, nous instancions l'algorithme à utiliser.

```
#chargement du package
library(mlr)

## Loading required package: ParamHelpers

#instancier le modèle
lrn <- makeLearner("classif.logreg",predict.type = "prob")
print(lrn)

## Learner classif.logreg from package stats
## Type: classif
## Name: Logistic Regression; Short name: logreg
## Class: classif.logreg
## Properties: twoclass,numerics,factors,prob,weights
## Predict-Type: prob
## Hyperparameters: model=FALSE
```

Nous créons un objet régression logistique, provenant du package "stats", on souhaite que l'outil produise les probabilités d'affectation aux classes.

Nous définissons ensuite la tâche à réaliser en indiquant notamment les données d'apprentissage à traiter, à savoir le data.frame DTrain, où "classe" fait office de variable cible.

```
#définir la tâche et les données à traiter
tache <- makeClassifTask(data=DTrain,target="classe")
print(tache)

## Supervised task: DTrain
## Type: classif
## Target: classe
## Observations: 499
## Features:
##    numerics     factors     ordered functionals
##          18           0           0           0
```

```
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Has coordinates: FALSE
## Classes: 2
##    begnin malignant
##       336       163
## Positive class: begnin
```

"mlr" effectue un check-up de la base, toujours salutaire avant de lancer un algorithme de machine learning.

Nous pouvons ensuite lancer le processus d'apprentissage.

```
#modélisation - apprentissage
classifieur <- train(lrn,tache)
print(classifieur)

## Model for learner.id=classif.logreg; learner.class=classif.logreg
## Trained on: task.id = DTrain; obs = 499; features = 18
## Hyperparameters: model=FALSE
```

Et afficher les coefficients estimés.

```
#afficher le modèle
print(getLearnerModel(classifieur))

##
## Call:  stats::glm(formula = f, family = "binomial", data = getTaskData(.task,
##     .subset), weights = .weights, model = FALSE)
##
## Coefficients:
## (Intercept)        clump    ucellsize   ucellshape   mgadhesion
##  -14.130185     0.697817    -0.020904     0.258290     0.255348
##      sepics      bnuclei   bchromatin     normnucl      mitoses
##    0.087488     0.475709     0.533563     0.090654     0.723860
##        rnd1         rnd2         rnd3         rnd4         rnd5
##    0.067459     0.079342    -0.146935     0.084521     0.269660
##        rnd6         rnd7         rnd8         rnd9
##    0.038936     0.138853     0.002517    -0.020374
##
## Degrees of Freedom: 498 Total (i.e. Null);  480 Residual
## Null Deviance:       630.5
## Residual Deviance: 73.8  AIC: 111.8
```

Puisqu'en sous-main, "mlr" s'est chargé d'appeler `glm()` de "stats" sur les mêmes données, il est tout à fait normal que nous ayons exactement les mêmes résultats que précédemment (section 4 ; coefficients estimés, residual deviance, etc.). L'intérêt de ce petit détour est que le package "iml" chargé de calculer l'importance des variables saura manipuler l'objet "classifieur".

Importance des variables

Il faut charger le package "iml" au préalable.

```
#package mesurer l'importance du modèle
library(iml)
```

Les étapes sont encapsulées dans des méthodes dédiées, elles sont identiques sur les échantillons d'apprentissage et de test.

Sur l'échantillon d'apprentissage

Il faut tout d'abord créer un objet de type Predictor spécifique à "iml", nous lui passons en paramètre le classifieur (déjà entraîné, il n'y a plus d'apprentissage ici), les descripteurs et la variable cible.

```
#objet de modelisation défini sur le TRAIN set
objetTrain <- Predictor$new(classifieur,data=XTrain,y=DTrain$classe)
print(objetTrain)

## Prediction task: classification
## Classes:
```

Nous pouvons ensuite calculer l'importance des variables en demandant 200 itérations (n.repetitions). Nous utilisons "ce" (classification error, taux d'erreur) pour évaluer la pertinence des variables.

```
#calculer l'importance
importanceTrain <- FeatureImp$new(objetTrain,loss="ce",n.repetitions = 200)
print(importanceTrain)

## Interpretation method:  FeatureImp
## error function: ce
##
## Analysed predictor:
## Prediction task: classification
## Classes:
##
## Analysed data:
## Sampling from data.frame with 499 rows and 18 columns.
##
## Head of results:
##       feature importance.05 importance importance.95 permutation.error
## 1    bnuclei     2.1818182   3.227273      4.645455        0.07114228
## 2      clump     1.8136364   2.954545      5.545455        0.06513026
## 3 bchromatin     1.5454545   2.090909      3.368182        0.04609218
## 4    mitoses     0.8181818   1.363636      5.090909        0.03006012
## 5 mgadhesion     0.8181818   1.272727      1.818182        0.02805611
## 6       rnd5     0.8181818   1.272727      1.731818        0.02805611
```

L'affichage par défaut fournit les résultats pour les 6 premières variables avec : la borne basse de l'intervalle de confiance à 90%, la moyenne de l'importance, la borne haute, et la moyenne du taux d'erreur mesuré lors de la neutralisation de la variable.

Via les propriétés de l'objet, nous pouvons obtenir l'erreur de référence :

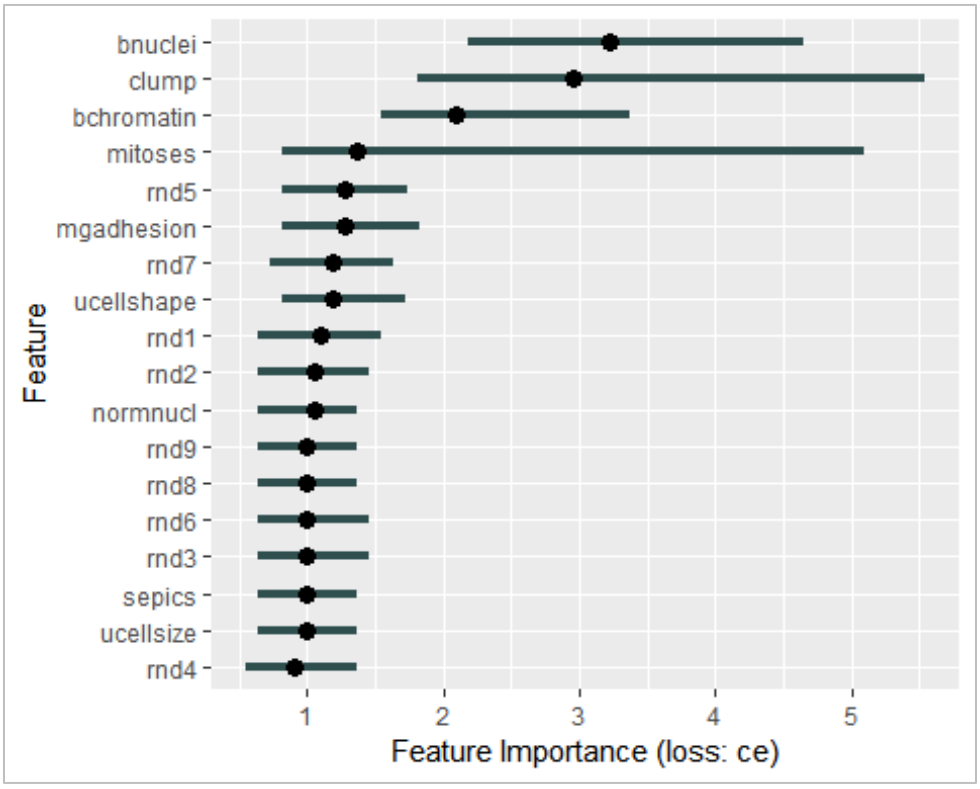
```
#erreur de référence utilisée
print(importanceTrain$original.error)

## [1] 0.02204409
```

Le détail des résultats pour l'ensemble des variables cette fois-ci.

```
#détail des résultats
print(importanceTrain$results)

##       feature importance.05 importance importance.95 permutation.error
## 1     bnuclei     2.1818182  3.2272727      4.645455        0.07114228
## 2       clump     1.8136364  2.9545455      5.545455        0.06513026
## 3  bchromatin     1.5454545  2.0909091      3.368182        0.04609218
## 4     mitoses     0.8181818  1.3636364      5.090909        0.03006012
## 5  mgadhesion     0.8181818  1.2727273      1.818182        0.02805611
## 6        rnd5     0.8181818  1.2727273      1.731818        0.02805611
## 7  ucellshape     0.8181818  1.1818182      1.727273        0.02605210
## 8        rnd7     0.7272727  1.1818182      1.636364        0.02605210
## 9        rnd1     0.6363636  1.0909091      1.545455        0.02404810
## 10   normnucl     0.6363636  1.0454545      1.363636        0.02304609
## 11       rnd2     0.6363636  1.0454545      1.454545        0.02304609
## 12  ucellsize     0.6363636  1.0000000      1.363636        0.02204409
## 13     sepics     0.6363636  1.0000000      1.368182        0.02204409
## 14       rnd3     0.6363636  1.0000000      1.459091        0.02204409
## 15       rnd6     0.6363636  1.0000000      1.454545        0.02204409
## 16       rnd8     0.6318182  1.0000000      1.363636        0.02204409
## 17       rnd9     0.6363636  1.0000000      1.363636        0.02204409
## 18       rnd4     0.5454545  0.9090909      1.368182        0.02004008
```

Un affichage graphique est également possible avec 3 valeurs par variable (bornes de l'intervalle de confiance et la moyenne).

```
#affichage
plot(importanceTrain)
```

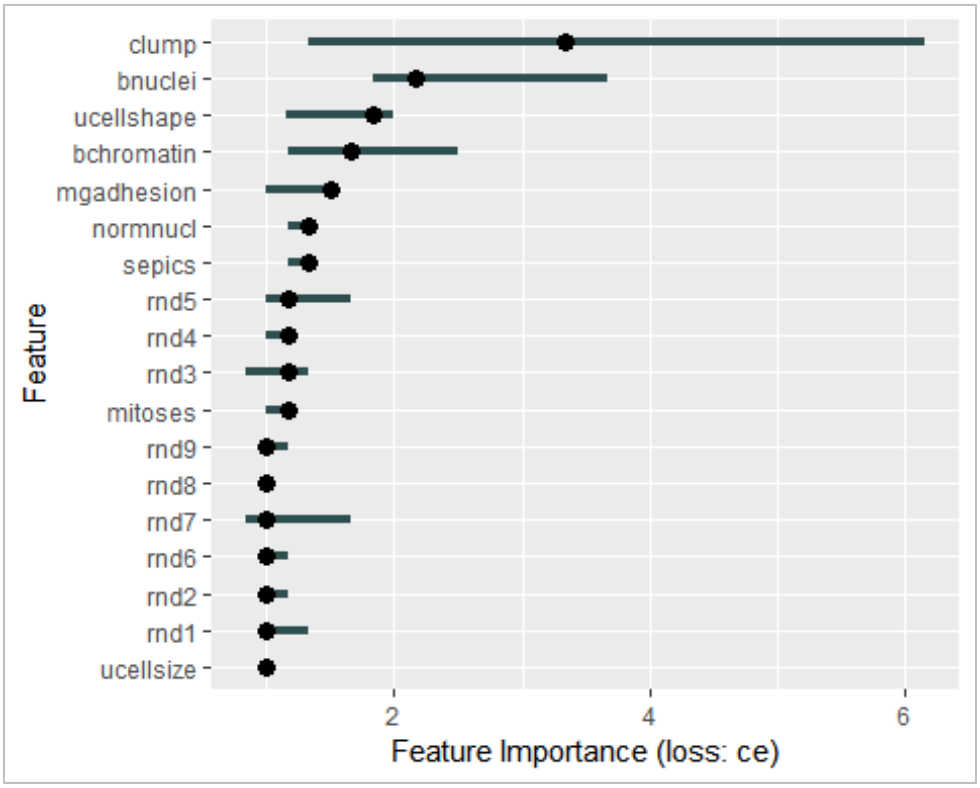


Les résultats sont cohérents avec ceux de notre implémentation, du moins en ce qui concerne le groupe des 4 premières variables. Nous constatons que la variabilité de MITOSES est étrangement très forte et, en moyenne, elle se démarque très peu de RND5 finalement.

Sur l'échantillon test

Nous pouvons réitérer les mêmes opérations sur l'échantillon test.

```
#objet de modelisation défini sur le test set
objetTest <- Predictor$new(classifieur,data=XTest,y=DTest$classe)

#calculer l'importance
importanceTest <- FeatureImp$new(objetTest,loss="ce",n.repetitions = 200)

#affichage
plot(importanceTest)
```



Ici également, UCELLSHAPE prend de l'importance (si je puis dire) sur l'échantillon test et s'intercale parmi les variables les plus influentes. Très étrangement en revanche, MITOSES semble être au niveau des variables non-pertinentes, à contre-courant des résultats obtenus précédemment. Je n'arrive pas à m'expliquer cette incongruité, surtout que 200 répétitions devait amener une certaine assise aux résultats.

7. Conclusion

Mesurer l'impact des variables dans les modèles prédictifs est crucial pour la compréhension des relations qui peuvent exister entre les descripteurs et la variable cible. Certaines méthodes de machine learning disposent de mécanismes intégrés qui permettent de mesurer naturellement les contributions des variables, les modèles linéaires notamment, ou encore dans une certaine mesure les modèles à base de règles. D'autres en revanche en sont dépourvues, on parle de "modèles boîtes noires", il s'agit des approches non linéaires souvent. Ils prédisent de manière satisfaisante sans qu'on ne sache réellement comment. Ce qui est gênant, surtout dans les contextes où nous devons justifier la prise de décision.

L'approche agnostique étudiée dans ce tutoriel, non dépendante des caractéristiques des modèles, est un outil simple et peu gourmand en calculs additionnels pour mesurer l'impact

des variables. Elle pallie les insuffisances des méthodes de machine learning qui n'en possèdent pas intrinsèquement. Elle ouvre également la porte à une sélection efficace et rapide des sous-ensembles de descripteurs pertinents en proposant un ordonnancement des variables selon leurs contributions.

8. Références

A. Fisher, C. Rudin, F. Dominici, "All Models are Wrong but many are Useful: Variable Importance for Black-Box, Proprietary, or Misspecified Prediction Models, using Model Class Reliance", novembre 2018 ; https://arxiv.org/abs/1801.01489v3.

C. Molnar, "Interpretable Machine Learning - A Guide for Making Black Box Models Explainable", 2019 ; https://christophm.github.io/interpretable-ml-book