Projet apprentissage statistique

Ndongo Seye;

2022-08-23

1 Chargement et préparation de données réelles

Nous nommerons le jeu de données de cancer du sein METABRIC téléchargé depuis kaggle par *datacancer* dans ce projet.

1.1 Chargement, description et préparation du jeu de données:

Le jeu de données compte 1904 obervations et 693 variables.

On voit que les patientes décédées du cancer ou non sont codés dans une variable catégorielle *death_from_cancer* à 3 niveaux:

- *Living*: si la patiente est en vie,
- *Died of Disease*: si la patiente est morte du cancer du sein,
- *Died of Other Causes*: si la patiente est morte d'une cause différente du cancer du sein

Pour chaque patiente, on attribue un identifiant unique *patient_id*. On donne l'âge de la patiente au moment où elle a été diagnostiquée porteuse du cancer (*age_at_dignosis*), si ellle suit une chimiothérapie ou pas (*chemotherapy*), le diagnostic du nombre de mutation du cancer (*mutation_count*), la taille de la tumeur (*tumor_size*) et son état de développement (*tumor_stage*).

Nous allons coder la variable *death from cancer* en bianaire où le niveau *Died of Disease*=1 et 0 pour les autres en:

a. créant une variable Y binaire qui vaut 1 pour les patientes décédées du cancer et 0 pour les autres,
b. ne gardant que les variables *age_at_dignosis*, *tumor_size*, *mutation_count*, *death_from_cancer* et celles allant de la colonne 32 à 693,

```r
datacancer <- datacancer %>%
  select( age_at_diagnosis,
          tumor_size, mutation_count,
          Y=death_from_cancer,
          all_of(32:693)
          ) %>%  # ici Y=death_from_cancer pour renommer la variable death_from_cance
  mutate( Y= if_else(Y=="Died of Disease",1,0) ) # pour le recodage en binaire
```

Le jeu de données compte maintenant 1904 observations et 666 colonnes.

c. modifiant les données de mutations en variables binaires: 1 pour chaque mutation, quelque soit le code qui la décrive.

Les données de mutations se terminent par *_mut*. Elles vont de la colonne 494 à la fin du tableau.

```r
datacancer[494:666] <- if_else(datacancer[494:666]==0,0,1) # si pas mutation: 0 sinon 1

datacancer %>%
  filter(is.na(mutation_count) | is.na(tumor_size)
         )# 65 données manquantes pour l'ensemble mutation_count et tumor_size

datacancer <- datacancer %>%
  drop_na(tumor_size,mutation_count)
any(is.na(datacancer)) # pour vérifier qu'il ne reste rien de données manquantes
```

Le jeu de données *datacancer* est en fin préparée pour les parties 2 et 3. J'ai préféré supprimer les données manquantes puisqu'il n'y a qu'en tout 65 au total.

2 Prédiction

2.1 Choisir trois méthodes de prédiction vues en cours

- Méthodes de prédiction avec forêt aléatoire:

  Tout d'abord un arbre de décision est un organigramme qui permet de classifier des données d'entrées ou de prédire la valeur de sortie de ces données. Il est facile à interprété et à visualiser et est péfèré des méthodes à noyau [1]. En particulier, les forêts aléatoires sont robustes et pratiques en apprentissage et impliquent plusieurs arbres de décision et assemblent leurs sorties [1]. Ils permettent aussi de diminuer le risque de sur-apprentissage.

- Méthode de prédiction SVM:

  Elle est simple à utiliser car nécessitant peu de paramètres. SVM est un algorithme de classification qui fonctionne en trouvant des «frontières de décision» séparant deux classes.

- Méthode de prédiction avec pénalité:

  La régression pénalisée permet de gérer les corrélations entre covariables ou encore la sélection des variables. Deux types de pénalités: Lasso pour la sélection de variable et Ridge pour la corrélation entre variables. La pénalité Elastic-net est un bon compromis entre les deux autres types de pénalités.

a. aucune réduction de dimension

1. Méthodes de prédiction avec forêt aléatoire

   - Séparation du jeu d'apprentissage et mise en facteur la variable Y

```
# On met la variable Y en facteur labellisée en (0="no" for Others cases) et ( 1="yes"
datacancer <- datacancer %>%
  mutate(Y=as.factor(if_else(Y==0,"no","yes") ) )

#  Séparation du jeu de données en jeu de test et d'apprentissage
inTrain <- createDataPartition(
  y <- datacancer$Y,
  p=.75,
  list = FALSE
)

# jeu d'apprentissage
cancerTraining <- datacancer %>%
  slice(n=inTrain)
# jeu de test
cancerTest <- datacancer %>%
  slice(n=-inTrain)

# crtl est un paramètre de contrôle avec validation croisée qu'on utilisera dans train
ctrl <- trainControl(
  #method = "boot",
  method="repeatedcv",
  number=10,
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)
```

```r
# Ajustement
rfFit <- train(
  Y ~.,
  data = cancerTraining,
  method = "rf",
  tuneLength = 13,
  trControl = ctrl,
  metric = "ROC"
)
```

```r
knitr::include_graphics("pictures/ROC_cross_validation.png", auto_pdf = T)
```

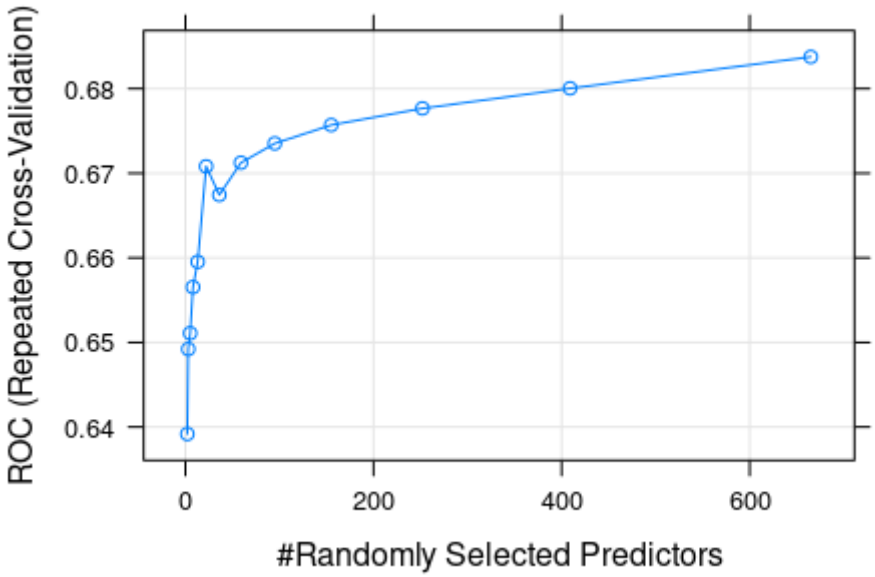

Figure 1: Courbe ROC avec validation croisée

Le modèle ayant la meilleure prédiction par validation croisée est alors le modèle final en sortie. On regardera ensuite les performances de l'arbre apris sur le jeu test

```r
decision <- rfFit$finalModel

#  Prédiction du modèle
prediction <- predict(decision,cancerTest)
# la matrice de confusion
confMat <- confusionMatrix(prediction,cancerTest$Y)
save(confMat, file = "data/confMat.RData")
```

```
## $positive
## [1] "no"
##
## $table
##           Reference
## Prediction  no yes
##        no  295 141
##        yes  12  11
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   6.666667e-01   4.235474e-02   6.214823e-01   7.096866e-01   6.688453e-01
## AccuracyPValue  McnemarPValue
##   5.612743e-01   4.264868e-25
##
## $byClass
##          Sensitivity          Specificity       Pos Pred Value
##           0.96091205           0.07236842           0.67660550
##       Neg Pred Value            Precision               Recall
##           0.47826087           0.67660550           0.96091205
##                   F1           Prevalence       Detection Rate
##           0.79407806           0.66884532           0.64270153
## Detection Prevalence    Balanced Accuracy
##           0.94989107           0.51664024
##
## $mode
## [1] "sens_spec"
##
## $dots
## list()
##
## attr(,"class")
## [1] "confusionMatrix"
```

l'Accuracy indique la proportion de bien classé (le pourcentage de nombres d'individus bien classés sur le jeu text). Il indique 67%. la sensitivité qui indique la proportion de vraies décédées du cancer est de 96%. On a 7% qui sont survécues ou qont mortes d'une autre façon différente non causée par le cancer du sein.

2. Méthode de prédiction SVM

    - Sur le jeu d'apprentissage créé précédemment, on apllique les deux méthodes de SVM: linéaire et à noyaux.

```
linearsvm <-  svm(formula = Y~.,
                  data = cancerTraining,
                  type = 'C-classification',
                  kernel = 'linear'
                  )


gaussiansvm <- svm(formula = Y~.,
                   data = cancerTraining,
                   type = 'C-classification',
                   kernel = 'radial'
                   )
```

- Prédiction sur le jeu test

```
linearpred   <- predict(linearsvm, newdata = cancerTest)
gaussianpred <- predict(gaussiansvm, newdata = cancerTest)

confMatSVM_lin <- confusionMatrix(data=linearpred,cancerTest$Y)
confMatSVM_gau <- confusionMatrix(data=gaussianpred,cancerTest$Y)
save(confMatSVM_lin,file="data/confMatSVM_lin.RData")
save(confMatSVM_gau,file="data/confMatSVM_gau.RData")
```

```
load(file="data/confMatSVM_lin.RData")
load(file="data/confMatSVM_gau.RData")
print(confMatSVM_lin)
```

```
## $positive
## [1] "no"
##
## $table
##            Reference
## Prediction  no yes
##        no  213  89
##        yes  94  63
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.6013072      0.1073930      0.5548900      0.6464103      0.6688453
## AccuracyPValue  McnemarPValue
##      0.9989714      0.7674680
##
## $byClass
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.6938111            0.4144737            0.7052980
##       Neg Pred Value            Precision               Recall
##            0.4012739            0.7052980            0.6938111
##                   F1           Prevalence       Detection Rate
##            0.6995074            0.6688453            0.4640523
## Detection Prevalence    Balanced Accuracy
##            0.6579521            0.5541424
##
## $mode
## [1] "sens_spec"
##
## $dots
## list()
##
## attr(,"class")
## [1] "confusionMatrix"
```

```
print(confMatSVM_gau)
```

```
## $positive
## [1] "no"
##
## $table
##           Reference
## Prediction  no yes
##        no  287 129
##        yes  20  23
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   6.753813e-01   1.052163e-01   6.304286e-01   7.180558e-01   6.688453e-01
## AccuracyPValue  McnemarPValue
##   4.041379e-01   8.933764e-19
##
## $byClass
##          Sensitivity          Specificity       Pos Pred Value
##            0.9348534            0.1513158            0.6899038
##       Neg Pred Value            Precision               Recall
##            0.5348837            0.6899038            0.9348534
##                   F1           Prevalence       Detection Rate
##            0.7939142            0.6688453            0.6252723
## Detection Prevalence    Balanced Accuracy
##            0.9063181            0.5430846
```

```
##
## $mode
## [1] "sens_spec"
##
## $dots
## list()
##
## attr(,"class")
## [1] "confusionMatrix"
```

Au vu des deux matrices de confusion précédentes, la méthode SVM par noyau gaussien est plus adaptée.

3. Méthode de prédiction avec pénalité

Nous disposons de trois méthodes pour la régression pénalisée: Lasso, Ridge et Elastic-net. Soient le jeu d'apprentissage *cancerTraining* et le jeu test *cancerTest* crées précédemment.

```
#jeu d'apprentissage
y.train <- cancerTraining %>%
  select(Y) %>%
  as.matrix()
y.test <- cancerTest %>%
  select(Y) %>%
  as.matrix()

x.train <- cancerTraining %>%
  select(-Y) %>%
  as.matrix()

x.test <- cancerTest %>%
  select(-Y) %>%
  as.matrix()
```

- Pénalité Ridge

```
model.ridge <- glmnet(x.train,y.train,family = "binomial",alpha=0)
plot(model.ridge)
```

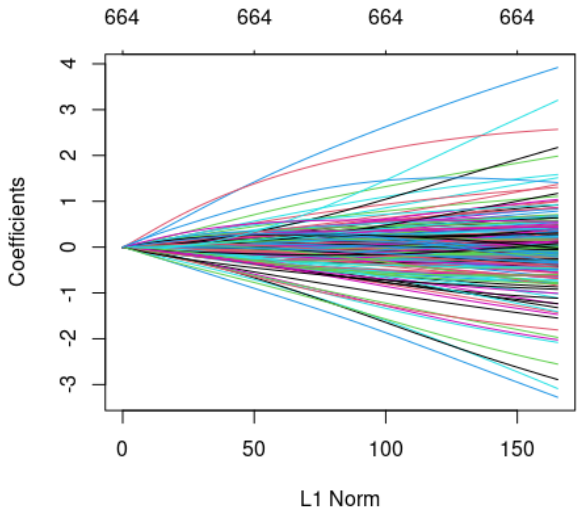

Figure 2: Modèle ridge

- Lasso

```
model.lasso <- glmnet(x.train,y.train,family = "binomial",alpha=1)
```

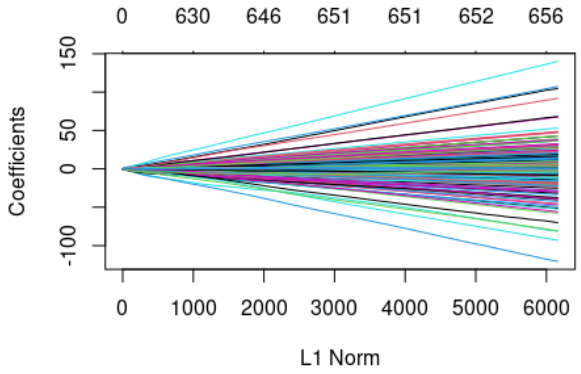

Figure 3: Modèle Lasso

- Elastic net

```
model.elastic_net <- glmnet(x.train,y.train,family = "binomial",alpha=.5)
```

Nous choisirons la méthode *elastic net* en procédant par validation croisée. Car c'est un bon compromis entre les deux.

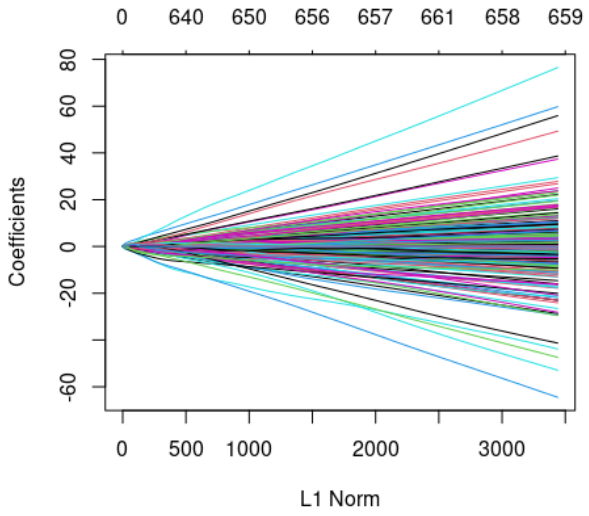

Figure 4: Modèle Elastic net

```
cancer.cv <- cv.glmnet(x.train,y.train,nfolds=20,family="binomial")
cancer.cv$lambda.min
model.EN.cv <- glmnet(x.train,y.train,family="binomial",alpha=0.5,nlambda=1,lambda=cance
```

- Prédiction du modèle sur le jeu test

```
cancerpredict.EN.cv <- predict(model.EN.cv,x.train,
                               type="response",
                               family="binomial"
                               )
results <- tibble(proba=cancerpredict.EN.cv,
                  y.pred=round(cancerpredict.EN.cv),
                  y.truth=(y.train=="yes")*1
                  )
results
cfMat.EN <- confusionMatrix(data = as_factor(results$y.pred),
                            as_factor(results$y.truth)
                            )
save(cfMat.EN,file = "pictures/cfMat_acp.RData")
```

```
## $positive
## [1] "0"
##
## $table
##           Reference
## Prediction   0   1
##          0 860 240
##          1  64 216
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.797101e-01   4.482323e-01   7.568981e-01   8.013246e-01   6.695652e-01
## AccuracyPValue  McnemarPValue
##   1.391578e-19   1.048789e-23
##
## $byClass
##          Sensitivity          Specificity       Pos Pred Value
##            0.9307359            0.4736842            0.7818182
##       Neg Pred Value            Precision               Recall
##            0.7714286            0.7818182            0.9307359
##                   F1           Prevalence       Detection Rate
##            0.8498024            0.6695652            0.6231884
## Detection Prevalence    Balanced Accuracy
##            0.7971014            0.7022101
##
## $mode
## [1] "sens_spec"
##
```

```
## $dots
## list()
##
## attr(,"class")
## [1] "confusionMatrix"
```

La matrice de confusion donne un bon taux de classement (78%) sur le jeu test avec une bonne sensitivité de 93% et une spécificité assez faible de 47.4%.

b. par ACP

- Réduction de dimension par ACP

Nous choisissons de prendre les variables nécessaires qui expliquent 70% du résultat de l'ACP

```
acp <- datacancer %>%
  select(-Y) %>% as.matrix() %>%
  prcomp()

# selection des variables avec 70%
nvar <- which(cumsum(acp[["sdev"]])/sum(acp[["sdev"]])>.7)

cancerTraining_ACP <- as_tibble(acp$x[,1:nvar]) %>%
                    mutate(Y=datacancer$Y,
                    .before="PC1"
                      ) %>%
                    slice(n=inTrain)

cancerTest_ACP <- as_tibble(acp$x[,1:nvar]) %>%
                    mutate(Y=datacancer$Y,
                    .before="PC1"
                      ) %>%
                    slice(n=-inTrain)
```

On a une diminution de nombre de variable de 244.

1. Méthodes de prédiction avec forêt aléatoire

```
rfFit_ACP <- train(
  Y ~ .,
  data = cancerTraining_ACP,
  method = "rf",
  tuneLength = 13, #donne le nombre de valeurs à essayer pour chaque paramètre à faire
```

```
  trControl = ctrl,
  metric = "ROC"
)
plot(rfit)
```

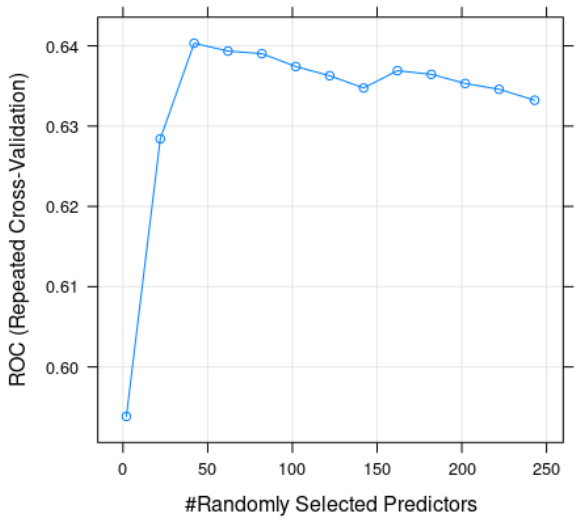

Figure 5: Courbe ROC par validation croisée/Avec réduction de dimension

```
## $positive
## [1] "no"
##
## $table
##           Reference
## Prediction  no yes
##        no  297 146
##        yes  10   6
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   6.601307e-01   8.913742e-03   6.147833e-01   7.033988e-01   6.688453e-01
## AccuracyPValue  McnemarPValue
##   6.738991e-01   3.132446e-27
##
## $byClass
##          Sensitivity          Specificity       Pos Pred Value
##            0.96742671           0.03947368           0.67042889
##       Neg Pred Value            Precision               Recall
##            0.37500000           0.67042889           0.96742671
```

```
##                   F1           Prevalence       Detection Rate
##           0.79200000           0.66884532           0.64705882
## Detection Prevalence    Balanced Accuracy
##           0.96514161           0.50345020
##
## $mode
## [1] "sens_spec"
##
## $dots
## list()
##
## attr(,"class")
## [1] "confusionMatrix"
```

66% des individus sont bien classés selon cette méthode avec un bon taux du nombre de patientes décédées (97%). Cependant la spécifité est très faible (4%)

2. Méthode de prédiction SVM

- On réutilise les jeux d'apprentissage et de test précédemment dans cette dsection (*cancderTraining_ACP* et *canccerTest_ACP*).

```
linearsvm_acp <-  svm(formula = Y~.,
                  data = cancerTraining_ACP,
                  type = 'C-classification',
                  kernel = 'linear')

gaussiansvm_acp <- svm(formula = Y~.,
                   data = cancerTraining_ACP,
                   type = 'C-classification',
                   kernel = 'radial')
```

- Prédiction sur le jeu test et matrice de confusion

```
linearpred_acp      <- predict(linearsvm_acp, newdata = cancerTest_ACP)
gaussianpred_acp    <- predict(gaussiansvm_acp, newdata = cancerTest_ACP)
confMat_svm_acp_lin <- confusionMatrix(data=linearpred_acp,cancerTest_ACP$Y)
confMat_svm_acp_gau <- confusionMatrix(data=gaussianpred_acp,cancerTest_ACP$Y)

save(confMat_svm_acp_lin, file = "data/confMat_svm_acp_lin.RData")
save(confMat_svm_acp_gau,file = "data/confMat_svm_acp_gau.RData")
```

```r
load("data/confMat_svm_acp_lin.RData")
load("data/confMat_svm_acp_gau.RData")

confMat_svm_acp_lin
```

```
## $positive
## [1] "no"
##
## $table
##           Reference
## Prediction  no yes
##        no  240 110
##        yes  67  42
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##    0.614379085    0.062541108    0.568139669    0.659134549    0.668845316
## AccuracyPValue  McnemarPValue
##    0.993830902    0.001594487
##
## $byClass
##          Sensitivity          Specificity       Pos Pred Value
##            0.7817590            0.2763158            0.6857143
##       Neg Pred Value            Precision               Recall
##            0.3853211            0.6857143            0.7817590
##                   F1           Prevalence       Detection Rate
##            0.7305936            0.6688453            0.5228758
## Detection Prevalence    Balanced Accuracy
##            0.7625272            0.5290374
##
## $mode
## [1] "sens_spec"
##
## $dots
## list()
##
## attr(,"class")
## [1] "confusionMatrix"
```

```r
confMat_svm_acp_gau
```

```
## $positive
## [1] "no"
```

```
##
## $table
##           Reference
## Prediction  no yes
##        no  305 145
##        yes   2   7
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   6.797386e-01   5.185279e-02   6.349081e-01   7.222341e-01   6.688453e-01
## AccuracyPValue  McnemarPValue
##   3.293343e-01   1.106874e-31
##
## $byClass
##          Sensitivity          Specificity       Pos Pred Value
##            0.99348534           0.04605263           0.67777778
##       Neg Pred Value            Precision               Recall
##            0.77777778           0.67777778           0.99348534
##                   F1           Prevalence       Detection Rate
##            0.80581242           0.66884532           0.66448802
## Detection Prevalence    Balanced Accuracy
##            0.98039216           0.51976899
##
## $mode
## [1] "sens_spec"
##
## $dots
## list()
##
## attr(,"class")
## [1] "confusionMatrix"
```

3. Méthode de prédiction avec pénalité

```
#jeu d'apprentissage
y.train_acp <- cancerTraining_ACP %>%
  select(Y) %>%
  as.matrix()

x.train_acp <- cancerTraining_ACP %>%
  select(-Y) %>%
  as.matrix()
```

- Pénalité Ridge

```
model.ridge_acp <- glmnet(x.train_acp,y.train_acp,family = "binomial" ,alpha=0)
plot(model.ridge_acp)
```

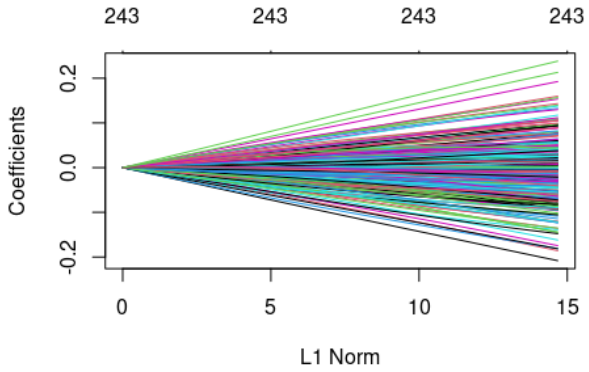

Figure 6: Méthde de Ridge

- Pénalité Lasso

```
model.lasso_acp <- glmnet(x.train_acp,y.train_acp,family = "binomial" , alpha=1)
plot(model.lasso_acp)
```

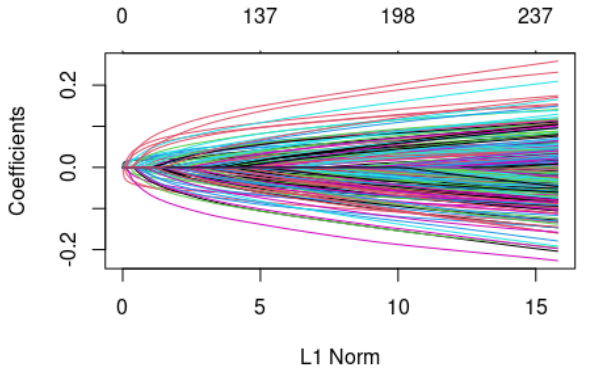

Figure 7: Pénalité Lasso

- Pénalité Elastic net

```
model.elastic_net_acp <- glmnet(x.train_acp,y.train_acp,family = "binomial" , alpha=.5)
plot(model.elastic_net_acp)
```

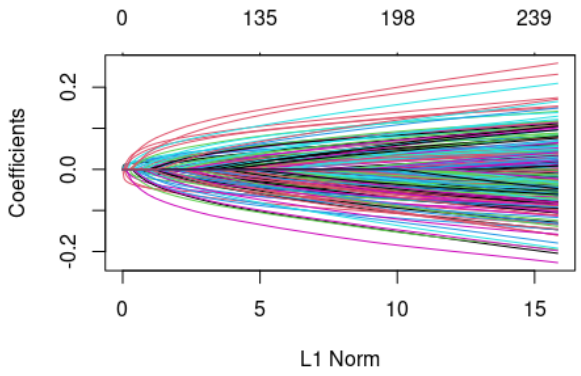

Figure 8: Elastic net

- Prédition avec elastic net

```
cancer.cv_acp <- cv.glmnet(x.train_acp,y.train_acp,
                           nfolds=10,
                           family="binomial"
                           )
cancer.cv_acp$lambda.min
model.EN.cv_acp <- glmnet(x.train_acp,
                          y.train_acp,
                          family="binomial" ,
                          alpha=0.5,nlambda=100,
                          lambda=cancer.cv_acp$lambda.min
                          )

cancerpredict.EN.cv_acp <- predict(model.EN.cv_acp,x.train_acp,
                                   type="response", family="binomial"
                                   )
results_acp <- tibble(proba=cancerpredict.EN.cv_acp,
                      pred=round(cancerpredict.EN.cv_acp),
                      y.truth=(y.train_acp=="yes") *1
                      )
results_acp
confMat_EN_acp <- confusionMatrix(data=as_factor(results_acp$pred),
                                  as_factor(results_acp$y.truth)
                                  )
save(confMat_EN_acp,file = "data/confMat_EN_acp.RData")
```

- Matrice de confusion

```
## $positive
## [1] "0"
##
## $table
##           Reference
## Prediction   0   1
##          0 876 313
##          1  48 143
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.384058e-01   3.068009e-01   7.143632e-01   7.614288e-01   6.695652e-01
## AccuracyPValue  McnemarPValue
##   1.750297e-08   6.817361e-44
##
## $byClass
##          Sensitivity          Specificity       Pos Pred Value
##            0.9480519            0.3135965            0.7367536
##       Neg Pred Value            Precision               Recall
##            0.7486911            0.7367536            0.9480519
##                   F1           Prevalence       Detection Rate
##            0.8291529            0.6695652            0.6347826
## Detection Prevalence    Balanced Accuracy
##            0.8615942            0.6308242
##
## $mode
## [1] "sens_spec"
##
## $dots
## list()
##
## attr(,"class")
## [1] "confusionMatrix"
```

On a 74% d'individus bien classés mais avec un mauvais taux de vrai négatif (seulement 31,4% des individus non décédées du cancer sont détectés). Par contre on a une sensitivité de 95%.

c. par auto-encodeur

Nous utiliserons le package *keras* pour cette partie auto-encodeur et considérons le jeu d'apprentissage *cancerTraining*

- Préparation des données

```
y <-  cancerTraining %>%
  mutate(Y=if_else(Y=="no",0,1)) %>%
  select(Y)  %>%
  as.matrix()
y <- to_categorical(y)

y.test <-  cancerTest %>%
  mutate(Y=if_else(Y=="no",0,1)) %>%
  select(Y) %>%
  as.matrix()
y.test <- to_categorical(y.test)


x <- cancerTraining %>%
  select(-Y) %>%
  as.matrix()
x <- array_reshape(x,dim(x))

x.test <- cancerTest %>%
  select(-Y) %>%
  as.matrix()
x.test <- array_reshape(x.test,dim(x.test))
```

- Apprentissage

```
model <- keras_model_sequential( input_shape = c(665))
model %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 32, activation = "sigmoid") %>%
  layer_dense(units = 2, activation = "softmax")

model

model$weights
```

```
# compile
model %>% compile(
#loss = 'binary_crossentropy',
loss = c("mean_squared_error","binary_crossentropy"),
optimizer = 'adam',
metrics = c('accuracy')
)

# Validation

# Nous utilisons l'argument validation_data pour apprendre sur le jeu de données test
# les erreurs quadratiques ansi que la proportion d'individus classée
history <- model %>% fit(
x, y,
epoch=100,
batch_size = 128,
validation_data=list(x.test,y.test)
#validation_split = 0.2,
#verbose=FALSE
)

str(history$metrics)
str(history$params)

history$metrics$val_accuracy[100]
history$metrics$val_loss[100]

plot(history)
```

Avec keras, on a une précision de 61% sur 100 itération et une erreur quadratique moyenne de .302.

- Prédicton

```
y.pred  <- model %>%
  predict(x.test, type = "response") %>%
  apply(1,which.max)-1

y.test_test <- y.test %>%
    apply(1,which.max)-1


head(predictions, 10)
```

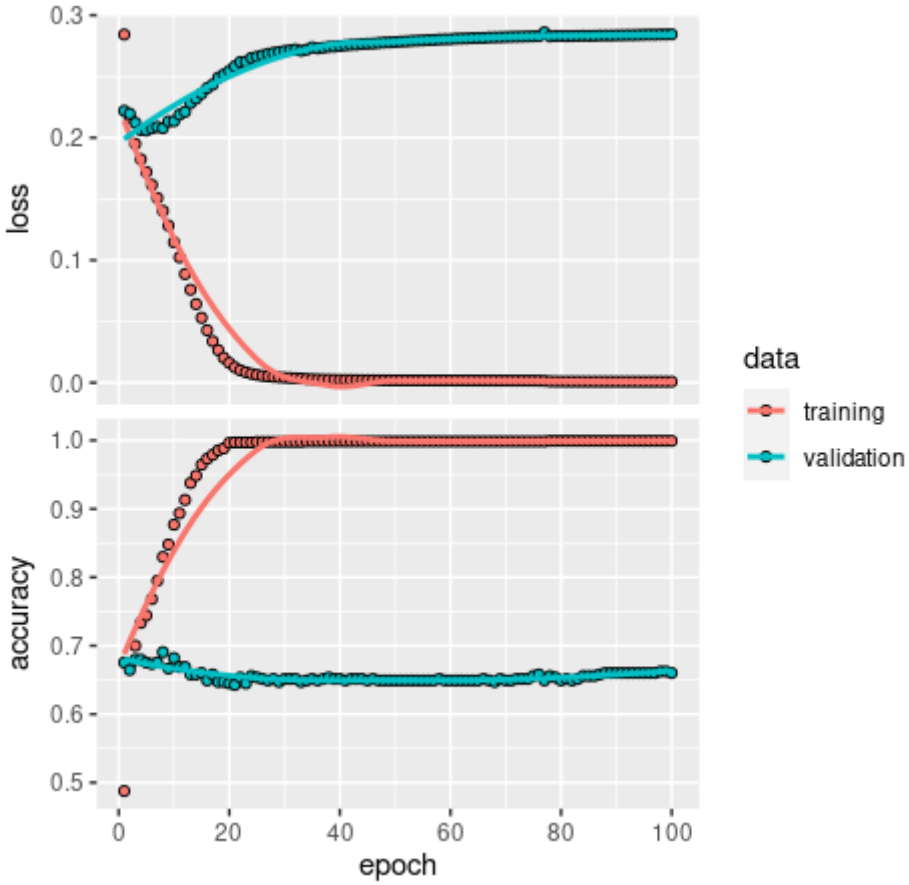

Figure 9: Erreur moyenne quadratique et prédiction

```
datapred <- data.frame(x1=predictions[,1],x2=predictions[,2],y=y.pred)

confusionMatrix(as_factor(y.pred), as_factor(y.test_test))
```

On a une sensitivité de 76% et une spécifité de 32%.

2.2 Discussion sur leurs avantages et inconvénients

- forêt aléatoire
  - Avantages: permet d'éviter le surapprentissage, paramètre facile à déterminer.
  - Inconvénients: computentionnellement lourd à réaliser quand le nombre d'arbres est grand
- Méthode SVM
  - Avantages: précision sur la prédiction, fonctionne bien sur de petits jeux de données
  - Inconvénients: ne convient pas aux jeux de données trop volumineux, Difficulté d'interprétations
- Régression pénalisée
  - Avantages: en utilisant elastic-net permet de conserver les avantages de Lasso et de Ridge
  - Inconvénients: il y a un nouveau paramètres à définir par cross-validation

Comme nous avons un jeu de données volumineux (1839x666), la réduction de dimension a été utile, réduisant le nombre de variables à 244.

3 Sélection

Proposons une manière d'effectuer une selection à l'aide d'une selection par stabilité et mettrons la en oeuvre. On lance B bootstraps et on apprend sur chacun d'eux un modèle de pénalité Elastic net ($\alpha = 0.5$) sans aucune réduction de dimension et on garde les ARN/mutaions les plus souvent selectionnés. B a été fixé à *2000*.

```r
# on réactualise le jeu d'apprentissage x.train en haut partie 'aucune réduction de di
B=2000 # nombre de bootstraps
ncandidats <- dim(x.train)[1]
selected <- rep(0,dim(x.train)[2])
for (b in 1:B){
  #echantillon bootstrap
  bootsample <- sample(1:ncandidats,ncandidats,replace=TRUE)
  x.boot <- x.train[bootsample,]
  y.boot <- y.train[bootsample,]
  #apprentissage
  model.boot <- glmnet(x.boot,y.boot,family="binomial",alpha=0.5,nlambda=1,lambda=.5*can
  selectedmutations <- model.boot$beta@i
  for (i in selectedmutations){
    selected[i] <- selected[i] +1
  }
}
```

```r
#restriction du jeu d'apprentissage aux mutations étant sélectionnés au moins 1/2 du t
selection <- which(selected>B/2)

x.train.stab <- data.frame(x.train[,selection]) %>%
  select(ends_with('_mut')) # selection des 130 mutations sur les 173 de départ

cancerTrain.stab <- data.frame(Y=y.train,x.train[,selection]) %>%
  as_tibble() %>%
  mutate(Y=if_else(Y=="no",0,1))

x.test <- data.frame(x.test) %>%
  as_tibble()


# apprentissage suivant un modèle linéaire classique
model <- glm(Y~.,cancerTrain.stab,family = binomial(link = "logit"))

predictions <- predict(model,newdata = x.test,type="response")

confMat_selection <- confusionMatrix(data=as_factor(round(predictions)),as_factor((y.tes
save(confMat_selection, file="data/confMat_selection.RData")
```

```
## $positive
## [1] "0"
##
## $table
##           Reference
```

```
## Prediction   0   1
##          0 244  98
##          1  63  54
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     0.64923747     0.15931197     0.60363860     0.69289896     0.66884532
## AccuracyPValue  McnemarPValue
##     0.82711787     0.00737156
##
## $byClass
##          Sensitivity          Specificity       Pos Pred Value
##            0.7947883            0.3552632            0.7134503
##       Neg Pred Value            Precision               Recall
##            0.4615385            0.7134503            0.7947883
##                   F1           Prevalence       Detection Rate
##            0.7519260            0.6688453            0.5315904
## Detection Prevalence    Balanced Accuracy
##            0.7450980            0.5750257
##
## $mode
## [1] "sens_spec"
##
## $dots
## list()
##
## attr(,"class")
## [1] "confusionMatrix"
```

On retrouve 130 variables des mutations et expressions de gènes sélectionnées la plus part du temps. Le taux de vrais positifs est assez bonne (79.5%) mais le taux de vrais négatifs est très faible (35.5%)

4 Références

github.com/ndongo4/Machine-learning-project/

[1] CHOLLET F., KALINOWSKI T., ALLAIRE J. J. Second Edition.[s.l.] : MANNING, 2022.