Cross validation

1. Importer les données:

Les données sont stockées dans le fichier **breast_cancer.csv** du répertoire **data** . On va lire le fichier en utilisant la fonction **read.csv()** et on va affecter la sortie de la fonction dans la variable **data**. Aprés on va appeler la fonction **head()** qui permet d'afficher les six premières lignes de la variable 'data'

```
data <- read.csv('C:/Users/asus/OneDrive/Desktop/Projet/data/breast_cancer.csv',
                 sep=';', row.names='id_sample')
head(data)
```

```
##                       NAT1    BIRC5     BAG1     BCL2    BLVRA    CCNB1    CCNE1
## TCGA-3C-AAAU-01A 7.100449 3.361004 3.972581 4.145669 4.765233 4.788987 2.164814
## TCGA-3C-AALI-01A 3.453640 4.501040 2.720929 1.493020 5.823480 5.281003 2.535437
## TCGA-3C-AALJ-01A 4.455574 4.164643 3.911511 4.191457 5.987255 5.229446 2.267963
## TCGA-3C-AALK-01A 4.297961 3.920234 3.688335 3.894904 5.211594 4.014641 0.951107
## TCGA-4H-AAAK-01A 1.695378 2.950846 4.110014 3.572843 4.317856 3.772768 1.103958
## TCGA-5L-AAT0-01A 3.839230 2.030192 3.649085 3.958096 5.186208 3.130406 1.138433
##                       CDC6    CDC20     CDH3    CENPF     EGFR    ERBB2     ESR1
## TCGA-3C-AAAU-01A 2.633598 4.131205 0.133455 3.182165 0.689091 5.239199 3.877698
## TCGA-3C-AALI-01A 2.734157 4.176553 0.110023 3.392215 0.954435 9.927166 0.382603
## TCGA-3C-AALJ-01A 3.379961 4.592752 0.236786 2.984169 0.821807 5.804556 4.859469
## TCGA-3C-AALK-01A 1.472950 3.806552 0.062392 2.893521 2.044761 7.382023 3.284898
## TCGA-4H-AAAK-01A 2.338953 3.473484 0.098773 2.514709 1.527022 5.942930 4.450677
## TCGA-5L-AAT0-01A 1.494393 2.720610 0.061163 2.014191 1.584587 5.579183 4.690124
##                      FGFR4    FOXC1     GRB7    FOXA1     KRT5    KRT14    KRT17
## TCGA-3C-AAAU-01A 0.923478 0.342602 3.845602 6.198849 0.034380 0.000000 0.009777
## TCGA-3C-AALI-01A 4.653488 0.794085 8.780099 6.303003 2.539948 2.954742 3.755362
## TCGA-3C-AALJ-01A 0.605796 1.628313 4.467610 5.905421 0.098767 0.105929 0.310048
## TCGA-3C-AALK-01A 1.462492 2.098339 4.968695 6.470010 5.303344 6.553266 6.691324
## TCGA-4H-AAAK-01A 3.292628 1.559426 4.087059 6.456412 5.081349 5.968474 5.939790
## TCGA-5L-AAT0-01A 0.853090 1.740289 3.603108 6.353737 5.114444 6.002268 5.181969
##                       MAPT     MDM2    MKI67    MMP11    MYBL2      MYC      PGR
## TCGA-3C-AAAU-01A 4.917417 5.880554 3.496616 6.383997 3.961628 5.324133 3.152494
## TCGA-3C-AALI-01A 0.767498 2.169065 3.293928 6.031836 6.328959 1.997876 0.127529
## TCGA-3C-AALJ-01A 3.530899 3.002628 2.222538 6.646212 5.776590 4.428182 1.399339
## TCGA-3C-AALK-01A 3.180234 2.629040 2.355445 5.776565 3.277660 4.540685 2.991844
## TCGA-4H-AAAK-01A 4.834376 2.414163 2.498481 7.791991 3.300091 6.156276 3.961307
## TCGA-5L-AAT0-01A 4.596884 2.475144 1.568097 7.717721 2.301077 4.805902 1.792283
##                       RRM2    SFRP1     TYMS      MIA     EXO1    PTTG1     MELK
## TCGA-3C-AAAU-01A 4.069746 2.610955 3.316276 0.056151 1.851411 3.551024 2.541704
## TCGA-3C-AALI-01A 5.110849 1.028520 3.660515 0.239768 2.893140 3.976765 3.232838
## TCGA-3C-AALJ-01A 3.858266 0.767785 3.567142 0.047283 1.735396 4.473512 3.361018
## TCGA-3C-AALK-01A 3.293315 3.701924 3.479655 1.252874 1.311587 3.117028 2.326996
## TCGA-4H-AAAK-01A 2.302735 4.090581 3.474526 1.224086 1.119937 2.956249 2.377493
```

```
## TCGA-5L-AAT0-01A 2.092475 4.301228 2.611648 1.128081 0.877216 2.300325 1.536737
##                      NDC80    KIF2C    UBE2C     ORC6   SLC39A6    PHGDH
## TCGA-3C-AAAU-01A 2.687899 3.422050 4.678216 1.837654 10.319962 2.991554
## TCGA-3C-AALI-01A 2.481391 3.674568 5.883007 2.762908  4.579932 2.973480
## TCGA-3C-AALJ-01A 3.015086 3.505261 5.706425 2.500333  7.747377 1.003328
## TCGA-3C-AALK-01A 1.788017 2.322912 4.378114 0.811602  7.921403 2.781136
## TCGA-4H-AAAK-01A 1.830427 2.509486 3.878444 0.816464  5.725175 2.728038
## TCGA-5L-AAT0-01A 1.253309 1.582389 3.057444 0.621261  5.251726 2.413841
##                    GPR160    UBE2T    CXXC5     ANLN    CEP55   ACTR3B     MLPH
## TCGA-3C-AAAU-01A 4.150233 4.106918 5.528618 3.073409 2.669860 1.928460 5.567999
## TCGA-3C-AALI-01A 5.561226 5.648057 4.711309 3.881110 3.357553 1.168684 7.064176
## TCGA-3C-AALJ-01A 2.859309 5.213461 6.152875 2.697093 2.599436 1.177678 5.222420
## TCGA-3C-AALK-01A 3.063807 4.166154 5.612184 2.645664 2.448027 1.026535 6.225590
## TCGA-4H-AAAK-01A 3.289418 3.437585 4.299617 2.068516 2.152652 1.513181 5.485277
## TCGA-5L-AAT0-01A 3.236713 3.555052 5.281731 1.050713 1.284239 1.413346 6.059373
##                      NUF2  TMEM45B         pam50
## TCGA-3C-AAAU-01A 2.536764 0.213597     luminal-A
## TCGA-3C-AALI-01A 3.124620 3.946538 HER2-enriched
## TCGA-3C-AALJ-01A 3.053335 0.281303     luminal-B
## TCGA-3C-AALK-01A 1.717959 3.289543     luminal-A
## TCGA-4H-AAAK-01A 1.537125 2.976903     luminal-A
## TCGA-5L-AAT0-01A 0.947315 2.884543     luminal-A
```

Afficher les dimensions des données:

```
#Définir la graine pour la reproductibilité
set.seed(123)
cat('data', dim(data),'\n')
```

```
## data 1016 51
```

Afficher les occurences de chaque niveau de la colonne pam50:

```
group_sizes <- table(data$pam50)
print(group_sizes)
```

```
##
##    basal-like HER2-enriched     luminal-A     luminal-B
##           190            82           543           201
```

2.Séparer les données d'expression et les étiquettes:

On va selectionner tout d'abord les colonnes de type numérique en utilisant la fonction **select_if** du package **dplyr** et afficher leurs dimensions en combinant les deux fonctions **cat()** et **dim()**

```r
library(dplyr)
```

```
##
## Attachement du package : 'dplyr'

## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag

## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
#Données d'expression de 50 gènes
X <- select_if(data, is.numeric)
cat('X', dim(X), '\n')
```

```
## X 1016 50
```

Extraire une colonne:

```r
#Etiquettes correspondantes (sous-types moléculaires)
y <- data$pam50
cat('y', length(y), '\n')
```

```
## y 1016
```

3.Créer une validation croisée:

```r
#Initialisation
 library(ggplot2)
 library(lattice)
 library(caret)
 num_folds <- 3
 folds <- createFolds(y, k = num_folds, list = TRUE, returnTrain = TRUE)

#Boucler sur les plis
  for (fold in 1:num_folds) {
    #Extraire les indices d'entraînement et de test
    train_indices <- unlist(folds[[fold]])
    test_indices <- setdiff(seq_along(y), train_indices)
    #Données d'entraînement et de test
    X_train <- X[train_indices, , drop = FALSE]
    X_test <- X[test_indices, , drop = FALSE]
    #Afficher les résultats pour chaque itération
    cat('Fold', fold, 'Train:', dim(X_train), 'Test:', dim(X_test), '\n')
  }
```

```
## Fold 1 Train: 678 50 Test: 338 50
## Fold 2 Train: 677 50 Test: 339 50
## Fold 3 Train: 677 50 Test: 339 50
```

```r
library(e1071)
 #Définir la graine pour la reproductibilité
 set.seed(0)
 #Créer une fonction pour calculer l'accuracy
 accuracy <- function(y_true, y_pred) {
   mean(y_true == y_pred)
 }

y <- factor(y)
#Initialiser le modèle SVM
classifier <- svm(pam50 ~ ., data = data.frame(X, pam50 = y),
                  kernel = "linear", class.weights = NULL)
#Initialiser la validation croisée
num_folds <- 3
folds <- createFolds(y, k = num_folds, list = TRUE, returnTrain = TRUE)
#Initialiser un vecteur pour stocker les précisions
accuracy_vector <- numeric(length = length(folds))

#Boucler sur les plis
   for (iteration in seq_along(folds)) {
     #Extraire les indices d'entraînement et de test
      train_indices <- unlist(folds[[iteration]])
      test_indices <- setdiff(seq_along(y), train_indices)

     #Données d'entraînement et de test
     X_train <- X[train_indices, , drop = FALSE]
     y_train <- y[train_indices]
     X_test <- X[test_indices, , drop = FALSE]
     y_test <- y[test_indices]

     #Normaliser les données
     scaler <- scale
     X_train_scaled <- scaler(X_train)
     X_test_scaled <- scaler(X_test, center =
                  attr(X_train_scaled,  "scaled:center"), scale =
                  attr(X_train_scaled, "scaled:scale"))

     #Entraîner le modèle SVM
     model <- svm(x = X_train_scaled, y = y_train,
                  kernel = "linear", class.weights = NULL)

     #Prédiction sur les données de test
     y_pred_test <- predict(model, newdata = X_test_scaled)

     #Calculer l'accuracy
     accuracy_vector[iteration] <- accuracy(y_test, y_pred_test)

     #Afficher les résultats pour chaque itération
     cat('Iteration', iteration,
         'Accuracy =', format(accuracy_vector[iteration],
```

```
          digits = 8), '\n')
  }
```

```
## Iteration 1 Accuracy = 0.94690265
## Iteration 2 Accuracy = 0.94690265
## Iteration 3 Accuracy = 0.93195266
```

```
  #Afficher la moyenne des précisions
  cat('Mean accuracy', format(mean(accuracy_vector), digits = 3), '\n')
```

```
## Mean accuracy 0.942
```

```
  #Créer un modèle SVM avec prétraitement
   svm_model <- svm(pam50 ~ ., data = data.frame(X, pam50 = y),
                    kernel = "linear", class.weights = NULL)
  #Créer un pipeline avec le modèle SVM et le prétraitement
  pipeline <- caret::train(
   pam50 ~ .,
   data = data.frame(X, pam50 = y),
   method = "svmLinear",
   trControl = trainControl(method = "cv", number = 3),  # 3-fold CV
   preProcess = c("center", "scale"),  #Centrer et réduire les données
   class.weights = "balanced"  #Poids de classe équilibrés
  )
print(pipeline)
```

```
## Support Vector Machines with Linear Kernel
##
## 1016 samples
##   50 predictor
##    4 classes: 'basal-like', 'HER2-enriched', 'luminal-A', 'luminal-B'
##
## Pre-processing: centered (50), scaled (50)
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 676, 678, 678
## Resampling results:
##
##   Accuracy   Kappa
##   0.9320629  0.8928915
##
## Tuning parameter 'C' was held constant at a value of 1
```