Assignment 4
Comparing Discriminant Rules. ROC Curve and other methods

Víctor Duque, Geraldo Gariza, Jan Leyva and Andreu Meca
Universitat Politècnica de Catalunya

9th of March, 2021

---

1. Use the script spam.R to read the data from the SPAM e-mail database.

```
source("spam.R")
```

```
## n = 4601, p = 57Proportion of spam e-mails =0.39
```

2. Divide the data into two parts: 2/3 for the training sample, 1/3 for the test sample. You should do it in a way that SPAM e-mail are 2/3 in the training sample and 1/3 in the test sample, and that the same happens for NO SPAM e-mails.

```
spam_1 <- which(spam$spam.01 == 1)
idx_train_1 <- sample((spam_1),
                      size = (2/3)*length(spam_1),
                      replace = FALSE)
spam_0 <- which(spam$spam.01 == 0)
idx_train_0 <- sample((spam_0),
                      size = (2/3)*length(spam_0),
                      replace = FALSE)
train_idx <- bind_rows(as_tibble(idx_train_0), as_tibble(idx_train_1))
train <- spam[train_idx$value,]
test <- spam[-(train_idx$value),]
rm(spam_1, idx_train_1, spam_0, idx_train_0, train_idx)
```

3. Consider the following three classification rules: Use the training sample to fix the tunning parameters (when needed) and to estimate the model parameters (when needed).

- Logistic regression fitted by maximum likelihood (IRWLS, glm).

```
mle <- glm(train$spam.01 ~., data = train, family = binomial())
```

- Logistic regression fitted by Lasso (glment).

```
lasso <- glmnet::glmnet(as.matrix(select(train, !spam.01)),
                        y=as.factor(train$spam.01),
                        alpha=1,
                        family="binomial",
                        lambda = glmnet::cv.glmnet(as.matrix(select(train,
                                                                    !spam.01)),
                                            y=as.factor(train$spam.01),
                                            alpha=1,
                                            family="binomial")$lambda.1se)
```

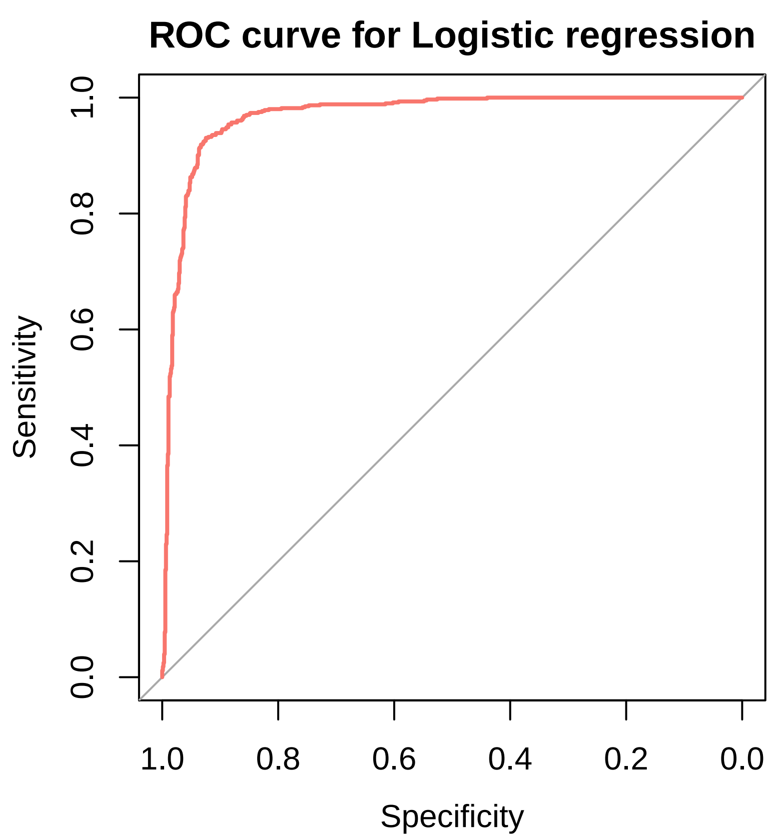
- k-nn binary regression (you can use your own implementation or functions knn and knn.cv from the R package class).

```
k_nn <- class::knn(train =
                train[1:57], test[1:57], k=10 ,
                   cl = as.factor(train$spam.01), prob = TRUE)

k_nn.cv <- class::knn.cv(train = train[1:57],
                         cl = as.factor(train$spam.01), prob = TRUE)
```

4. Use the test sample to compute and plot the ROC curve for each rule.

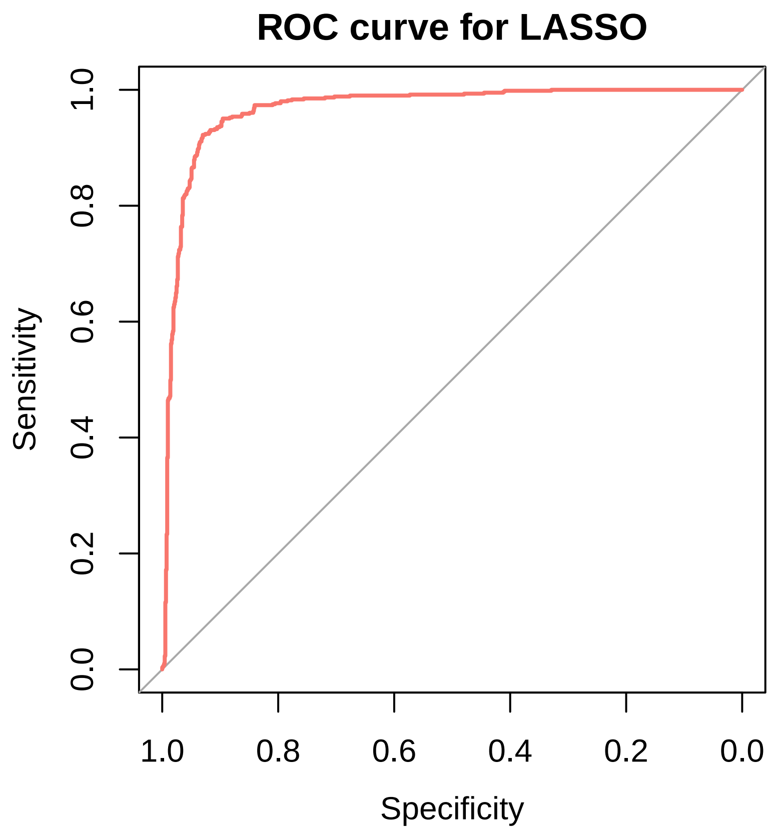
- Logistic regression fitted by maximum likelihood (IRWLS, glm).

```r
par(pty = "s")
mle_predict <- predict(mle, test, type = "response")

# roc(test$spam.01, mle_predict, plot = FALSE, legacy.axes = TRUE)
plot.roc((roc(test$spam.01, mle_predict,
              plot = FALSE, legacy.axes = TRUE)),
                col = "#F8766D",
                  main = "ROC curve for Logistic regression")
```



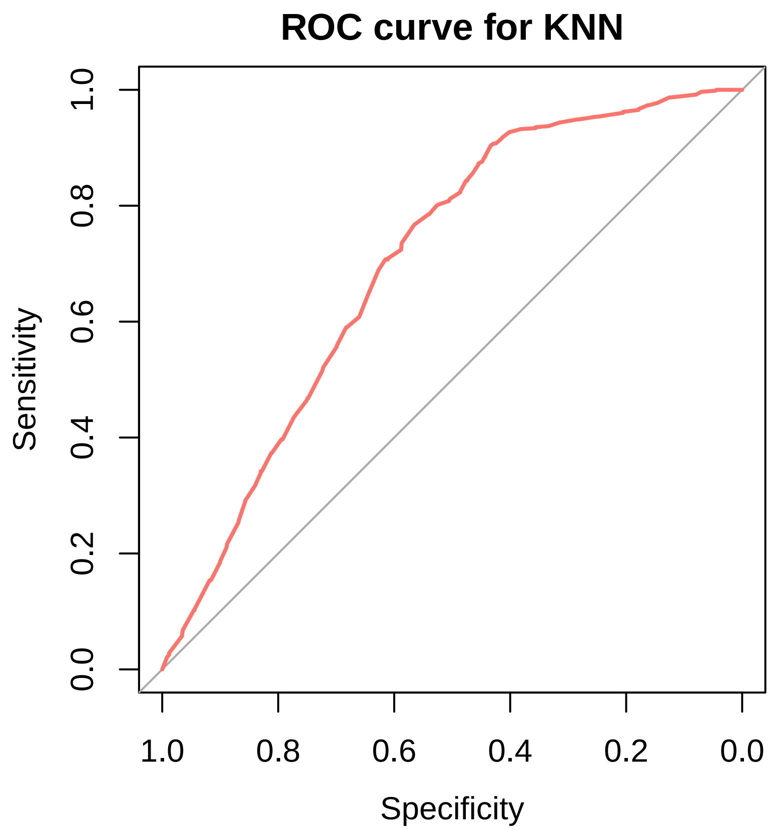
- Logistic regression fitted by Lasso (glment).

```r
par(pty = "s")
lasso_predict <- predict(lasso,newx =
                           as.matrix(select(test, !spam.01)), type = "response")

# roc(test$spam.01, lasso_predict, plot = FALSE, legacy.axes = TRUE)
plot.roc(roc(test$spam.01, lasso_predict, legacy.axes = TRUE), col = "#F8766D", main = '
```



- k-nn binary regression (you can use your own implementation or functions knn and knn.cv from the R package class).

```
par(pty = "s")
k_nn <- class::knn(train = train[1:57],
      test[1:57], k=100 , cl = as.factor(train$spam.01), prob = TRUE)

k_nn.cv <- class::knn.cv(train = train[1:57],
                         cl = as.factor(train$spam.01), prob = TRUE)
t<-attr(k_nn, "prob")

# roc(test$spam.01, t, plot = FALSE, legacy.axes = TRUE)
plot.roc((roc(test$spam.01, t, legacy.axes = TRUE)),
         col = "#F8766D", main = "ROC curve for KNN")
```



5. Compute also the misclassification rate for each rule when using the cut point c = 1/2.

We've created a custom function called misclassification to compute the misclassification rate. The function has 3 parameters, actual and predict which are the vectors of the actual and predicted values and are needed for the function to work and the third and optional parameter is the cut point, set by default to 1/2.

```
misclassification <- function(actual, predict, cut = 0.75){
  #format actual vector
  actual <- as_tibble(actual)
  colnames(actual)[1] <- "actual"
  #format predict vector
  predict <- as_tibble(predict)
  colnames(predict)[1] <- "predict"
  #compute misclassification rate
  df <- bind_cols(actual, predict) %>%
    mutate(predict = ifelse(predict >= cut,
```

```r
                            1,
                            0)) %>%
    mutate(error = ifelse(actual + predict == 1,
                           1,
                           0)) %>%
    summarise(misclassification_rate = sum(error)/nrow(.))

  cat("The misclassification rate is ",
    df$misclassification_rate*100,
    "% \n",
    sep = "")
}
```

```r
confusion_matrix <- function(actual, predict, cut = 0.5){
  #format actual vector
  actual <- as_tibble(actual)
  colnames(actual)[1] <- "actual"
  #format predict vector
  predict <- as_tibble(predict)
  colnames(predict)[1] <- "predict"
  #compute misclassification rate
  df <- bind_cols(actual, predict) %>%
    mutate(predict = ifelse(predict >= cut,
                            1,
                            0))

  matrix <- data.frame("Actual_0" =
  c(0, 0), "Actual_1" = c(0, 0), row.names = c("Predicted_0", "Predicted_1"))

  matrix[1, 1] <- nrow(filter(df, actual == 0 & predict == 0))
  matrix[1, 2] <- nrow(filter(df, actual == 0 & predict == 1))
  matrix[2, 1] <- nrow(filter(df, actual == 1 & predict == 0))
  matrix[2, 2] <- nrow(filter(df, actual == 1 & predict == 1))

  knitr::kable(matrix)
```

```
}
```

- Logistic regression fitted by maximum likelihood.

```r
confusion_matrix(actual = test$spam.01, predict = mle_predict)
```

	Actual_0	Actual_1
Predicted_0	871	59
Predicted_1	56	549

```r
misclassification(actual = test$spam.01, predict = mle_predict)
```

```
## The misclassification rate is 9.446254%
```

- Logistic regression fitted by Lasso.

```r
confusion_matrix(actual = test$spam.01, predict = lasso_predict)
```

	Actual_0	Actual_1
Predicted_0	875	55
Predicted_1	69	536

```r
misclassification(actual = test$spam.01, predict = lasso_predict)
```

```
## The misclassification rate is 12.443%
```

- k-nn binary regression.

```r
confusion_matrix(actual = test$spam.01, predict = as.numeric(levels(k_nn))[k_nn])
```

	Actual_0	Actual_1
Predicted_0	758	172
Predicted_1	246	359

```
cat("missclassification:",
    round((1 - length(which(k_nn == test$spam.01))
           /length(k_nn)),4)*100,"%")
```

```
## missclassification: 27.23 %
```

It's clear that logistic regression perform the best classification in this data set. When is compared the differences between the three models logistic regression get the best puntuation.

6. Compute l_{val} for each rule.

We've created a custom function called lval to compute the l_{val}. It needs two inputs, actual and predict, which are the vectors of actual and predicted values by a model.

```
lval <- function(actual, predict){
  y <- as.matrix(actual)
  x <- as.matrix(predict)
  left <- t(y)%*%(log(x))
  right <- t(1-y)%*%(log(1-x))
  lval <- (left+right)/nrow(y)
  cat("The lval is ", lval, ".", sep = "")
}
```

- Logistic regression fitted by maximum likelihood.

```
lval(actual = test$spam.01, predict = mle_predict)
```

```
## The lval is -0.2732304.
```

- Logistic regression fitted by Lasso.

```r
lasso_predict[587] <- 0.999999
lval(actual = test$spam.01, predict = lasso_predict)
```

```
## The lval is -0.276765.
```

- k-nn binary regression.

```r
lval(actual = test$spam.01, predict = t)
```

```
## The lval is -Inf.
```