ROC Curve Assignment

Ignasi Mane, Antoni Company & Chiara Barbi

3/2/2019

Introduction

Write a report that contains the results of the computations that you are asked to carry out below, as well as the explanation of what you are doing.

Libraries

```
library(glmnet)
library(caret)
library(class)
library(ROC632)
library(e1071)
```

Questions

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

```
spam <- read.table("spambase.data",sep=",")

spam.names <- c(read.table("spambase.names",sep=":",skip=33,nrows=53,as.is=TRUE)[,1],
                "char_freq_#",
                read.table("spambase.names",sep=":",skip=87,nrows=3,as.is=TRUE)[,1],
                "spam.01")

names(spam) <- spam.names
spam$spam.01 <- as.factor(spam$spam.01)
spam.01 <- spam$spam.01
```

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 trainig sample and 1/3 in the test sample, and that the same happens for NO SPAM e-mails

```
set.seed(5745)

n<-dim(spam)[1]
p<-dim(spam)[2]-1

spam.1 <- which(spam.01==1)
spam.0 <- which(spam.01==0)
n1 <- length(spam.1)
n0 <- length(spam.0)
```

```
spam.1.tr <- sort(sample(spam.1, round(2*n1/3)))
spam.0.tr <- sort(sample(spam.0, round(2*n0/3)))

spam.1.test <- setdiff(spam.1,spam.1.tr)
spam.0.test <- setdiff(spam.0,spam.0.tr)

spam.tr <- union(spam.1.tr,spam.0.tr)
spam.test <- union(spam.1.test,spam.0.test)

n.tr <- length(spam.tr)
n.test <- length(spam.test)

train <- spam[spam.tr,]
test <- spam[spam.test,]
```

3. Consider the following three classification rules:

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

Logistic regression fitted by Lasso (glment).

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

Use the trainig sample to fix the tunning parameters (when needed) and to estimate the model parameters (when needed)

3.1. Simple Glm with binary response

Using logit as a link, we fit a generalized linear model with binary response to the training set.

```
glm.spam.tr <- glm(spam.01~., data=train, family = binomial(link = logit))
```

3.2. Lasso Regression with binary response

We will split the data with the required variables into a matrix and a response vector. Then, using the training set, we will apply 10-fold Cross-Validation to chose the optimal value of λ^*, with which we will fit the final lasso regression model to obtain the value of $\hat{\beta}$.

```
y <- train$spam.01
x <- as.matrix(train[,!names(train) %in% c("spam.01")])

glm.spam.lasso <- cv.glmnet(y=y, x=x, family = "binomial", nfolds=10,
                            standardize=TRUE, intercept=TRUE, alpha = 1)

lambda.opt <- glm.spam.lasso$lambda.min
```

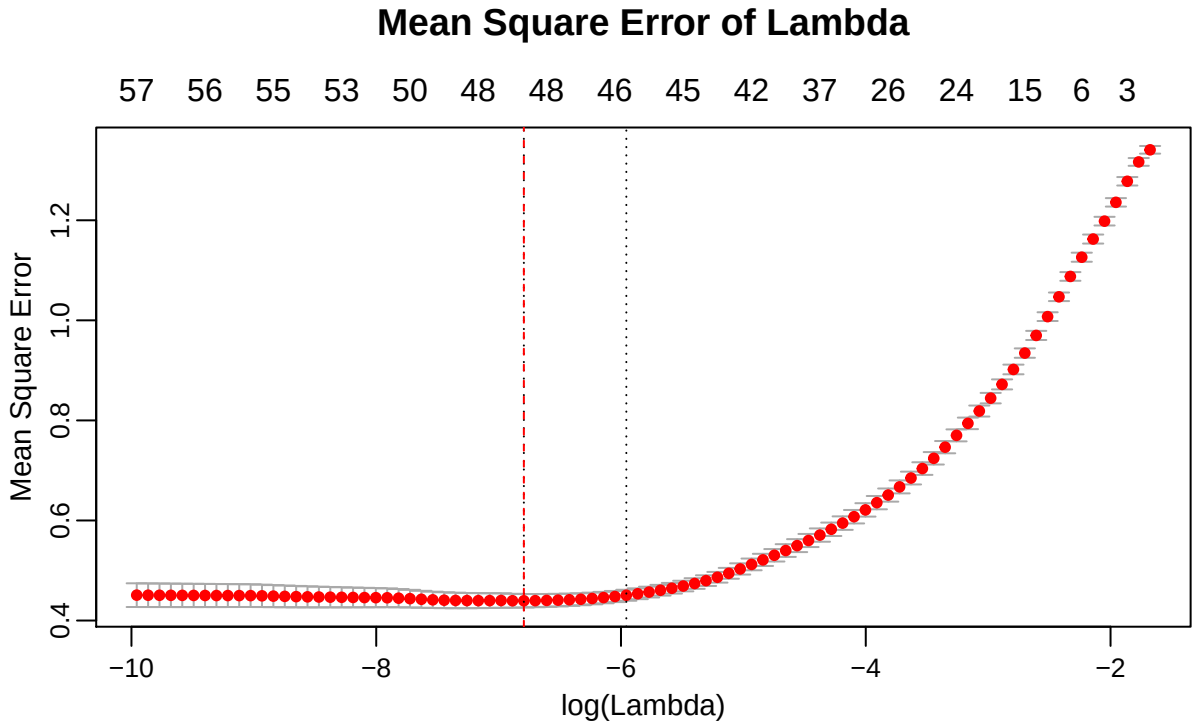
Once the 10-fold Cross Validation for the Lasso regression is done, it is seen that the lambda that minimizes the expected value of the mean square error is:

```
lambda.opt
```

```
## [1] 0.001120846
```

Graphically, this value can be represented (red line) in the following plot:

```
par(mar = c(4,3,5.5,1), mgp=c(1.5,0.5,0))
plot(glm.spam.lasso, ylab="Mean Square Error",
     cex.axis=0.8, cex.lab=0.9, main="Mean Square Error of Lambda")
abline(v=log(glm.spam.lasso$lambda.min),col=2,lty=2)
```



Finally, using the optimal value of λ^*, we apply the glmnet function to estimate the vector of $\hat{\beta}$ coefficients.

```
glm.lasso <- glmnet(y=y, x=x,lambda = lambda.opt, family = "binomial",
                    standardize = TRUE, intercept = TRUE, alpha = 1)
```

In the optimum, 48 of the 57 variables have non zero coefficients, so the degrees of freedom of the model is 48.

3.3. Knn algorithm

First, we will split the data with the required variables into a matrix and the response vector. Then, using the training set, we will apply 10-fold Cross-Validation to chose the optimal value of k^* from a vector of possible candidates defined as a sequence of positive integers from 1 to 50.

```
k <- seq(from=1, to=50, by=1)
y <- train$spam.01
x <- train[,!names(train) %in% c("spam.01")]
```

In order to fit the knn model for classification, we will use the function $train() $ from the caret library. According to its documentation, this function, which is one of the most widely used in machine learning, "sets up a grid of tuning parameters for a number of classification and regression routines, fits each model and calculates a re-sampling based performance measure". Basically it allows us to use fit a great variety of

algorithms such as knn, ridge regression.. from different packages glmnet, class... to the data controlling different re-sampling methods to calibrate the tuning parameters,

Therefore, using *train*() along with 10-fold cross-validation in the training sample is used to determine the optimal value k^* that minimizes the expected value of the "accuracy" -proportion of right classifications-.

```
knn.spam <- train(x=x,y=y, method="knn",
                  trControl = trainControl(method = "cv", number = 10),
                  metric="Accuracy", tuneGrid = expand.grid(k = k),
                  preProcess = c("center","scale"))
k.opt <- knn.spam$bestTune
```

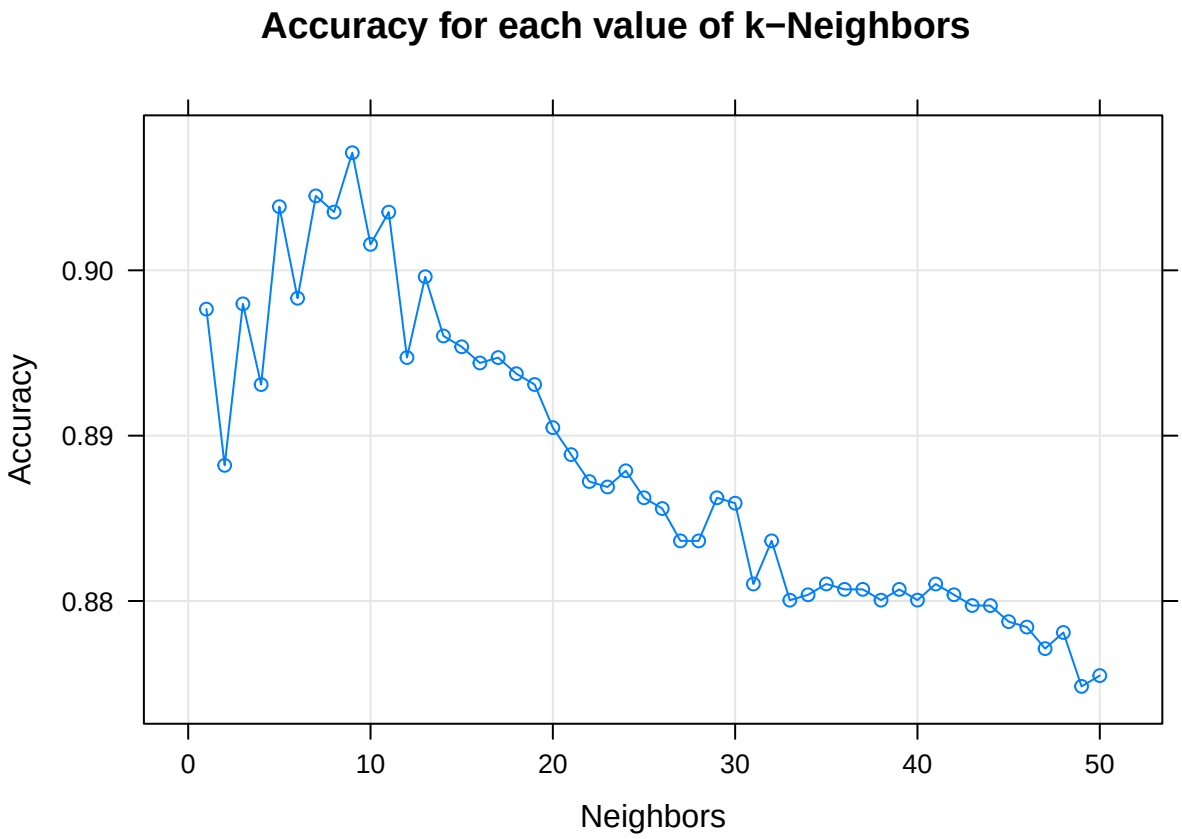
The optimal value k is the following:

```
print(k.opt)
```

```
##   k
## 9 9
```

It can be represented graphically in the following plot:

```
par(mar = c(4,3,5.5,1), mgp=c(1.5,0.5,0))
plot(knn.spam, ylab="Accuracy", xlab= "Neighbors",cex.axis=0.8, cex.lab=0.9, main="Accuracy for each va
```



Now, we fit the final knn model using the value k^* provided from the 10-fold cross-validation method applied in the train set .

```
knn <- train(x = x, y=y, method="knn", metric="Accuracy",
             tuneGrid = expand.grid(k = k.opt), preProcess = c("center","scale"))
```

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

Once we have trained the regression models, we proceed to compute the ROC to determine how good the regressions are when we use them to predict the response variable in the test set.

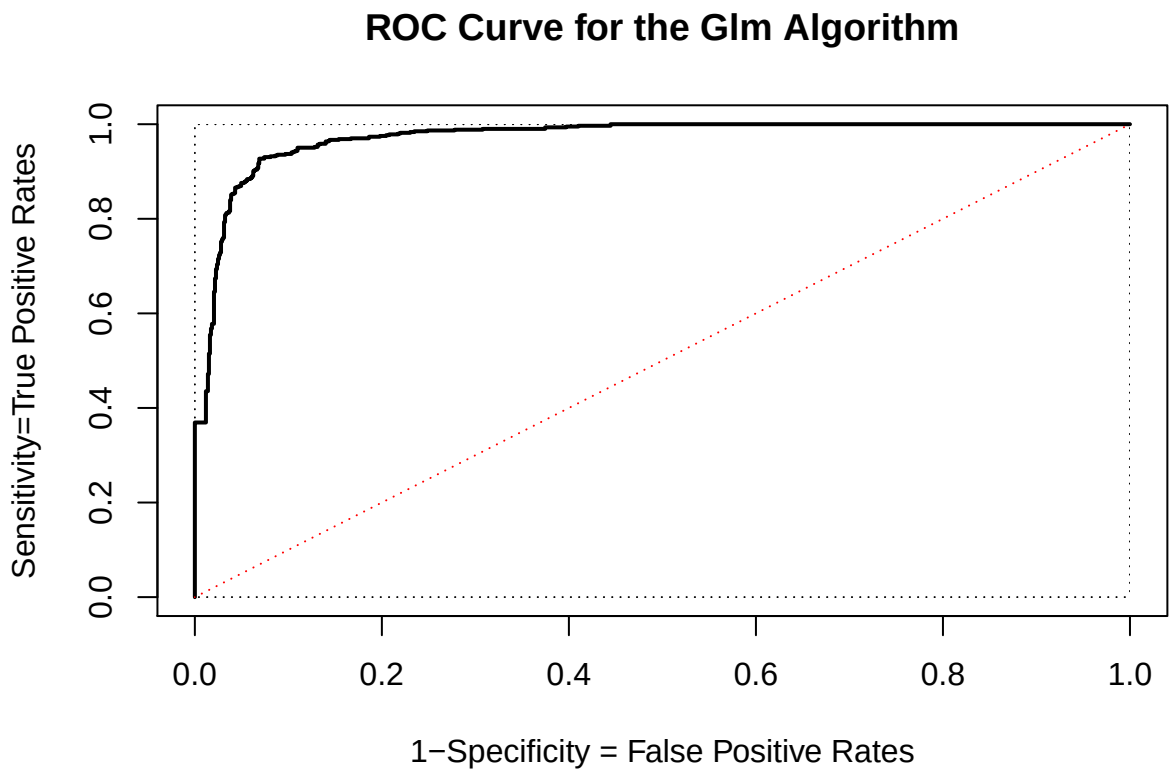
4.1. Simple Glm with binari response

We use the generalized line model to predict the response variable in the test set.

```
pred.glm.spam.test <- predict(glm.spam.tr, newdata = test, type="response")
```

In order to study the accuracy of the predictions, we analyse the ROC curve.

```
J <- 201
cut.points <- (0:J)/J
ROC.obj <- ROC(status=test$spam.01, marker=pred.glm.spam.test, cut.values=cut.points)
plot(ROC.obj$FP, ROC.obj$TP, ylab="Sensitivity=True Positive Rates",
     xlab="1-Specificity = False Positive Rates", type="s", lwd=2,
     main="ROC Curve for the Glm Algorithm")
clip(x1=0,x2=1,y1=0,y2=1)
abline(a=0, b=1,col=2,lty=3)
abline(v=0,col=1,lty=3)
abline(v=1,col=1,lty=3)
abline(h=1,col=1,lty=3)
abline(h=0,col=1,lty=3)
```



If we analyse the graphical results obtained, we can see that the accuracy is relay high. This affirmation relies on that the remaining area under the ROC curve is very large and on that the ROC curve is clearly above the line that represents random classification.

The value of the AUC is the following:

```
ROC.obj$AUC
```

```
## [1] 0.9682231
```

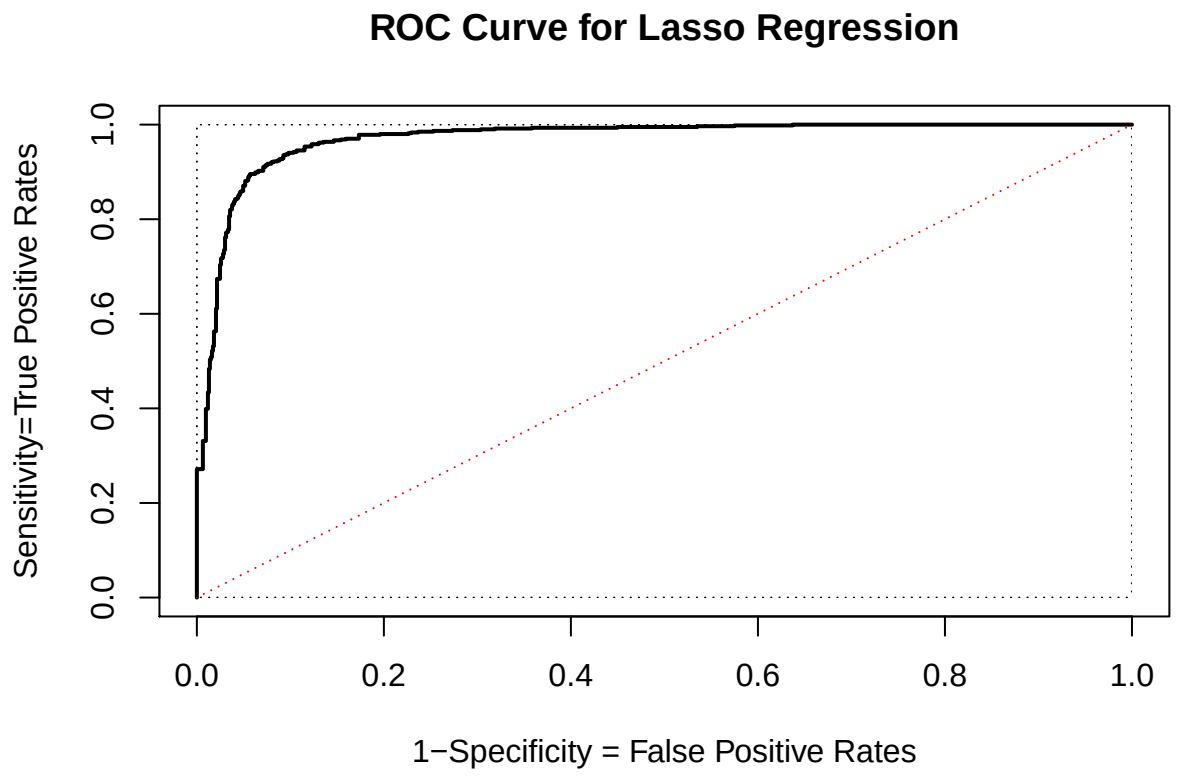
4.2. Lasso Regression with binari response

We use the lasso regression model to predict the response variable in the test set.

```
x <- as.matrix(test[,!names(test) %in% c("spam.01")])
pred.glm.lasso.test <- predict(glm.lasso, newx = x, type="response")
```

In order to study the accuracy of the predictions, we analyse the ROC curve.

```
J <- 201
cut.points <- (0:J)/J
ROC.obj <- ROC(status=test$spam.01, marker=pred.glm.lasso.test, cut.values=cut.points)
plot(ROC.obj$FP, ROC.obj$TP, ylab="Sensitivity=True Positive Rates",
     xlab="1-Specificity = False Positive Rates", type="s", lwd=2,
     main="ROC Curve for Lasso Regression")
clip(x1=0,x2=1,y1=0,y2=1)
abline(a=0, b=1,col=2,lty=3)
abline(v=0,col=1,lty=3)
abline(v=1,col=1,lty=3)
abline(h=1,col=1,lty=3)
abline(h=0,col=1,lty=3)
```



Like in the previous case, the ROC curve is well above the line that represents random classification. Again the area under curve is very large and close to the unit, the value that represents perfect rate of classification.

The AUC value is:

```
ROC.obj$AUC
```

```
## [1] 0.9681251
```

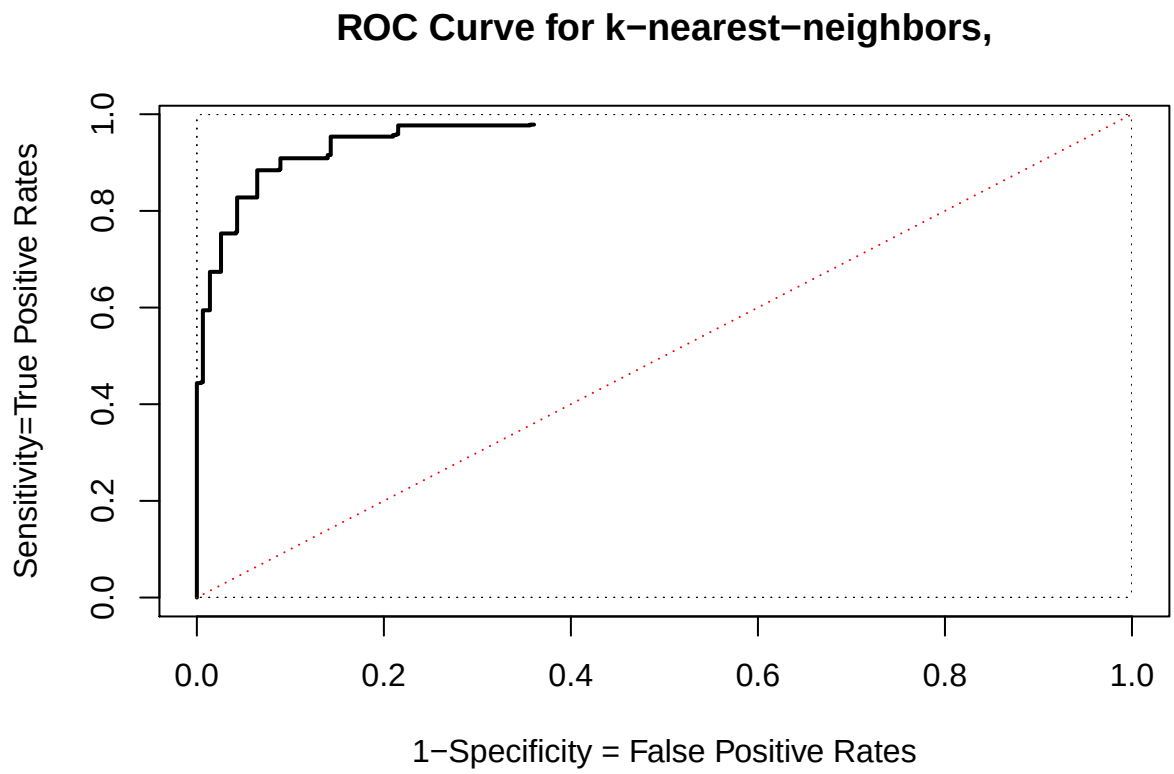
4.3. Knn algorithm

We use the knn algorithm to predict the response variable in the test set

```
x <- test[,!names(test) %in% c("spam.01")]
pred.glm.knn.test <- predict(knn, newdata = x, type="prob")
```

In order to study the accuracy of the predictions, we analyse the ROC curve.

```
J <- 201
cut.points <- (0:J)/J
ROC.obj <- ROC(status=test$spam.01, marker=pred.glm.knn.test$`1`, cut.values=cut.points)
plot(ROC.obj$FP, ROC.obj$TP, ylab="Sensitivity=True Positive Rates",
     xlab="1-Specificity = False Positive Rates", type="s", lwd=2,
     main="ROC Curve for k-nearest-neighbors,", xlim=c(0,1))
clip(x1=0,x2=1,y1=0,y2=1)
abline(a=0, b=1,col=2,lty=3)
abline(v=0,col=1,lty=3)
abline(v=1,col=1,lty=3)
abline(h=1,col=1,lty=3)
abline(h=0,col=1,lty=3)
```



Again, the ROC curve is well above the line that represents random classification. However, in this case the area under curve is slightly smaller than for the others two models.

The AUC value is:

```
ROC.obj$AUC
```

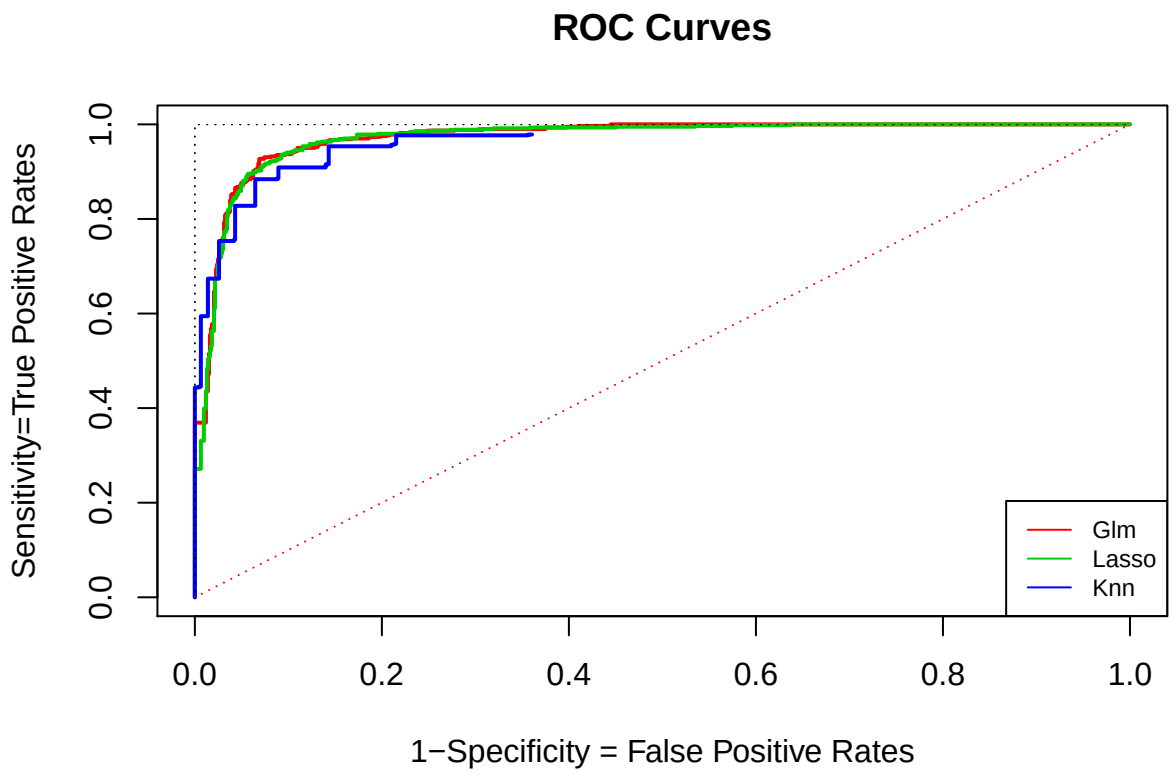
```
## [1] 0.953106
```

4.4. ROC Curves Comparaison

To compare the accuracy in the predictions of the three models used, we will plot the ROC curves from the three models all together.

```
J <- 201
cut.points <- (0:J)/J
ROC.obj.glm <- ROC(status=test$spam.01, marker=pred.glm.spam.test, cut.values=cut.points)
ROC.obj.lasso <- ROC(status=test$spam.01, marker=pred.glm.lasso.test, cut.values=cut.points)
ROC.obj.knn <- ROC(status=test$spam.01, marker=pred.glm.knn.test$`1`, cut.values=cut.points)

plot(ROC.obj.glm$FP, ROC.obj.glm$TP, ylab="Sensitivity=True Positive Rates",
     xlab="1-Specificity = False Positive Rates", type="s", lwd=2,
     main="ROC Curves", col=2)
lines(ROC.obj.lasso$FP, ROC.obj.lasso$TP, ylab="Sensitivity=True Positive Rates",
     xlab="1-Specificity = False Positive Rates", type="s", lwd=2, col=3)
lines(ROC.obj.knn$FP, ROC.obj.knn$TP, ylab="Sensitivity=True Positive Rates",
     xlab="1-Specificity = False Positive Rates", type="s", lwd=2, col=4)
legend(x="bottomright",c("Glm", "Lasso", "Knn"), cex = 0.75, lty = 1, col = c(2,3,4))
clip(x1=0,x2=1,y1=0,y2=1)
abline(a=0, b=1,col=2,lty=3)
abline(v=0,col=1,lty=3)
abline(h=1,col=1,lty=3)
```



When comparing the ROC curves from the predictions of the three models we can clearly see that the results obtained from the Glm and the Lasso predictions are nearly the same, whereas, the accuracy of the Knn model is a little bit lower.

We will compare the AUC values for the three models to quantify the difference in the accuracy.

```
sumry <- rbind(ROC.obj.glm$AUC,ROC.obj.lasso$AUC,ROC.obj.knn$AUC)
colnames(sumry) <- "AUC"
rownames(sumry) <- c("glm","lasso","knn")

print(sumry)
```

```
##              AUC
## glm   0.9682231
## lasso 0.9681251
## knn   0.9531060
```

We can see that the accuracy obtained for the predictions with the Glm and Lasso models are practically identical, if not identical. Also, we can see that the accuracy generated by the Knn model is inferior to the previous two, even though they are very high.

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

The Glm prediction has a missclassifaction rate of:

```
msr.glm <-table( spam$spam.01[spam.test], pred.glm.spam.test>.5 )
msr.glm
```

```
##
##     FALSE TRUE
##   0   882   47
##   1    76  528
```

```
paste("Missclassification rate of the Glm prediction is: ",round((msr.glm[1,2]+msr.glm[2,1])/sum(msr.glm
```

```
## [1] "Missclassification rate of the Glm prediction is:  0.0802"
```

The Lasso prediction has a missclassifaction rate of:

```
msr.lasso <- table( spam$spam.01[spam.test], pred.glm.lasso.test>.5 )
msr.lasso
```

```
##
##     FALSE TRUE
##   0   883   46
##   1    84  520
```

```
paste("Missclassification rate of the Lasso prediction is: ",round((msr.lasso[1,2]+msr.lasso[2,1])/sum(m
```

```
## [1] "Missclassification rate of the Lasso prediction is:  0.0848"
```

The Knn prediction has a missclassifaction rate of:

```
msr.knn <- table( spam$spam.01[spam.test], pred.glm.knn.test$`1`>.5 )
msr.knn
```

```
##
##     FALSE TRUE
```

```
##   0   870   59
##   1   104  500
```

```
paste("Missclassification rate of the Knn prediction is: ",round((msr.knn[1,2]+msr.knn[2,1])/sum(msr.kn
```

```
## [1] "Missclassification rate of the Knn prediction is:  0.1063"
```

```
msr <- matrix(0,ncol=2,nrow=3)
msr[,1] <- c("Glm", "Lasso", "Knn")
msr[,2] <- c(round((msr.glm[1,2]+msr.glm[2,1])/sum(msr.glm),4),
             round((msr.lasso[1,2]+msr.lasso[2,1])/sum(msr.lasso),4),
             round((msr.knn[1,2]+msr.knn[2,1])/sum(msr.knn),4))
colnames(msr) <- c("Model", "Missclassifiaction Rate")
msr
```

```
##      Model   Missclassifiaction Rate
## [1,] "Glm"   "0.0802"
## [2,] "Lasso" "0.0848"
## [3,] "Knn"   "0.1063"
```

If we compare the results of missclassifaction obtained in each model we can see that the models with the lowest missclassifaction rates are the Glm and the Lasso both with values close to 0.08. Among them, the model with the lowest missclassification rate corresponds to the Glm. Meanwhile, the model with the worst rate of missclassification corresponds to the Knn with a value close to 0.11.

6. Compute l_{val} for each rule.

This value is calculated using the validation set as follows:

$$\ell_{\text{val}}(g_S) = \frac{1}{m}\sum_{j=1}^{m}\left(y_j^v \log g_S(x_j^v) + (1-y_j^v)\log(1-g_S(x_j^v))\right)$$

This value can be understood as the log likelihood of the model. Thus, the larger the value the more plausible the model is.

Notice that that some issues arise when the model estimates that $g_S(x) = P(Y=1|X=x) = 1$ when the the real value of $Y = 0$ because the value of the log evaluated at 0 is minus infinity. To overcome this problem, we define a value epsilon as the minimum value of $g_S(x)$ subject to $g_S(x)$ is not equal to 1 or 0 and, then, establish that $g_S(x) = epsilon$ if $g_S(x) = 0$ and that $g_S(x) = 1 - epsilon$ if $g_S(x) = 1$

Glm model

We compute l_{val}:

```
pred.glm.spam.test <- predict(glm.spam.tr, newdata = test, type="response")

mean(as.numeric(as.character(test$spam.01))*log(pred.glm.spam.test) +
                  (1-as.numeric(as.character(test$spam.01)))*log(1-pred.glm.spam.test))
```

```
## [1] -0.2501307
```

Knn model

We compute l_{val}:

```
x <- test[,!names(test) %in% c("spam.01")]
pred.glm.knn.test <- predict(knn, newdata = x, type="prob")

filter <- !pred.glm.knn.test$`1` %in% c(0,1)
epsi <- min(pred.glm.knn.test$`1`[filter],1-pred.glm.knn.test$`1`[filter])

filt0 <- pred.glm.knn.test$`1`==0
filt1 <- pred.glm.knn.test$`1`==1

pred.glm.knn.test$`1`[filt0] <- epsi
pred.glm.knn.test$`1`[filt1] <- 1-epsi

mean(as.numeric(as.character(test$spam.01))*log(pred.glm.knn.test$`1`) +
                   (1-as.numeric(as.character(test$spam.01)))*log(1-pred.glm.knn.test$`1`))
```

```
## [1] -0.2670949
```

Lasso Regression

We compute l_{val}:

```
x <- as.matrix(test[,!names(test) %in% c("spam.01")])
pred.glm.lasso.test <- predict(glm.lasso, newx = x, type="response")

mean(as.numeric(as.character(test$spam.01))*log(pred.glm.lasso.test) +
                   (1-as.numeric(as.character(test$spam.01)))*log(1-pred.glm.lasso.test))
```

```
## [1] -0.244448
```