# Comparing Discriminant Rules

Marcel Pons Cloquells

December 5, 2020

## 1. Spam Dataset

The `Spam Database` consists of 57 variables and 4601 rows, each one corresponding to an e-mail message. The last column named `spam.01` is a factor denoting whether the email message is *spam* or *nonspam* (response variable in our study).

In order to proceed with the proposed classification techniques, the data is divided into two sets: 2/3 of the data is selected for the training set and the remaining 1/3 for the test set. Furthermore, since we want to keep the number of **spam emails** balanced between the two sets (proportion 0.39 in the whole dataset), 2/3 of spam emails are selected for the training set and 1/3 for the test set, following the same procedure for the **no-spam** e-mails.

## 2. Classification rules

### About the $\ell_{val}$ measure

In order to compute the $\ell_{val}$ measure for the three performed models, we have created the function `l_val()`[1] that computes this measure given the vector of probabilities for spam/no-spam ( using the `test_data` to predict them).

### 2.1. Logistic Regression fitted by Maximum Likelihood

A logistic regression model is performed using the `glm()` function with `family=binomial`. Since we are going to use the centered and standardized data for the *lasso* and *knn* models, we have chosen to do the same for the linear regression (although it could be performed without doing so), in this way the comparisons are going to be with the same scale for all the three models.

```
glm.spam <- glm(spam.01 ~ .,data = spam_train_s, family=binomial)
```

With the `glm.spam` model we predict the probability of a message to be spam for the unseen `spam_test` data. These probabilities are going to be used to plot the *ROC* and compute the *AUC* and $\ell_{val}$. By setting a cut point, all messages with higher probability than 1/2 are going to be considered **spam** (1) and **no-spam** (0) otherwise. We use this classification to build the confusion matrix and measure the misclassification rate.

```
# Predicting the test set
prob_pred <- predict(glm.spam, type = "response", newdata = spam_test_s[,-58])
y_pred = ifelse(prob_pred > 0.5, 1, 0)

# Building the Confusion Matrix
cm = table(spam_test[, 58], y_pred > 0.5)
```

[1] R code of the function is provided in the Annex

### 2.2. Logistic regression fitted by Lasso (glmnet)

With the lasso regression model the same procedure will be followed, but first the penalization factor lambda ($\lambda$) that reaches the best classification has to be found. To that end, we perform 10-fold CV using `cv.glmnet()` function.

```
cv.spam.lasso = cv.glmnet(x=as.matrix(spam_train[,-58]),
                          y=spam_train[,58],
                          alpha = 1,
                          nfolds=10,
                          family = "binomial")
```

From the cross validation, we decide to use the *lambda.1se* because the accuracy (Binomial Deviance in this case) with respect *lambda.min* does not look very diminished. Moreover, in this way we obtain a simplified model with fewer variables[2].

```
spam.lasso <- glmnet(x=as.matrix(spam_train[,-58]),
                     y=spam_train[,58],
                     alpha = 1,
                     family = "binomial",
                     intercept = TRUE,
                     standardize = TRUE,
                     lambda = cv.spam.lasso$lambda.1se)
```

### 2.3. K-nearest neighbours binary regression

For the K-nearest neighbours, we first need to tune the number of neighbours (hyperparameter $k$). To that end, we create a function `KNN.KCV` that performs the desired number of cross validations and returns the misclassification error for each considered number of neighbours[3].

```
MR <- KNN.KCV(spam_train, 5, neighbours) # vector of Ks
```

By perfroming a 5 cross validation, we conclude that considering 3 neighbours gives the lowest misclassification rate (in the validation). Therefore, we use $k = 3$ in `knn()` of the `class` library.

```
y_pred = knn(train = spam_train[, -58],
             test = spam_test[, -58],
             cl = spam_train[, 58],
             k = 3,
             prob = TRUE)
```

[2]The plot of the different values of Binomial Deviance for the considered lambdas is provided in the Annex
[3]The plot of the Misclassification Rate for the different Ks is provided in the Annex

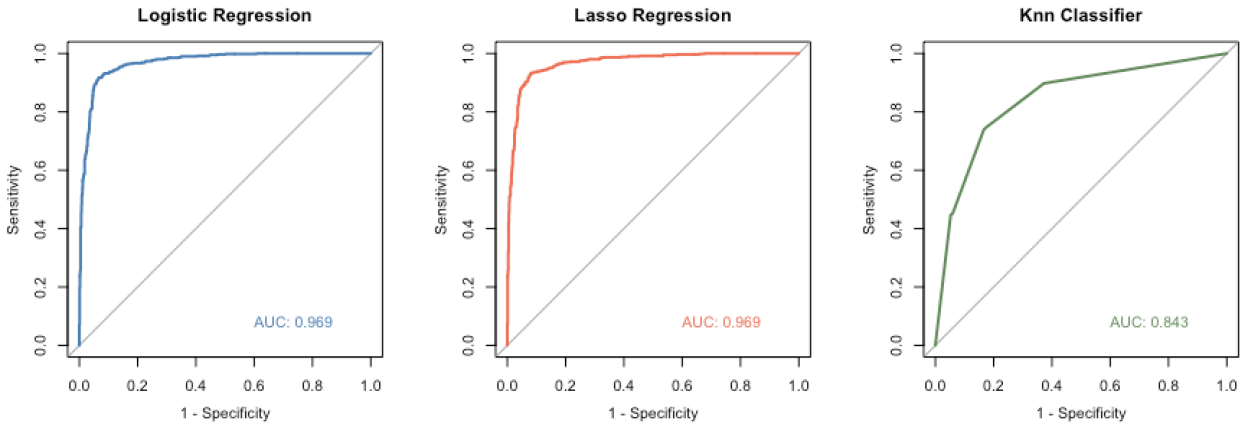
## 3. Comparison of Discriminant Rules

From the three models, the respective predictions on the test set data have been performed and for each one a confusion matrix is build to assess the number of emails correctly classified (*true negatives* + *true positives*) and incorrectly classified (*false negatives* + *false positives*). The misclassification rate of each model is obtained by the calculation of the proportion of incorrectly classified emails in all the predictions.

| | `Logistic Regression` | `Lasso Regression` | `Knn` |
|---|---|---|---|
| Misclassification Rate | `0.0737` | `0.0763` | `0.2048` |
| $\ell_{val}$ | `-0.2371` | `-0.2304` | `-5.9891` |

The model that gives the lowest missclassification rate is the Logistic regression performed with `glm()`, although it is very close to the rate achieved by Lasso Regression. Moreover, the $\ell_{val}$ measure of both models is almost identical too. It can be said that the two models have identical classification performance.

On the other hand, the classification performed by `Knn` is the worst one among the three models, with a misclassification rate of 0.2 and a lower $\ell_{val}$ measure.



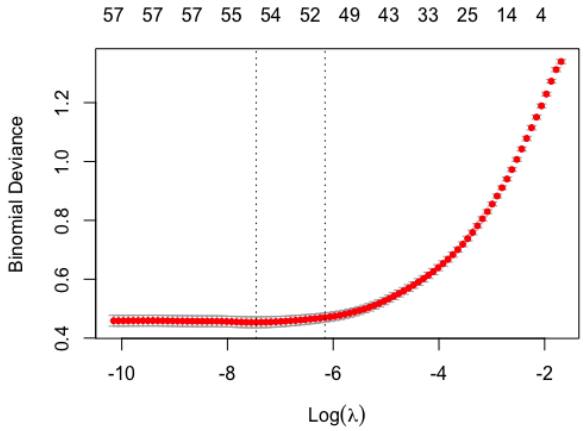

Having a look at the ROC curves and the AUC, the same scenario is encountered, Logistic and Lasso Regression are identical, with practically a perfect discrimination of emails; meanwhile Knn performs in a worse way than the other models.

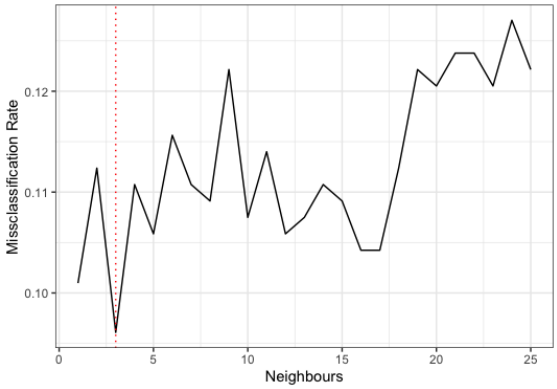
### Conclusions

To conclude, the best choice of model would be the Lasso Regression with `glmnet()`, because in addition to show a precise classification, it performs a feature selection not considering some of the variables that Logistic Regression does. Also, the addition of the penalization parameter $\lambda$ makes Lasso less prone to over-fitting.

# Annex

### Hyperparemeter tunning in Lasso and Knn




### $\ell_{val}$ function R code

```
l_val <- function(prob_pred){
  sum_l = 0
  for (i in 1:length(prob_pred)){
    iter = spam_test[i,58]*log(as.vector(prob_pred[i])+1e-6) +␣
 ↪(1-spam_test[i,58])*log(1-as.vector(prob_pred[i])+1e-6)
    if(is.nan(iter) == FALSE){
    sum_l = sum_l + iter
    }
  }
  l_val = sum_l/length(prob_pred)
  return(l_val)
}
```