Lab 1 - Group A14

Stefano Toffol (steto820), Andreas Stasinakis (andst745), Mim Kemal Tekin (mimte666)

24 November 2018

Contributions:

- Assignment 1 - Stefano Toffol;
- Assignment 2 - Mim Kemal Tekin;
- Assignment 3 - Andreas Stasinakis;

Assignment 1 - *by: Stefano Toffol*

Task 1.1

On the dataset `spambase.xlsx` the following functions are applied:

It results in a 50%/50% split of the data between train (Table 1.1.1) and test (Table 1.1.2).

Table 1.1.1: First lines of the train dataset.

Word1	Word2	Word3	...	Word46	Word47	Word48	Spam
0.00	0.00	0.00	...	0.00	0	0.00	1
0.00	0.00	0.00	...	0.00	0	0.00	1
0.00	0.00	0.00	...	0.00	0	0.00	1
0.00	0.00	0.00	...	0.00	0	1.49	0
0.00	0.00	0.19	...	1.72	0	0.00	0
0.68	0.17	0.51	...	0.00	0	0.00	0

Table 1.1.2: First lines of the test dataset.

Word1	Word2	Word3	...	Word46	Word47	Word48	Spam
0.47	0.31	0.47	...	0.00	0	0	0
0.49	0.28	0.40	...	0.00	0	0	1
0.00	0.00	0.28	...	0.00	0	0	0
0.00	0.00	0.28	...	0.86	0	0	0
0.46	0.31	0.46	...	1.91	0	0	0
0.47	0.31	0.47	...	0.00	0	0	1

Task 1.2

A `glm` model is computed on the train dataset, considering the column *Spam* as Y variable, while the frequency of the words are used as explanatory variables. The model belongs to the *binomial* family with probability parameter p_i, that is equivalent to $E[Y_i]$, and uses the *logistic* link function. We supposed the observations are distributed independently ($Y_i \backsim Bernoulli(p_i)$) and that $logit\,(p_i) = \beta_0 + \beta_1 \cdot X_1 + ... + \beta_n \cdot X_n$. In other words we're modelling the phenomena as follows:

$$log(\frac{p}{1-p}) = \beta_0 + \beta_1 \cdot X_{w1} + \beta_2 \cdot X_{w2} + ... + \beta_{48} \cdot X_{w48}$$

The predicted values will assume the following values:

$$\hat{Y_i} = \begin{cases} 1, & \text{if } p(Y_i = 1|X) > 0.5 \\ 0, & \text{otherwise} \end{cases}$$

The resulting confusion matrices are the following:

Table 1.2.1: Confusion matrix of the train dataset (threshold 0.5).

	Predicted values		
Real values	Non-spam	Spam	Frequencies
Non-spam	803	142	945
Spam	81	344	425
Frequencies	884	486	1370

Table 1.2.2: Confusion matrix of the test dataset (threshold 0.5).

	Predicted values		
Real values	Non-spam	Spam	Frequencies
Non-spam	791	146	937
Spam	97	336	433
Frequencies	888	482	1370

The two matrices are similar and the real data appear to be distributed equally among the two datasets (945 vs 937 *non-spam* mails). However the predictions on the train dataset, as one may expect, appears to perform slightly better overall: the model correctly classifies *spam* mails in rougthly 81 % of the times, whereas in the case of the train dataset the same percentage is almost 78 %. Nonetheless in both cases the models perform fairly well: the misclassifications rate for the train dataset is 0.1628, while in the test dataset is 0.1774.

Task 1.3

The question resamble the previous one, however this time the predictions will assume the following values:

$$\hat{Y}_i = \begin{cases} 1, & \text{if } p(Y_i = 1|X) > 0.9 \\ 0, & \text{otherwise} \end{cases}$$

The different threshold drastically change the outcome of the matrixes: now in both the predictions almost all the emails are classified as non-spam, resulting in a conseguent shift of the misclassification error. In both cases only one email has been uncorrectly classified as *spam*, while on the other hand more than 400, almost one third of the total data, has been labelled as *non-spam* when the opposite was true. The misclassifications rate rises substancially: is equal to 0.3066 in the train dataset and to 0.3124 in the test dataset.

Table 1.3.1: Confusion matrix of the train dataset (threshold 0.9).

	Predicted values		
Real values	Non-spam	Spam	Frequencies
Non-spam	944	1	945
Spam	419	6	425
Frequencies	1363	7	1370

Table 1.3.2: Confusion matrix of the test dataset (threshold 0.9).

	Predicted values		
Real values	Non-spam	Spam	Frequencies
Non-spam	936	1	937
Spam	427	6	433
Frequencies	1363	7	1370

Task 1.4

On the same train and test datasets is now applied a *Weighted k-Nearest Neighbor Classifier* to predict both the train and the test data.
Regarding the train dataset (Table 1.4.1), the `kknn` function performs slighltly better than the `glm` in classifing the *non-spam* mails (807 vs 803) while the opposite happens for *spam* mails (327 vs 344). Overall the misclassifications rate remains similar, being 0.1723 using the clustering method and 0.1628 applying the regression model.
On the other hand in the test dataset (Table 1.4.2) the performanses of the `kknn` algorithm drops drastically, resulting in more than 100 more misclassified units for both *spam* and *non-spam* emails if compared to the `glm`. The misclassification rate follows the same pattern, increasing to 0.3299 from the previous 0.1774 optained through `glm`.

Table 1.4.1: Confusion matrix on the train dataset (kk method - 30 neighbors).

	Predicted values		
Real values	Non-spam	Spam	Frequencies
Non-spam	807	138	945
Spam	98	327	425
Frequencies	905	465	1370

Table 1.4.2: Confusion matrix on the test dataset (kk method - 30 neighbors).

	Predicted values		
Real values	Non-spam	Spam	Frequencies
Non-spam	672	265	937
Spam	187	246	433
Frequencies	859	511	1370

Task 1.5

Using only one neighbor to define the clusters, the model will obviously overfit the data since the prediction of each observation will be based on the closest unit of the dataset, being strongly inflenced by the random error present in the phenomena.
As a matter of fact, in the train dataset the model perfectly discriminate between *spam* and *non-spam* mails (Table 1.5.1), whereas, when applied to the test dataset, the model makes several mistakes. The obtained confusion matrix (Table 1.5.2) is similar to the one computed setting `k`= 30, but with a small improvment in correctly predicting *spam* mails, balanced by a small decrease of correctly classified *non-spam* mails. In this case the misclassification error is equal to 0.346, pretty close to the one resulting in Question 4.

Table 1.5.1: Confusion matrix on the train dataset (kk method - 1 neighbor).

	Predicted values		
Real values	Non-spam	Spam	Frequencies
Non-spam	945	0	945
Spam	0	425	425
Frequencies	945	425	1370

Table 1.5.2: Confusion matrix on the test dataset (kk method - 1 neighbor).

	Predicted values		
Real values	Non-spam	Spam	Frequencies
Non-spam	640	297	937
Spam	177	256	433
Frequencies	817	553	1370

Assignment 3 - *by: Mim Kemal Tekin*

Task 3.1

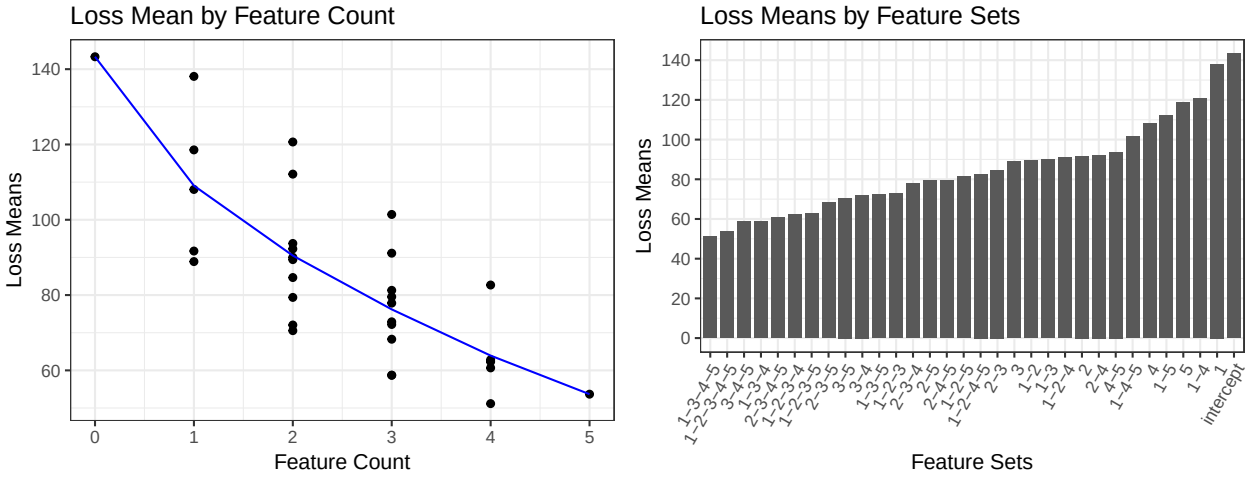
Implement an R function that performs feature selection (best subset selection) in linear regression by using k-fold cross-validation without using any specialized function like lm() (use only basic R functions). Your function should depend on:
- X: matrix containing X measurements.
- Y: vector containing Y measurements.
- Nfolds: number of folds in the cross-validation.
You may assume in your code that matrix X has 5 columns. The function should plot the CV scores computed for various feature subsets against the number of features, and it should also return the optimal subset of features and the corresponding cross-validation (CV) score. Before splitting into folds, the data should be permuted, and the seed 12345 should be used for that purpose.

Cross-validation algorithm is a model selection algorithm. It splits whole data to k pieces. It uses all pieces as a test data once while other rest of the data as a train data. By this way, it collects model prediction errors for k combinations of data by using specific loss function for model and return their mean.
In this task, we try to choose optimal features for our regression model and we use MSE as a loss function. So we will create different models with all combinations of the features, after this we will calculate loss means of them to find the best model.
Function does respectively shuffling data, iteration of all feature combinations and in this loop it uses k-fold cross-validation algorithm to get the mean of losses. In k-fold cross-validation algorithm, we set a seed in order to get always same indexes to split data. By this, we are able to compare errors of the models which use the same portion for all folds.

The code of CV function is at appendix.

Task 3.2

Test your function on data set swiss available in the standard R repository:
- Fertility should be Y
- All other variables should be X
- Nfolds should be 5
Report the resulting plot and interpret it. Report the optimal subset of features and comment whether it is reasonable that these specific features have largest impact on the target.




```
## [1] "Optimal Features:"
```

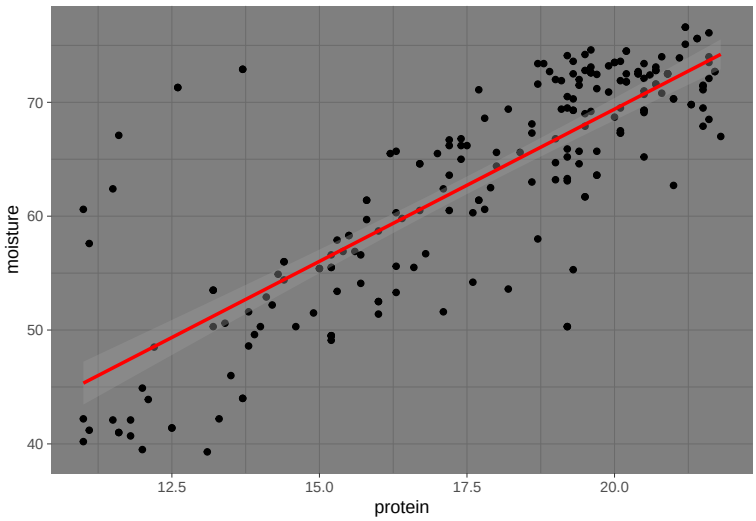
```
## [1] "Agriculture"      "Education"        "Catholic"
## [4] "Infant.Mortality"
```

In this task we used "swiss" dataset which is inside R. This dataset about Standardized fertility measure and socio-economic indicators for each of 47 French-speaking provinces of Switzerland at about 1888. Dataset has 5 predictor which are respectively "Agriculture" (1), "Examination" (2), "Education" (3), "Catholic" (4), "Infant.Mortality" (5).

The plots above, we can see loss mean changes by feature count in left one and loss means of all combinations in right one. We can consider by examining left plot, generally increasing feature count decreases loss means, but we catch the optimal value (the lowest loss mean) in the models which have 4 feature.
We can see this optimal combination is "1-3-4-5" columns in right plot. And the feature names are specified as "Agriculture", "Education", "Catholic" and "Infant.Mortality".
When we think about this results and examine closely right plot, we can see lonely "Education" feature is also gives better predictions than the others and it is also in the best pairs which are 3-5 and 3-4. When we think about real life, "Education" is really effective variable for fertility measurement. Another fact that we can see in this plot, while "Agriculture" has the worst loss mean when we use it alone, we can see it is almost inside all the feature groups of lowest loss mean. So we can conclude this as "Agriculture" is also an effective feature when we use it other effective features.

Assingment 4 - *by: Andreas Stasinakis*

Task 4.1



From the plot, we can assume that the data are described well by this linear model. There are some outliers, for example left in the corner creating a cluster, which increase the bias of the model, but in general it fits well. In some points, the variance is quite high, as a result the MSE may be effected and increased.

Task 4.2

Probabilistic model :

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \varepsilon.$$

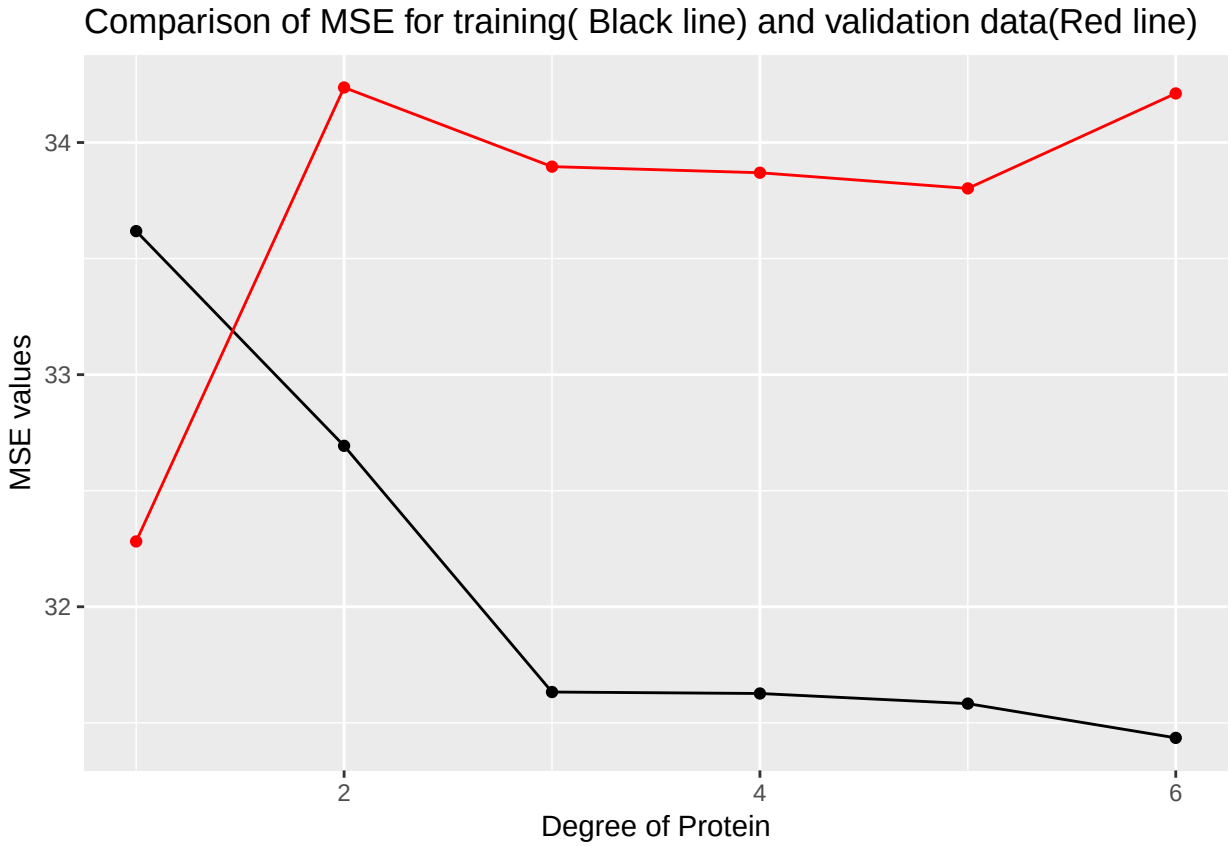
MSE criterion :

MSE criterion help us understand if our data fit well in the model we selected. The formula which is used for the calculation of MSE is :

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

From the formula, it is obvious that the lower MSE's value is, the closer we are to find the line of best fit. Of course it is impossible to have $MSE = 0$ in order to have perfect fit, but we can use MSE to compare different models and choose the one with the lowest MSE.

Task 4.3



From the plot, we can observe that while i is increasing, the MSE for the training data(black line) is reducing significant but the MSE for the validation data is increasing. Therefore, for the training data, the best model is the last one when for the validation data is the first one. It can be also said that while we are doing the model more complicated, we increase the probability overfitting. To be more specific, lets think about the following model : $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \varepsilon$. All independent variables x^2, x^3 are depend on variable x, which means that the higher n (power of variable x) the more biased will be our model. So the MSE values we have in the plot is an expected result, but we have to choose the model in respect to the validation data. So the model with the lowest MSE, so the best one, for the validation data is the simple linear model

$$y = \beta_0 + \beta_1 x + \varepsilon.$$

.

For the bias - variance tradeoff, it is good to mention the following definitions :

i) *Bias* captures the difference between the true values and the predicted values.
ii) *Variance* is the difference between data sets.

So our goal choosing a model is the one with the lowest bias and variance. We can reduce the bias by choosing a more complex model but the problem is that the more complex model the higher variance. That is the reason why we have to choose a model with balance between bias and variance.

For the plot above, it is clear that making our model more complex, reduces the bias *but* one the other hand the last model has a very high variance value. We can observe also that the MSE of the last model for the training data is around 31.3 while for the validation is approximately 34.5. That means that the model is overfitting because it may fit the training data well but the predictions have high variance.

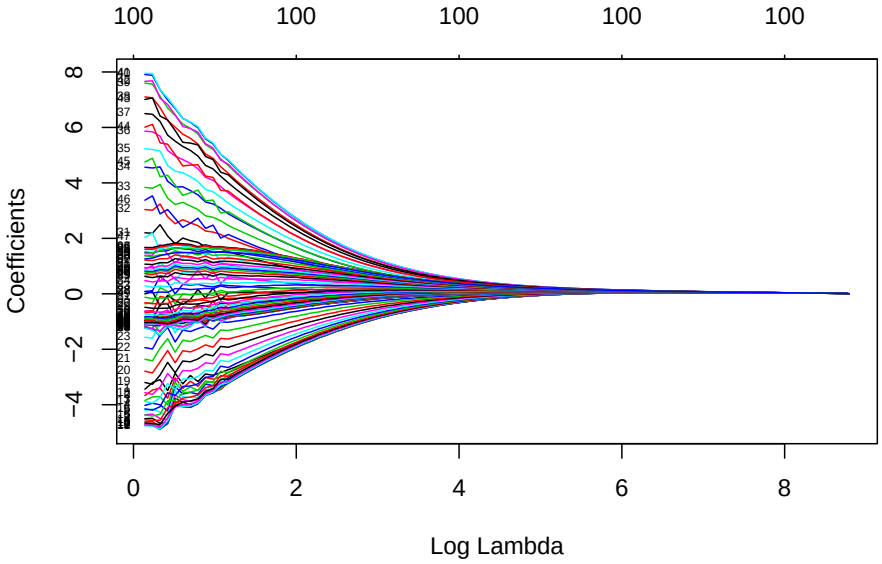
Task 4.4

```
## $number_of_variable_selected
## [1] 63
```

In this question, we perform variable selection which means that we eliminate all the independent variables we do not need for the prediction. More specific, 37 out of 100 predictors are "unnecessary" who make our model more complex and they do not have any strong dependency with the target value. Therefore, using stepAIC, we keep only 63 of them.

Task 4.5

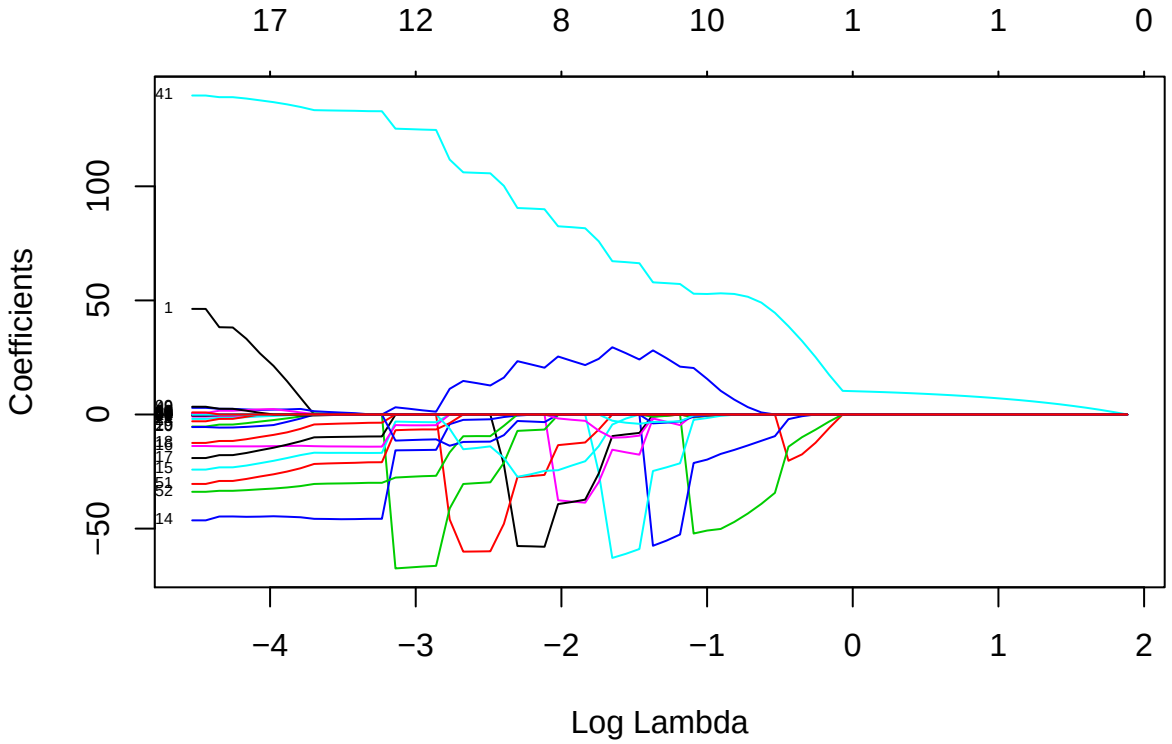
The plot of the Ridge regression : Log(Lambda) Vs Coefficients



As mentioned before we have to choose a model which not only is low biased but its variance is also low. We can achieve that using some regularization methods such as the Ridge regression. The concept of ridge regression is that, especially when we do not have a huge amount of data, we introduce a small amount of bias in our model as a result the variance of the model is reduced. More specific, we fit the data in a slightly worse fit, but with a lower variance which means more accurate predictions. Finally, ridge regression does not remove any variable but minimize the beta coefficients.

In this example, we can not say which λ is the best because we need the graph between the error vs the λ. From the above plot, it is clear that while λ increases, until λ is approximately equal to 3, the coefficients go asymptotically to zero. After this value of λ, the coefficients will continue to be near 0, *but* they never reach 0.

Task 4.6

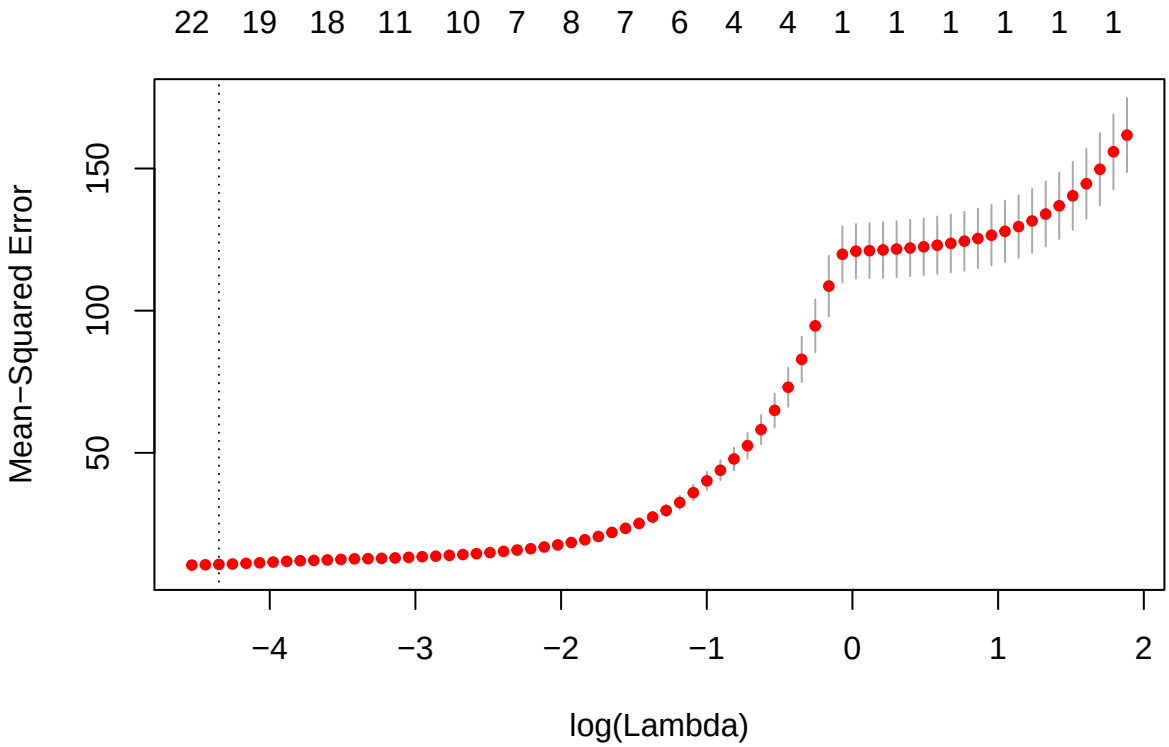


Another regularization method is LASSO regression. LASSO regression is similar to Ridge regression but they have a very important difference. As mentioned before, Ridge regression does not remove variables, it just shrinks their coefficients. In contrary, LASSO regression shrinks the coefficients and after a specific value of *lambda*, eliminates all the useless predictors. Finally, which of them is a better methods depends only from the data. If the data has many unnecessary variables, is better to use LASSO, while if most of the variables are important for the prediction, we should choose Ridge. So in the plot above, as λ increases, the coefficients reach zero. As a result, the higher λ, the less coefficients the model has because LASSO removes the variables with coefficient equal to zero.

Task 4.7

```
## $Optimal_value
## [1] 0
```

The optimal value of λ is 0. This means that the best model is the simple linear regression without removing any variable, so the number of variable selected is 100.



From the plot, we can conclude that the best model is the linear one because it has the lowest MSE. When λ goes to 0, the $limlog(\lambda)$ is going asymptotically to $-\infty$. Therefore, despite the fact that we can not see it in the plot, we have the lowest MSE and this is the reason why the simple linear model is the best for the data. Moreover, while λ increases, the MSE also increases. We can also mention that if we want a less complex model we can choose one while $log(\lambda)$ is less than -2 because the differences between the MSEs are not so important. After $log(\lambda)$ is greater than -2, the MSEs are increasing exponentially until 0, when they continue increasing exponentially with a different exponentially function.

Task 4.8

For the same data, in the question 4.4 the best model include only 63 predictors out of 100, while in 4.7 the best model does not remove any of the independent variables. We have this difference because in 4.4 we use stepAIC function in a linear model while in 4.7 we have a LASSO regression.

StepAIC selects the model based on Akaike Information Criteria(AIC). The goal is to find the model with the smallest AIC by removing or adding variables in model. This causes bad results when it comes to the test data, as a result the model does not generalize well. On the other hand, LASSO include the penalization which leads to more accurate predictions.

Appendix

```r
knitr::opts_chunk$set(echo = F, warning = F, message = F, error = F)

# Load libraries
library(readxl)
library(kableExtra)
library(kknn)
library(ggplot2)
library(dplyr)
library(MASS)
library(glmnet)


# ------------------------------------------------------------------------------
# A1 - Q1
# ------------------------------------------------------------------------------

# Read data
data <- read_xlsx("../spambase.xlsx")


# Split in train-test
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]


# Kable train
kable(data.frame(head(train[,1:3]), "..."=rep("...", 6), head(train[,46:49])),
      "latex", booktabs = T, align = "c",
      caption = "Table 1.1.1: First lines of the train dataset.") %>%
  kable_styling(latex_options = "hold_position", font_size = 6)


# Kable test
kable(data.frame(head(test[,1:3]), "..."=rep("...", 6), head(test[,46:49])),
      "latex", booktabs = T, align = "c",
      caption = "Table 1.1.2: First lines of the test dataset.") %>%
  kable_styling(latex_options = "hold_position", font_size = 6)


# ------------------------------------------------------------------------------
# A1 - Q2
# ------------------------------------------------------------------------------

# Set Y as factor
train$Spam <- as.factor(train$Spam)
test$Spam <- as.factor(test$Spam)

# Fit model
```

```r
fit1_glm <- glm(Spam ~ ., family = binomial(link = "logit"), data = train)

# Predict on the train dataset
predicted_train1 <- predict(fit1_glm, type = "response")
predicted_train1[predicted_train1>0.5] <- 1
predicted_train1[predicted_train1<=0.5] <- 0

# Predict on the test dataset
predicted_test1 <- predict(fit1_glm, newdata = test, type = "response")
predicted_test1[predicted_test1>0.5] <- 1
predicted_test1[predicted_test1<=0.5] <- 0


# Confusion matrix train
temp <- table(train$Spam, predicted_train1)
kable(data.frame(
      c("Non-spam", "Spam", "Frequencies"), c(temp[1:2], sum(temp[1:2])),
      c(temp[3:4], sum(temp[3:4])), c(sum(temp[c(1,3)]), sum(temp[c(2,4)]), 1370)),
    col.names = c("Real values", "Non-spam", "Spam", "Frequencies"),
    "latex", booktabs = T, align = "c",
    caption = "Table 1.2.1: Confusion matrix of the train dataset  (threshold 0.5).") %>%
  add_header_above(c(" " = 1, "Predicted values" = 2, " " = 1)) %>%
  column_spec(c(1,3), border_right = T) %>%
  row_spec(2, hline_after = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 8)


# Confusion matrix test
temp <- table(test$Spam, predicted_test1)
kable(data.frame(
      c("Non-spam", "Spam", "Frequencies"), c(temp[1:2], sum(temp[1:2])),
      c(temp[3:4], sum(temp[3:4])), c(sum(temp[c(1,3)]), sum(temp[c(2,4)]), 1370)),
    col.names = c("Real values", "Non-spam", "Spam", "Frequencies"),
    "latex", booktabs = T, align = "c",
    caption = "Table 1.2.2: Confusion matrix of the test dataset (threshold 0.5).") %>%
  add_header_above(c(" " = 1, "Predicted values" = 2, " " = 1)) %>%
  column_spec(c(1,3), border_right = T) %>%
  row_spec(2, hline_after = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 8)


# ------------------------------------------------------------------------------
# A1 - Q3
# ------------------------------------------------------------------------------

# Predict on the train dataset
predicted_train2 <- predict(fit1_glm, type = "response")
predicted_train2[predicted_train2>0.9] <- 1
predicted_train2[predicted_train2<=0.9] <- 0

# Predict on the test dataset
predicted_test2 <- predict(fit1_glm, newdata = test, type = "response")
predicted_test2[predicted_test2>0.9] <- 1
```

```r
predicted_test2[predicted_test2<=0.9] <- 0


# Confusion matrix train
temp <- table(train$Spam, predicted_train2)
kable(data.frame(
      c("Non-spam", "Spam", "Frequencies"), c(temp[1:2], sum(temp[1:2])),
      c(temp[3:4], sum(temp[3:4])), c(sum(temp[c(1,3)]), sum(temp[c(2,4)]), 1370)),
    col.names = c("Real values", "Non-spam", "Spam", "Frequencies"),
    "latex", booktabs = T, align = "c",
    caption = "Table 1.3.1: Confusion matrix of the train dataset (threshold 0.9).") %>%
  add_header_above(c(" " = 1, "Predicted values" = 2, " " = 1)) %>%
  column_spec(c(1,3), border_right = T) %>%
  row_spec(2, hline_after = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 8)


# COnfusion matrix test
temp <- table(test$Spam, predicted_test2)
kable(data.frame(
      c("Non-spam", "Spam", "Frequencies"), c(temp[1:2], sum(temp[1:2])),
      c(temp[3:4], sum(temp[3:4])), c(sum(temp[c(1,3)]), sum(temp[c(2,4)]), 1370)),
    col.names = c("Real values", "Non-spam", "Spam", "Frequencies"),
    "latex", booktabs = T, align = "c",
    caption = "Table 1.3.2: Confusion matrix of the test dataset (threshold 0.9).") %>%
  add_header_above(c(" " = 1, "Predicted values" = 2, " " = 1)) %>%
  column_spec(c(1,3), border_right = T) %>%
  row_spec(2, hline_after = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 8)


# ------------------------------------------------------------------------------
# A1 - Q4
# ------------------------------------------------------------------------------

kk_classifier_train30 <- kknn(Spam ~ ., train, train, k = 30)
kk_classifier_test30 <- kknn(Spam ~ ., train, test, k = 30)


# Confusion matrix train
temp <- table(train$Spam, kk_classifier_train30$fitted.values)
kable(data.frame(
      c("Non-spam", "Spam", "Frequencies"), c(temp[1:2], sum(temp[1:2])),
      c(temp[3:4], sum(temp[3:4])), c(sum(temp[c(1,3)]), sum(temp[c(2,4)]), 1370)),
    col.names = c("Real values", "Non-spam", "Spam", "Frequencies"),
    "latex", booktabs = T, align = "c",
    caption = "Table 1.4.1: Confusion matrix on the train dataset
    (kk method - 30 neighbors).") %>%
  add_header_above(c(" " = 1, "Predicted values" = 2, " " = 1)) %>%
  column_spec(c(1,3), border_right = T) %>%
  row_spec(2, hline_after = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 8)
```

```r
# Confusion matrix test
temp <- table(test$Spam, kk_classifier_test30$fitted.values)
kable(data.frame(
      c("Non-spam", "Spam", "Frequencies"), c(temp[1:2], sum(temp[1:2])),
      c(temp[3:4], sum(temp[3:4])), c(sum(temp[c(1,3)]), sum(temp[c(2,4)]), 1370)),
    col.names = c("Real values", "Non-spam", "Spam", "Frequencies"),
    "latex", booktabs = T, align = "c",
    caption = "Table 1.4.2: Confusion matrix on the test dataset
    (kk method  - 30 neighbors).") %>%
  add_header_above(c(" " = 1, "Predicted values" = 2, " " = 1)) %>%
  column_spec(c(1,3), border_right = T) %>%
  row_spec(2, hline_after = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 8)


# ------------------------------------------------------------------------------
# A1 - Q5
# ------------------------------------------------------------------------------

kk_classifier_train1 <- kknn(Spam ~ ., train, train, k = 1)
kk_classifier_test1 <- kknn(Spam ~ ., train, test, k = 1)


# Confusion matrix train
temp <- table(train$Spam, kk_classifier_train1$fitted.values)
kable(data.frame(
      c("Non-spam", "Spam", "Frequencies"), c(temp[1:2], sum(temp[1:2])),
      c(temp[3:4], sum(temp[3:4])), c(sum(temp[c(1,3)]), sum(temp[c(2,4)]), 1370)),
    col.names = c("Real values", "Non-spam", "Spam", "Frequencies"),
    "latex", booktabs = T, align = "c",
    caption = "Table 1.5.1: Confusion matrix on the train dataset
    (kk method - 1 neighbor).") %>%
  add_header_above(c(" " = 1, "Predicted values" = 2, " " = 1)) %>%
  column_spec(c(1,3), border_right = T) %>%
  row_spec(2, hline_after = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 8)


# Confusion matrix test
temp <- table(test$Spam, kk_classifier_test1$fitted.values)
kable(data.frame(
      c("Non-spam", "Spam", "Frequencies"), c(temp[1:2], sum(temp[1:2])),
      c(temp[3:4], sum(temp[3:4])), c(sum(temp[c(1,3)]), sum(temp[c(2,4)]), 1370)),
    col.names = c("Real values", "Non-spam", "Spam", "Frequencies"),
    "latex", booktabs = T, align = "c",
    caption = "Table 1.5.2: Confusion matrix on the test dataset
    (kk method  - 1 neighbor).") %>%
  add_header_above(c(" " = 1, "Predicted values" = 2, " " = 1)) %>%
  column_spec(c(1,3), border_right = T) %>%
  row_spec(2, hline_after = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 8)

####################
```

```
##### TASK 3.1 #####
####################
library(ggplot2)
library(dplyr)
# function that calculates coefficients
calculate_coefficients = function(Y, X){
  X = as.data.frame(X)
  Xx = cbind(data.frame(intercept = rep(1, nrow(X))), X)
  Xx = as.matrix(Xx)
  coef_matrix = solve(t(Xx) %*% Xx) %*% t(Xx) %*% Y
  return(coef_matrix)
}
# function that predict values by coef_matrix and test values
predict_values = function(coef_matrix, X){
  X = as.data.frame(X)
  Xx = cbind(data.frame(intercept = rep(1, nrow(X))), X)
  Xx = as.matrix(Xx)
  Y_hat = Xx %*% coef_matrix
  return(Y_hat)
}
# function that calculate loss
calculate_loss = function(Y_actual, Y_estimated){
  loss = mean((Y_actual - Y_estimated)^2)
  return(loss)
}
CV = function(X, Y, Nfolds){
  # k-fold function
  k_fold = function(X, Y, N){
    # When we select only one feature in X (column),
    # R convert df to vector automatically
    # We have to convert it df again (this also used in other funcs)
    X = as.data.frame(X)
    # store length of our data
    n = length(Y)
    # get indexes of our data
    all_indexes = 1:n
    # create temporary variables
    loss_values = c()
    set.seed(12345)

    # iterate all combinations
    for(i in 1:N){
      # divide data into training and test sets
      # find the test ids, other ids will be training observations
      id = sample(all_indexes, floor(n/N))
      all_indexes = all_indexes[!all_indexes %in% id]
      test = X[id,]
      train = X[-id,]

      # get actual values
      test_actual = Y[id]
      train_actual = Y[-id]
```

```
    # train our linear regression
    coef_mat = calculate_coefficients(train_actual, train)

    # predict values
    predicts = predict_values(coef_mat, test)

    # calculate loss
    loss = calculate_loss(Y_actual = test_actual, Y_estimated = predicts)

    # store loss
    loss_values = c(loss_values, loss)
  }
  # print(loss_values[1])
  return(mean(loss_values))
  # return(loss_values)
}

# store length of our data
n = length(Y)
# get indexes of our data
all_indexes = 1:n

# shuffle data
set.seed(12345)
id = sample(all_indexes, n)
X = as.data.frame(X[id,])
Y = Y[id]

# create temporary variables
loss_values = c()
feature_sets = c()
counter = 1

# in this assignment feature_qty is case specific.
# we know how many feature we have
feature_qty = 5
feature_indexes = 1:feature_qty

# iterate all combinations of features
for(i in 0:(2^feature_qty-1)){
  # find the combination as binary string
  cmb = intToBits(i)
  # select features
  selected_features = feature_indexes[cmb[1:feature_qty]==01]
  # run k-fold cross validation for selected features
  loss = k_fold(as.data.frame(X[,selected_features]), Y, Nfolds)
  # store loss value and feature sets
  loss_values = c(loss_values, loss)
  if(i==0)
    feature_sets = c(feature_sets, "intercept")
  else
    feature_sets = c(feature_sets, paste(selected_features, collapse = "-"))
}
```

```
  # feature dataframe
  feat_count = sapply(strsplit(feature_sets, "-"), length)
  feat_count[1] = 0
  fsets = data.frame(feat_count = feat_count,
                     set = feature_sets,
                     loss_mean = loss_values, stringsAsFactors = FALSE)
  # plot 1
  loss_means = fsets %>%
    group_by(feat_count) %>%
    summarise(mean = mean(loss_mean))
  plot1 = ggplot(fsets, aes(x = feat_count, y = loss_mean)) +
    geom_point() +
    geom_line(data = loss_means, aes(x = feat_count, y = mean), color = "blue") +
    labs(x="Feature Count", y="Loss Means", title="Loss Mean by Feature Count") +
    theme_bw() +
    scale_y_continuous(breaks = round(seq(0, max(fsets$loss_mean)+20, by = 20), 1))
  # plot 2
  plot2 = ggplot(fsets, aes(x = reorder(set, +loss_mean) , y = loss_mean)) +
    geom_bar(stat = "identity", width=0.8) +
    scale_y_continuous(breaks = round(seq(0, max(fsets$loss_mean)+20, by = 20), 1)) +
    labs(x="Feature Sets", y="Loss Means", title="Loss Means by Feature Sets") +
    theme_bw() +
    theme(axis.text.x = element_text(angle = 60, hjust = 1))
  # get best features
  bfeats_index = strsplit(fsets[which.min(fsets$loss_mean),2],split = "-")[[1]]
  bfeats = colnames(X)[as.numeric(bfeats_index)]
  response = list(fsets = fsets,
                  plot1 = plot1,
                  plot2 = plot2,
                  best_features = bfeats)
  return(response)
}

####################
##### TASK 3.2 #####
####################
library(gridExtra)
# import data
data = swiss
# select dataw
X = data[,2:6]
# actual values
Y = data[,1]

kfcv_result = CV(X, Y, 5)

grid.arrange(grobs=list(kfcv_result$plot1, kfcv_result$plot2), ncol=2)
print("Optimal Features:")
kfcv_result$best_features
# kfcv_result$fsets

#import the data from excel file
data <- read_xlsx("../tecator.xlsx")
```

```r
#Create the plot
moisture <- data$Moisture
protein <- data$Protein
my_data <- data.frame(moisture, protein)
ggplot(my_data, mapping = aes(protein, moisture)) +
  geom_point(color = "black") +
  theme_dark() +
  geom_smooth(method = "lm", color = "red")

#Fitted the model
my_model <- lm(formula = moisture ~ protein, data = my_data)
#create linear model
M1 <- lm(formula = moisture ~ protein, data = data)

#calculate MSE for the linear model
MSE1 <- mean((M1$residuals)^2)

#create the quadratic model
M2 <- lm(formula = moisture ~ protein + I(protein^2), data = data)

#calculate the MSE for the quadratic
MSE2 <- mean((M2$residuals)^2)

# Splitting the data into 50% training and 50% test.
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = data[id, ]
valid = data[-id, ]


# function for creating formulas
create_formula <- function(X,Y,i){
  vec <- c()
  for (k in 1:i) {
    vec <- c(vec,"I(",X,"^",as.character(k),")","+")
  }
  vec <- c(Y," ~",vec)
   return(as.formula(paste(vec[-length(vec)], collapse = " ")))
}

train_MSE <- c()
valid_MSE <- c()
my_list <- list()
my_prediction <- list()

# Fitted the values
for (i in 1:6) {
  #fit the model for training data
  my_formula <- create_formula("Protein", "Moisture" , i)
  my_list[[i]] <- lm(formula = my_formula, data = train )

  #calculate MSE for training data
```

```
  temp_MSE <- mean(my_list[[i]]$residuals^2)
  train_MSE <- c(train_MSE,temp_MSE)

  # Predicted values with validation data
  my_prediction[[i]] <-  predict(object = my_list[[i]] ,  newdata = valid)

  #calculate the MSE for the validation data
  temp_MSE <- mean((valid$Moisture - my_prediction[[i]])^2)
  valid_MSE <- c(valid_MSE,temp_MSE)
}

  MSE_df_train <- data.frame("Models" =  c(1:6),"MSE"= train_MSE)
  MSE_df_valid <- data.frame("Models" =  c(1:6), "MSE"= valid_MSE)

  ggplot() +
    geom_line(data = MSE_df_train, aes(x= MSE_df_train$Models, y = MSE_df_train$MSE), color='black') +
    geom_line(data = MSE_df_valid, aes(x = MSE_df_valid$Models, y = MSE_df_valid$MSE), color='red') +
    geom_point(mapping = aes(x= MSE_df_train$Models, y = MSE_df_train$MSE)) +
    geom_point(aes(x = MSE_df_valid$Models, y = MSE_df_valid$MSE), color='red') +
    labs(title = "Comparison of MSE for training( Black line) and validation data(Red line)", x = "Degr

# Take the 1-100 variables for the data
  X = data[, 2:102]

# Fit the model with my data
model <- lm(formula = Fat ~ ., data = X)

# Variable selection with MASS package
 best_model = stepAIC(object = model, trace = 0)

#Number of variables were selected
 print(list('number_of_variable_selected' =  best_model$rank - 1))
predictors = as.matrix(data[, 2:101])
response = data$Fat

#fit the Ridge model with X data
ridge_model <- glmnet(x = predictors, y = response, family = "gaussian", alpha = 0)
#plot for the coefficients
plot(ridge_model, xvar="lambda", label=TRUE)

# Fit the LASSO model
lasso_model <- glmnet(x = predictors, y = response, family = "gaussian", alpha = 1)

#plot for the coefficients
plot(lasso_model, xvar="lambda", label=TRUE)

cv_lasso = cv.glmnet(predictors , response, alpha=1, family="gaussian", lambda = c(lasso_model$lambda,
print(list('Optimal_value' =  cv_lasso$lambda.min))
plot(cv_lasso)
```