Lab_1

Andreas Stasinakis

November 12, 2018

Assignment 1 Spam classification with nearest neighbors

The data file spambase.xlsx contains information about the frequency of various words, characters etc for a total of 2740 e-mails. Furthermore, these e-mails have been manually classified as spams (spam = 1) or regular e-mails (spam = 0).

1.1)

Import the data into R and divide it into training and test sets (50%/50%)

```
library(readxl)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```
library(kknn)
library(ggplot2)
library(MASS)

#import the data from excel file
data <- read_excel("../dataset/spambase.xlsx")
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))

#spliting them in train and test data
train = data[id, ]
test = data[-id, ]
```

1.2)

Use logistic regression (functions glm(), predict()) to classify the training and test data by the classification principle ????Y = 1 for P>0.5 otherwise Y = 0 and report the confusion matrices (use table()) and the misclassification rates for training and test data. Analyse the obtained results.

Table 1: confusion matrix for train data

	predicted(no-spam)	predicted(spam)
real(no-spam)	803	142
real(spam)	81	344

Table 2: confusion matrix for test data

	predicted(no-spam)	predicted(spam)
real(no-spam)	791	146
real(spam)	97	336

Table 3: misclassification rates

misclassification.rate.train.	0.1627737
misclassification.rate.test.	0.1773723

Analysis

In general, we split the data to train and test data. We use the train data to fit the model and test data to make predictions. In this case though, we have predictions for both train and test data. The results are really similar each other and both misclassification rates are not that high, around 0.17. That means that approximately 83% of the data was clarified correctly, which is efficient enough.

We can also analyse more these confusion matrices calculating the true positives(TP), true negatives(TN), false positives(FP) and false negatives(FN). In both cases, the false negatives were higher than the false positive. As a result, both models predicted incorrect that a no-spam email was spam more times than that a spam email was no-spam.

We can also calculate the Accuracy, which is the percentage of times that the classifier was correct, with 2 ways :

i) By the formula:

$$(TP + TN)/total$$

ii) By the formula : misclasification rates + Accuracy = 1.

Therefore,for the training data, we have Accuracy = 0.8372263, when in the test data is 0.8226277.

1.3

Use logistic regression to classify the test data by the classification principle (Y = 1 for P>0.9) and report the confusion matrices (use table()) and the misclassification rates for training and test data. Compare the results. What effect did the new rule have?

Table 4: confusion matrix for train data

	predicted(no-spam)	predicted(spam)
real(no-spam)	944	1
real(spam)	419	6

Table 5: confusion matrix for test data

	predicted(no-spam)	predicted(spam)
real(no-spam)	936	1
real(spam)	427	6

Table 6: misclassification rates

misclassification.rate.train.	0.3065693
misclassification.rate.test.	0.3124088

Analysis

Keeping the same models but changing the classification principle effects the results significantly. It is obvious that classifying with 0.9 probability a mail as a no - spam increases the e-mails detected correctly as no - spam but also many of spam e-mails are classified as no - spam. Therefore, the misclassification rate is almost the double, both in train and test data, from the first case with classification probability 0.5.

We can also observe that in contrast to the 1.2, in both models, the number of false positives is significantly higher than the number of false negatives (419 >>1 and 427>>1).

Finally, the accuracy in both of these cases is less than in question 1.2.

1.4

Use standard classifier kknn() with K=30 from package kknn, report the the misclassification rates for the training and test data and compare the results with step 2.

Table 7: confusion matrix for train data

	predicted(no-spam)	predicted(spam)
real(no-spam)	807	138
real(spam)	98	327

Table 8: confusion matrix for test data

	predicted(no-spam)	predicted(spam)
real(no-spam)	672	265
real(spam)	187	246

Table 9: misclassification rates

misclassification.rate.train.	0.1722628
misclassification.rate.test.	0.3299270

Analysis

Using standard classifier, gives us different misclassification rates between test and training data. More specific, the misclassification rate for the training data (0.1722628) is almost the half of the misclassification rate for the test data (0.3299270). Therefore we can conclude that the model we fit our data is overfitting or underfitting. In order to understand if is overfitting or underfitting, we have to do the classifier for many different k.

Comparing to 1.2, we can say that the model is overfitting, as in this case, because the misclassification rates for both data are similar.

1.5

Repeat step 4 for K=1 and compare the results with step 4. What effect does the decrease of K lead to and why?

Table 10: confusion matrix for train data

	predicted(no-spam)	predicted(spam)
real(no-spam)	945	0
real(spam)	0	425

Table 11: confusion matrix for test data

	predicted(no-spam)	predicted(spam)
real(no-spam)	640	297
real(spam)	177	256

Table 12: misclassification rates

misclassification.rate.train.	0.0000000
misclassification.rate.test.	0.3459854

Analysis

Choosing the number of parameters k is not that easy. Actually, a small value of k means a simple algorithm but not so accurate. On the other hand, a large value for k can make it computational expensive. In many problems, the formula below could be useful :

$$k = \sqrt{n}$$

where n = number of observations and we choose the closest odd number.

Using kknn for only one neighbor, makes the model also overfitting. This is obvious from the table, because for the train data we predict all the e-mails correctly, when for the test data the misclassification rate is around 0.346. Comparing to $k = 30$, i believe that there is a better way to choose K, as mentioned before, because both of them are overfitting and the clarification rates for the test data are quite high.

Assignment 3. Feature selection by cross-validation in a linear model.

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. ### My code for the function which performs feature selection

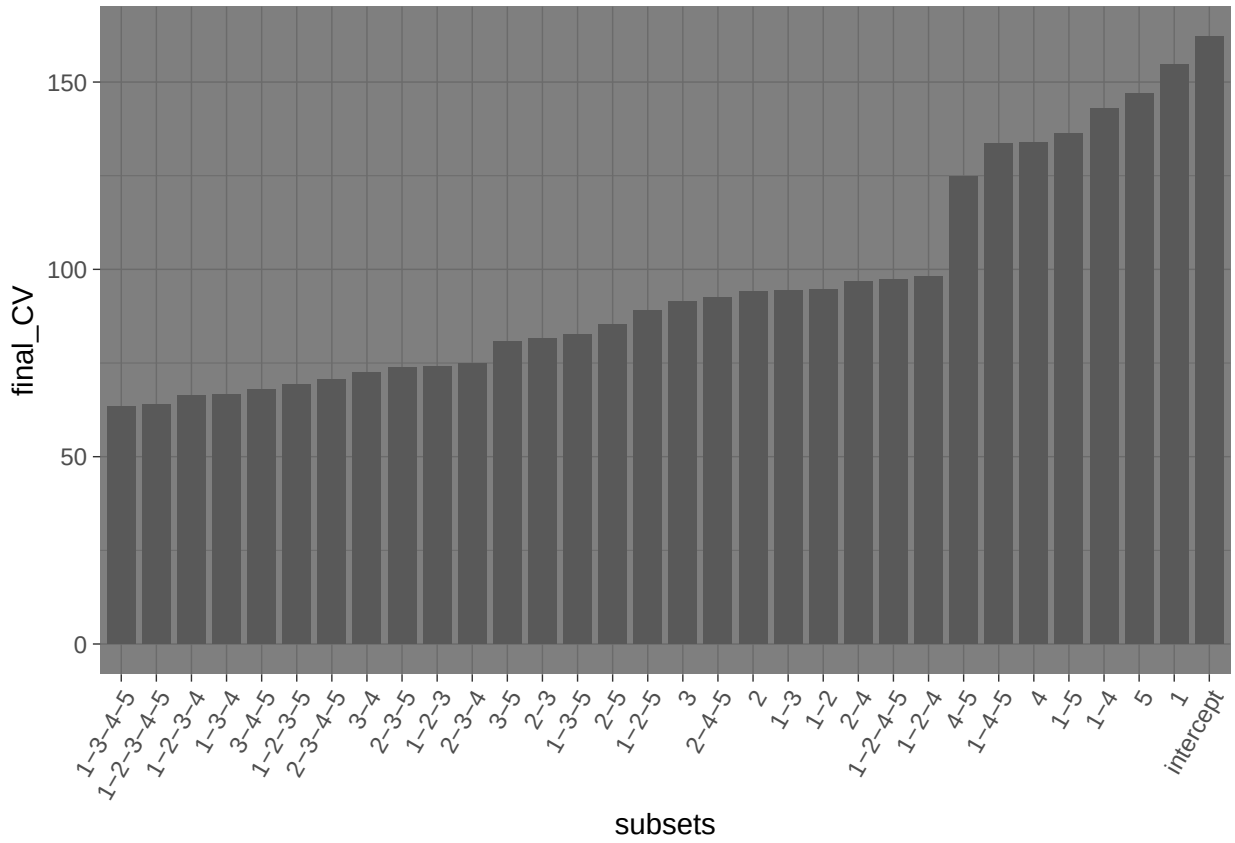
```r
my_CV <- function(X,Y,Nfolds){

  #shuffle the data
  set.seed(12345)
  id = sample(1:nrow(X))
  X = X[id,]
  Y = Y[id]
  data = X

  #create empty vectors
  CVs = c()
  CVs_vector <- c()
  final_CV = c()
  subsets = c()

  # take all permutations
  for (k in 0:(ncol(data))) {

      # take all combinations
      v = combn(c(1:ncol(data)), k)
    for (l in 1:dim(v)[2]) {

      # K-fold cross-validation
      for (i in 1:Nfolds) {

        #Spliting the data into training and test
        test_id = seq(from = i, nrow(data), by = Nfolds)
        train_X = as.matrix(data[-test_id, v[,l]])
        train_Y = as.matrix(Y[-test_id])
        test_X = as.matrix(data[test_id, v[,l]])
        test_Y = as.matrix(Y[test_id])

        #Calculate the coefficients for the model
        train_X = as.matrix(cbind("intercept" = rep(1,nrow(train_X)), train_X))
        test_X = as.matrix(cbind("intercept" = rep(1,nrow(test_X)), test_X))
        B = solve(t(train_X) %*% train_X) %*% t(train_X) %*% train_Y

        #calculate the estimated Y values
        Y_hat = (test_X %*% B)

        #Calculate the SSE
        temp_CVs <- mean((test_Y - Y_hat)^2)
        CVs_vector <- c(CVs_vector,temp_CVs)

      }

      #take the intercept
      if (k==0) {
        subsets = c(subsets,"intercept")
      }else {
        subsets = c(subsets,paste(v[,l],collapse = "-"))
      }
```

```
      #store CV for every model
      CVs <- mean(CVs_vector)
      final_CV <- c(final_CV,CVs)
      CVs_vector <- c()
    }
  }

  #final data frame with all permutations and CV values
  df = data.frame(subsets, final_CV)
  df = df[order(df$final_CV),]
  rownames(df) <- c(1:nrow(df))
  df$subsets = factor(df$subsets, levels = df$subsets)

  #take the lowest CV and the best feature subset
  best_feature <- colnames(data)[ as.numeric(strsplit(as.character(df$subsets[1]), '-')[[1]])]
  lowest_CV <- df[1,2]


  #plot for the df
  my_plot = ggplot(df, aes(x = subsets , y = final_CV)) +
    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_dark() +
    theme(axis.text.x = element_text(angle = 60, hjust = 1))

  #list all them
  final_results <- list(  plot = my_plot, best_subset = best_feature , lowest_CV_value = lowest_CV)

  #return all results
  return(final_results)

}
```

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.

```
## $plot
```




```
##
## $best_subset
## [1] "Agriculture"      "Education"        "Catholic"
## [4] "Infant.Mortality"
##
## $lowest_CV_value
## [1] 63.40326
```

Analysis

Using the k-folds cross validation, help us "remove" the variables which are not related with the dependent variable. Therefore the model with the lowest CV score is the best one and the variables which are missing are not that important for the prediction. As a result, from the graph is obvious that this feature subset is the best one because it has the lowest CV score.

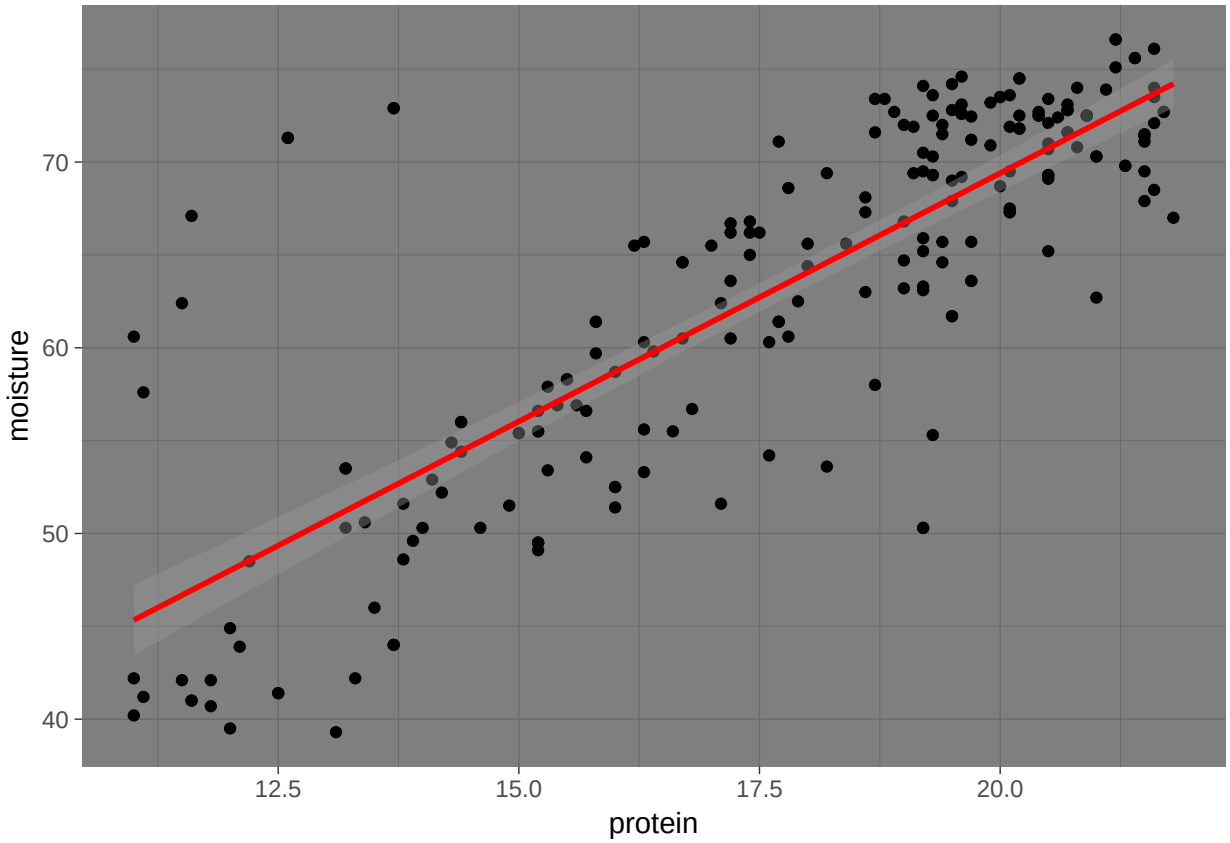
We can also observe that the number of variables is not important. To be more specific, there are models with the same number of variables but totally different CV scores. Also it is clear that in most of the cases, a model with only one or two variables has larger CV score from a more complex model.

Assignment 4. Linear regression and regularization

The Excel file tecator.xlsx contains the results of study aimed to investigate whether a near infrared absorbance spectrum can be used to predict the fat content of samples of meat. For each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is -log10 of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry.

4.1)

Import data to R and create a plot of Moisture versus Protein. Do you think that these data are described well by a linear model?



Analysis

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.

4.2

Consider model ???????????????? in which Moisture is normally distributed, and the expected Moisture is a polynomial function of Protein including the polynomial terms up to power ???????? (i.e M1 is a linear model, M2 is a quadratic model and so on). Report a probabilistic model that describes ????????????????. Why is it appropriate to use MSE criterion when fitting this model to a training data?

Analysis

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

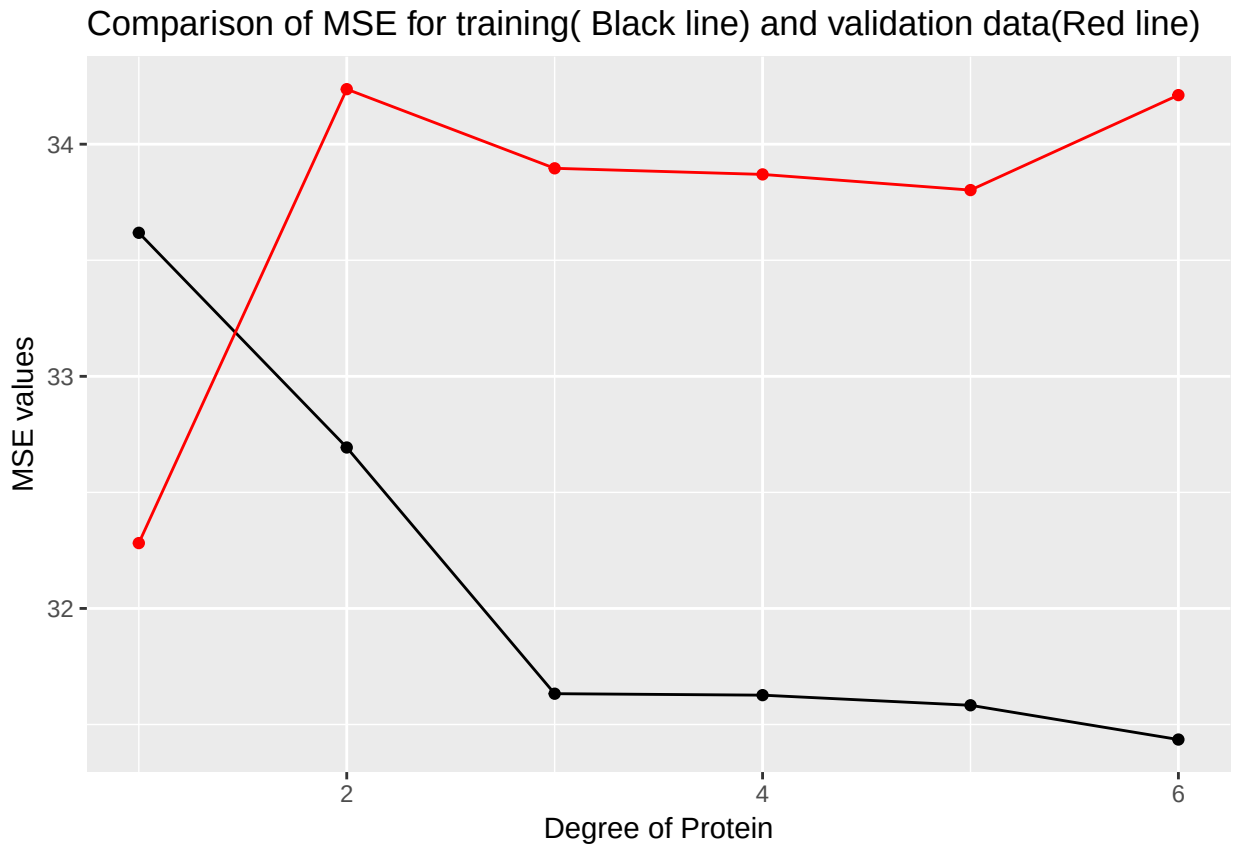
the calculation of MSE is :

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

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.

4.3

Divide the data into training and validation sets(50%/50%) and fit models $M_i, i = 1...6$. For each model, record the training and the validation MSE and present a plot showing how training and validation MSE depend on i (write some R code to make this plot). Which model is best according to the plot? How do the MSE values change and why? Interpret this picture in terms of bias-variance tradeoff.



Analysis

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.

4.4

Perform variable selection of a linear model in which Fat is response and Channel1-Channel100 are predictors by using stepAIC. Comment on how many variables were selected.

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

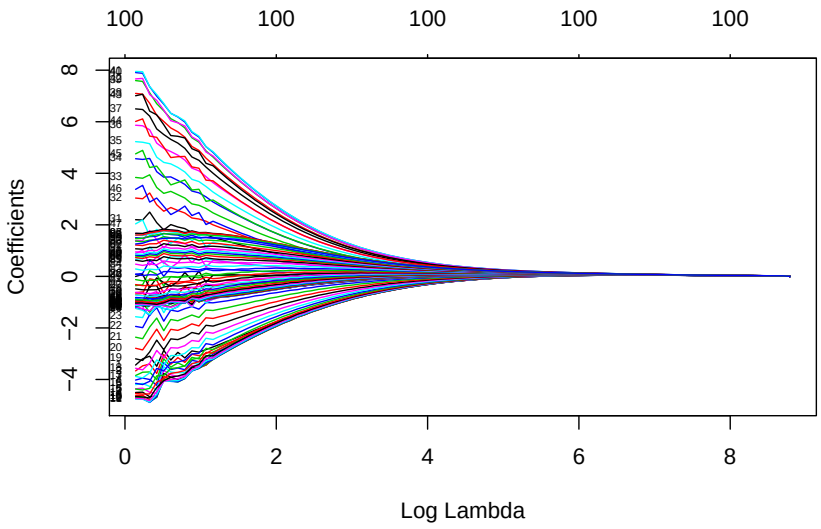
Analysis

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.

4.5

Fit a Ridge regression model with the same predictor and response variables. Present a plot showing how model coefficients depend on the log of the penalty factor ???????? and report how the coefficients change with λ. ???????.

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



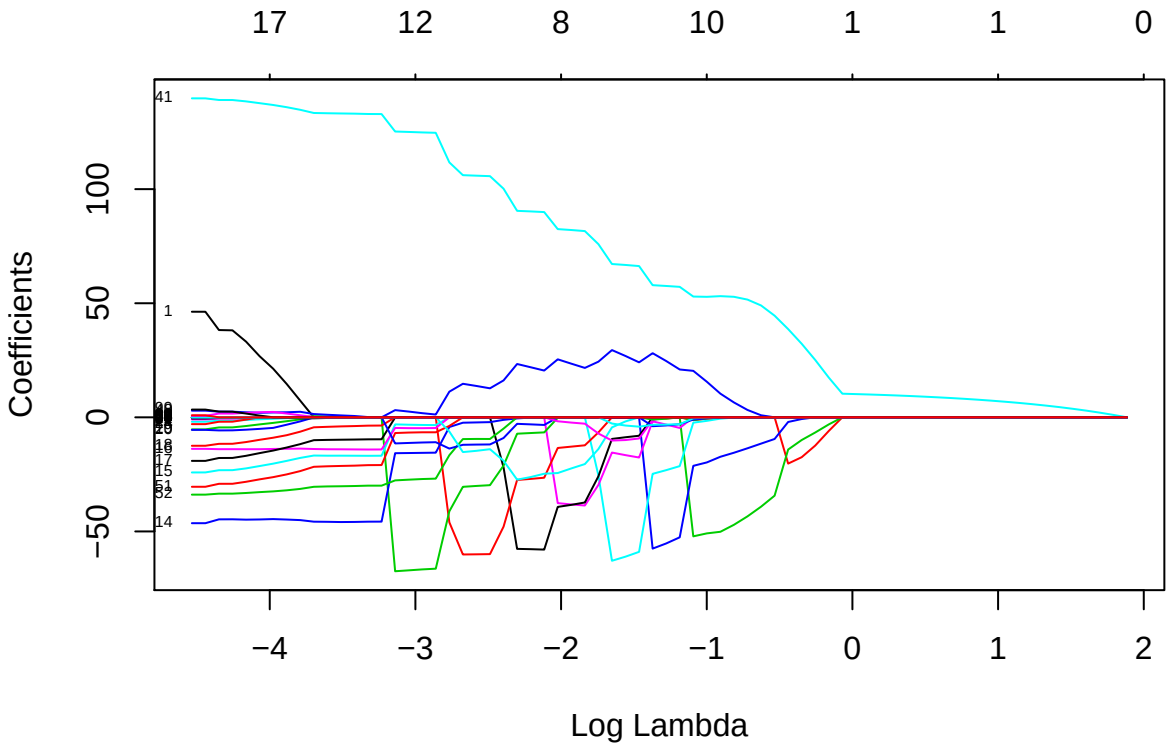
Analysis

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.

4.6

Repeat step 5 but fit LASSO instead of the Ridge regression and compare the plots from steps 5 and 6. Conclusions?



Analysis

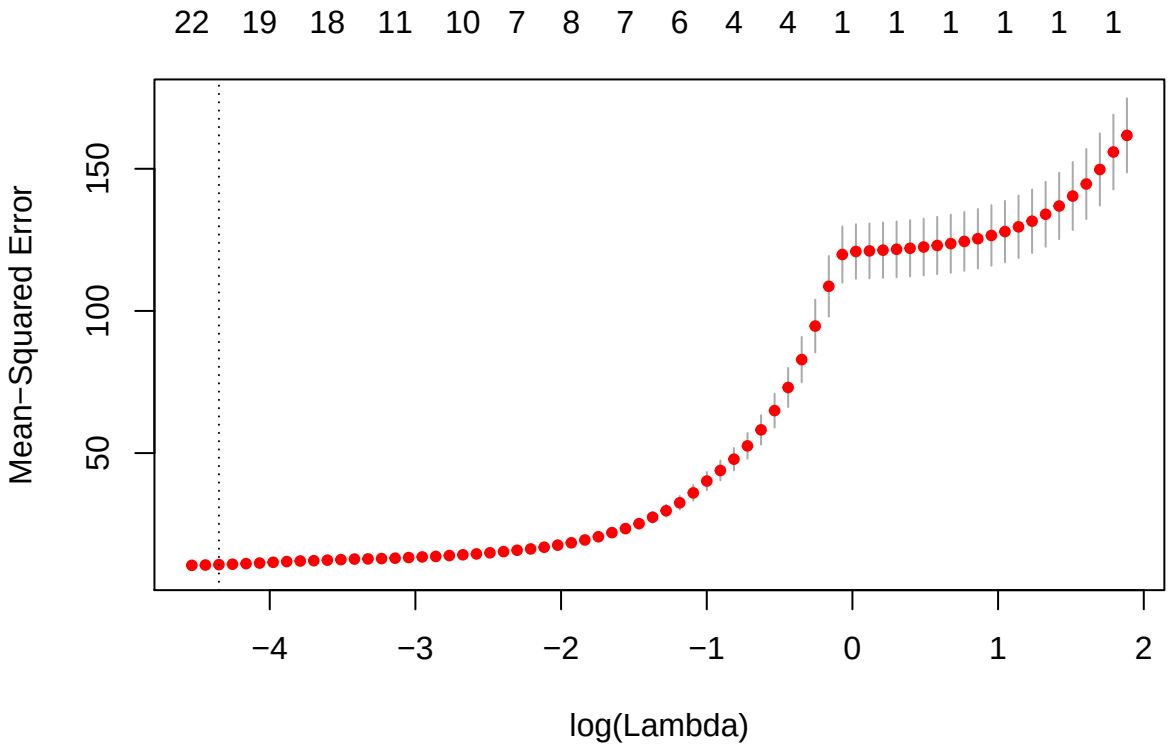
Another regularization method is LASSO regression. LASSO regression is similar to Ridge regression but they have a very important difference. As mentioned before, Ridge regression does not remove variables, it just shrinks their coefficients. In contrary, LASSO regression shrinks the coefficients and 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.

4.7

Use cross-validation to find the optimal LASSO model (make sure that case λ???????? = 0 is also considered by the procedure) , report the optimal λ ???????? and how many variables were chosen by the model and make conclusions. Present also a plot showing the dependence of the CV score and comment how the CV score changes with λ???????.

```
## $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.



Analysis

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.

4.8

Compare the results from steps 4 and 7.

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 = FALSE, warning = FALSE)
library(readxl)
library(glmnet)
library(kknn)
library(ggplot2)
library(MASS)

#import the data from excel file
data <- read_excel("../dataset/spambase.xlsx")
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))

#spliting them in train and test data
train = data[id, ]
test = data[-id, ]
  #fitting the train data in a model
  my_model = glm( formula = Spam ~ ., data = train, family = binomial)

  #predict values with train data
  predict_train = predict(object = my_model, train, type = "response")

  #predict values with test data
  predict_test = predict(object = my_model, test, type = "response")

  #use classification principle for both predicted vectors
  predict_test[which(predict_test > 0.5)] = 1
  predict_test[which(predict_test <= 0.5)] = 0
  predict_train[which(predict_train > 0.5)] = 1
  predict_train[which(predict_train <= 0.5)] = 0

  #reporting the confusion matrix for train data
  confusion_matrix_train <- table( train$Spam, predict_train)
  rownames(confusion_matrix_train) <- c("real(no-spam)","real(spam)")
  colnames(confusion_matrix_train) <- c("predicted(no-spam)", "predicted(spam)")

  #reporting the confusion matrix for test data
  confusion_matrix_test <- table( test$Spam, predict_test)
  rownames(confusion_matrix_test) <- c("real(no-spam)","real(spam)")
  colnames(confusion_matrix_test) <- c("predicted(no-spam)", "predicted(spam)")

  #print both confusion matrices
  knitr::kable(x = confusion_matrix_train, caption = "confusion matrix for train data")
  knitr::kable(x = confusion_matrix_test, caption = "confusion matrix for test data")

  #misclassification rates for both confusion matrices
  mis_rate_train <-1-sum(diag(confusion_matrix_train))/sum(confusion_matrix_train)
  mis_rate_test <-1-sum(diag(confusion_matrix_test))/sum(confusion_matrix_test)
  mis_rate <- data.frame("misclassification rate(train)" = mis_rate_train,"misclassification rate(test)
  knitr::kable(x = t(mis_rate), caption = "misclassification rates")
```

```r
# we will keep the same model as before : my_model
  # Also the same predicted models : predict_test and predict_train
  #predict values with train data
  predict_train = predict(object = my_model, train, type = "response")

  #predict values with test data
  predict_test = predict(object = my_model, test, type = "response")

  #We change the classification principle for both
  predict_test[which(predict_test > 0.9)] = 1
  predict_test[which(predict_test <= 0.9)] = 0
  predict_train[which(predict_train > 0.9)] = 1
  predict_train[which(predict_train <= 0.9)] = 0

  #reporting the new confusion matrix for train data
  confusion_matrix_train <- table( train$Spam, predict_train)
  rownames(confusion_matrix_train) <- c("real(no-spam)","real(spam)")
  colnames(confusion_matrix_train) <- c("predicted(no-spam)", "predicted(spam)")

  #reporting the confusion matrix for test data
  confusion_matrix_test <- table( test$Spam, predict_test)
  rownames(confusion_matrix_test) <- c("real(no-spam)","real(spam)")
  colnames(confusion_matrix_test) <- c("predicted(no-spam)", "predicted(spam)")

  #print both confusion matrices
  knitr::kable(x = confusion_matrix_train, caption = "confusion matrix for train data")
  knitr::kable(x = confusion_matrix_test, caption = "confusion matrix for test data")

  #misclassification rates for both confusion matrices
  mis_rate_train <- 1-sum(diag(confusion_matrix_train))/sum(confusion_matrix_train)
  mis_rate_test <- 1-sum(diag(confusion_matrix_test))/sum(confusion_matrix_test)
  mis_rate <- data.frame("misclassification rate(train)" = mis_rate_train,"misclassification rate(test)
  knitr::kable(x = t(mis_rate), caption = "misclassification rates")


#using weighted k - Nearest Neighbor Classifier

# for train data
  kknn_train <- kknn(formula = Spam ~ ., train = train, test = train, k = 30)

  # for test data
  kknn_test <- kknn(formula = Spam ~ ., train = train, test = test, k = 30)

  #use the classification principle for both data
  kknn_train$fitted.values[which(kknn_train$fitted.values > 0.5)] = 1
  kknn_train$fitted.values[which(kknn_train$fitted.values <= 0.5)] = 0
  kknn_test$fitted.values[which(kknn_test$fitted.values > 0.5)] = 1
  kknn_test$fitted.values[which(kknn_test$fitted.values <= 0.5)] = 0

  #confusion matrix for train data
  cm_train <- table( train$Spam,  kknn_train$fitted.values)
  rownames(cm_train) <- c("real(no-spam)","real(spam)")
  colnames(cm_train) <- c("predicted(no-spam)", "predicted(spam)")
```

```
  #confusion matrix for test data
  cm_test <- table( test$Spam,  kknn_test$fitted.values)
  rownames(cm_test) <- c("real(no-spam)","real(spam)")
  colnames(cm_test) <- c("predicted(no-spam)", "predicted(spam)")

  #print both confusion matrices
  knitr::kable(x = cm_train, caption = "confusion matrix for train data")
  knitr::kable(x = cm_test, caption = "confusion matrix for test data")

  #misclassification rates for both confusion matrices
  mr_train <- 1-sum(diag(cm_train))/sum(cm_train)
  mr_test <- 1-sum(diag(cm_test))/sum(cm_test)
  mr <- data.frame("misclassification rate(train)" = mr_train,"misclassification rate(test)" =mr_test)
  knitr::kable(x = t(mr), caption = "misclassification rates")

  # for train data
  kknn_train <- kknn(formula = Spam ~ ., train = train, test = train, k = 1)

  # for test data
  kknn_test <- kknn(formula = Spam ~ ., train = train, test = test, k = 1)

  #use the classification principle for both data
  kknn_train$fitted.values[which(kknn_train$fitted.values > 0.5)] = 1
  kknn_train$fitted.values[which(kknn_train$fitted.values <= 0.5)] = 0
  kknn_test$fitted.values[which(kknn_test$fitted.values > 0.5)] = 1
  kknn_test$fitted.values[which(kknn_test$fitted.values <= 0.5)] = 0

  #confusion matrix for train data
  cm_train <- table( train$Spam,  kknn_train$fitted.values)
  rownames(cm_train) <- c("real(no-spam)","real(spam)")
  colnames(cm_train) <- c("predicted(no-spam)", "predicted(spam)")

  #confusion matrix for test data
  cm_test <- table( test$Spam,  kknn_test$fitted.values)
  rownames(cm_test) <- c("real(no-spam)","real(spam)")
  colnames(cm_test) <- c("predicted(no-spam)", "predicted(spam)")

  #print both confusion matrices
  knitr::kable(x = cm_train, caption = "confusion matrix for train data")
  knitr::kable(x = cm_test, caption = "confusion matrix for test data")

  #misclassification rates for both confusion matrices
  mr_train <- 1-sum(diag(cm_train))/sum(cm_train)
  mr_test <- 1-sum(diag(cm_test))/sum(cm_test)
  mr <- data.frame("misclassification rate(train)" = mr_train,"misclassification rate(test)" =  mr_test)
  knitr::kable(x = t(mr), caption = "misclassification rates")


my_CV <- function(X,Y,Nfolds){

  #shuffle the data
  set.seed(12345)
  id = sample(1:nrow(X))
```

```
X = X[id,]
Y = Y[id]
data = X

#create empty vectors
CVs = c()
CVs_vector <- c()
final_CV = c()
subsets = c()

# take all permutations
for (k in 0:(ncol(data))) {

    # take all combinations
    v = combn(c(1:ncol(data)), k)
  for (l in 1:dim(v)[2]) {

    # K-fold cross-validation
    for (i in 1:Nfolds) {

      #Spliting the data into training and test
      test_id = seq(from = i, nrow(data), by = Nfolds)
      train_X = as.matrix(data[-test_id, v[,l]])
      train_Y = as.matrix(Y[-test_id])
      test_X = as.matrix(data[test_id, v[,l]])
      test_Y = as.matrix(Y[test_id])

      #Calculate the coefficients for the model
      train_X = as.matrix(cbind("intercept" = rep(1,nrow(train_X)), train_X))
      test_X = as.matrix(cbind("intercept" = rep(1,nrow(test_X)), test_X))
      B = solve(t(train_X) %*% train_X) %*% t(train_X) %*% train_Y

      #calculate the estimated Y values
      Y_hat = (test_X %*% B)

      #Calculate the SSE
      temp_CVs <- mean((test_Y - Y_hat)^2)
      CVs_vector <- c(CVs_vector,temp_CVs)

    }

    #take the intercept
    if (k==0) {
      subsets = c(subsets,"intercept")
    }else {
      subsets = c(subsets,paste(v[,l],collapse = "-"))
    }

    #store CV for every model
    CVs <- mean(CVs_vector)
    final_CV <- c(final_CV,CVs)
    CVs_vector <- c()
  }
```

```
  }

  #final data frame with all permutations and CV values
  df = data.frame(subsets, final_CV)
  df = df[order(df$final_CV),]
  rownames(df) <- c(1:nrow(df))
  df$subsets = factor(df$subsets, levels = df$subsets)

  #take the lowest CV and the best feature subset
  best_feature <- colnames(data)[ as.numeric(strsplit(as.character(df$subsets[1]), '-')[[1]])]
  lowest_CV <- df[1,2]


  #plot for the df
  my_plot = ggplot(df, aes(x = subsets , y = final_CV)) +
    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_dark() +
    theme(axis.text.x = element_text(angle = 60, hjust = 1))

  #list all them
  final_results <- list(  plot = my_plot, best_subset = best_feature , lowest_CV_value = lowest_CV)

  #return all results
  return(final_results)

}
X = swiss[,-1]
Y = swiss$Fertility
Nfolds = 5
my_CV(X ,Y, Nfolds = Nfolds)


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

#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)
```

```r
#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

data <- read_excel("../dataset/tecator.xlsx")

# 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)
```