Lab01 Machine Learning

Machine Learning - 732A99

Thijs Quast

22 November 2018

Contents

Assignment 1

1

```r
# Import Excel file spambase.xlsx
library(readxl)
data <- read_excel("spambase.xlsx")

# Divide excel file into training and tests sets (50%/50%)
n <- dim(data)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- data[id,]
test <- data[-id, ]
```

2

After splitting the dataset into training and test sets, a logistic regression is ran.

```r
# Fitting and predicting
fit <- glm(Spam ~ ., data = train, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
predict_on_train <- predict(fit, train, type = "response")
predict_on_test <- predict(fit, test, type = "response")

# Classifying results for training and test datasets p>0.5
predict_on_train[predict_on_train > 0.5] <- 1
predict_on_train[predict_on_train <= 0.5] <- 0

predict_on_test[predict_on_test > 0.5] <- 1
predict_on_test[predict_on_test <= 0.5] <- 0

# Confusion matrices
train_confusion <- table(train$Spam, predict_on_train)
test_confusion <- table(test$Spam, predict_on_test)

# Misclassifcation rates
total_observations_train <- sum(train_confusion)
total_observations_test <- sum(test_confusion)

fp_train <- train_confusion[1,2]
fn_train <- train_confusion[2,1]
fp_test <- test_confusion[1,2]
fn_test <- test_confusion[2,1]

missclassification_train <- (fp_train + fn_train)/total_observations_train
missclassification_test <- (fp_test + fn_test)/total_observations_test
```

```r
library(knitr)
kable(train_confusion, caption = "Confusion matrix training data")
```

Table 1: Confusion matrix training data

	0	1
0	803	142
1	81	344

The misclassification rate on the training dataset is: 0.1627737

```
kable(test_confusion ,caption = "Confusion matrix test data")
```

Table 2: Confusion matrix test data

	0	1
0	791	146
1	97	336

The misclassification rate on the test dataset is: 0.1773723

From the obtained results from of the regression on training and test data, one can say that the regression performs better on the training data. The number of False Positives and False Negatives is smaller for the training dataset. Logically, the misclassification rates on the training data is smaller. In practice this means that the regression model is probably slightly overfitting on the training data.

3

```
# Fitting and predicting
fit_2 <- glm(Spam ~ ., data = train, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
predict_on_train_2 <- predict(fit_2, train, type = "response")
predict_on_test_2 <- predict(fit_2, test, type = "response")

# Classifying results for training and test datasets p>0.9
predict_on_train_2[predict_on_train_2 > 0.9] <- 1
predict_on_train_2[predict_on_train_2 <= 0.9] <- 0

predict_on_test_2[predict_on_test_2 > 0.9] <- 1
predict_on_test_2[predict_on_test_2 <= 0.9] <- 0

# Confusion matrices
train_confusion_2 <- table(train$Spam, predict_on_train_2)
test_confusion_2 <- table(test$Spam, predict_on_test_2)

# Misclassifcation rates
total_observations_train_2 <- sum(train_confusion_2)
total_observations_test_2 <- sum(test_confusion_2)

fp_train_2 <- train_confusion_2[1,2]
fn_train_2 <- train_confusion_2[2,1]
fp_test_2 <- test_confusion_2[1,2]
fn_test_2 <- test_confusion_2[2,1]
```

```
missclassification_train_2 <- (fp_train_2 + fn_train_2)/total_observations_train_2
missclassification_test_2 <- (fp_test_2 + fn_test_2)/total_observations_test_2
```

```
kable(train_confusion_2, caption = "Confusion matrix training data")
```

Table 3: Confusion matrix training data

	0	1
0	944	1
1	419	6

The misclassification rate on the training dataset is: 0.3065693

```
kable(test_confusion_2, caption = "Confusion matrix test data")
```

Table 4: Confusion matrix test data

	0	1
0	936	1
1	427	6

The misclassification rate on the test dataset is: 0.3124088

Resulting from the new rule, the misclassification rates for both the training dataset and the test dataset have gone increased. Also, the confusion matrices have changed in their performance. For both the training and test data, the number of False Negatives have increased and the number of False Positives have decreased. In practice this means that the model more often classifies a spam email as a non-spam email. Which is not what one wants the spam classifier to do. I would say the new rule has made the model worse.

4

```
library(kknn)

# Not sure whether to use train.kknn function or kknn function. For now I will go with kknn.
kknn_model <- kknn(Spam ~ ., train = train, test = test, k = 30)
predict_on_train_3 <- fitted(kknn_model)
predict_on_test_3 <- fitted(kknn_model)

# Classifying results for training and test datasets p>0.5
predict_on_train_3[predict_on_train_3 > 0.5] <- 1
predict_on_train_3[predict_on_train_3 <= 0.5] <- 0

predict_on_test_3[predict_on_test_3 > 0.5] <- 1
predict_on_test_3[predict_on_test_3 <= 0.5] <- 0

# Confusion matrices
train_confusion_3 <- table(train$Spam, predict_on_train_3)
test_confusion_3 <- table(test$Spam, predict_on_test_3)

# Misclassifcation rates
```

```r
total_observations_train_3 <- sum(train_confusion_3)
total_observations_test_3 <- sum(test_confusion_3)

fp_train_3 <- train_confusion_3[1,2]
fn_train_3 <- train_confusion_3[2,1]
fp_test_3 <- test_confusion_3[1,2]
fn_test_3 <- test_confusion_3[2,1]

missclassification_train_3 <- (fp_train_3 + fn_train_3)/total_observations_train_3
missclassification_test_3 <- (fp_test_3 + fn_test_3)/total_observations_test_3
```

The misclassification rate on the training dataset is: 0.449635

The misclassification rate on the test dataset is: 0.329927

Compared the step 2, the misclassification rate on the training dataset is severly worse (0.449635 compared to 0.1627737). The misclassification on the test dataset is also worse, however the difference smaller than on the training dataset, (0.329927 compared to 0.1773723).

5

```r
# 1.5 ####
kknn_model_2 <- kknn(Spam ~ ., train = train, test = test, k = 1)
predict_on_train_4 <- fitted(kknn_model_2)
predict_on_test_4 <- fitted(kknn_model_2)

# Classifying results for training and test datasets p>0.5
predict_on_train_4[predict_on_train_4 > 0.5] <- 1
predict_on_train_4[predict_on_train_4 <= 0.5] <- 0

predict_on_test_4[predict_on_test_4 > 0.5] <- 1
predict_on_test_4[predict_on_test_4 <= 0.5] <- 0

# Confusion matrices
train_confusion_4 <- table(train$Spam, predict_on_train_4)
test_confusion_4 <- table(test$Spam, predict_on_test_4)

# Misclassifcation rates
total_observations_train_4 <- sum(train_confusion_4)
total_observations_test_4 <- sum(test_confusion_4)

fp_train_4 <- train_confusion_4[1,2]
fn_train_4 <- train_confusion_4[2,1]
fp_test_4 <- test_confusion_4[1,2]
fn_test_4 <- test_confusion_4[2,1]

missclassification_train_4 <- (fp_train_4 + fn_train_4)/total_observations_train_4
missclassification_test_4 <- (fp_test_4 + fn_test_4)/total_observations_test_4
```

After setting K=1, the results are the following: The misclassification rate on the training dataset is: 0.470073 The misclassification rate on the test dataset is: 0.3459854

Compared to question 4, the misclassification rate on the training dataset and the test dataset have both increased.

Assignment 3

1

```
select_my_features <- function(x, y, nfolds){
  # set seed and reshuffle data
  set.seed(12345)
  intercept <- rep(1, nrow(x))
  matrix_xy <- cbind(intercept, x, y)
  n <- dim(x)[1]
  id <- sample(1:n, floor(n))
  matrix_xy <- matrix_xy[id, ]
  matrix_x <- matrix_xy[, 1:6]
  matrix_y <- matrix_xy[, 7]

  # Create folds and empty vectors
  folds <- c(1:nfolds)
  residuals_folds <- c()
  res_model <- c()
  n_features <- c()

  # Possible combinations of features including an intercept, intercept is always selected
  combinations_matrix <- expand.grid(c(T, F), c(T, F), c(T, F), c(T, F),
                                     c(T, F))
  intercept_true <- rep(TRUE, 32)
  combinations_matrix <- cbind(intercept_true, combinations_matrix)

    # Loop over each possible model
    for (i in 1:32){
      model_i <- as.logical(combinations_matrix[i,])
      data <- matrix_x[, model_i]
      folds <- c(1:nfolds)
      data_xy <- cbind(data, matrix_y, folds)
      dim_x <- ncol(data_xy) - 2

      #loop over each fold
      for (each in 1:nfolds){
        #training and test data
        train <- data_xy[data_xy[, "folds"] != each,]
        train_x <- train[, 1:dim_x]
        y_dim <- dim_x + 1
        train_y <- train[, y_dim]

        test <- data_xy[data_xy[, "folds"] == each,]
        test_x <- test[, 1:dim_x]
        test_y <- test[, y_dim]

        # computing linear regressions
        Xt_i <- t(train_x)
        XtX_i <- solve(Xt_i %*% train_x)
        betaestimates_i <- XtX_i %*% Xt_i %*% train_y
        yfit_i <- test_x %*% betaestimates_i
        res <- test_y - yfit_i
```

```
        mse <- mean(res^2)

        #storing outcomes in vectors
        residuals_folds[each] <- mse
        mean_mse <- mean(residuals_folds)
      }
      # storing outcomes in other empty vectors, one level above previous loop
  res_model[i] <- mean_mse
  n_features[i] <- dim_x - 1
   }
  # extracting the best model
  best_model <- which.min(res_model)
  possible_regressors <- colnames(matrix_x)
  x <- as.logical(combinations_matrix[best_model,])
  final_model <- possible_regressors[x]

  df <- cbind(res_model, n_features)

  # Compute end result
  list_of_results <- list(final_model)
  list_of_results$plot <- barplot(height = res_model, names.arg = n_features)
  list_of_results$cv_score <- df[best_model, 1]
  return(list_of_results)
}
```

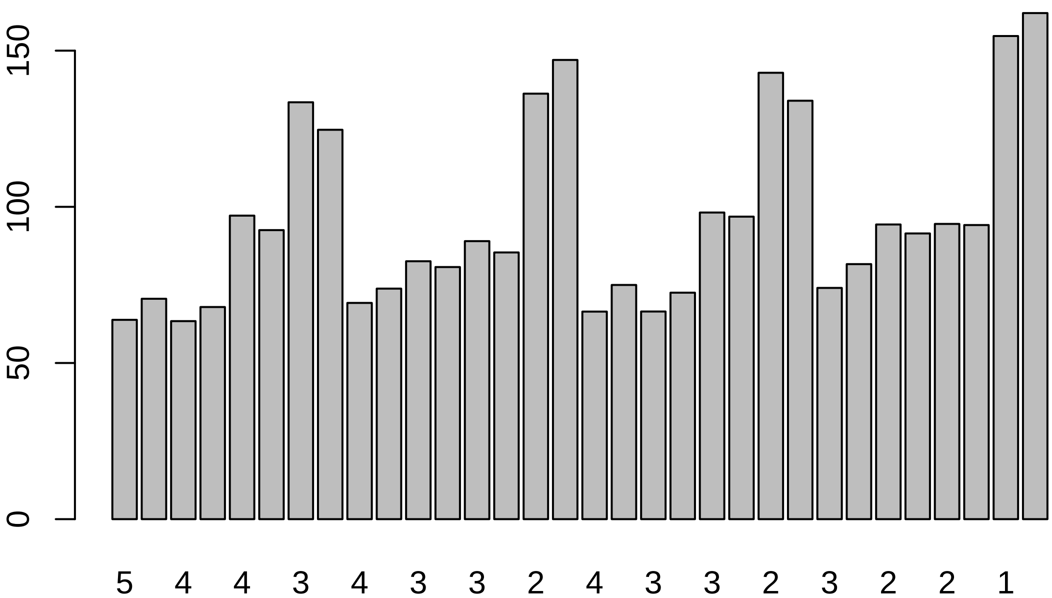
2

```
swiss_y <- as.matrix(swiss[, 1])
swiss_x <- as.matrix(swiss[, 2:6])
select_my_features(swiss_x, swiss_y, 5)
```

```
## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)
```

```
## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)
```

```
## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)

## Warning in cbind(data, matrix_y, folds): number of rows of result is not a
## multiple of vector length (arg 3)
```




```
## [[1]]
## [1] "intercept"        "Agriculture"      "Education"
## [4] "Catholic"         "Infant.Mortality"
##
## $plot
##       [,1]
##  [1,]  0.7
##  [2,]  1.9
##  [3,]  3.1
##  [4,]  4.3
##  [5,]  5.5
##  [6,]  6.7
##  [7,]  7.9
##  [8,]  9.1
```

```
##  [9,] 10.3
## [10,] 11.5
## [11,] 12.7
## [12,] 13.9
## [13,] 15.1
## [14,] 16.3
## [15,] 17.5
## [16,] 18.7
## [17,] 19.9
## [18,] 21.1
## [19,] 22.3
## [20,] 23.5
## [21,] 24.7
## [22,] 25.9
## [23,] 27.1
## [24,] 28.3
## [25,] 29.5
## [26,] 30.7
## [27,] 31.9
## [28,] 33.1
## [29,] 34.3
## [30,] 35.5
## [31,] 36.7
## [32,] 37.9
##
## $cv_score
## res_model
##  63.40326
```

In general I would say that as the number of features increases the model performance increases as well. The optimal subset of featues is 4 (excluding the intercept). The optimal model therefore is:

Fertility ~ Intercept + X1"Agriculture" + X2"Education" + X3"Catholic" + X4"Infant.Mortality"

Resulting in a cross validation score of : 63.40326.

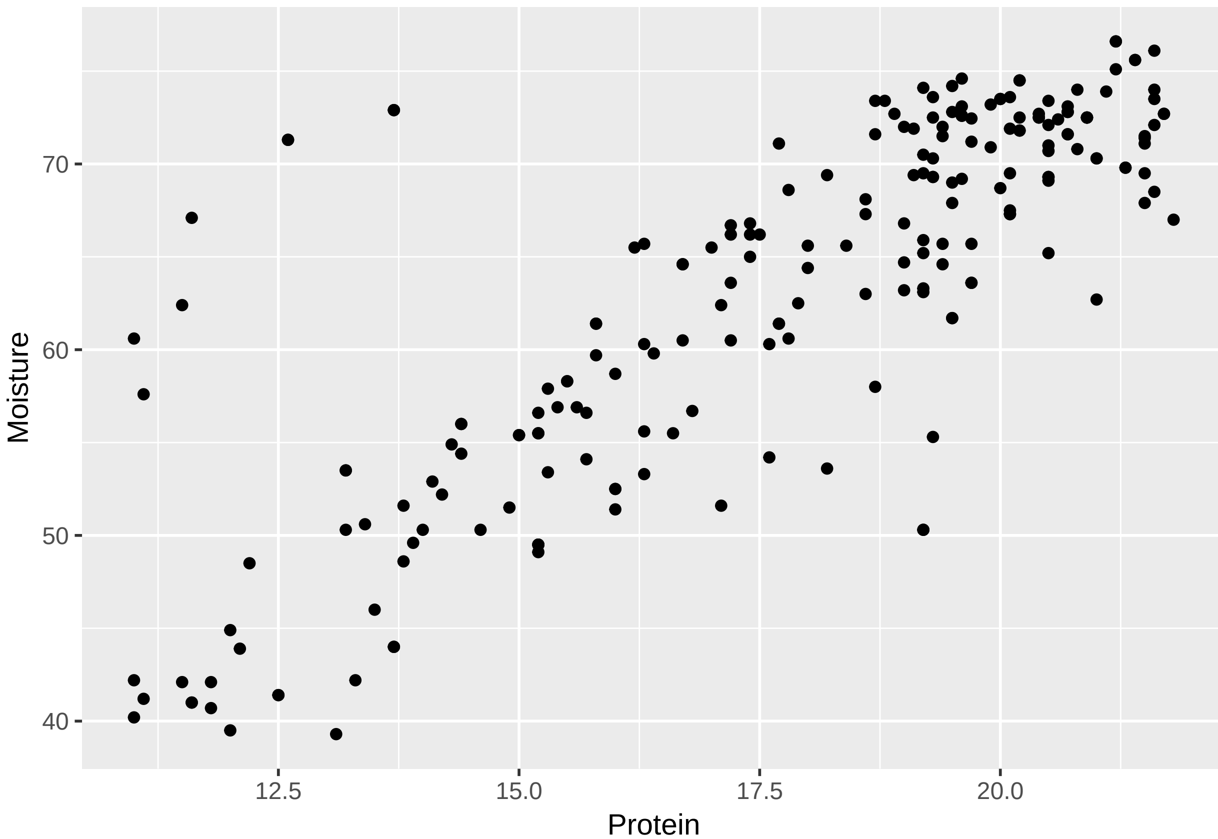
I would say for none of the independent variables it is reasonable to have an impact on the fertility of people. When reasoning, there is no explanation as to why working in agriculture, having a degree, religion or death of childs could affect the fertility of people. Therefore when computing these models it is always important to reason whether the results make sense.

Infant Mortality is more a result of fertility. Therefore it could be an indea to have "Fertility" as independent variable and "Infant.Mortality" as dependent variable.

Assignment 4

1

```r
library(ggplot2)
#4.1
tecator <- read_excel("tecator.xlsx")
plot <- ggplot(tecator, aes(x=Protein, y=Moisture)) + geom_point()
plot
```



Yes, althought there are some outliers, there seems to be positive linear relationship between Protein and Moisture. Meaning, if Protein increaes, Moisture increases. Therefore, I argue the data is well described by a linear model.

2

The probabilistic model that describes Mi is: Moisture = B0 + B1Protein + B2Protein^2 + … + BiProtein^i + error.

It is appropriate to use MSE in this case, because MSE is an unbiased estimator of the variance of the error term. Because MSE is the average of the squared value of all the error terms, and it is computed by dividing by the degrees of freedom, the MSE measures how well the model fits the data.

3

```r
# 4.3
# Dividing data into 50%/50% train/test
n <- dim(tecator)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- tecator[id,]
test <- tecator[-id, ]

# M1
model_m1 <- lm(Moisture ~ Protein, data = train)
m1_predict_test <- predict(model_m1, test)
```

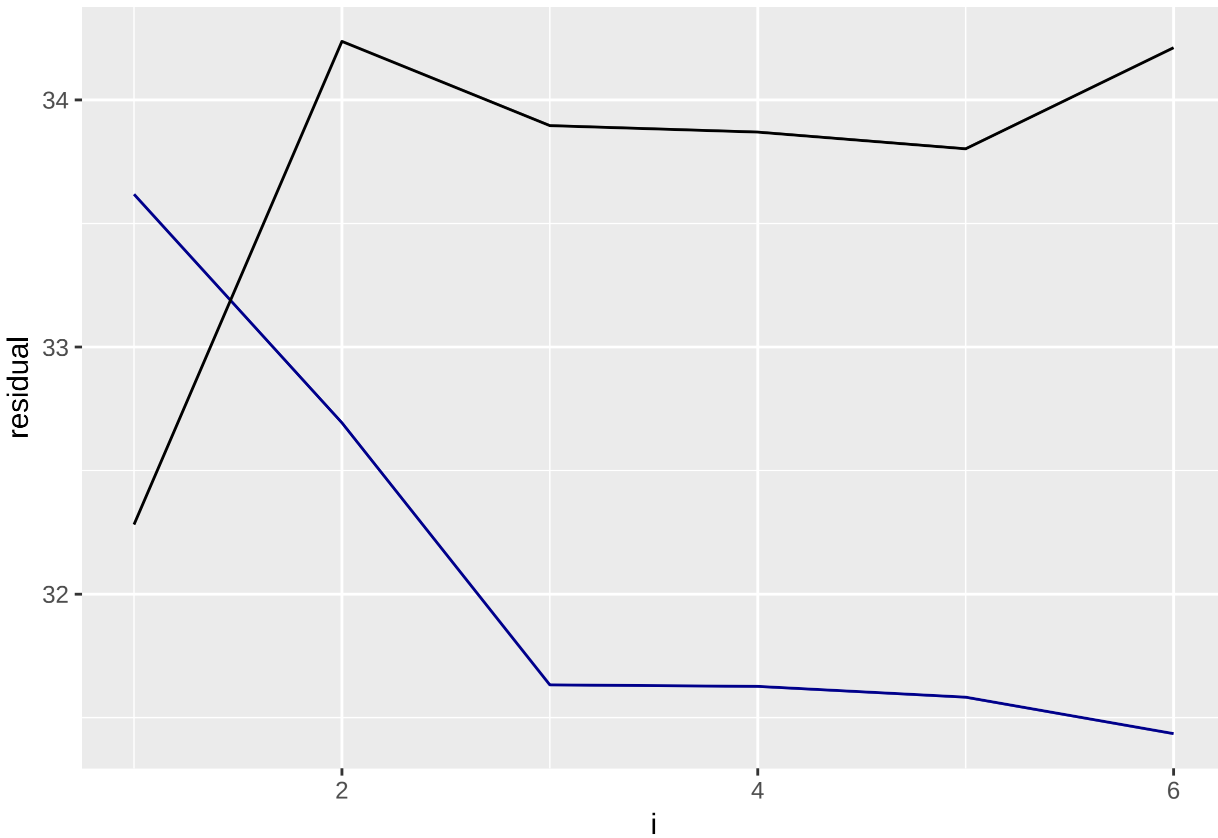
```r
m1_predict_train <- predict(model_m1, train)
m1_residual_test <- (sum((test$Moisture - m1_predict_test)^2))/nrow(test)
m1_residual_train <- (sum((train$Moisture - m1_predict_train)^2))/nrow(train)

#M2
model_m2 <- lm(Moisture ~ Protein + I(Protein^2), data = train)
m2_predict_test <- predict(model_m2, test)
m2_predict_train <- predict(model_m2, train)
m2_residual_test <- (sum((test$Moisture - m2_predict_test)^2))/nrow(test)
m2_residual_train <- (sum((train$Moisture - m2_predict_train)^2))/nrow(train)

#M3
model_m3 <- lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3), data = train)
m3_predict_test <- predict(model_m3, test)
m3_predict_train <- predict(model_m3, train)
m3_residual_test <- (sum((test$Moisture - m3_predict_test)^2))/nrow(test)
m3_residual_train <- (sum((train$Moisture - m3_predict_train)^2))/nrow(train)

#M4
model_m4 <- lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3) + I(Protein^4), data = train)
m4_predict_test <- predict(model_m4, test)
m4_predict_train <- predict(model_m4, train)
m4_residual_test <- (sum((test$Moisture - m4_predict_test)^2))/nrow(test)
m4_residual_train <- (sum((train$Moisture - m4_predict_train)^2))/nrow(train)

#M5
model_m5 <- lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3) + I(Protein^4)
               + I(Protein^5), data = train)
m5_predict_test <- predict(model_m5, test)
m5_predict_train <- predict(model_m5, train)
m5_residual_test <- (sum((test$Moisture - m5_predict_test)^2))/nrow(test)
m5_residual_train <- (sum((train$Moisture - m5_predict_train)^2))/nrow(train)

#M6
model_m6 <- lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3) + I(Protein^4)
               + I(Protein^5) + I(Protein^6), data = train)
m6_predict_test <- predict(model_m6, test)
m6_predict_train <- predict(model_m6, train)
m6_residual_test <- (sum((test$Moisture - m6_predict_test)^2))/nrow(test)
m6_residual_train <- (sum((train$Moisture - m6_predict_train)^2))/nrow(train)

# Create dataframe of MSE outcomes
mse_matrix <- matrix(NA, ncol = 3, nrow = 6)
mse_matrix
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
## [3,]   NA   NA   NA
## [4,]   NA   NA   NA
## [5,]   NA   NA   NA
## [6,]   NA   NA   NA
```

```r
colnames(mse_matrix) <- c("train", "test", "i")
model_i <- c(1:6)
mse_matrix[,3] <- model_i
mse_matrix[1,1] <- m1_residual_train
mse_matrix[1,2] <- m1_residual_test
mse_matrix[2,1] <- m2_residual_train
mse_matrix[2,2] <- m2_residual_test
mse_matrix[3,1] <- m3_residual_train
mse_matrix[3,2] <- m3_residual_test
mse_matrix[4,1] <- m4_residual_train
mse_matrix[4,2] <- m4_residual_test
mse_matrix[5,1] <- m5_residual_train
mse_matrix[5,2] <- m5_residual_test
mse_matrix[6,1] <- m6_residual_train
mse_matrix[6,2] <- m6_residual_test

mse_df <- as.data.frame(mse_matrix)
mse_df
```

```
##      train     test i
## 1 33.61836 32.28154 1
## 2 32.69342 34.23708 2
## 3 31.63266 33.89615 3
## 4 31.62641 33.86992 4
## 5 31.58273 33.80234 5
## 6 31.43513 34.21152 6
```

```r
# Plot MSE's of different models on training and test data

plot_mse <- ggplot(data = mse_df, aes(x=i, y=train)) + geom_line(color = 'darkblue') + geom_line(aes(x=
plot_mse <- plot_mse + ylab("residual")
plot_mse
```



The darkblue line represents the residuals on the training data against the number of i implemented in the model. The black line represents the residuals on the test data against the number of i implemented in the model. The best model according to the plot is where i = 1. This model has the lowest residual on the test dataset and therefore performs best. In this plot, one can see a clear example of a bias-variance tradeoff. As the model is more specifically fit to the training dataset, the performance on the training dataset increaes (The residual decreases) however the model is biased towards the training dataset. As soon as the model is presented a new dataset (the test dataset) with different variance, the model performs worse. Therefore it is always a tradeoff between overfitting on the training dataset to increase model performance and accounting for variance of new unseen data.

4

```
library("MASS")

# Subsetting dataframe for the model
df_4 <- tecator
class(tecator)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
tecator_q4 <- tecator[, 2:102]

full_model <- lm(Fat ~ ., data = tecator_q4)
summary(full_model)
```

```
##
## Call:
## lm(formula = Fat ~ ., data = tecator_q4)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9833 -0.4982  0.0135  0.4864  3.1727
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)      7.302      1.876   3.892 0.000168 ***
## Channel1     10898.047   3003.614   3.628 0.000428 ***
## Channel2    -12174.864   5520.233  -2.205 0.029426 *
## Channel3     -5953.285   8868.517  -0.671 0.503398
## Channel4     23229.862  15426.530   1.506 0.134875
## Channel5    -28386.219  19758.501  -1.437 0.153554
## Channel6     12748.270  17381.421   0.733 0.464794
## Channel7    -11422.335  11454.169  -0.997 0.320769
## Channel8      7102.332   7123.935   0.997 0.320892
## Channel9       783.655   5228.808   0.150 0.881130
## Channel10     3512.239   6787.803   0.517 0.605856
## Channel11   -10547.574  10580.407  -0.997 0.320926
## Channel12    34638.288  18344.772   1.888 0.061543 .
## Channel13   -38705.447  23098.395  -1.676 0.096542 .
## Channel14    28895.947  19952.355   1.448 0.150293
## Channel15   -13726.347  13312.307  -1.031 0.304676
## Channel16    -7062.769   8172.878  -0.864 0.389308
## Channel17     2571.597   6279.661   0.410 0.682932
## Channel18     5263.427   6183.397   0.851 0.396432
## Channel19     8860.827   8925.154   0.993 0.322914
## Channel20   -12149.937  15184.189  -0.800 0.425276
## Channel21   -19284.872  20536.132  -0.939 0.349680
## Channel22    36626.953  22847.592   1.603 0.111680
## Channel23   -11165.390  19302.712  -0.578 0.564111
## Channel24   -15008.939  13616.072  -1.102 0.272655
## Channel25    16698.992   8582.462   1.946 0.054151 .
## Channel26    -4891.852   5901.456  -0.829 0.408880
## Channel27    -6334.752   6072.685  -1.043 0.299084
## Channel28    24043.786   8144.906   2.952 0.003834 **
## Channel29   -39940.900  12335.575  -3.238 0.001578 **
## Channel30    33309.092  17674.622   1.885 0.062034 .
## Channel31   -23174.509  20974.708  -1.105 0.271539
## Channel32    18764.305  18959.821   0.990 0.324423
## Channel33    -3747.892  13458.994  -0.278 0.781158
## Channel34    -6671.747   9353.448  -0.713 0.477122
## Channel35    -5318.549   7534.861  -0.706 0.481716
## Channel36    10488.898   5773.159   1.817 0.071869 .
## Channel37    -8410.539   5892.265  -1.427 0.156202
## Channel38     -408.228   7970.269  -0.051 0.959241
## Channel39    19815.971  11338.219   1.748 0.083206 .
## Channel40   -23690.179  15971.026  -1.483 0.140748
## Channel41    29398.659  19340.032   1.520 0.131256
## Channel42   -32055.252  20639.448  -1.553 0.123170
## Channel43    11826.000  17491.895   0.676 0.500356
## Channel44    -9994.257  11435.392  -0.874 0.383969
## Channel45    23017.798   8927.175   2.578 0.011200 *
## Channel46    -9041.633   6218.630  -1.454 0.148705
```

```
## Channel47    -4846.799   3520.124  -1.377 0.171246
## Channel48     1536.042   4401.789   0.349 0.727764
## Channel49     2188.418   7363.225   0.297 0.766848
## Channel50   -13170.870   9829.843  -1.340 0.182947
## Channel51    26420.737  13371.372   1.976 0.050580 .
## Channel52   -23565.834  16339.395  -1.442 0.151968
## Channel53    -2005.210  16742.496  -0.120 0.904878
## Channel54    30327.413  14023.378   2.163 0.032658 *
## Channel55   -31802.344  10650.780  -2.986 0.003461 **
## Channel56    12428.271   6395.916   1.943 0.054463 .
## Channel57     -102.107   4676.993  -0.022 0.982620
## Channel58      210.251   4388.133   0.048 0.961869
## Channel59    -7679.011   4511.526  -1.702 0.091465 .
## Channel60    11590.949   3967.244   2.922 0.004199 **
## Channel61    -6559.639   3756.703  -1.746 0.083485 .
## Channel62     2533.819   3939.248   0.643 0.521370
## Channel63    11950.924   5296.267   2.256 0.025947 *
## Channel64   -18515.851   7070.171  -2.619 0.010021 *
## Channel65     4051.697   8539.248   0.474 0.636066
## Channel66      222.861   9691.472   0.023 0.981694
## Channel67    10439.030  10111.231   1.032 0.304061
## Channel68   -22570.742   9493.417  -2.378 0.019094 *
## Channel69    17285.149   8168.742   2.116 0.036520 *
## Channel70      -45.036   7357.838  -0.006 0.995127
## Channel71    -8134.714   6796.093  -1.197 0.233802
## Channel72    -1768.780   6344.295  -0.279 0.780905
## Channel73    15744.948   5531.706   2.846 0.005246 **
## Channel74   -11219.545   5666.910  -1.980 0.050132 .
## Channel75     5289.427   5067.718   1.044 0.298810
## Channel76    -2454.612   4760.274  -0.516 0.607101
## Channel77      740.608   4922.688   0.150 0.880677
## Channel78    -5730.806   5518.607  -1.038 0.301257
## Channel79    12166.493   6026.835   2.019 0.045863 *
## Channel80   -22688.979   7023.823  -3.230 0.001616 **
## Channel81    14991.763   8595.338   1.744 0.083824 .
## Channel82     3331.367   9984.910   0.334 0.739264
## Channel83    -6651.082  11358.746  -0.586 0.559337
## Channel84    -6752.949  12405.922  -0.544 0.587276
## Channel85    16271.066  12434.546   1.309 0.193323
## Channel86     5512.031  13689.180   0.403 0.687955
## Channel87   -21092.220  15770.171  -1.337 0.183730
## Channel88     9657.690  15143.593   0.638 0.524921
## Channel89      273.586  13103.448   0.021 0.983379
## Channel90    -5489.915  13927.199  -0.394 0.694180
## Channel91     2891.941  15479.740   0.187 0.852133
## Channel92    10160.850  14407.777   0.705 0.482103
## Channel93    -3183.235  11882.686  -0.268 0.789269
## Channel94    -7330.650  10959.287  -0.669 0.504913
## Channel95     5551.521   9450.485   0.587 0.558075
## Channel96    -3320.415   8349.562  -0.398 0.691613
## Channel97    -2512.787   7974.922  -0.315 0.753272
## Channel98    -5979.563   7355.289  -0.813 0.417935
## Channel99     8283.253   7911.765   1.047 0.297336
## Channel100    -101.926   3591.166  -0.028 0.977407
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.22 on 114 degrees of freedom
## Multiple R-squared:  0.9951, Adjusted R-squared:  0.9908
## F-statistic:   232 on 100 and 114 DF,  p-value: < 2.2e-16
```

```
stepwise <- stepAIC(full_model, direction = "both", trace = FALSE)
summary(stepwise)
```

```
##
## Call:
## lm(formula = Fat ~ Channel1 + Channel2 + Channel4 + Channel5 +
##     Channel7 + Channel8 + Channel11 + Channel12 + Channel13 +
##     Channel14 + Channel15 + Channel17 + Channel19 + Channel20 +
##     Channel22 + Channel24 + Channel25 + Channel26 + Channel28 +
##     Channel29 + Channel30 + Channel32 + Channel34 + Channel36 +
##     Channel37 + Channel39 + Channel40 + Channel41 + Channel42 +
##     Channel45 + Channel46 + Channel47 + Channel48 + Channel50 +
##     Channel51 + Channel52 + Channel54 + Channel55 + Channel56 +
##     Channel59 + Channel60 + Channel61 + Channel63 + Channel64 +
##     Channel65 + Channel67 + Channel68 + Channel69 + Channel71 +
##     Channel73 + Channel74 + Channel78 + Channel79 + Channel80 +
##     Channel81 + Channel84 + Channel85 + Channel87 + Channel88 +
##     Channel92 + Channel94 + Channel98 + Channel99, data = tecator_q4)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.82961 -0.57129 -0.00696  0.58152  2.86375
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)      7.093      1.453   4.882 2.64e-06 ***
## Channel1     10559.894   2333.430   4.525 1.21e-05 ***
## Channel2    -12636.967   3467.995  -3.644 0.000369 ***
## Channel4      8489.323   4637.993   1.830 0.069164 .
## Channel5    -10408.967   4771.350  -2.182 0.030689 *
## Channel7     -5376.018   3851.782  -1.396 0.164847
## Channel8      7215.595   4246.489   1.699 0.091342 .
## Channel11    -9505.520   5721.115  -1.661 0.098692 .
## Channel12    37240.918  12290.648   3.030 0.002878 **
## Channel13   -41564.547  15892.375  -2.615 0.009817 **
## Channel14    34938.179  13290.454   2.629 0.009454 **
## Channel15   -23761.451   6584.006  -3.609 0.000417 ***
## Channel17     4296.572   3189.730   1.347 0.179998
## Channel19    14279.808   5017.407   2.846 0.005042 **
## Channel20   -23855.616   5153.161  -4.629 7.85e-06 ***
## Channel22    18444.906   3381.683   5.454 1.97e-07 ***
## Channel24   -20138.426   4946.417  -4.071 7.52e-05 ***
## Channel25    18137.432   5374.094   3.375 0.000938 ***
## Channel26    -7670.318   3859.006  -1.988 0.048660 *
## Channel28    20079.898   4991.631   4.023 9.06e-05 ***
## Channel29   -36351.014   7655.223  -4.749 4.72e-06 ***
## Channel30    18071.276   5863.802   3.082 0.002446 **
## Channel32     3838.013   2722.862   1.410 0.160729
```

```
## Channel34    -9242.884   2225.926  -4.152 5.48e-05 ***
## Channel36     8070.938   3317.588   2.433 0.016152 *
## Channel37    -9045.588   3536.621  -2.558 0.011522 *
## Channel39    18664.454   5986.730   3.118 0.002183 **
## Channel40   -20069.709  10701.902  -1.875 0.062677 .
## Channel41    22257.776  11122.533   2.001 0.047169 *
## Channel42   -21760.853   5833.811  -3.730 0.000270 ***
## Channel45    18145.804   2985.416   6.078 9.50e-09 ***
## Channel46    -8225.696   3715.367  -2.214 0.028330 *
## Channel47    -4986.549   2558.694  -1.949 0.053165 .
## Channel48     2876.075   2014.985   1.427 0.155546
## Channel50   -13009.410   4535.797  -2.868 0.004720 **
## Channel51    29251.161   6554.297   4.463 1.57e-05 ***
## Channel52   -26833.976   4389.473  -6.113 7.97e-09 ***
## Channel54    30954.862   4392.339   7.047 6.06e-11 ***
## Channel55   -35183.287   5646.314  -6.231 4.39e-09 ***
## Channel56    14912.986   2810.889   5.305 3.93e-07 ***
## Channel59    -8030.278   1887.431  -4.255 3.66e-05 ***
## Channel60    13071.416   2629.374   4.971 1.79e-06 ***
## Channel61    -7850.189   2246.864  -3.494 0.000625 ***
## Channel63    15059.275   3231.692   4.660 6.90e-06 ***
## Channel64   -19909.466   4727.696  -4.211 4.35e-05 ***
## Channel65     4190.184   3486.766   1.202 0.231346
## Channel67    13850.508   3909.121   3.543 0.000526 ***
## Channel68   -25873.365   5304.223  -4.878 2.69e-06 ***
## Channel69    18362.385   3331.483   5.512 1.50e-07 ***
## Channel71    -9223.910   1558.752  -5.917 2.11e-08 ***
## Channel73    12456.498   2386.255   5.220 5.82e-07 ***
## Channel74    -5624.411   1933.590  -2.909 0.004177 **
## Channel78    -7927.105   2176.860  -3.642 0.000372 ***
## Channel79    15473.188   3812.200   4.059 7.89e-05 ***
## Channel80   -22391.895   4490.714  -4.986 1.67e-06 ***
## Channel81    13852.453   3105.934   4.460 1.59e-05 ***
## Channel84   -11442.630   3457.064  -3.310 0.001167 **
## Channel85    20228.671   4081.863   4.956 1.91e-06 ***
## Channel87   -15938.315   4102.273  -3.885 0.000153 ***
## Channel88     5647.072   3236.286   1.745 0.083033 .
## Channel92     6595.995   1864.595   3.537 0.000537 ***
## Channel94    -5497.846   1847.113  -2.976 0.003397 **
## Channel98    -8728.596   2489.314  -3.506 0.000598 ***
## Channel99     8554.587   1898.010   4.507 1.31e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.107 on 151 degrees of freedom
## Multiple R-squared:  0.9947, Adjusted R-squared:  0.9925
## F-statistic: 447.9 on 63 and 151 DF,  p-value: < 2.2e-16
```

```r
# The intercept does not count as a variable, therefore subtract 1
number_of_variables <- length(stepwise$coefficients) - 1
number_of_variables
```

```
## [1] 63
```

Excluding the intercept, a total of 63 independent variables are selected in this model.

5

```r
# 4.5 ####
# Installing and importing packages
library(glmnet)
```

```
## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-16
```

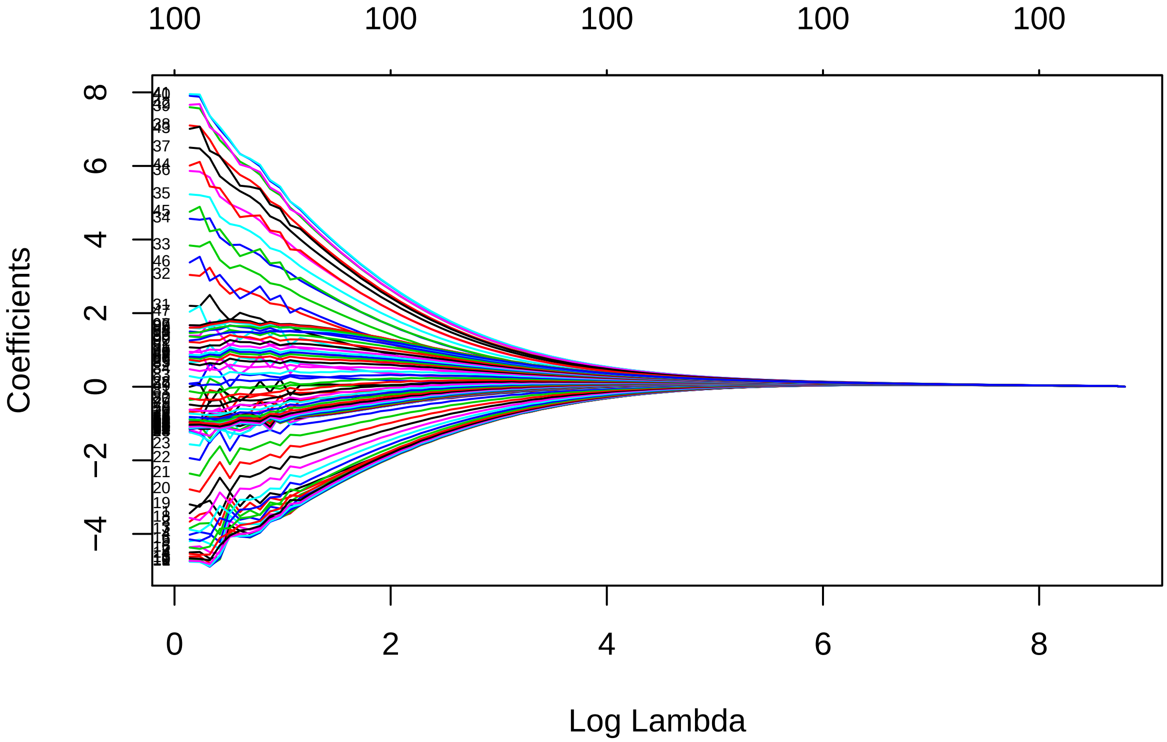
```r
# Preparing data
dim(tecator_q4)
```

```
## [1] 215 101
```

```r
covariates <- tecator_q4[, 1:100]
response <- tecator_q4[, 101]
dim(response)
```

```
## [1] 215   1
```

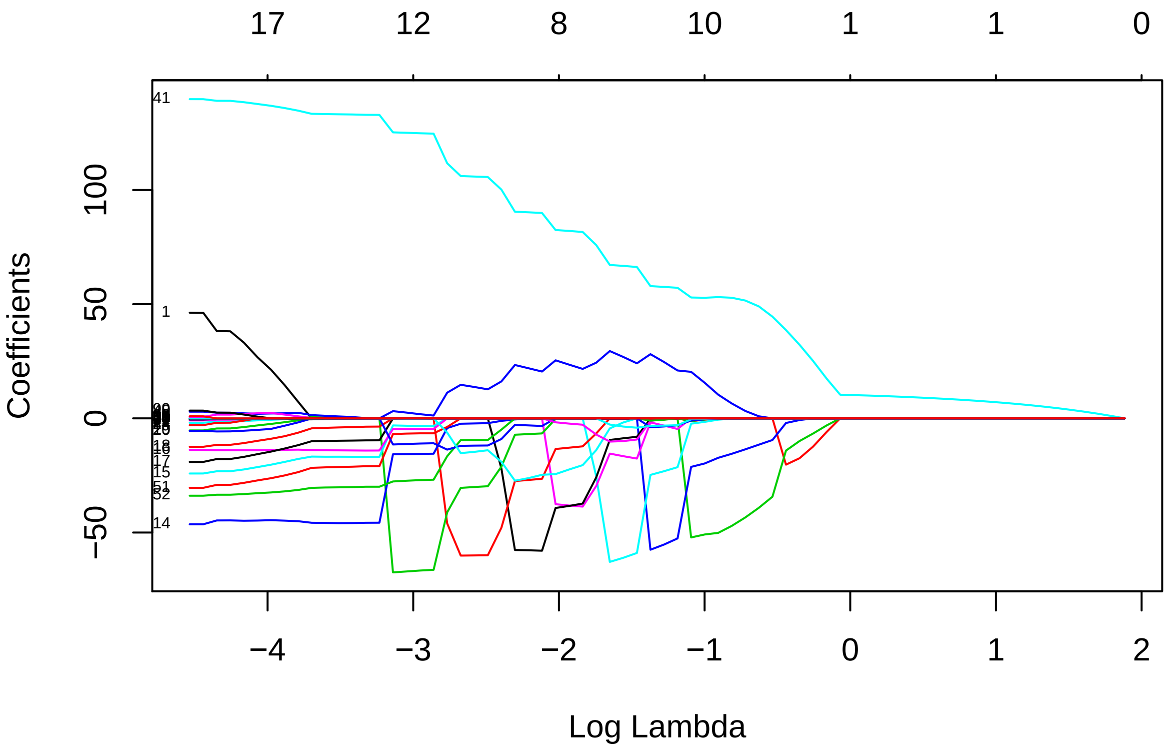
```r
# 4.5 ####
model_ridge <- glmnet(as.matrix(covariates), as.matrix(response), alpha = 0, family = "gaussian")
plot(model_ridge, xvar = "lambda", label = TRUE)
```



All coefficients converge to zero, as log(lambda) increases.

6

```r
model_lasso <- glmnet(as.matrix(covariates), as.matrix(response), alpha = 1, family = "gaussian")
plot(model_lasso, xvar = "lambda", label = TRUE)
```



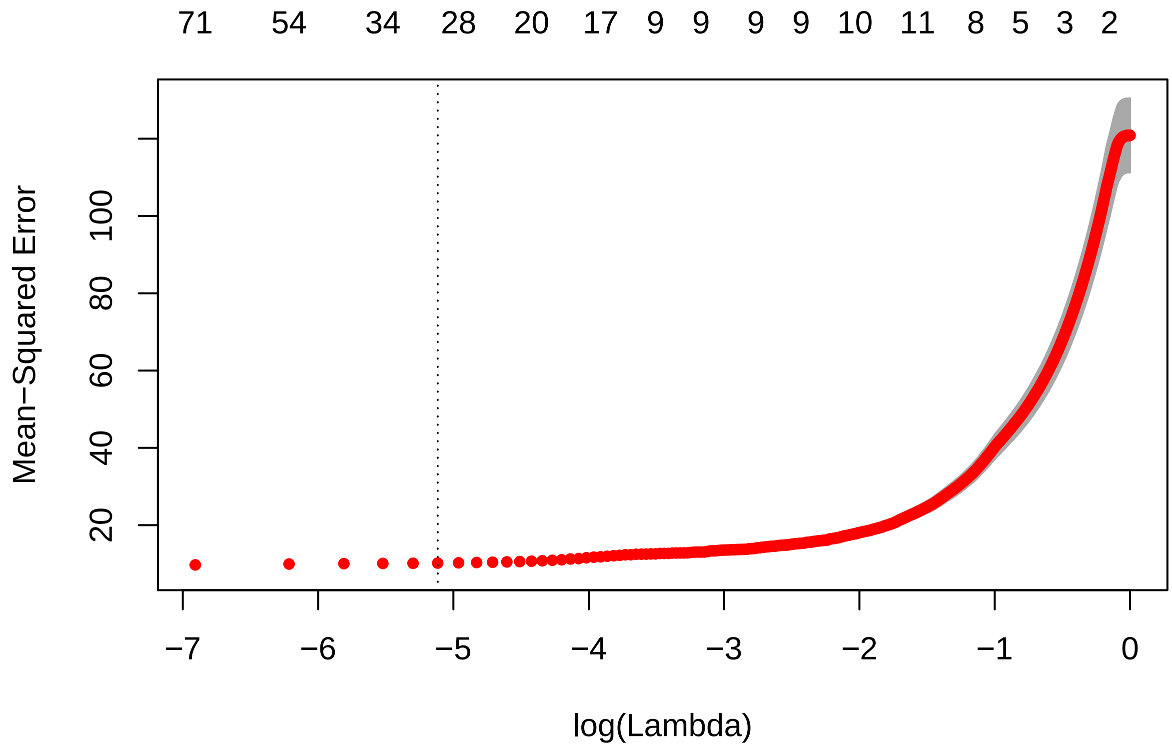
Comparing results from in the LASSO regression, less variables are selected. As log Lambda increases the coefficients in both models converge to zero. However, in the LASSO regression, the coefficients converge much faster as log(lambda) increases.

7

```r
# Use cv.glmnet function, set alpha = 1, as a LASSO model is implemented
model_cv <- cv.glmnet(as.matrix(covariates), as.matrix(response), alpha = 1, family = "gaussian",
                      lambda = seq(0,1,0.001))
model_cv$lambda.min
```

```
## [1] 0
```

```r
plot(model_cv)
```




```r
coef(model_cv, s="lambda.min")
```

```
## 101 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  1.218582e+01
## Channel1     2.243351e+01
## Channel2    -5.477212e+01
## Channel3     2.028288e+01
## Channel4     1.790966e+01
## Channel5     1.599792e+01
## Channel6     1.496901e+01
## Channel7     4.930552e+01
## Channel8     1.410886e+01
## Channel9     1.100549e+01
## Channel10   -3.946699e+01
## Channel11   -3.413441e+01
## Channel12   -3.361613e+00
## Channel13    1.501427e+02
## Channel14    2.372668e-01
## Channel15   -4.867299e+01
## Channel16   -4.549011e+01
## Channel17   -3.891421e+01
## Channel18   -3.095813e+01
## Channel19   -2.488455e+01
## Channel20   -2.152392e+01
## Channel21   -1.721212e+01
## Channel22   -1.288916e+01
## Channel23   -7.467292e+00
## Channel24    3.858146e-01
## Channel25    6.141436e+00
```

```
## Channel26    7.339759e+00
## Channel27    4.952231e+00
## Channel28    9.256704e-01
## Channel29   -1.845498e+00
## Channel30   -6.201764e+00
## Channel31   -1.240428e+01
## Channel32   -1.608200e+01
## Channel33   -1.533188e+01
## Channel34   -8.878726e+00
## Channel35   -1.222622e+00
## Channel36    4.986875e+00
## Channel37    8.663342e+00
## Channel38    9.507388e+00
## Channel39    5.835387e+00
## Channel40    5.486958e+00
## Channel41    1.437055e+02
## Channel42    2.782707e+01
## Channel43    9.118839e+00
## Channel44   -9.479487e+00
## Channel45   -2.436273e+00
## Channel46   -7.205962e+00
## Channel47   -7.485645e-04
## Channel48   -1.494150e+01
## Channel49   -1.826387e+01
## Channel50   -2.915222e+01
## Channel51   -6.977177e+01
## Channel52   -1.219494e+01
## Channel53    7.071090e+00
## Channel54   -9.397099e+00
## Channel55   -3.634639e-01
## Channel56    4.870685e+00
## Channel57    3.943612e+01
## Channel58    1.141937e+01
## Channel59    4.812258e+00
## Channel60    5.489478e+00
## Channel61    1.251985e+01
## Channel62    2.021891e+01
## Channel63   -7.405961e+00
## Channel64   -2.464353e+01
## Channel65    3.321800e+01
## Channel66   -9.834681e+00
## Channel67    1.262018e+01
## Channel68   -3.217675e+00
## Channel69    8.903740e+00
## Channel70    9.503287e-01
## Channel71    3.949972e+00
## Channel72    1.789541e+01
## Channel73   -2.043606e+01
## Channel74   -1.276925e+01
## Channel75   -8.831294e+00
## Channel76   -2.240484e+01
## Channel77    8.138153e-01
## Channel78   -2.734175e+01
## Channel79   -2.300507e+01
```

```
## Channel80    2.379017e-04
## Channel81    1.391527e-04
## Channel82    9.866988e-05
## Channel83    1.303538e-06
## Channel84   -8.303980e-05
## Channel85   -4.297569e-05
## Channel86   -3.530336e-05
## Channel87    5.470530e-05
## Channel88    8.946494e-05
## Channel89    9.648548e-05
## Channel90   -2.640082e+01
## Channel91    6.485393e-01
## Channel92    7.039651e-01
## Channel93   -1.993822e+01
## Channel94    2.455602e+01
## Channel95    2.755495e+01
## Channel96    4.056484e-02
## Channel97    8.631612e+00
## Channel98    4.537254e+00
## Channel99    5.168688e+00
## Channel100   1.640315e+01
```

The optimal lambda is 0. Excluding the intercept, a total of 100 independent variables were selected. In this dataset, this means that all the independent variables were incorporated in the model.

8

The model implemented in question 4, by means of stepAIC, selects a total of 63 independent variables, whereas the model in question 7 selects 100 independent variables.

Appendix

```r
# Import Excel file spambase.xlsx
library(readxl)
data <- read_excel("spambase.xlsx")

# Divide excel file into training and tests sets (50%/50%)
n <- dim(data)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- data[id,]
test <- data[-id, ]
# Fitting and predicting
fit <- glm(Spam ~ ., data = train, family = binomial)
predict_on_train <- predict(fit, train, type = "response")
predict_on_test <- predict(fit, test, type = "response")

# Classifying results for training and test datasets p>0.5
predict_on_train[predict_on_train > 0.5] <- 1
predict_on_train[predict_on_train <= 0.5] <- 0

predict_on_test[predict_on_test > 0.5] <- 1
```

```r
predict_on_test[predict_on_test <= 0.5] <- 0

# Confusion matrices
train_confusion <- table(train$Spam, predict_on_train)
test_confusion <- table(test$Spam, predict_on_test)

# Misclassifcation rates
total_observations_train <- sum(train_confusion)
total_observations_test <- sum(test_confusion)

fp_train <- train_confusion[1,2]
fn_train <- train_confusion[2,1]
fp_test <- test_confusion[1,2]
fn_test <- test_confusion[2,1]

missclassification_train <- (fp_train + fn_train)/total_observations_train
missclassification_test <- (fp_test + fn_test)/total_observations_test
library(knitr)
kable(train_confusion, caption = "Confusion matrix training data")
kable(test_confusion ,caption = "Confusion matrix test data")
# Fitting and predicting
fit_2 <- glm(Spam ~ ., data = train, family = binomial)
predict_on_train_2 <- predict(fit_2, train, type = "response")
predict_on_test_2 <- predict(fit_2, test, type = "response")

# Classifying results for training and test datasets p>0.9
predict_on_train_2[predict_on_train_2 > 0.9] <- 1
predict_on_train_2[predict_on_train_2 <= 0.9] <- 0

predict_on_test_2[predict_on_test_2 > 0.9] <- 1
predict_on_test_2[predict_on_test_2 <= 0.9] <- 0

# Confusion matrices
train_confusion_2 <- table(train$Spam, predict_on_train_2)
test_confusion_2 <- table(test$Spam, predict_on_test_2)

# Misclassifcation rates
total_observations_train_2 <- sum(train_confusion_2)
total_observations_test_2 <- sum(test_confusion_2)

fp_train_2 <- train_confusion_2[1,2]
fn_train_2 <- train_confusion_2[2,1]
fp_test_2 <- test_confusion_2[1,2]
fn_test_2 <- test_confusion_2[2,1]

missclassification_train_2 <- (fp_train_2 + fn_train_2)/total_observations_train_2
missclassification_test_2 <- (fp_test_2 + fn_test_2)/total_observations_test_2
kable(train_confusion_2, caption = "Confusion matrix training data")
kable(test_confusion_2, caption = "Confusion matrix test data")
library(kknn)

# Not sure whether to use train.kknn function or kknn function. For now I will go with kknn.
kknn_model <- kknn(Spam ~ ., train = train, test = test, k = 30)
```

```r
predict_on_train_3 <- fitted(kknn_model)
predict_on_test_3 <- fitted(kknn_model)

# Classifying results for training and test datasets p>0.5
predict_on_train_3[predict_on_train_3 > 0.5] <- 1
predict_on_train_3[predict_on_train_3 <= 0.5] <- 0

predict_on_test_3[predict_on_test_3 > 0.5] <- 1
predict_on_test_3[predict_on_test_3 <= 0.5] <- 0

# Confusion matrices
train_confusion_3 <- table(train$Spam, predict_on_train_3)
test_confusion_3 <- table(test$Spam, predict_on_test_3)

# Misclassifcation rates
total_observations_train_3 <- sum(train_confusion_3)
total_observations_test_3 <- sum(test_confusion_3)

fp_train_3 <- train_confusion_3[1,2]
fn_train_3 <- train_confusion_3[2,1]
fp_test_3 <- test_confusion_3[1,2]
fn_test_3 <- test_confusion_3[2,1]

missclassification_train_3 <- (fp_train_3 + fn_train_3)/total_observations_train_3
missclassification_test_3 <- (fp_test_3 + fn_test_3)/total_observations_test_3

# 1.5 ####
kknn_model_2 <- kknn(Spam ~ ., train = train, test = test, k = 1)
predict_on_train_4 <- fitted(kknn_model_2)
predict_on_test_4 <- fitted(kknn_model_2)

# Classifying results for training and test datasets p>0.5
predict_on_train_4[predict_on_train_4 > 0.5] <- 1
predict_on_train_4[predict_on_train_4 <= 0.5] <- 0

predict_on_test_4[predict_on_test_4 > 0.5] <- 1
predict_on_test_4[predict_on_test_4 <= 0.5] <- 0

# Confusion matrices
train_confusion_4 <- table(train$Spam, predict_on_train_4)
test_confusion_4 <- table(test$Spam, predict_on_test_4)

# Misclassifcation rates
total_observations_train_4 <- sum(train_confusion_4)
total_observations_test_4 <- sum(test_confusion_4)

fp_train_4 <- train_confusion_4[1,2]
fn_train_4 <- train_confusion_4[2,1]
fp_test_4 <- test_confusion_4[1,2]
fn_test_4 <- test_confusion_4[2,1]

missclassification_train_4 <- (fp_train_4 + fn_train_4)/total_observations_train_4
missclassification_test_4 <- (fp_test_4 + fn_test_4)/total_observations_test_4
```

```r
select_my_features <- function(x, y, nfolds){
  # set seed and reshuffle data
  set.seed(12345)
  intercept <- rep(1, nrow(x))
  matrix_xy <- cbind(intercept, x, y)
  n <- dim(x)[1]
  id <- sample(1:n, floor(n))
  matrix_xy <- matrix_xy[id, ]
  matrix_x <- matrix_xy[, 1:6]
  matrix_y <- matrix_xy[, 7]

  # Create folds and empty vectors
  folds <- c(1:nfolds)
  residuals_folds <- c()
  res_model <- c()
  n_features <- c()

  # Possible combinations of features including an intercept, intercept is always selected
  combinations_matrix <- expand.grid(c(T, F), c(T, F), c(T, F), c(T, F),
                                     c(T, F))
  intercept_true <- rep(TRUE, 32)
  combinations_matrix <- cbind(intercept_true, combinations_matrix)

    # Loop over each possible model
    for (i in 1:32){
      model_i <- as.logical(combinations_matrix[i,])
      data <- matrix_x[, model_i]
      folds <- c(1:nfolds)
      data_xy <- cbind(data, matrix_y, folds)
      dim_x <- ncol(data_xy) - 2

      #loop over each fold
      for (each in 1:nfolds){
        #training and test data
        train <- data_xy[data_xy[, "folds"] != each,]
        train_x <- train[, 1:dim_x]
        y_dim <- dim_x + 1
        train_y <- train[, y_dim]

        test <- data_xy[data_xy[, "folds"] == each,]
        test_x <- test[, 1:dim_x]
        test_y <- test[, y_dim]

        # computing linear regressions
        Xt_i <- t(train_x)
        XtX_i <- solve(Xt_i %*% train_x)
        betaestimates_i <- XtX_i %*% Xt_i %*% train_y
        yfit_i <- test_x %*% betaestimates_i
        res <- test_y - yfit_i
        mse <- mean(res^2)

        #storing outcomes in vectors
        residuals_folds[each] <- mse
```

```
        mean_mse <- mean(residuals_folds)
      }
      # storing outcomes in other empty vectors, one level above previous loop
   res_model[i] <- mean_mse
   n_features[i] <- dim_x - 1
    }
  # extracting the best model
  best_model <- which.min(res_model)
  possible_regressors <- colnames(matrix_x)
  x <- as.logical(combinations_matrix[best_model,])
  final_model <- possible_regressors[x]

  df <- cbind(res_model, n_features)

  # Compute end result
  list_of_results <- list(final_model)
  list_of_results$plot <- barplot(height = res_model, names.arg = n_features)
  list_of_results$cv_score <- df[best_model, 1]
  return(list_of_results)
}
swiss_y <- as.matrix(swiss[, 1])
swiss_x <- as.matrix(swiss[, 2:6])
select_my_features(swiss_x, swiss_y, 5)
library(ggplot2)
#4.1
tecator <- read_excel("tecator.xlsx")
plot <- ggplot(tecator, aes(x=Protein, y=Moisture)) + geom_point()
plot
# 4.3
# Dividing data into 50%/50% train/test
n <- dim(tecator)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- tecator[id,]
test <- tecator[-id, ]

# M1
model_m1 <- lm(Moisture ~ Protein, data = train)
m1_predict_test <- predict(model_m1, test)
m1_predict_train <- predict(model_m1, train)
m1_residual_test <- (sum((test$Moisture - m1_predict_test)^2))/nrow(test)
m1_residual_train <- (sum((train$Moisture - m1_predict_train)^2))/nrow(train)

#M2
model_m2 <- lm(Moisture ~ Protein + I(Protein^2), data = train)
m2_predict_test <- predict(model_m2, test)
m2_predict_train <- predict(model_m2, train)
m2_residual_test <- (sum((test$Moisture - m2_predict_test)^2))/nrow(test)
m2_residual_train <- (sum((train$Moisture - m2_predict_train)^2))/nrow(train)

#M3
model_m3 <- lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3), data = train)
m3_predict_test <- predict(model_m3, test)
```

```
m3_predict_train <- predict(model_m3, train)
m3_residual_test <- (sum((test$Moisture - m3_predict_test)^2))/nrow(test)
m3_residual_train <- (sum((train$Moisture - m3_predict_train)^2))/nrow(train)

#M4
model_m4 <- lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3) + I(Protein^4), data = train)
m4_predict_test <- predict(model_m4, test)
m4_predict_train <- predict(model_m4, train)
m4_residual_test <- (sum((test$Moisture - m4_predict_test)^2))/nrow(test)
m4_residual_train <- (sum((train$Moisture - m4_predict_train)^2))/nrow(train)

#M5
model_m5 <- lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3) + I(Protein^4)
               + I(Protein^5), data = train)
m5_predict_test <- predict(model_m5, test)
m5_predict_train <- predict(model_m5, train)
m5_residual_test <- (sum((test$Moisture - m5_predict_test)^2))/nrow(test)
m5_residual_train <- (sum((train$Moisture - m5_predict_train)^2))/nrow(train)

#M6
model_m6 <- lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3) + I(Protein^4)
               + I(Protein^5) + I(Protein^6), data = train)
m6_predict_test <- predict(model_m6, test)
m6_predict_train <- predict(model_m6, train)
m6_residual_test <- (sum((test$Moisture - m6_predict_test)^2))/nrow(test)
m6_residual_train <- (sum((train$Moisture - m6_predict_train)^2))/nrow(train)

# Create dataframe of MSE outcomes
mse_matrix <- matrix(NA, ncol = 3, nrow = 6)
mse_matrix
colnames(mse_matrix) <- c("train", "test", "i")
model_i <- c(1:6)
mse_matrix[,3] <- model_i
mse_matrix[1,1] <- m1_residual_train
mse_matrix[1,2] <- m1_residual_test
mse_matrix[2,1] <- m2_residual_train
mse_matrix[2,2] <- m2_residual_test
mse_matrix[3,1] <- m3_residual_train
mse_matrix[3,2] <- m3_residual_test
mse_matrix[4,1] <- m4_residual_train
mse_matrix[4,2] <- m4_residual_test
mse_matrix[5,1] <- m5_residual_train
mse_matrix[5,2] <- m5_residual_test
mse_matrix[6,1] <- m6_residual_train
mse_matrix[6,2] <- m6_residual_test

mse_df <- as.data.frame(mse_matrix)
mse_df

# Plot MSE's of different models on training and test data

plot_mse <- ggplot(data = mse_df, aes(x=i, y=train)) + geom_line(color = 'darkblue') + geom_line(aes(x=
plot_mse <- plot_mse + ylab("residual")
```

```r
plot_mse
library("MASS")

# Subsetting dataframe for the model
df_4 <- tecator
class(tecator)
tecator_q4 <- tecator[, 2:102]

full_model <- lm(Fat ~ ., data = tecator_q4)
summary(full_model)
stepwise <- stepAIC(full_model, direction = "both", trace = FALSE)
summary(stepwise)

# The intercept does not count as a variable, therefore subtract 1
number_of_variables <- length(stepwise$coefficients) - 1
number_of_variables
# 4.5 ####
# Installing and importing packages
library(glmnet)

# Preparing data
dim(tecator_q4)
covariates <- tecator_q4[, 1:100]
response <- tecator_q4[, 101]
dim(response)

# 4.5 ####
model_ridge <- glmnet(as.matrix(covariates), as.matrix(response), alpha = 0, family = "gaussian")
plot(model_ridge, xvar = "lambda", label = TRUE)
model_lasso <- glmnet(as.matrix(covariates), as.matrix(response), alpha = 1, family = "gaussian")
plot(model_lasso, xvar = "lambda", label = TRUE)
# Use cv.glmnet function, set alpha = 1, as a LASSO model is implemented
model_cv <- cv.glmnet(as.matrix(covariates), as.matrix(response), alpha = 1, family = "gaussian",
                      lambda = seq(0,1,0.001))
model_cv$lambda.min
plot(model_cv)
coef(model_cv, s="lambda.min")
```