Machine Learning Lab 01

Maximilian Pfundstein (maxpf364)

11/16/2018

Contents

Assignment 1 - Spam Classification with Nearest Neighbors

1.1

Let's import the data and have a look at it.

Table 1: spambase.xlsx

Word1	Word2	Word3	Word4	Word5	Word6	Word7	Word8	Word9	Word10
0.46	0.30	0.46	0	0.05	0.12	0.05	0.28	0.43	0.74
0.47	0.31	0.47	0	0.05	0.13	0.05	0.26	0.44	0.76
0.49	0.28	0.40	0	0.09	0.11	0.02	0.21	0.42	0.75
0.49	0.32	0.46	0	0.05	0.16	0.05	0.24	0.46	0.79
0.00	0.00	0.28	0	0.16	0.18	0.00	0.00	0.00	0.00
0.00	0.00	0.28	0	0.16	0.18	0.00	0.00	0.00	0.00

We split the data in 50% training and 50% testing.

1.2

Let's see what the confusion matrices look like for the Logistic Regression with 0.5 threshold. First one is for the training, second one is for the test data.

Table 2: Confusion Matrix (Training Data)

	Normal Mail	Spam
Classified as Spam	803	81
Classified as no-Spam	142	344

Table 3: Confusion Matrix (Test Data)

	Normal Mail	Spam
Classified as Spam	791	97
Classified as no-Spam	146	336

Misclassification rate for the training data:

```
## [1] "Spam not detected: 0.15026455026455"
```

```
## [1] "Mail missclassified as spam: 0.190588235294118"
```

```
## [1] "Misclassification rate: 0.162773722627737"
```

And now for the test data:

```
## [1] "Spam not detected: 0.155816435432231"
```

```
## [1] "Mail missclassified as spam: 0.224018475750577"
```

```
## [1] "Misclassification rate: 0.177372262773723"
```

We can observe that the model is of course working better for the training data. We have a high error rate (α and β) errors which means the model is not working that well.

1.3

Let's see what the confusion matrices look like for the Logistic Regression with 0.9 threshold. First one is for the training, second one is for the test data.

Table 4: Confusion Matrix (Training Data)

	Normal Mail	Spam
Classified as Spam	944	419
Classified as no-Spam	1	6

Table 5: Confusion Matrix (Test Data)

	Normal Mail	Spam
Classified as Spam	936	427
Classified as no-Spam	1	6

Misclassification rate for the training data:

```
## [1] "Spam not detected: 0.00105820105820106"
```

```
## [1] "Mail missclassified as spam: 0.985882352941176"
```

```
## [1] "Misclassification rate: 0.306569343065693"
```

And now for the test data:

```
## [1] "Spam not detected: 0.00106723585912487"
```

```
## [1] "Mail missclassified as spam: 0.986143187066975"
```

```
## [1] "Misclassification rate: 0.312408759124088"
```

We can see, that the overall misclassification rate increases drastically and we shifted the α and β errors. This means we have very few spam that is not detected, but this is due to the fact that almost all mail is classified as spam and therefore we have a really high error rate on mails that are classified as spam.

1.4

Train the model with K = 30 and apply it on the training and test data using kknn.

Confusion matrices:

Table 6: Confusion Matrix (Test Data)

	Normal Mail	Spam
Classified as Spam	807	98
Classified as no-Spam	138	327

Table 7: Confusion Matrix (Test Data)

	Normal Mail	Spam
Classified as Spam	672	187
Classified as no-Spam	265	246

Result training:

```
## [1] "Spam not detected: 0.146031746031746"
```

```
## [1] "Mail missclassified as spam: 0.230588235294118"
```

```
## [1] "Misclassification rate: 0.172262773722628"
```

Result test:

```
## [1] "Spam not detected: 0.28281750266809"
```

```
## [1] "Mail missclassified as spam: 0.431870669745958"
```

```
## [1] "Misclassification rate: 0.32992700729927"
```

The misclassification rate for kknn for the training set is almost the same. When trying to classifiy on the training data we see, that the misclassification rate almost doubles. It looks like this model is overfitting.

1.5

Train the model with K = 1 and apply it on the training and test data using kknn.

Confusion matrices:

Table 8: Confusion Matrix (Test Data)

	Normal Mail	Spam
Classified as Spam	945	0
Classified as no-Spam	0	425

Table 9: Confusion Matrix (Test Data)

	Normal Mail	Spam
Classified as Spam	640	177
Classified as no-Spam	297	256

Result training:

```
## [1] "Spam not detected: 0"

## [1] "Mail missclassified as spam: 0"

## [1] "Misclassification rate: 0"
```

Result test:

```
## [1] "Spam not detected: 0.316969050160085"

## [1] "Mail missclassified as spam: 0.408775981524249"

## [1] "Misclassification rate: 0.345985401459854"
```

We can see that this model is drastically overfitting. Due to k = 1 it's basically learning every point, so it's performing really well on the training set but struggles on the test set.

Assignment 3 - Feature Selection by Cross-Validation in a Linear Model

We are going to define two functions. The first one is doing the cross validation. We will use this function for the feature selection. You can find them in the appendix.

Let's have a look at the results:

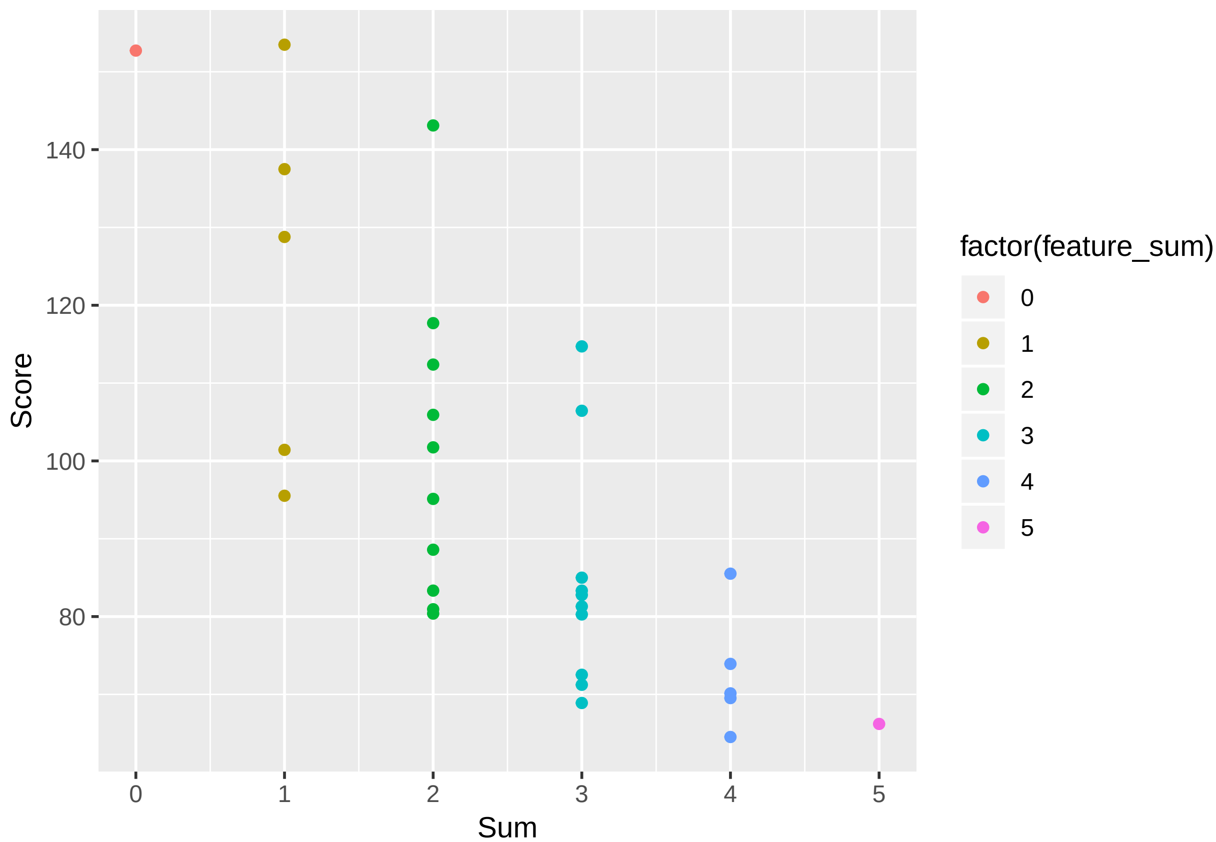


Table 10: Best Feature Selection

	Sum	Score	V3	V4	V5	V6	V7
24	4	64.5268	1	0	1	1	1

We can see that the range of the score improves with an increasing number of features until a certain threshold, which makes sense as the features give more information but as soon as you select to many features which are unrelated to the predicted value, they a noise and decrease the score again. So in this case we find the optimum with 4 features, as seen in the plot and the table. The feature that is not selected, is the second one, V4.

Assignment 4 - Linear Regression and Regularization

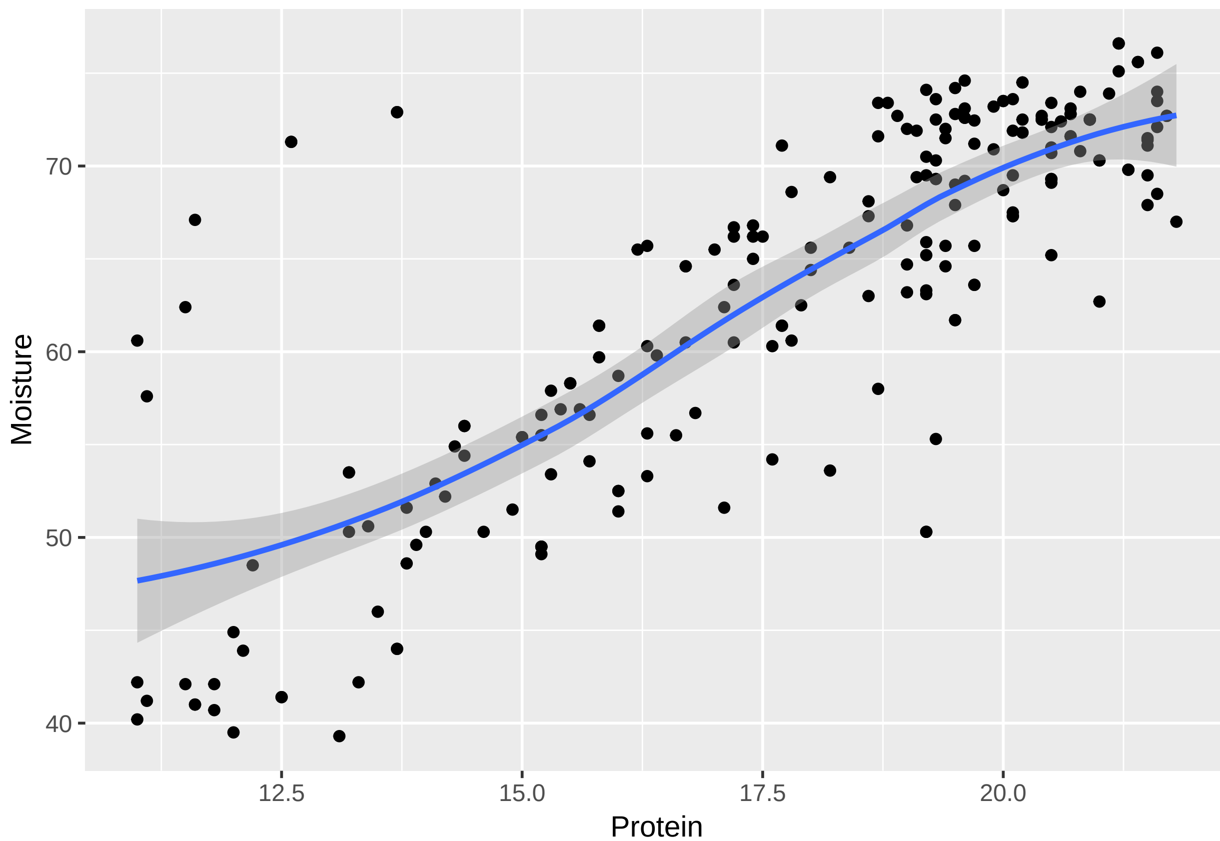
4.1

Let's import the data as well and have a look at it.

Table 11: tecator.xlsx

Sample	Channel1	Channel2	Channel3	Channel4	Channel5	Channel6	Channel7	Channel8	Channel9
1	2.61776	2.61814	2.61859	2.61912	2.61981	2.62071	2.62186	2.62334	2.62511
2	2.83454	2.83871	2.84283	2.84705	2.85138	2.85587	2.86060	2.86566	2.87093
3	2.58284	2.58458	2.58629	2.58808	2.58996	2.59192	2.59401	2.59627	2.59873
4	2.82286	2.82460	2.82630	2.82814	2.83001	2.83192	2.83392	2.83606	2.83842

Sample	Channel1	Channel2	Channel3	Channel4	Channel5	Channel6	Channel7	Channel8	Channel9
5	2.78813	2.78989	2.79167	2.79350	2.79538	2.79746	2.79984	2.80254	2.80553
6	3.00993	3.01540	3.02086	3.02634	3.03190	3.03756	3.04341	3.04955	3.05599



A linear model might work for this.

4.2

We know that the Model M_i normally distributed an has an error ϵ:

$$M_i \sim N(\mu, \sigma^2)$$

We know that the expected value is a polynomial function. The polynomal function looks like:

$$E[M_i] = \mu = \beta_0 + \beta_1 * X + \beta_2 * X^2 + ... + \beta_{i-1} * X^{i-1}$$

Let's write this down as a sum:

$$\mu = \sum_{n=0}^{i-1} \beta_n * X^n$$

If we substite μ in the formula for the normal distribution, we get our model:

$$M_i \sim N(\sum_{n=0}^{i-1} \beta_n * X^n, \sigma^2)$$

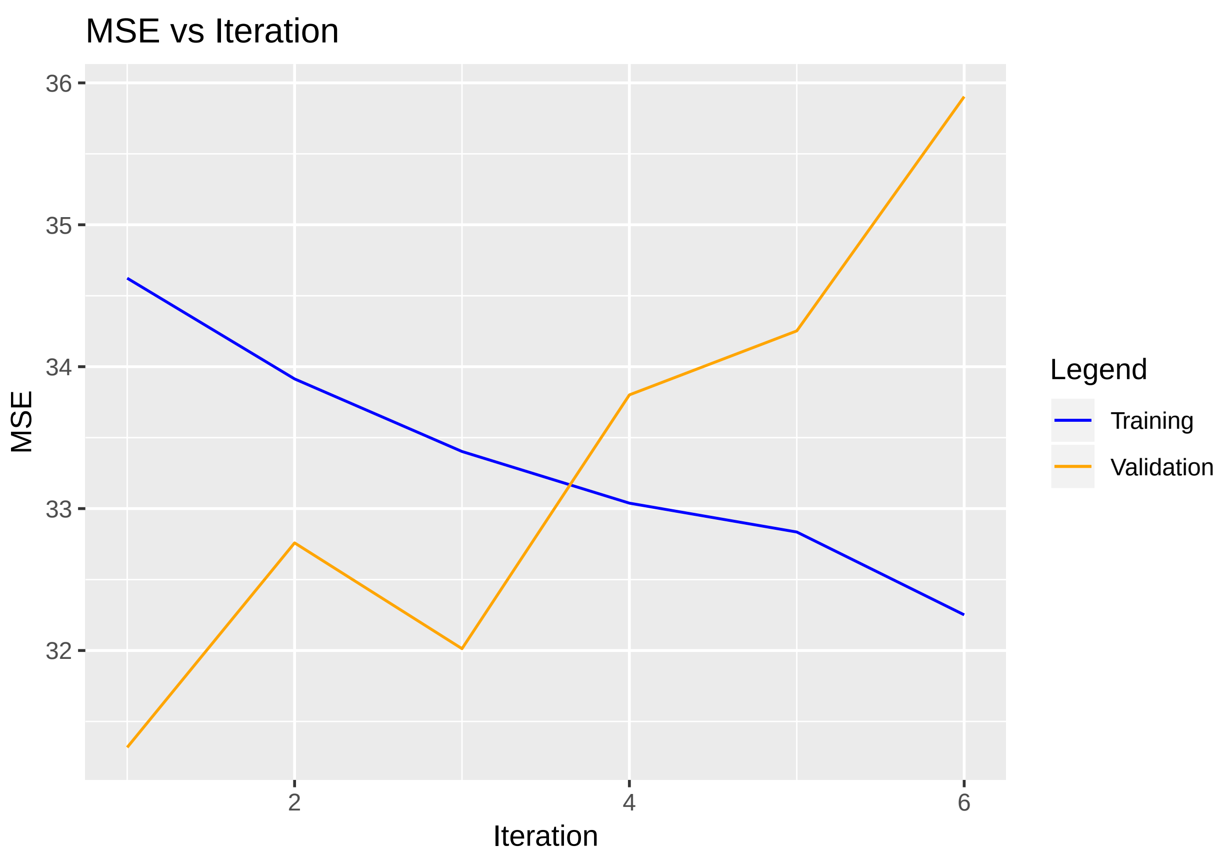
It's appropiate to use the MSE criterion due the properties that erros are always positive, so we can search for a minimum. Furthermor it highly punishes missclassified data.

4.3

Let's look at the data and plot it.

Table 12: MSE of Training and Validation

Iteration	MSE_Training	MSE_Validation
1	34.62363	31.31733
2	33.91379	32.75837
3	33.40243	32.01300
4	33.03846	33.80135
5	32.83490	34.25232
6	32.25127	35.90301



The best model is the model that performs best on the validation data. In this case it's Model M_1. The bias-variance tradeoff is basically a different viewpoint on overfitting. If a model has a high variance but performing well on the training dataset, its probably overfitting. We don't see the exact values of the model here, but we can see that while the model with increasing i is working better on the training dataset, it performs worse on the validation data set. This is exactly due to the mentioned overfitting.

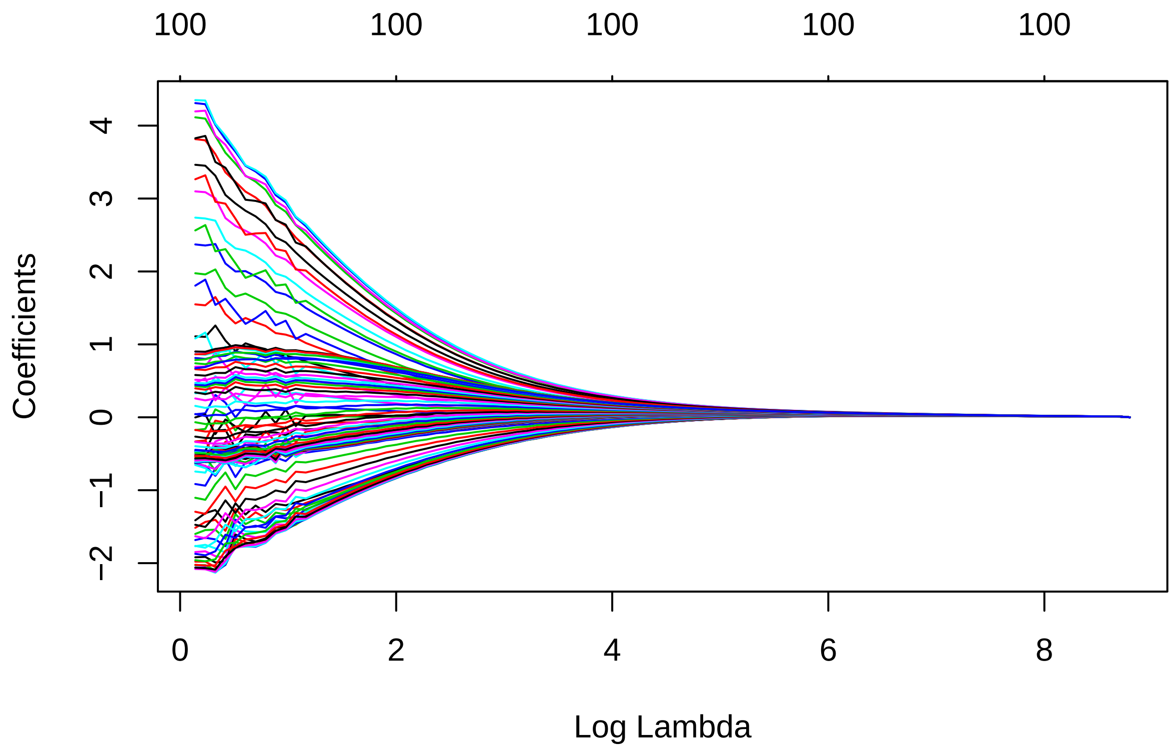
4.4

```
##
## 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_data)
##
## 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
```

The model selected 63 channels producing a standard error of 1.107 and having 151 degrees of freedom.

4.5

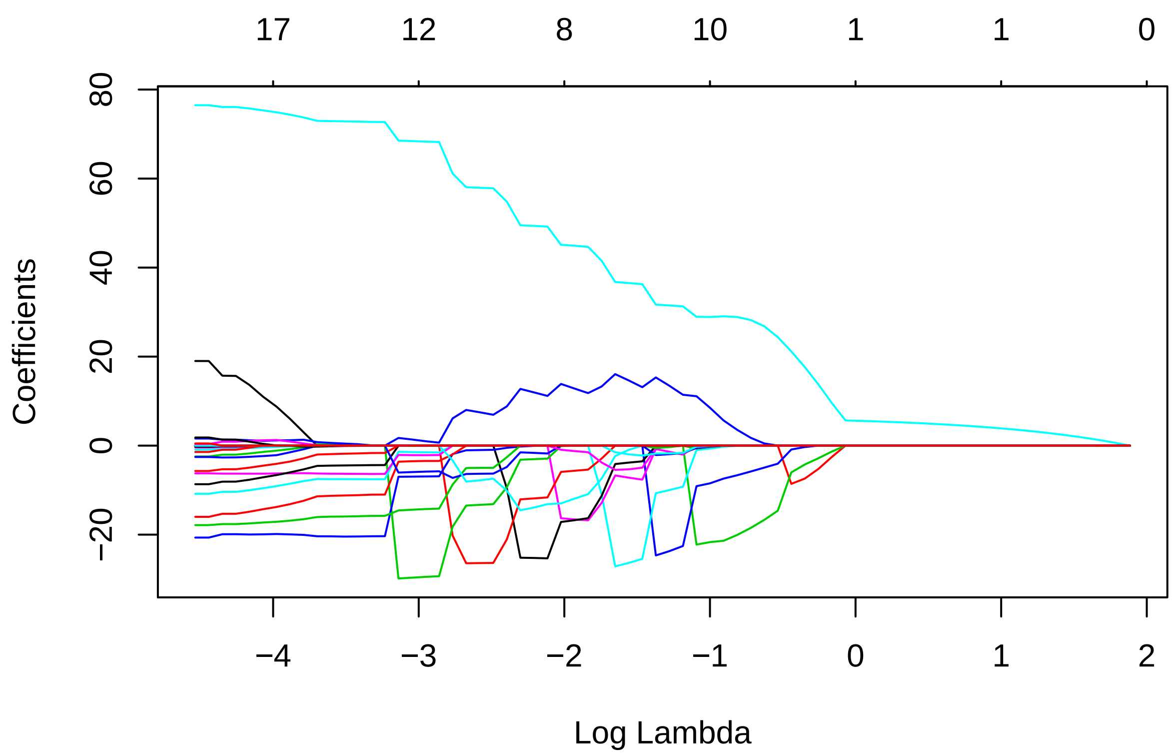


λ, also called the *Shrinkage Parameter*, penalizes the coefficients to decrease the number of coefficients to prevent overfitting. This is due to the fact that the *Shrinkage Penalty*

$$\lambda \sum_j \beta_j^2$$

is added to the term of calculating the estimates $\hat{\beta}^R$. If $\lambda = 0$ we're back to least squares estimates.

4.6

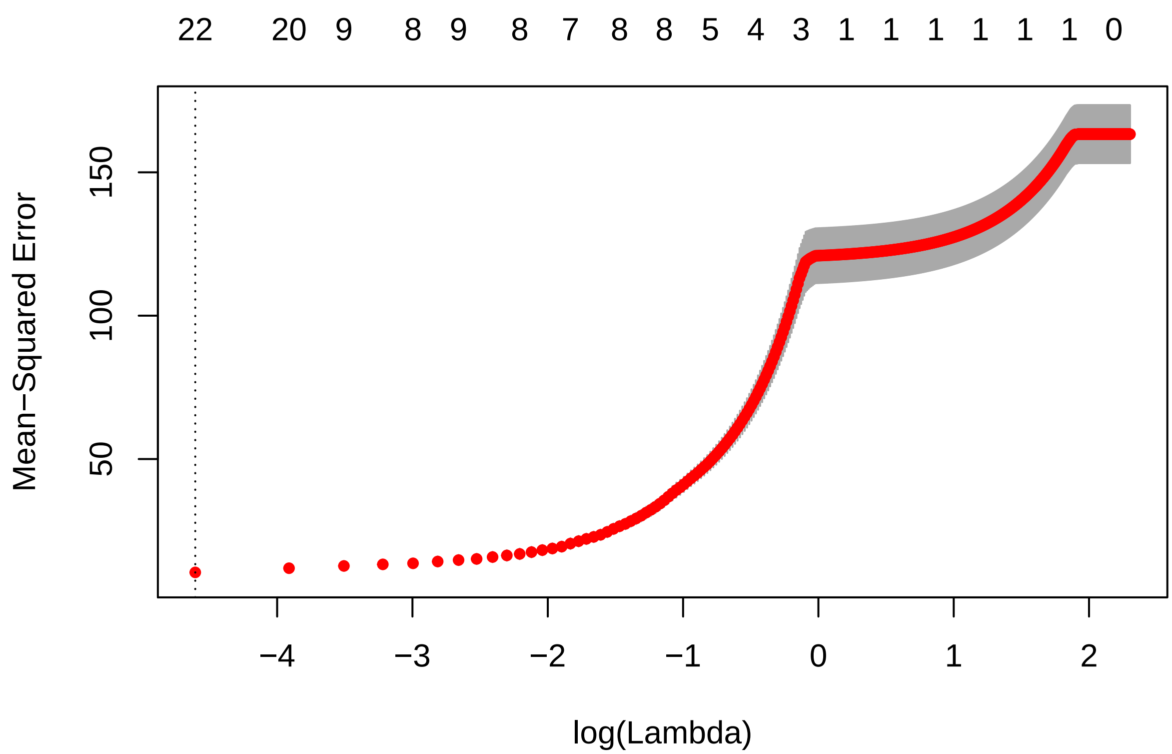


Ridge Regression has the problem, that even with high $\lambda's$ the coefficients will never be zero, this would only be the case for $lim_{\lambda \to \infty}$. This time the *Shrinkage Penalty* is

$$\lambda \sum_j |\beta_j|$$

Due to the fact that high $\lambda's$ can set the coefficients to zero, LASSO perform a *Variable Selection*. This is what can bee seen in the plot, with increasing $\ln(\lambda)$ more and more variables are set to 0.

4.7




```
## [1] "Best lambda score: 0"
```

We see that the best MSE is for $\lambda = lim_{\lambda\to 0} = 0$, so we now that we are performing least squares estimate which includes **all** coefficients.

4.8

In 4.4 63 channels are selected by the variable selection. In 4.5 the channels are not selected, but rather they're penalized, but due to the fact that they're only shrinked they don't disappear. 4.6 handles this problem as it actually can set the channels to 0 (exclude them) with increasing labda. In 4.7 we can see that for this exercise it's best to include all of the coefficients, so all of them seem to be related the the prediction parameter. This is not always the case so it's not possible to generalize how many channels/predictors should be used.

Appendix: Source Code

```
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
library(readxl)
library(MASS)
library(glmnet)
library(kknn)
```

```r
library(knitr)
set.seed(12345)
#### Assigment 1.1 ####
spambase = read_excel("./spambase.xlsx")
kable(head(spambase[,1:10]), caption = "spambase.xlsx")
set.seed(12345)
n = dim(spambase)[1]
id = sample(1:n, floor(n*0.5))
train = spambase[id,]
test = spambase[-id,]
#### Assigment 1.3 ####
spambase_model = glm(Spam ~ ., data = train, family = "binomial")

## Train
spambase_predict_train =
  predict(object = spambase_model, newdata = train, type = "response")
spambase_predict_train =
  apply(as.matrix(spambase_predict_train), c(1),
        FUN = function(x) return(x > 0.5))
confusion_matrix_train =
  as.matrix(table(spambase_predict_train, train$Spam))
colnames(confusion_matrix_train) =
  c("Normal Mail", "Spam")
rownames(confusion_matrix_train) =
  c("Classified as Spam","Classified as no-Spam")

## Test
spambase_predict_test =
  predict(object = spambase_model, newdata = test, type = "response")
spambase_predict_test =
  apply(as.matrix(spambase_predict_test), c(1),
        FUN = function(x) return(x > 0.5))
confusion_matrix_test =
  as.matrix(table(spambase_predict_test, test$Spam))
colnames(confusion_matrix_test) =
  c("Normal Mail", "Spam")
rownames(confusion_matrix_test) =
  c("Classified as Spam","Classified as no-Spam")
kable(confusion_matrix_train, caption = "Confusion Matrix (Training Data)")
kable(confusion_matrix_test,caption = "Confusion Matrix (Test Data)")
spam_not_detected_train =
  confusion_matrix_train[2,1]/sum(confusion_matrix_train[,1])
mail_missclassified_train =
  confusion_matrix_train[1,2]/sum(confusion_matrix_train[,2])

print(paste(sep = "", "Spam not detected: ",
            spam_not_detected_train))
print(paste(sep = "", "Mail missclassified as spam: ",
            mail_missclassified_train))
print(paste(sep ="", "Misclassification rate: ",
            (confusion_matrix_train[1,2]+confusion_matrix_train[2,1])/
              sum(confusion_matrix_train)))
spam_not_detected_test =
```

```
  confusion_matrix_test[2,1]/sum(confusion_matrix_test[,1])
mail_missclassified_test =
  confusion_matrix_test[1,2]/sum(confusion_matrix_test[,2])
print(paste(sep = "", "Spam not detected: ",
            spam_not_detected_test))
print(paste(sep = "", "Mail missclassified as spam: ",
            mail_missclassified_test))
print(paste(sep ="", "Misclassification rate: ",
            (confusion_matrix_test[1,2]+confusion_matrix_test[2,1])/
              sum(confusion_matrix_test)))
#### Assigment 1.3 ####
spambase_model_two = glm(Spam ~ ., data = train, family = "binomial")

## train_two
spambase_predict_train_two =
  predict(object = spambase_model_two, newdata = train, type = "response")
spambase_predict_train_two =
  apply(as.matrix(spambase_predict_train_two), c(1),
        FUN = function(x) return(x > 0.9))
confusion_matrix_train_two =
  as.matrix(table(spambase_predict_train_two, train$Spam))
colnames(confusion_matrix_train_two) =
  c("Normal Mail", "Spam")
rownames(confusion_matrix_train_two) =
  c("Classified as Spam","Classified as no-Spam")

## test_two
spambase_predict_test_two =
  predict(object = spambase_model_two, newdata = test, type = "response")
spambase_predict_test_two =
  apply(as.matrix(spambase_predict_test_two), c(1),
        FUN = function(x) return(x > 0.9))
confusion_matrix_test_two =
  as.matrix(table(spambase_predict_test_two, test$Spam))
colnames(confusion_matrix_test_two) =
  c("Normal Mail", "Spam")
rownames(confusion_matrix_test_two) =
  c("Classified as Spam","Classified as no-Spam")
kable(confusion_matrix_train_two, caption = "Confusion Matrix (Training Data)")
kable(confusion_matrix_test_two, caption = "Confusion Matrix (Test Data)")
spam_not_detected_train_two =
  confusion_matrix_train_two[2,1]/sum(confusion_matrix_train_two[,1])
mail_missclassified_train_two =
  confusion_matrix_train_two[1,2]/sum(confusion_matrix_train_two[,2])
print(paste(sep = "", "Spam not detected: ", spam_not_detected_train_two))
print(paste(sep = "", "Mail missclassified as spam: ",
            mail_missclassified_train_two))
print(paste(sep ="", "Misclassification rate: ",
            (confusion_matrix_train_two[1,2]+confusion_matrix_train_two[2,1])/
              sum(confusion_matrix_train_two)))
spam_not_detected_test_two =
  confusion_matrix_test_two[2,1]/sum(confusion_matrix_test_two[,1])
mail_missclassified_test_two =
```

```r
  confusion_matrix_test_two[1,2]/sum(confusion_matrix_test_two[,2])
print(paste(sep = "", "Spam not detected: ",
            spam_not_detected_test_two))
print(paste(sep = "", "Mail missclassified as spam: ",
            mail_missclassified_test_two))
print(paste(sep ="", "Misclassification rate: ",
            (confusion_matrix_test_two[1,2]+confusion_matrix_test_two[2,1])/
              sum(confusion_matrix_test_two)))
#### Assigment 1.4 ####
kknn_model_train = kknn(formula = Spam ~ ., train = train, test = train, k = 30)
kknn_model_test = kknn(formula = Spam ~ ., train = train, test = test, k = 30)

# Train
y_hat_train =
  apply(as.matrix(kknn_model_train$fitted.values), c(1),
        FUN = function(x) return(x > 0.5))
confusion_matrix_kkn_train = as.matrix(table(y_hat_train, train$Spam))
colnames(confusion_matrix_kkn_train) =
  c("Normal Mail", "Spam")
rownames(confusion_matrix_kkn_train) =
  c("Classified as Spam","Classified as no-Spam")

# Test
y_hat_test =
  apply(as.matrix(kknn_model_test$fitted.values), c(1),
        FUN = function(x) return(x > 0.5))
confusion_matrix_kkn_test = as.matrix(table(y_hat_test, test$Spam))
colnames(confusion_matrix_kkn_test) =
  c("Normal Mail", "Spam")
rownames(confusion_matrix_kkn_test) =
  c("Classified as Spam","Classified as no-Spam")
kable(confusion_matrix_kkn_train, caption = "Confusion Matrix (Test Data)")
kable(confusion_matrix_kkn_test, caption = "Confusion Matrix (Test Data)")
spam_not_detected_train_kknn =
  confusion_matrix_kkn_train[2,1]/sum(confusion_matrix_kkn_train[,1])
mail_missclassified_train_kknn =
  confusion_matrix_kkn_train[1,2]/sum(confusion_matrix_kkn_train[,2])
print(paste(sep = "", "Spam not detected: ",
            spam_not_detected_train_kknn))
print(paste(sep = "", "Mail missclassified as spam: ",
            mail_missclassified_train_kknn))
print(paste(sep ="", "Misclassification rate: ",
            (confusion_matrix_kkn_train[1,2]+confusion_matrix_kkn_train[2,1])/
              sum(confusion_matrix_kkn_train)))
spam_not_detected_test_kknn =
  confusion_matrix_kkn_test[2,1]/sum(confusion_matrix_kkn_test[,1])
mail_missclassified_test_kknn =
  confusion_matrix_kkn_test[1,2]/sum(confusion_matrix_kkn_test[,2])
print(paste(sep = "", "Spam not detected: ",
            spam_not_detected_test_kknn))
print(paste(sep = "", "Mail missclassified as spam: ",
            mail_missclassified_test_kknn))
print(paste(sep ="", "Misclassification rate: ",
```

```r
            (confusion_matrix_kkn_test[1,2]+confusion_matrix_kkn_test[2,1])/
              sum(confusion_matrix_kkn_test)))
#### Assigment 1.5 ####
kknn_model_train_1 =
  kknn(formula = Spam ~ ., train = train, test = train, k = 1)
kknn_model_test_1 =
  kknn(formula = Spam ~ ., train = train, test = test, k = 1)

# Train
y_hat_train_1 =
  apply(as.matrix(kknn_model_train_1$fitted.values), c(1),
        FUN = function(x) return(x > 0.5))
confusion_matrix_kkn_train_1 = as.matrix(table(y_hat_train_1, train$Spam))
colnames(confusion_matrix_kkn_train_1) =
  c("Normal Mail", "Spam")
rownames(confusion_matrix_kkn_train_1) =
  c("Classified as Spam","Classified as no-Spam")

# Test
y_hat_test_1 =
  apply(as.matrix(kknn_model_test_1$fitted.values), c(1),
        FUN = function(x) return(x > 0.5))
confusion_matrix_kkn_test_1 = as.matrix(table(y_hat_test_1, test$Spam))
colnames(confusion_matrix_kkn_test_1) =
  c("Normal Mail", "Spam")
rownames(confusion_matrix_kkn_test_1) =
  c("Classified as Spam","Classified as no-Spam")
kable(confusion_matrix_kkn_train_1, caption = "Confusion Matrix (Test Data)")
kable(confusion_matrix_kkn_test_1, caption = "Confusion Matrix (Test Data)")
spam_not_detected_train_kknn_1 =
  confusion_matrix_kkn_train_1[2,1]/sum(confusion_matrix_kkn_train_1[,1])
mail_missclassified_train_kknn_1 =
  confusion_matrix_kkn_train_1[1,2]/sum(confusion_matrix_kkn_train_1[,2])
print(paste(sep = "", "Spam not detected: ",
            spam_not_detected_train_kknn_1))
print(paste(sep = "", "Mail missclassified as spam: ",
            mail_missclassified_train_kknn_1))
print(paste(sep ="", "Misclassification rate: ",
        (confusion_matrix_kkn_train_1[1,2]+confusion_matrix_kkn_train_1[2,1])/
          sum(confusion_matrix_kkn_train_1)))
spam_not_detected_test_kknn_1 =
  confusion_matrix_kkn_test_1[2,1]/sum(confusion_matrix_kkn_test_1[,1])
mail_missclassified_test_kknn_1 =
  confusion_matrix_kkn_test_1[1,2]/sum(confusion_matrix_kkn_test_1[,2])
print(paste(sep = "", "Spam not detected: ",
            spam_not_detected_test_kknn_1))
print(paste(sep = "", "Mail missclassified as spam: ",
            mail_missclassified_test_kknn_1))
print(paste(sep ="", "Misclassification rate: ",
          (confusion_matrix_kkn_test_1[1,2]+confusion_matrix_kkn_test_1[2,1])/
            sum(confusion_matrix_kkn_test_1)))
#### Assigment 3 ####
c_cross_validation = function(k = 5, Y, X) {
```

```
if (!is.numeric(X) && ncol(X) == 0) {
  y_hat = mean(Y)
  return(mean((y_hat-Y)^2))
}

Y = as.matrix(Y)
X = as.matrix(X)
X = cbind(1, X)

# Create a list of 5 matrices with the appropiate
# size (these will hold the subsets)
X_subsets = list()
Y_subsets = list()

# We fill the list entries with the subsets
for (i in 1:k) {
  percentage_marker = nrow(X)/k
  start = floor(percentage_marker*(i-1)+1)
  end = floor(percentage_marker*i)
  X_subsets[[i]] = X[start:end,]
  Y_subsets[[i]] = Y[start:end]
}

# Now we take one matrix at a time for training and
# everything else as the testing
scores = 0
for (i in 1:k) {

  ## Initial
  X_train = matrix(0, ncol = ncol(X))
  Y_train = c()

  # Get validation and training data
  current_subset_X = X_subsets[-i]
  current_subset_Y = Y_subsets[-i]
  for (j in 1:(length(X_subsets)-1)) {
    X_train = rbind(X_train, current_subset_X[[j]])
    Y_train = c(Y_train, current_subset_Y[[j]])
  }

  # Because of R
  X_train = X_train[-1,]

  # Model
  betas =
    as.matrix((solve(t(X_train) %*% X_train)) %*% t(X_train) %*% Y_train)

  ## Select the training data and transform them to
  # one matrix X_test and one vector Y_test
  X_val = X_subsets[[i]]
  Y_val = Y_subsets[[i]]

  ## Now we get our y_hat and y_real. y_real is
```

```
    # only used to clarify the meaning
    y_hat = as.vector(X_val %*% betas)
    y_real = Y_val

    ## Get MSE and add to the scores list
    scores = c(scores, mean((y_hat - y_real)^2))

  }
  # Return the mean of our scores
  scores = scores[-1]
  return(mean(scores))
}

c_best_subset_selection = function(Y, X) {

  # Shuffle X and Y via indexes
  ids = sample(x = 1:nrow(X), nrow(X))
  X = X[ids,]
  Y = Y[ids]

  # Get all combinations
  comb_matrix = matrix(0, ncol = ncol(X))
  for (i in c(1:(2^(ncol(X))-1))) {
    comb_matrix =
      rbind(comb_matrix, tail(rev(as.numeric(intToBits(i))), ncol(X)))
  }

  results = c()

  # Do cross validation for each feature set
  for (j in 1:nrow(comb_matrix)) {
    comb = as.logical(comb_matrix[j,])
    feature_select = X[,comb]
    res = c_cross_validation(5, Y, feature_select)
    results = c(results, res)
  }
  models = matrix(results, ncol = 1)
  models = cbind(models, comb_matrix)

  # Add column with the sum of the features for plotting
  feature_sum = c()
  for (k in 1:nrow(comb_matrix)) {
    row_sum = sum(comb_matrix[k,])
    feature_sum = c(feature_sum, row_sum)
  }
  models = as.data.frame(cbind(feature_sum, models))
  colnames(models)[1:2] = c("Sum", "Score")
  print(ggplot(models, aes(x = Sum, y = Score, colour = factor(feature_sum))) +
    geom_point())
   stat_summary(fun.y = min, colour = "red", geom = "point", size = 5)
  return(models[min(models[,2]) == models[,2],])
}
kable(c_best_subset_selection(swiss[,1], swiss[,2:6]), caption = "Best Feature Selection")
```

```r
#### Assigment 4.1 ####
tecator_data = read_excel("./tecator.xlsx")
kable(head(tecator_data[,1:10]), caption = "tecator.xlsx")
ggplot(tecator_data, aes(x = Protein, y = Moisture)) +
  geom_point() + geom_smooth()

#### Assigment 4.3 ####
n = 6
model_tecator_data = data.frame(Y = tecator_data$Moisture)
for (i in c(1:n)) {
  model_tecator_data = data.frame(model_tecator_data, (tecator_data$Protein)^i)
}

for (i in c(1:n)) {
  names(model_tecator_data)[i+1] = paste("Protein", i, sep="")
}

# Shuffle
ids = sample(x = 1:nrow(model_tecator_data), nrow(model_tecator_data))
model_tecator_data = model_tecator_data[ids,]
tecator_data_training =
  model_tecator_data[1:(nrow(model_tecator_data)/2),]
tecator_data_validation =
  model_tecator_data[(nrow(model_tecator_data)/2):nrow(model_tecator_data),]
results = data.frame(Iteration = as.character(),
                     Training_SSE = as.numeric(), Validation_SSE = as.numeric())
for (i in c(1:n)) {

  formula = "Y ~ Protein1"

  if (i != 1) {
    for (j in c(2:i)) {
      formula = paste(formula, " + Protein", j, sep="")
    }
  }

  linreg = lm(as.formula(formula), data = tecator_data_training)
  mse_training = mean(linreg$residuals^2)
  y_hat = predict(object = linreg, newdata = tecator_data_validation)
  mse_validation = mean((y_hat - tecator_data_validation$Y)^2)
  results = rbind(results, list(i, mse_training, mse_validation))
}
names(results) = list("Iteration", "MSE_Training", "MSE_Validation")
kable(results, caption = "MSE of Training and Validation")
ggplot(results) +
  geom_line(aes(x = Iteration, y = MSE_Training, colour = "Training")) +
  geom_line(aes(x = Iteration, y = MSE_Validation, colour = "Validation")) +
  labs(title="MSE vs Iteration", y="MSE", x="Iteration", color = "Legend") +
  scale_color_manual(values = c("blue", "orange"))
#### Assigment 4.4 ####
# Filtering of colums and then just using "Fat ~ ." would also be possible.
c_formula = "Fat ~ Channel1"
for (i in 2:100) {
```

```r
  c_formula = paste(c_formula, " + Channel", i, sep = "")
}
model = lm(formula = c_formula, data = tecator_data)
model.stepAIC = stepAIC(model, direction = c("both"), trace = FALSE)
summary(model.stepAIC)
#### Assigment 4.5 ####
covariates_ridge = scale(tecator_data[,2:(ncol(tecator_data)-3)])
response_ridge = tecator_data$Fat

glm_model_ridge = glmnet(as.matrix(covariates_ridge),
                         response_ridge, alpha = 0, family="gaussian")
plot(glm_model_ridge, xvar="lambda")
#### Assigment 4.6 ####
covariates_lasso = scale(tecator_data[,2:(ncol(tecator_data)-3)])
response_lasso = tecator_data$Fat

glm_model_lasso = glmnet(as.matrix(covariates_ridge),
                         response_ridge, alpha = 1, family="gaussian")
plot(glm_model_lasso, xvar="lambda")
#### Assigment 4.7 ####
lasso_vc = cv.glmnet(y = response_lasso, x = covariates_lasso,
                     alpha = 1, lambda = seq(from = 0, to = 10, by = 0.01))
plot(lasso_vc)
print(paste(sep = "", "Best lambda score: ", lasso_vc$lambda.min))
```