machine learning(732A99) lab1

Anubhav Dikshit(anudi287), Lennart Schilling(lensc874), Thijs Quast(thiqu264)

26 November 2018

Contents

Contributions

During the lab, Lenart focused on assignment 1, Thijs focused on assignment 3 and Anuhav focused on assignment 4. All codes and analysis was indepedently done and is also reflected in the individual reports.

Assignment 1

1.1 Import the data into R and divide it into training and test sets (50%/50%) by using the following code

At first, the data from the Excel file *spambase.xlsx* will be imported and splitted into train and test data (50%:50%)

```
# Importing data
data = read.xlsx("spambase.xlsx", sheetName = "spambase_data")
tecator_data <- read.xlsx("tecator.xlsx", sheetName = "data")
tecator_data <- tecator_data[,2:NCOL(tecator_data)] # removing sample column

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

1.2 Use logistic regression (functions glm(), predict()) to classify the training and test data by the classification principles and report the confusion matrices (use table()) and the misclassification rates for training and test data. Analyse the obtained results.

Using the train data, a logistic regression model will be created. Analysing the p-values of the coefficients, it can be seen which independent variables have a significant influence on the dependent variable *Spam*. For the sake of clarity, this is not explicitly stated in this report.

```
# Fitting model
logitModel = glm(Spam ~ ., data = train, family = binomial)
summary(logitModel)
```

```
##
## Call:
## glm(formula = Spam ~ ., family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5205  -0.4402  -0.0005   0.6584   3.6196
##
## Coefficients:
##                 Estimate    Std. Error z value           Pr(>|z|)
## (Intercept)     1.5081289     0.2011152   7.499 0.0000000000000644 ***
## Word1          -0.7191881     0.5014624  -1.434           0.151520
## Word2           0.0399446     0.3014438   0.133           0.894580
## Word3          -0.3528561     0.1839013  -1.919           0.055019 .
## Word4           0.1369914     0.1117461   1.226           0.220230
## Word5           0.1221434     0.1413138   0.864           0.387400
## Word6           0.2887448     0.4153010   0.695           0.486888
## Word7          -0.2947580     0.3348237  -0.880           0.378676
## Word8          -0.1033859     0.3509944  -0.295           0.768337
## Word9          -0.1084832     0.4053447  -0.268           0.788983
```

```
## Word10         -0.0309116     0.1668155  -0.185           0.852991
## Word11         -0.6088010     0.6789682  -0.897           0.369902
## Word12          0.1614068     0.1072619   1.505           0.132378
## Word13          0.7810953     0.3572174   2.187           0.028771 *
## Word14         -0.4818689     0.3003186  -1.605           0.108598
## Word15         -0.1305229     0.3884464  -0.336           0.736861
## Word16          0.3170597     0.2331654   1.360           0.173891
## Word17         -0.0920067     0.2822914  -0.326           0.744479
## Word18          0.0232959     0.2321682   0.100           0.920074
## Word19          0.0003694     0.0574594   0.006           0.994871
## Word20          0.0168793     0.3181387   0.053           0.957687
## Word21         -0.0282058     0.1072320  -0.263           0.792524
## Word22         -0.4767281     0.3200436  -1.490           0.136337
## Word23          0.2541081     0.3412766   0.745           0.456525
## Word24         -0.2483026     0.6254580  -0.397           0.691372
## Word25         -0.0782762     0.0585199  -1.338           0.181027
## Word26          0.0037333     0.1358210   0.027           0.978071
## Word27         -0.2238407     0.1077417  -2.078           0.037749 *
## Word28          0.1319556     0.1880126   0.702           0.482776
## Word29         -0.0813140     0.0911918  -0.892           0.372564
## Word30         -1.8150700     0.6195133  -2.930           0.003391 **
## Word31         -4.6944446     1.8530163  -2.533           0.011296 *
## Word32       -119.4495067 15134.1847623  -0.008           0.993703
## Word33         -2.8994424     0.6794146  -4.268 0.0000197622951954 ***
## Word34         -3.7098525     4.3523571  -0.852           0.394004
## Word35         -7.0325274     1.9964683  -3.522           0.000428 ***
## Word36         -1.6779354     0.3810251  -4.404 0.0000106400723118 ***
## Word37         -0.8583374     0.2174799  -3.947 0.0000792212684576 ***
## Word38         -0.6043466     1.2790588  -0.472           0.636575
## Word39         -1.8771543     0.4281730  -4.384 0.0000116464815552 ***
## Word40          0.0739289     0.3400368   0.217           0.827885
## Word41       -332.5541421 16559.5233247  -0.020           0.983978
## Word42         -5.3516109     1.3024490  -4.109 0.0000397576929183 ***
## Word43         -2.5919298     0.7352904  -3.525           0.000423 ***
## Word44         -2.9313923     0.6600870  -4.441 0.0000089575885095 ***
## Word45         -1.1408036     0.1681342  -6.785 0.0000000000116025 ***
## Word46         -3.2880431     0.5178052  -6.350 0.0000000002153691 ***
## Word47         -3.7410844     2.0304190  -1.843           0.065399 .
## Word48         -4.3898995     1.4729045  -2.980           0.002878 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1696.82  on 1369  degrees of freedom
## Residual deviance:  928.54  on 1321  degrees of freedom
## AIC: 1026.5
##
## Number of Fisher Scoring iterations: 23
```

In the next step, the *logitModel* will be used to classify emails of the training and test data. To prevent duplicate code in 1.3, the *classificationLogit*-function was coded. Giving data and a threshold as an input, a list with the specified threshold to decide which probabilities lead to a spam classification, the confusion matrix and the misclassification rate will be returned.

```r
# Classifying & evaluating results
classificationLogit = function(data, threshold = 0.5) {
  # Classifying emails with the model
  yFit = predict(logitModel,
                 newdata = data[,!colnames(data) %in% "Spam"],
                 type='response')
  yFit = ifelse(yFit > threshold, 1, 0)
  # Evaluating classification results
  confusionMatrix = table(y = data$Spam, yFit)
  misclassificationRate <- mean(yFit != data$Spam)
  # Returning results
  return(
    list(
      threshold = threshold,
      confusionMatrix = confusionMatrix,
      misclassificationRate = misclassificationRate
    )
  )
}
```

Using the train data and the default threshold (0.5) as the input leads to the following confusion matrix and misclassification rate.

```r
classificationLogitTrain = classificationLogit(data = train)
classificationLogitTrain$confusionMatrix
```

```
##    yFit
## y     0   1
##   0 803 142
##   1  81 344
```

```r
classificationLogitTrain$misclassificationRate
```

```
## [1] 0.1627737
```

Instead, using the test data and the default threshold (0.5) as the input leads to the following confusion matrix and misclassification rate.

```r
classificationLogitTest = classificationLogit(data = test)
classificationLogitTest$confusionMatrix
```

```
##    yFit
## y     0   1
##   0 791 146
##   1  97 336
```

```r
classificationLogitTest$misclassificationRate
```

```
## [1] 0.1773723
```

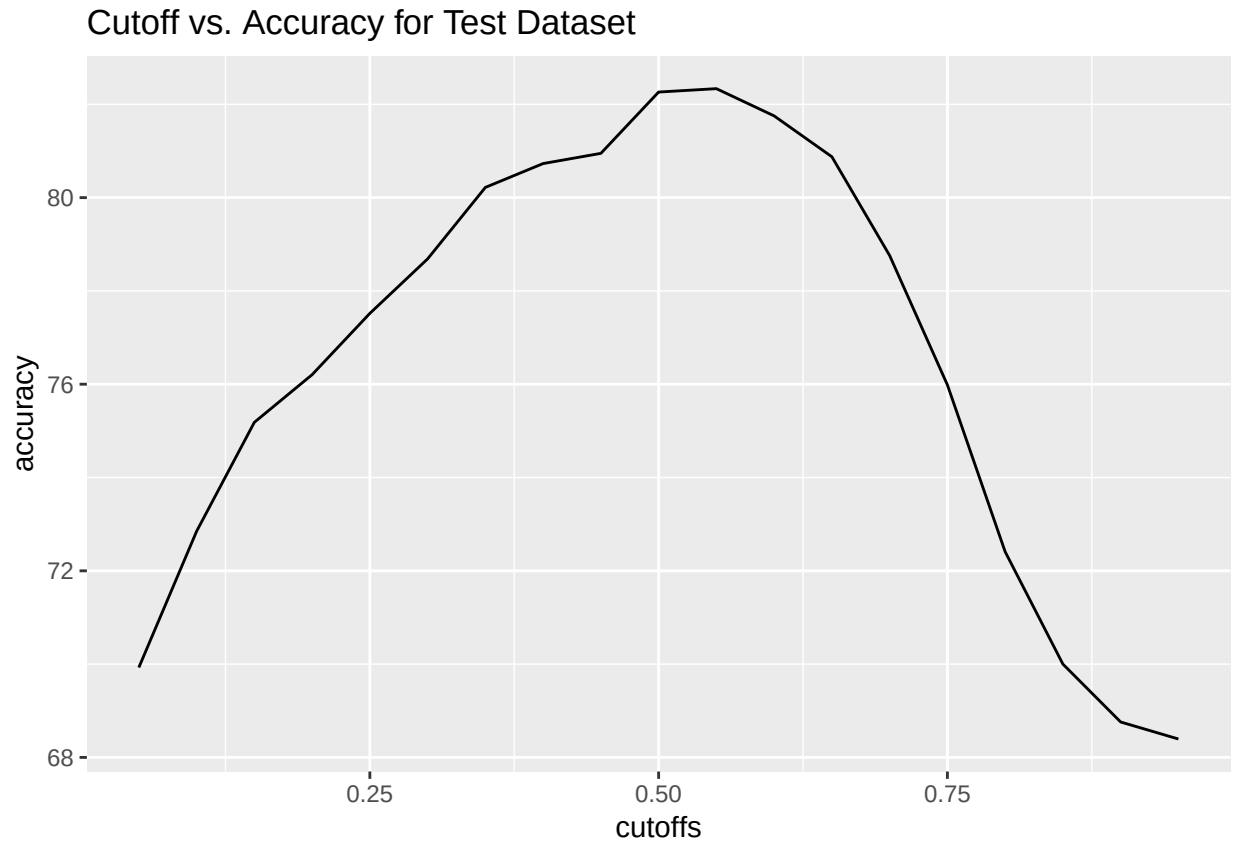
Analysis: It can be seen that the model performs about equally well for both data. The misclassification rate is slightly better for the training data (16.2%) than for the training data (17.7%). This is an indication that there is not too much overfitting on the training data.

1.3 Use logistic regression to classify the test data by the classification principle probability>90%.Assessing the Fit on train dataset for 90% 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?

Now we are changing the classification principle and therefore the input threshold from 0.5 to 0.9, however lets also find the best cutoff value

Choosing the best cutoff for test

```
cutoffs <- seq(from = 0.05, to = 0.95, by = 0.05)
accuracy <- NULL
prediction_prob <- predict(logitModel, newdata = test , type = "response")


for (i in seq_along(cutoffs)){
    prediction <- ifelse(prediction_prob >= cutoffs[i], 1, 0) #Predicting for cut-off

    accuracy <- c(accuracy,length(which(test$Spam == prediction))/length(prediction)*100)}

cutoff_data <- as.data.frame(cbind(cutoffs, accuracy))

ggplot(data = cutoff_data, aes(x = cutoffs, y = accuracy)) +
  geom_line() +
  ggtitle("Cutoff vs. Accuracy for Test Dataset")
```



Using the train data and the threshold = 0.9 as the input leads to the following confusion matrix and

misclassification rate.

```
classificationLogitTrainAdjThreshold = classificationLogit(data = train, threshold = 0.9)
classificationLogitTrainAdjThreshold$confusionMatrix
```

```
##    yFit
## y     0   1
##   0 944   1
##   1 419   6
```

```
classificationLogitTrainAdjThreshold$misclassificationRate
```

```
## [1] 0.3065693
```

Using the test data and the threshold = 0.9 as the input leads to the following confusion matrix and misclassification rate.

```
classificationLogitTestAdjThreshold = classificationLogit(data = test, threshold = 0.9)
classificationLogitTestAdjThreshold$confusionMatrix
```

```
##    yFit
## y     0   1
##   0 936   1
##   1 427   6
```

```
classificationLogitTestAdjThreshold$misclassificationRate
```

```
## [1] 0.3124088
```

Analysis: Our small detour suggests that the cutoff value of ~60% was the best for our purpose and going higher than this leads to worse results, at 70% and above the accuracy drastically reduces which is what we see when we make cutoff as 90%.

For both test and train, it can be seen that the classification quality decreases a lot (misclassification rates about 30%). Because the threshold is now much higher than before, the number of false negative predictions has increased strongly.

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 1.2.

In the following, the standard classifier *kknn()* was used to predict spam mails. Again, to prevent duplicate code in 1.5, the *classificationKknn*-function was coded. Giving data, number of k and a threshold as an input, a list with the same elements as the *classificationLogit*-function is returned.

```
library(kknn)
# Classifying & evaluating results
classificationKknn = function(data, k, threshold = 0.5) {
  # Classifying emails
  kknnModel <- kknn(formula = Spam ~ .,
                    train = train,
                    test = data,
                    k = k)
  kknnModel$fitted.values = ifelse(kknnModel$fitted.values > threshold, 1, 0)
  # Evaluating classification results
  confusionMatrix = table(y = data$Spam, yFit = kknnModel$fitted.values)
  misclassificationRate <- mean(kknnModel$fitted.values != data$Spam)
  # Returning results
```

```
  return(
    list(
      threshold = threshold,
      confusionMatrix = confusionMatrix,
      misclassificationRate = misclassificationRate
    )
  )
}
```

Using the train data and k = 30 as the input leads to the following misclassification rate.

```
classificationKnnTrain = classificationKknn(data = train, k = 30)
classificationKnnTrain$misclassificationRate
```

```
## [1] 0.1722628
```

Instead, using the test data and k = 30 as the input leads to the following misclassification rate.

```
classificationKnnTest = classificationKknn(data = test, k = 30)
classificationKnnTest$misclassificationRate
```

```
## [1] 0.329927
```

Analysis: Here, a big difference between the prediction power of the model related to the train data (misclassification rate: 17.2%) and the test data (misclassification rate: 32.9%) can be observed. This leads to the assumption that the model is overfitting on the training data. Compared to the results of the logistic regression model with the threshold = 0.5, this model does not deliver such accurate predictions.

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?

Now we are changing the k from 30 to 1.

Using the train data and k = 1 as the input leads to the following misclassification rate.

```
classificationKnnTrain = classificationKknn(data = train, k = 1)
classificationKnnTrain$misclassificationRate
```

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

Using the test data and k = 1 as the input leads to the following misclassification rate.

```
classificationKnnTest = classificationKknn(data = test, k = 1)
classificationKnnTest$misclassificationRate
```

```
## [1] 0.3459854
```

Analysis: This example shows very clearly that the model is strongly overfitted on the training data. While it classifies every mail for the training data correctly, the misclassification rate for the test data is almost 35%. With k = 1, the classification depends only on the nearest neighbor (the value of the dependent variable of this observation in the training data where the independent variables have the lowest distance to the obsvervation which shall be classified) which leads to a much higher dependency on the training data.

Explaination: The KKNN works in the following way, An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. If k = 1, then the object is simply assigned to the class of that single nearest neighbor. Thus K=1, makes the seperation boundary to be very complex and locally optimised (lots of local clusters), while as K goes higher, the decision boundary becomes more linear/simple.

Assignment 3

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)

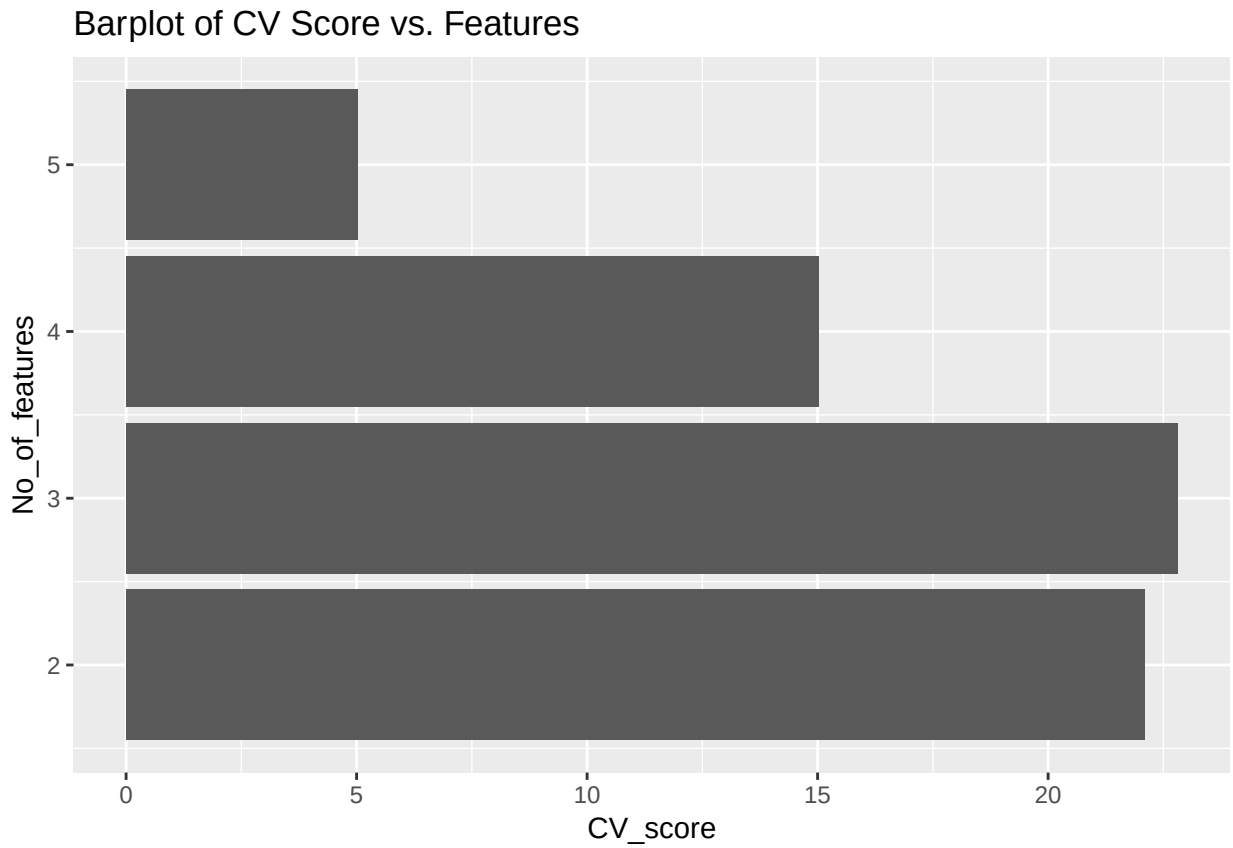
```
select_my_features <- function(x, y, nfolds){

  # 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)

  data <- cbind(abs(res), n_features)
  colnames(data) <- c("CV_score", "No_of_features")
  data <- as.data.frame(data)

  list_of_results$plot <- ggplot(data = data, aes(x = No_of_features, y = CV_score)) +
    geom_bar(stat="identity") +
    ggtitle("Barplot of CV Score vs. Features") + coord_flip()


  list_of_results$cv_score <- df[best_model, 1]
  return(list_of_results)
}
```

2 Test your function on data set swiss available in the standard R repository:

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

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




```
##
## $cv_score
## res_model
##  63.40326
```

Analysis: 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.

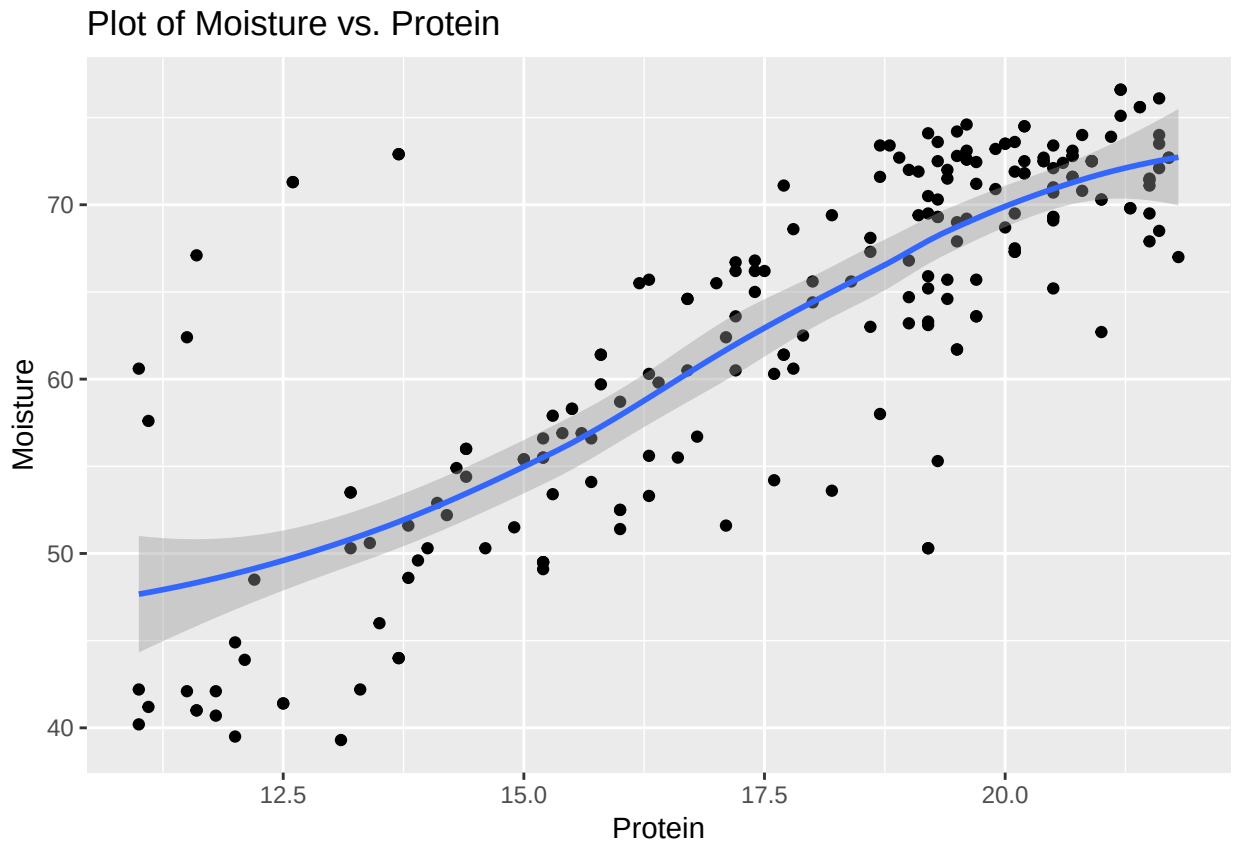
I would say for none of the independent variables it is reasonable to have an impact on the fertility of people. When reasoning, there is no explanation as to why working in agriculture, having a degree, religion or death of childs could affect the fertility of people. Therefore when computing these 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 idea to have “Fertility” as independent variable and “Infant.Mortality” as dependent variable.

Assignment 4 Linear regression and regularization

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

```r
ggplot(data = tecator_data, aes(x = Protein, y = Moisture)) +
  geom_point() +
  geom_smooth( method = 'loess') +
  ggtitle("Plot of Moisture vs. Protein")
```



Analysis: The data seems fairly linear in nature however there are many outliers. As we can see that data is fairly distributed around the line drawn (above and below) thus there is little bias.

4.2 Consider model M in which Moisture is normally distributed, and the expected Moisture is a polynomial function of Protein including the polynomial terms up to power of i. Report a probabilistic model that describes M. Why is it appropriate to use MSE criterion when fitting this model to a training data?

$$M_i = \sum_{i=0}^{p} X^i Protein * \beta i + \epsilon$$

$$\epsilon \sim N\left(0, \sigma^2\right)$$

$$\epsilon = M_i - \sum_{i=0}^{p} X^i Protein * \beta i$$

$$M_i \sim N\left(\sum_{i=0}^{p} X^i Protein * \beta i, \sigma_M^2\right)$$

or

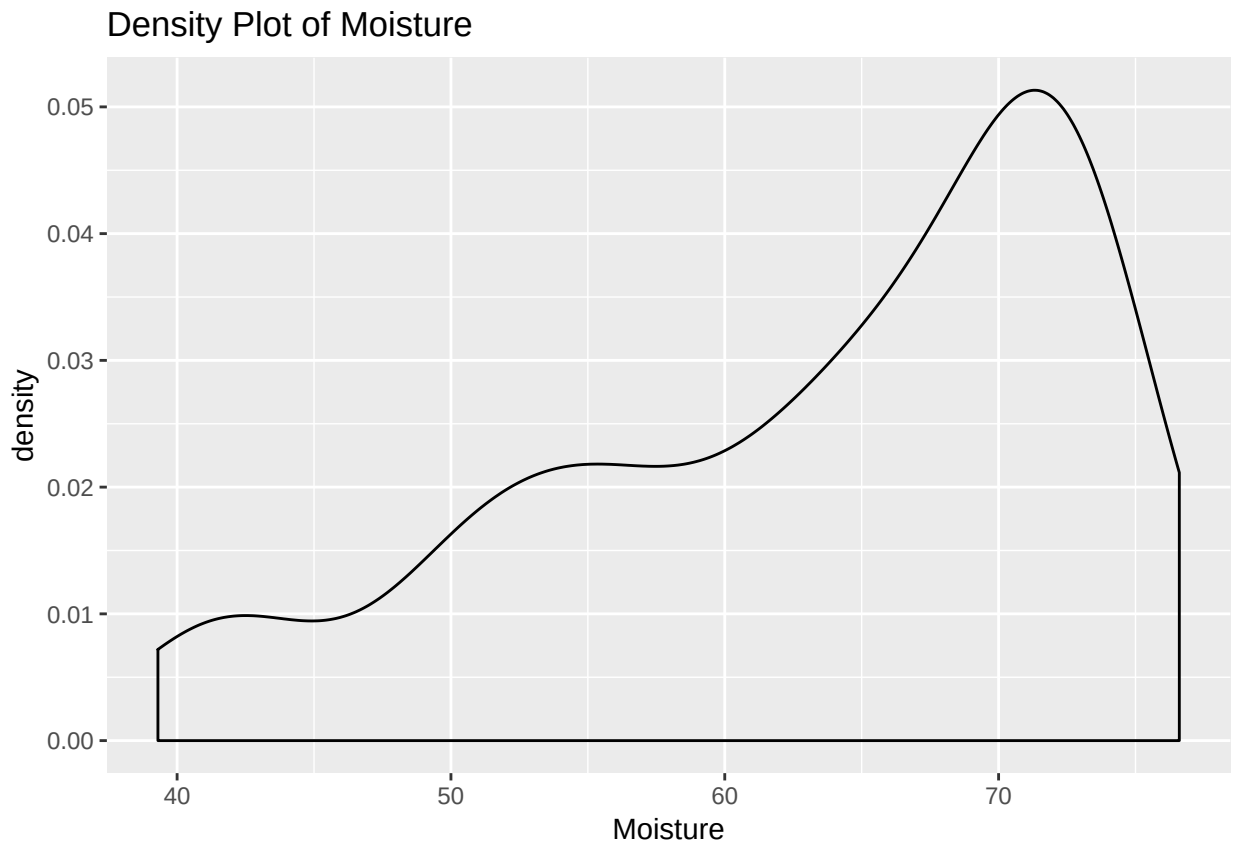
$$P\left(M_i|X_{Protein},\vec{\beta}\right) = N\left(\sum_{i=0}^{p} X^i Protein * \beta i, \sigma_M^2\right)$$

$$Where,$$

σ_M^2 : variance of Moisture

p : degree of the polynomial

```
ggplot(data = tecator_data, aes(x = Moisture)) +
  geom_density() +
  ggtitle("Density Plot of Moisture")
```



Analysis: In this case we are given that mositure is normally distributed, thus the loss function to minimize had to be (actual-predicted)^2, if we were to minimize the absoulte value, then this would assume a Laplace distribution.

4.3 Divide the data into training and validation sets (50/50) and fit models M (i=1,2,3,..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.

```
final_data <- tecator_data

magic_function <- function(df, N)
{
df2 <- df
for(i in 2:N)
{
  df2[paste("Protein_",i,"_power", sep="")] <- (df2$Protein)^i
  }

df2 <- df2[c("Protein_2_power", "Protein_3_power",
             "Protein_4_power", "Protein_5_power",
             "Protein_6_power")]

df <- cbind(df,df2)
return(df)
}

final_data <- magic_function(final_data, 6)

set.seed(12345)
n =  NROW(final_data)
id = sample(1:n, floor(n*0.5))
train = final_data[id,]
test = final_data[-id,]

# model building
M_1 <- lm(data = train, Moisture~Protein)
M_2 <- lm(data = train, Moisture~Protein+Protein_2_power)
M_3 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power)
M_4 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power+
            Protein_4_power)
M_5 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power+
            Protein_4_power+Protein_5_power)
M_6 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power+
            Protein_4_power+Protein_5_power+Protein_6_power)

train$type <- "train"
test$type <- "test"

final_data <- rbind(test, train)

# predicting new values
M_1_predicted <- predict(M_1, newdata = final_data)
M_2_predicted <- predict(M_2, newdata = final_data)
```

```r
M_3_predicted <- predict(M_3, newdata = final_data)
M_4_predicted <- predict(M_4, newdata = final_data)
M_5_predicted <- predict(M_5, newdata = final_data)
M_6_predicted <- predict(M_6, newdata = final_data)

# calculating the MSE
final_data$M_1_error <- (final_data$Moisture - M_1_predicted)^2
final_data$M_2_error <- (final_data$Moisture - M_2_predicted)^2
final_data$M_3_error <- (final_data$Moisture - M_3_predicted)^2
final_data$M_4_error <- (final_data$Moisture - M_4_predicted)^2
final_data$M_5_error <- (final_data$Moisture - M_5_predicted)^2
final_data$M_6_error <- (final_data$Moisture - M_6_predicted)^2

# Chainning like Chainsaw
final_error_data <- final_data %>% select(type, M_1_error, M_2_error, M_3_error,
                                          M_4_error, M_5_error, M_6_error) %>%
  gather(variable, value, -type) %>%
  separate(variable, c("model", "power", "error"), "_") %>%
  group_by(type, power) %>%
  summarise(MSE = mean(value, na.rm=TRUE))

ggplot(final_error_data, aes(x = power, y = MSE, color=type)) + geom_point() +
  ggtitle("Mean squared error vs. model complexitiy by dataset type")
```

Analysis: As evident from the plot above, we see that as we increase the model complexitiy (higher powers of the 'protein'), the trainning error reduces however the model becomes too biased towards the trainning set

(overfits) and misses the test datasets prediction by larger margins in higher powers.

The best model is M1, that is Moisture~Protein as evident from the least test error (MSE).

The above is a classical case of bias-varience trade-off, which is as follows, as one makes the model fit the trainning dataset better the model becomes more biased and its ability to handle variation to new dataset decreases(varience), thus one should also maintain a good trade off between these two.

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

```
min.model1 = lm(Fat ~ 1, data=tecator_data[,-1])
biggest1 <- formula(lm(Fat ~.,  data=tecator_data[,-1]))

step.model1 <- stepAIC(min.model1, direction ='forward', scope=biggest1, trace = FALSE)
summ(step.model1)
```

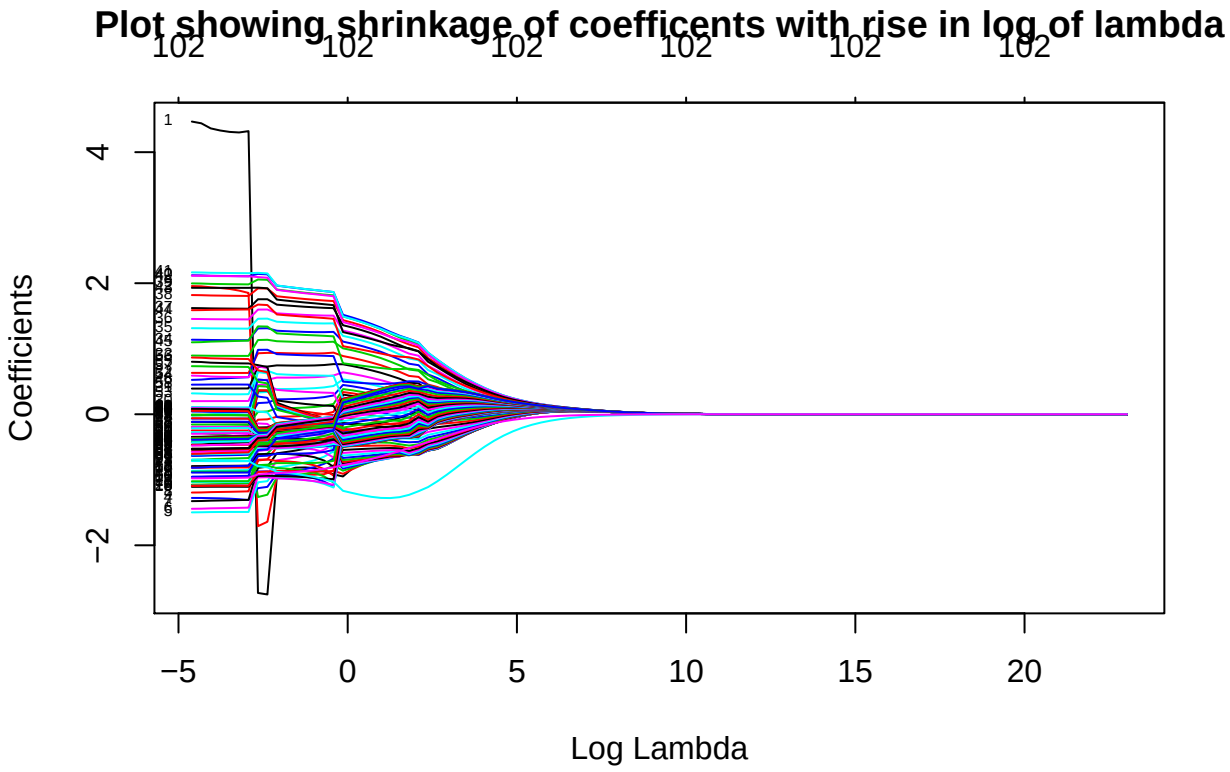
```
## MODEL INFO:
## Observations: 215
## Dependent Variable: Fat
## Type: OLS linear regression
##
## MODEL FIT:
## F(29,185) = 4775.35, p = 0.00
## R² = 1.00
## Adj. R² = 1.00
##
## Standard errors: OLS
##                  Est.    S.E. t val.    p
## (Intercept)    93.46    1.59  58.86 0.00 ***
## Moisture       -1.03    0.02 -54.25 0.00 ***
## Protein        -0.64    0.06 -10.91 0.00 ***
## Channel100     66.56   48.18   1.38 0.17
## Channel41   -3268.11  826.92  -3.95 0.00 ***
## Channel7      -64.03   20.80  -3.08 0.00  **
## Channel48   -2022.46  254.46  -7.95 0.00 ***
## Channel42    4934.22 1124.96   4.39 0.00 ***
## Channel50    1239.52  236.09   5.25 0.00 ***
## Channel45    4796.22  783.38   6.12 0.00 ***
## Channel66    2435.79 1169.85   2.08 0.04   *
## Channel56    2373.00  540.06   4.39 0.00 ***
## Channel90    -258.27  247.22  -1.04 0.30
## Channel60    -264.27  708.11  -0.37 0.71
## Channel70      14.25  327.12   0.04 0.97
## Channel67   -2015.92  543.74  -3.71 0.00 ***
## Channel59     635.71  996.31   0.64 0.52
## Channel65    -941.61 1009.23  -0.93 0.35
## Channel58    1054.24  927.95   1.14 0.26
## Channel44   -5733.84 1079.19  -5.31 0.00 ***
## Channel18     299.80   88.43   3.39 0.00 ***
## Channel78    2371.11  361.25   6.56 0.00 ***
## Channel84    -428.99  338.35  -1.27 0.21
```

```
## Channel62    3062.97  769.59   3.98 0.00 ***
## Channel53    -804.39  203.44  -3.95 0.00 ***
## Channel75   -1461.42  402.26  -3.63 0.00 ***
## Channel57   -3266.79  876.71  -3.73 0.00 ***
## Channel63   -2844.66  906.40  -3.14 0.00  **
## Channel24    -308.71   97.87  -3.15 0.00  **
## Channel37     401.64  151.76   2.65 0.01  **
```

Analysis: 29 variables were choose out of 107. Even among these there are many which have very low p values thus statistically it is a practice to remove variables which have p values above 0.05, thus the true variables may not even include these many.

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 lambda and report how the coefficients change with lambda.

```
y <- tecator_data %>% select(Fat) %>% data.matrix()
x <- tecator_data %>% select(-c(Fat)) %>% data.matrix()

lambda <- 10^seq(10, -2, length = 100)

ridge_fit <- glmnet(x, y, alpha = 0, family = "gaussian", lambda = lambda)
plot(ridge_fit, xvar = "lambda", label = TRUE,
     main = "Plot showing shrinkage of coefficents with rise in log of lambda")
```



```
## Change of coefficent with respect to lambda
result <- NULL
for(i in lambda){
temp <- t(coef(ridge_fit, i)) %>% as.matrix()
temp <- cbind(temp, lambda = i)
result <- rbind(temp, result)
}
result <- result %>% as.data.frame() %>% arrange(lambda)
```

```
table_cofe <- head(result, 10) %>% select(Channel1, Channel2, Channel84, Channel62,
                                          Channel53, Channel75, Channel57,Protein,
                                          lambda)

knitr::kable(table_cofe, caption = "Coefficent of Ridge Regression vs. Lambda")
```

Table 1: Coefficent of Ridge Regression vs. Lambda

Channel1	Channel2	Channel84	Channel62	Channel53	Channel75	Channel57	Protein	lambda
4.466912	1.9563431	0.0934552	-0.5633505	0.3187870	-0.0043661	0.7335944	-0.6928952	0.0100000
4.440394	1.9525143	0.0926891	-0.5609320	0.3156013	-0.0053703	0.7321219	-0.6936043	0.0132194
4.362982	1.9376731	0.0905061	-0.5544721	0.3084684	-0.0079319	0.7284455	-0.6967560	0.0174753
4.330747	1.9271671	0.0891937	-0.5512426	0.3059969	-0.0092315	0.7268339	-0.6998292	0.0231013
4.309285	1.9124776	0.0876178	-0.5481419	0.3045920	-0.0105692	0.7252852	-0.7032970	0.0305386
4.300527	1.8870336	0.0852251	-0.5445233	0.3044751	-0.0123127	0.7233819	-0.7083824	0.0403702
4.320767	1.8517518	0.0822553	-0.5413104	0.3063425	-0.0141825	0.7213962	-0.7129651	0.0533670
-2.727500	-1.7083641	-0.2094827	-0.3675587	0.6435452	-0.2488630	0.4954374	-0.7940538	0.0705480
-2.750491	-1.6412309	-0.2118732	-0.3713265	0.6236700	-0.2534746	0.4702021	-0.8107922	0.0932603
-0.936478	-0.9505301	-0.1484618	-0.4389420	0.1282598	-0.2328265	0.0679230	-0.8779587	0.1232847

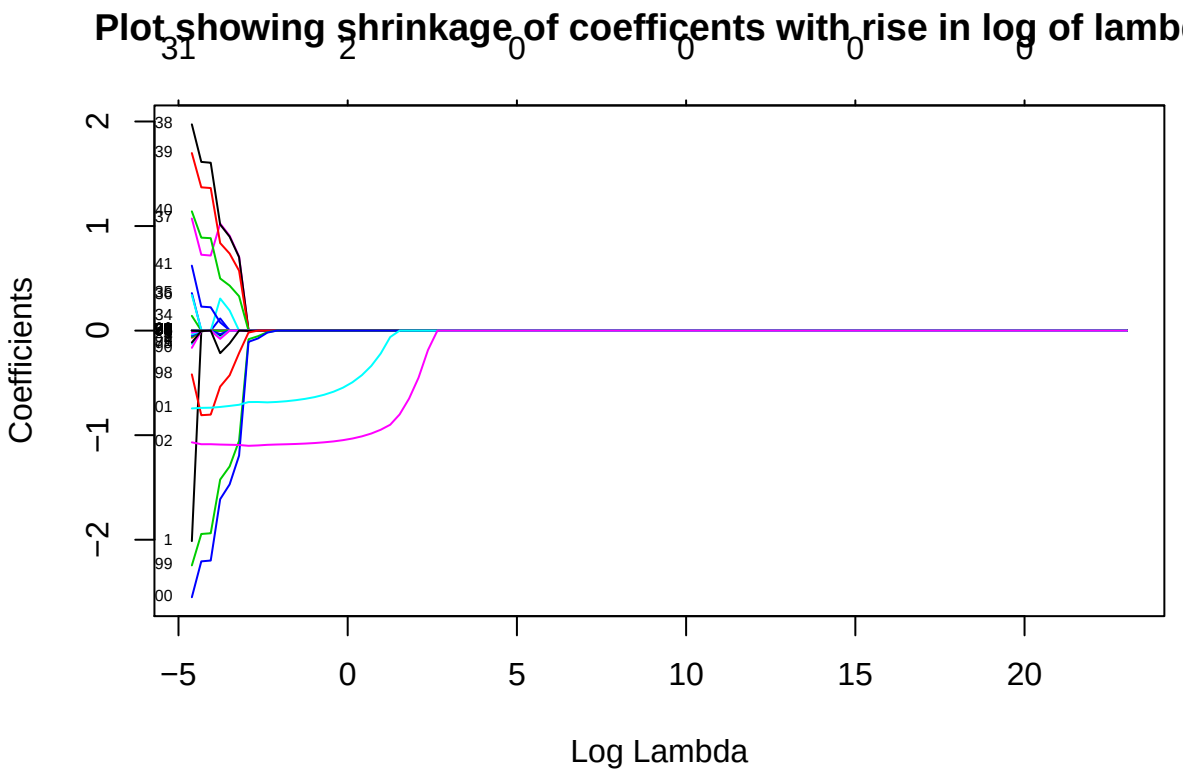
Analysis: The idea of lasso and ridge regression is to introduce bias to variables in order to reduce/account for multicollinearity. Introducing bias (lambda) to covarience matrix is done by multiplying the diagonal elements by lambda(often 1+lambda), this inflates the covarience of predictors compared to correleations of predictors. The idea is to test the stability of betas that is how likely are the betas/coefficents of regressions to be stable if we keep introducing bias.

We can clearly see that 'Channel1' and 'Channel2' betas go from positive to negative with very little bias introduced while terms like 'Channel75' dont change the beta signs. Thus the practice is exclude the terms whose beta/coefficent dont change drastically much within say first 10 introduction of lambda.

We see that many of the terms/coefficent tend to zero at around log(lambda) that is ~5.

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

```
lambda <- 10^seq(10, -2, length = 100)

lasso_fit <- glmnet(x, y, alpha = 1, family = "gaussian", lambda = lambda)
plot(lasso_fit, xvar = "lambda", label = TRUE,
     main = "Plot showing shrinkage of coefficents with rise in log of lambda")
```



Analysis: We quickly see that very little introduction of penalisation/bias is all it takes to make many terms/coefficent to zero. This implies for the full dataset lasso is much better suited for regularisation compared to ridge.

At lambda around 1 (log lambda is 0) we get only two or three non zero terms.

4.7 Use cross-validation to find the optimal LASSO model, report the optimal lambda 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 lambda.

```
#find the best lambda from our list via cross-validation

lambda_lasso <- 10^seq(10, -2, length = 100)
lambda_lasso[101] <- 0
lasso_cv <- cv.glmnet(x,y, alpha=1, lambda = lambda_lasso, type.measure="mse")
coef(lasso_cv, lambda = lasso_cv$lambda.min)
```

```
## 103 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) 100.1826458
## Channel1      .
## Channel2      .
## Channel3      .
## Channel4      .
## Channel5      .
```

```
## Channel6      .
## Channel7      .
## Channel8      .
## Channel9      .
## Channel10     .
## Channel11     .
## Channel12     .
## Channel13     .
## Channel14     .
## Channel15     .
## Channel16     .
## Channel17     .
## Channel18     .
## Channel19     .
## Channel20     .
## Channel21     .
## Channel22     .
## Channel23     .
## Channel24     .
## Channel25     .
## Channel26     .
## Channel27     .
## Channel28     .
## Channel29     .
## Channel30     .
## Channel31     .
## Channel32     .
## Channel33     .
## Channel34     .
## Channel35     .
## Channel36     .
## Channel37     0.7085073
## Channel38     0.7023525
## Channel39     0.5726923
## Channel40     0.3285077
## Channel41     .
## Channel42     .
## Channel43     .
## Channel44     .
## Channel45     .
## Channel46     .
## Channel47     .
## Channel48     .
## Channel49     .
## Channel50     .
## Channel51     .
## Channel52     .
## Channel53     .
## Channel54     .
## Channel55     .
## Channel56     .
## Channel57     .
## Channel58     .
## Channel59     .
```

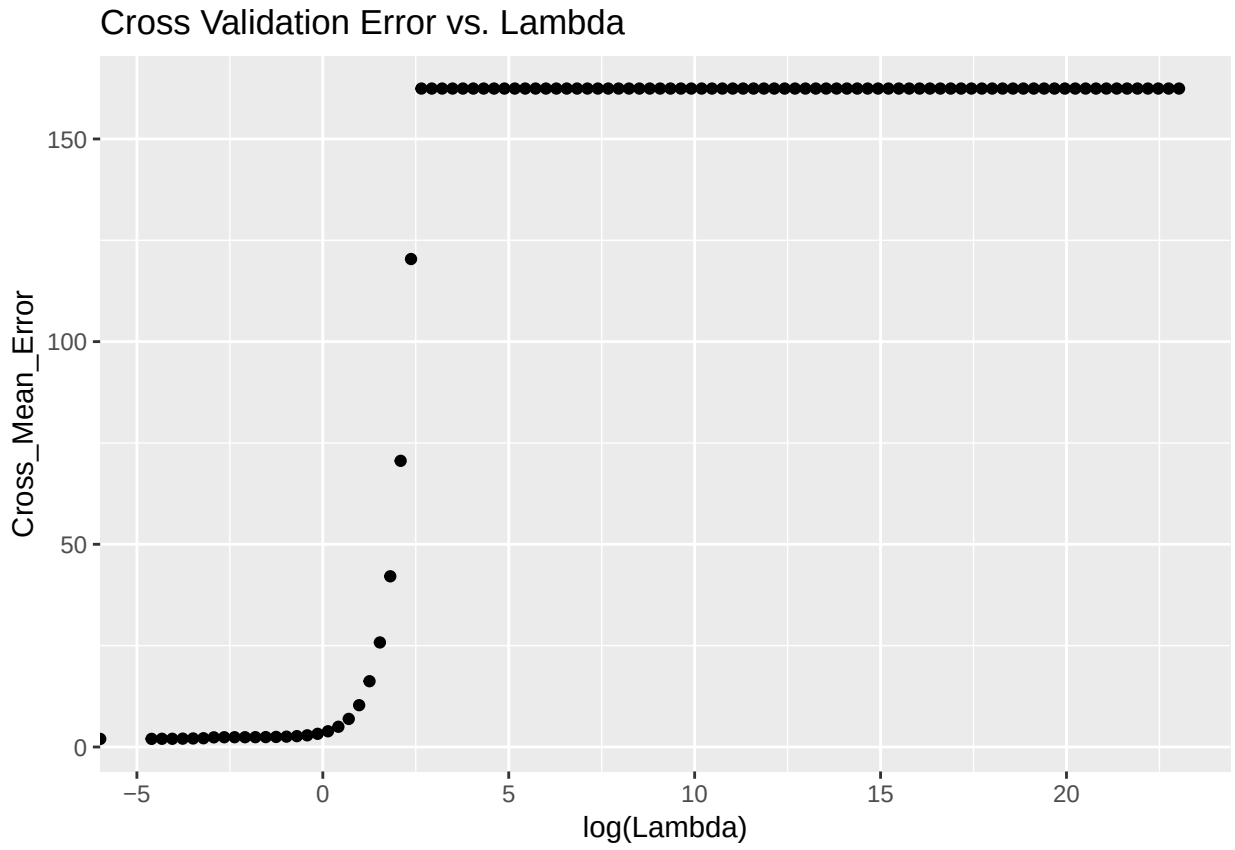
```
## Channel60     .
## Channel61     .
## Channel62     .
## Channel63     .
## Channel64     .
## Channel65     .
## Channel66     .
## Channel67     .
## Channel68     .
## Channel69     .
## Channel70     .
## Channel71     .
## Channel72     .
## Channel73     .
## Channel74     .
## Channel75     .
## Channel76     .
## Channel77     .
## Channel78     .
## Channel79     .
## Channel80     .
## Channel81     .
## Channel82     .
## Channel83     .
## Channel84     .
## Channel85     .
## Channel86     .
## Channel87     .
## Channel88     .
## Channel89     .
## Channel90     .
## Channel91     .
## Channel92     .
## Channel93     .
## Channel94     .
## Channel95     .
## Channel96     .
## Channel97     .
## Channel98    -0.2147181
## Channel99    -1.0547607
## Channel100   -1.1957408
## Protein      -0.7093446
## Moisture     -1.0939227
```

```
lasso_cv$lambda.min
```

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

```
## Change of coefficent with respect to lambda
result_lasso <- NULL
for(i in 1:length(lambda_lasso)){
temp <- lasso_cv$cvm[i] %>% as.matrix()
temp <- cbind(CVM_error = temp, lambda = lasso_cv$lambda[i])
result_lasso <- rbind(temp, result_lasso)
}
```

```
result_lasso <- result_lasso %>% as.data.frame() %>% arrange(lambda)
colnames(result_lasso) <- c("Cross_Mean_Error", "Lambda")

ggplot(result_lasso, aes(x = log(Lambda), y = Cross_Mean_Error)) + geom_point() +
  ggtitle("Cross Validation Error vs. Lambda")
```



Analysis: The minimum value of lambda was 0, implies zero penalisation. The variables selected are: Channel98, Channel99, Channel100, Protein, Moisture, Channel37, Channel38, Channel39, Channel40 along with intercept.

We see that Cross validation error is lowest at lambda= 0 and remains low till lambda~1 (log lambda 0) after which the error drastically increases at log(lambda) ~ 2.5, the error maxes out and remains about the same for higher values of lambda. This implies that more bias introduction will lead to worse performance.

4.8 Compare the results from steps 4 and 7.

Analysis: In order to find the best predictors for a given model we employ various techniques.

In step4 we analytically arrive at the best variables to model 'Fat' using multiple variables, using stepAIC we arrive at 29 variables excluding the Intercept.

In step5 we use regularisation techniques and start introducing bias to eliminate the variables which maybe corellated with each other, employing this we get further reduction of about 10 variables at log lambda ~ 5.

Using Lasso in step6 we get only about 5 variables at lambda close to 1, however the exact number of variables to choose is depended on the lambda value and the corresponding error. However having used the whole dataset, we need to employee cross validation to get the precise value of lambda.

In step 7 we get the best value of lambda for lasso for which the mean squared error is the least and here we are left with 9 variables excluding the intercept. The mean squared error is also low (~10).

Thus we have learned how to perform regression and how to account for multicorrlinearity and possible ways to detect and negate the same.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
if (!require("pacman")) install.packages("pacman")
pacman::p_load(xlsx, glmnet, MASS, jtools, huxtable, ggplot2,
               ggthemes, gridExtra, ROCR, broom, caret, e1071,
               knnn, tidyr, dplyr,reshape2, glmnet)

options("jtools-digits" = 2, scipen = 999)

# Importing data
data = read.xlsx("spambase.xlsx", sheetName = "spambase_data")
tecator_data <- read.xlsx("tecator.xlsx", sheetName = "data")
tecator_data <- tecator_data[,2:NCOL(tecator_data)] # removing sample column

# Dividing data into train and test set
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = data[id,]
test = data[-id,]
# Fitting model
logitModel = glm(Spam ~ ., data = train, family = binomial)
summary(logitModel)
# Classifying & evaluating results
classificationLogit = function(data, threshold = 0.5) {
  # Classifying emails with the model
  yFit = predict(logitModel,
                 newdata = data[,!colnames(data) %in% "Spam"],
                 type='response')
  yFit = ifelse(yFit > threshold, 1, 0)
  # Evaluating classification results
  confusionMatrix = table(y = data$Spam, yFit)
  misclassificationRate <- mean(yFit != data$Spam)
  # Returning results
  return(
    list(
      threshold = threshold,
      confusionMatrix = confusionMatrix,
      misclassificationRate = misclassificationRate
    )
  )
}
classificationLogitTrain = classificationLogit(data = train)
classificationLogitTrain$confusionMatrix
classificationLogitTrain$misclassificationRate
```

```r
classificationLogitTest = classificationLogit(data = test)
classificationLogitTest$confusionMatrix
classificationLogitTest$misclassificationRate

cutoffs <- seq(from = 0.05, to = 0.95, by = 0.05)
accuracy <- NULL
prediction_prob <- predict(logitModel, newdata = test , type = "response")


for (i in seq_along(cutoffs)){
    prediction <- ifelse(prediction_prob >= cutoffs[i], 1, 0) #Predicting for cut-off

    accuracy <- c(accuracy,length(which(test$Spam == prediction))/length(prediction)*100)}

cutoff_data <- as.data.frame(cbind(cutoffs, accuracy))

ggplot(data = cutoff_data, aes(x = cutoffs, y = accuracy)) +
  geom_line() +
  ggtitle("Cutoff vs. Accuracy for Test Dataset")

classificationLogitTrainAdjThreshold = classificationLogit(data = train, threshold = 0.9)
classificationLogitTrainAdjThreshold$confusionMatrix
classificationLogitTrainAdjThreshold$misclassificationRate
classificationLogitTestAdjThreshold = classificationLogit(data = test, threshold = 0.9)
classificationLogitTestAdjThreshold$confusionMatrix
classificationLogitTestAdjThreshold$misclassificationRate
library(kknn)
# Classifying & evaluating results
classificationKknn = function(data, k, threshold = 0.5) {
  # Classifying emails
  kknnModel <- kknn(formula = Spam ~ .,
                    train = train,
                    test = data,
                    k = k)
  kknnModel$fitted.values = ifelse(kknnModel$fitted.values > threshold, 1, 0)
  # Evaluating classification results
  confusionMatrix = table(y = data$Spam, yFit = kknnModel$fitted.values)
  misclassificationRate <- mean(kknnModel$fitted.values != data$Spam)
  # Returning results
  return(
    list(
      threshold = threshold,
      confusionMatrix = confusionMatrix,
      misclassificationRate = misclassificationRate
    )
  )
}
classificationKnnTrain = classificationKknn(data = train, k = 30)
classificationKnnTrain$misclassificationRate
classificationKnnTest = classificationKknn(data = test, k = 30)
classificationKnnTest$misclassificationRate
classificationKnnTrain = classificationKknn(data = train, k = 1)
classificationKnnTrain$misclassificationRate
```

```
classificationKnnTest = classificationKknn(data = test, k = 1)
classificationKnnTest$misclassificationRate
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)

  data <- cbind(abs(res), n_features)
  colnames(data) <- c("CV_score", "No_of_features")
  data <- as.data.frame(data)

  list_of_results$plot <- ggplot(data = data, aes(x = No_of_features, y = CV_score)) +
    geom_bar(stat="identity") +
    ggtitle("Barplot of CV Score vs. Features") + coord_flip()


  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)
ggplot(data = tecator_data, aes(x = Protein, y = Moisture)) +
  geom_point() +
  geom_smooth( method = 'loess') +
  ggtitle("Plot of Moisture vs. Protein")
ggplot(data = tecator_data, aes(x = Moisture)) +
  geom_density() +
  ggtitle("Density Plot of Moisture")

final_data <- tecator_data

magic_function <- function(df, N)
{
df2 <- df
for(i in 2:N)
{
  df2[paste("Protein_",i,"_power", sep="")] <- (df2$Protein)^i
  }

df2 <- df2[c("Protein_2_power", "Protein_3_power",
```

```
              "Protein_4_power", "Protein_5_power",
              "Protein_6_power")]

df <- cbind(df,df2)
return(df)
}

final_data <- magic_function(final_data, 6)

set.seed(12345)
n =  NROW(final_data)
id = sample(1:n, floor(n*0.5))
train = final_data[id,]
test = final_data[-id,]

# model building
M_1 <- lm(data = train, Moisture~Protein)
M_2 <- lm(data = train, Moisture~Protein+Protein_2_power)
M_3 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power)
M_4 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power+
            Protein_4_power)
M_5 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power+
            Protein_4_power+Protein_5_power)
M_6 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power+
            Protein_4_power+Protein_5_power+Protein_6_power)

train$type <- "train"
test$type <- "test"

final_data <- rbind(test, train)

# predicting new values
M_1_predicted <- predict(M_1, newdata = final_data)
M_2_predicted <- predict(M_2, newdata = final_data)
M_3_predicted <- predict(M_3, newdata = final_data)
M_4_predicted <- predict(M_4, newdata = final_data)
M_5_predicted <- predict(M_5, newdata = final_data)
M_6_predicted <- predict(M_6, newdata = final_data)

# calculating the MSE
final_data$M_1_error <- (final_data$Moisture - M_1_predicted)^2
final_data$M_2_error <- (final_data$Moisture - M_2_predicted)^2
final_data$M_3_error <- (final_data$Moisture - M_3_predicted)^2
final_data$M_4_error <- (final_data$Moisture - M_4_predicted)^2
final_data$M_5_error <- (final_data$Moisture - M_5_predicted)^2
final_data$M_6_error <- (final_data$Moisture - M_6_predicted)^2

# Chainning like Chainsaw
final_error_data <- final_data %>% select(type, M_1_error, M_2_error, M_3_error,
                                          M_4_error, M_5_error, M_6_error) %>%
  gather(variable, value, -type) %>%
  separate(variable, c("model", "power", "error"), "_") %>%
  group_by(type, power) %>%
```

```
  summarise(MSE = mean(value, na.rm=TRUE))

ggplot(final_error_data, aes(x = power, y = MSE, color=type)) + geom_point() +
  ggtitle("Mean squared error vs. model complexitiy by dataset type")

min.model1 = lm(Fat ~ 1, data=tecator_data[,-1])
biggest1 <- formula(lm(Fat ~.,  data=tecator_data[,-1]))

step.model1 <- stepAIC(min.model1, direction ='forward', scope=biggest1, trace = FALSE)
summ(step.model1)
y <- tecator_data %>% select(Fat) %>% data.matrix()
x <- tecator_data %>% select(-c(Fat)) %>% data.matrix()

lambda <- 10^seq(10, -2, length = 100)

ridge_fit <- glmnet(x, y, alpha = 0, family = "gaussian", lambda = lambda)
plot(ridge_fit, xvar = "lambda", label = TRUE,
     main = "Plot showing shrinkage of coefficents with rise in log of lambda")


## Change of coefficent with respect to lambda
result <- NULL
for(i in lambda){
temp <- t(coef(ridge_fit, i)) %>% as.matrix()
temp <- cbind(temp, lambda = i)
result <- rbind(temp, result)
}
result <- result %>% as.data.frame() %>% arrange(lambda)
table_cofe <- head(result, 10) %>% select(Channel1, Channel2, Channel84, Channel62,
                                          Channel53, Channel75, Channel57,Protein,
                                          lambda)

knitr::kable(table_cofe, caption = "Coefficent of Ridge Regression vs. Lambda")
lambda <- 10^seq(10, -2, length = 100)

lasso_fit <- glmnet(x, y, alpha = 1, family = "gaussian", lambda = lambda)
plot(lasso_fit, xvar = "lambda", label = TRUE,
     main = "Plot showing shrinkage of coefficents with rise in log of lambda")

#find the best lambda from our list via cross-validation

lambda_lasso <- 10^seq(10, -2, length = 100)
lambda_lasso[101] <- 0
lasso_cv <- cv.glmnet(x,y, alpha=1, lambda = lambda_lasso, type.measure="mse")
coef(lasso_cv, lambda = lasso_cv$lambda.min)

lasso_cv$lambda.min

## Change of coefficent with respect to lambda
result_lasso <- NULL
for(i in 1:length(lambda_lasso)){
temp <- lasso_cv$cvm[i] %>% as.matrix()
temp <- cbind(CVM_error = temp, lambda = lasso_cv$lambda[i])
```

```
result_lasso <- rbind(temp, result_lasso)
}

result_lasso <- result_lasso %>% as.data.frame() %>% arrange(lambda)
colnames(result_lasso) <- c("Cross_Mean_Error", "Lambda")

ggplot(result_lasso, aes(x = log(Lambda), y = Cross_Mean_Error)) + geom_point() +
  ggtitle("Cross Validation Error vs. Lambda")
```