machine learning(732A99) lab1

Anubhav Dikshit(anudi287)

26 November 2018

Assignment 1

Loading The Libraries

Loading Input files

```
spam_data <- read.xlsx("spambase.xlsx", sheetName = "spambase_data")
spam_data$Spam <- as.factor(spam_data$Spam)

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

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

```
set.seed(12345)

n =  NROW(spam_data)
id = sample(1:n, floor(n*0.5))
train = spam_data[id,]
test = spam_data[-id,]
```

1.2 Use logistic regression (functions glm(), predict()) to classify the training and test data by the classification principles

Manual Feature Selection

```
best_model <- glm(formula = Spam ~., family = binomial, data = train)
summ(best_model)
```

```
## MODEL INFO:
## Observations: 1370
## Dependent Variable: Spam
## Type: Generalized linear model
##   Family: binomial
##   Link function: logit
##
## MODEL FIT:
## <U+0001D6D8>²(48) = 768.27, p = 0.00
## Pseudo-R² (Cragg-Uhler) = 0.60
## Pseudo-R² (McFadden) = 0.45
## AIC = 1026.54, BIC = 1282.45
```

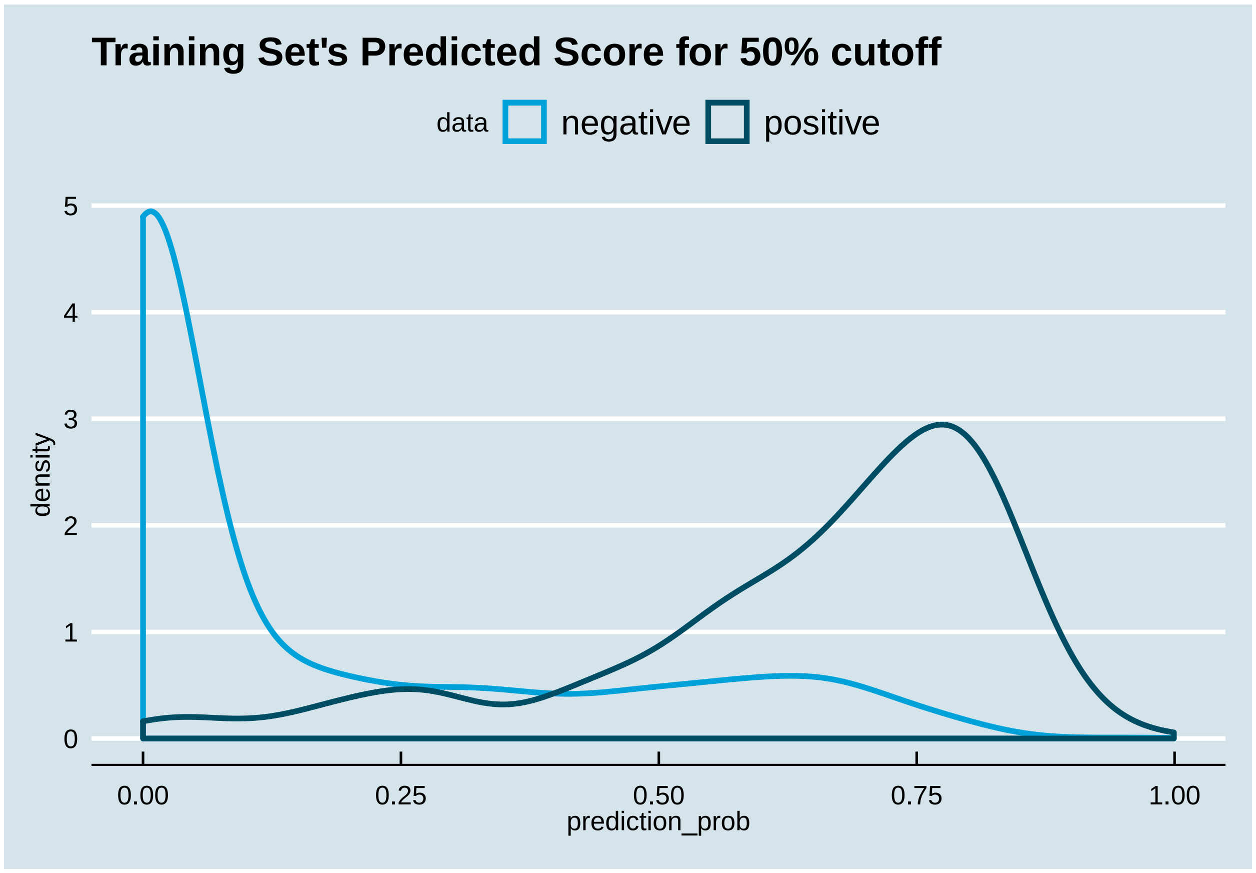
```
##
## Standard errors: MLE
##                   Est.     S.E. z val.    p
## (Intercept)       1.51     0.20   7.50 0.00 ***
## Word1            -0.72     0.50  -1.43 0.15
## Word2             0.04     0.30   0.13 0.89
## Word3            -0.35     0.18  -1.92 0.06   .
## Word4             0.14     0.11   1.23 0.22
## Word5             0.12     0.14   0.86 0.39
## Word6             0.29     0.42   0.70 0.49
## Word7            -0.29     0.33  -0.88 0.38
## Word8            -0.10     0.35  -0.29 0.77
## Word9            -0.11     0.41  -0.27 0.79
## Word10           -0.03     0.17  -0.19 0.85
## Word11           -0.61     0.68  -0.90 0.37
## Word12            0.16     0.11   1.50 0.13
## Word13            0.78     0.36   2.19 0.03   *
## Word14           -0.48     0.30  -1.60 0.11
## Word15           -0.13     0.39  -0.34 0.74
## Word16            0.32     0.23   1.36 0.17
## Word17           -0.09     0.28  -0.33 0.74
## Word18            0.02     0.23   0.10 0.92
## Word19            0.00     0.06   0.01 0.99
## Word20            0.02     0.32   0.05 0.96
## Word21           -0.03     0.11  -0.26 0.79
## Word22           -0.48     0.32  -1.49 0.14
## Word23            0.25     0.34   0.74 0.46
## Word24           -0.25     0.63  -0.40 0.69
## Word25           -0.08     0.06  -1.34 0.18
## Word26            0.00     0.14   0.03 0.98
## Word27           -0.22     0.11  -2.08 0.04   *
## Word28            0.13     0.19   0.70 0.48
## Word29           -0.08     0.09  -0.89 0.37
## Word30           -1.82     0.62  -2.93 0.00  **
## Word31           -4.69     1.85  -2.53 0.01   *
## Word32         -119.45 15134.18  -0.01 0.99
## Word33           -2.90     0.68  -4.27 0.00 ***
## Word34           -3.71     4.35  -0.85 0.39
## Word35           -7.03     2.00  -3.52 0.00 ***
## Word36           -1.68     0.38  -4.40 0.00 ***
## Word37           -0.86     0.22  -3.95 0.00 ***
## Word38           -0.60     1.28  -0.47 0.64
## Word39           -1.88     0.43  -4.38 0.00 ***
## Word40            0.07     0.34   0.22 0.83
## Word41         -332.55 16559.52  -0.02 0.98
## Word42           -5.35     1.30  -4.11 0.00 ***
## Word43           -2.59     0.74  -3.53 0.00 ***
## Word44           -2.93     0.66  -4.44 0.00 ***
## Word45           -1.14     0.17  -6.79 0.00 ***
## Word46           -3.29     0.52  -6.35 0.00 ***
## Word47           -3.74     2.03  -1.84 0.07   .
## Word48           -4.39     1.47  -2.98 0.00  **
```

Prediction for probability greater than 50% and 90%

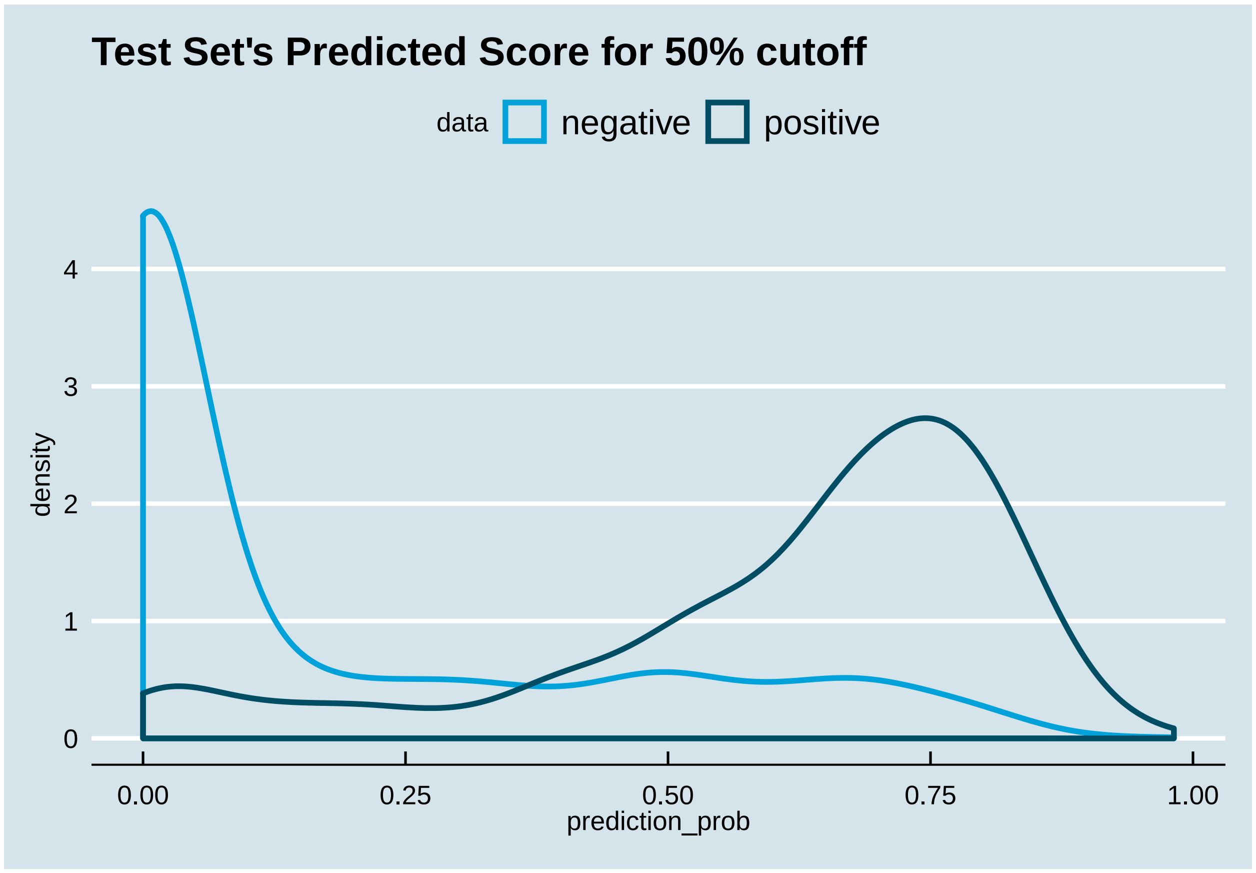
```
# prediction
train$prediction_prob <- predict(best_model, newdata = train, type = "response")
test$prediction_prob <- predict(best_model, newdata = test , type = "response")

train$prediction_class_50 <- ifelse(train$prediction_prob > 0.50, 1, 0)
test$prediction_class_50 <- ifelse(test$prediction_prob > 0.50, 1, 0)

train$prediction_class_90 <- ifelse(train$prediction_prob > 0.90, 1, 0)
test$prediction_class_90 <- ifelse(test$prediction_prob > 0.90, 1, 0)
```

Assessing the Model

```
# plots
ggplot(train, aes(prediction_prob, color = Spam)) +
geom_density(size = 1) + ggtitle("Training Set's Predicted Score for 50% cutoff") +
  scale_color_economist(name = "data", labels = c("negative", "positive")) +
  theme_economist()
```



```
ggplot(test, aes(prediction_prob, color = Spam)) +
geom_density(size = 1) + ggtitle("Test Set's Predicted Score for 50% cutoff") +
  scale_color_economist(name = "data", labels = c("negative", "positive")) +
  theme_economist()
```



1.2 Assessing the Fit on train dataset for 50%

```r
#confusion table
conf_train <- table(train$Spam, train$prediction_class_50)
names(dimnames(conf_train)) <- c("Actual Train", "Predicted Train")
confusionMatrix(conf_train)
```

```
## Confusion Matrix and Statistics
##
##             Predicted Train
## Actual Train   0   1
##            0 803 142
##            1  81 344
##
##                Accuracy : 0.8372
##                  95% CI : (0.8166, 0.8564)
##     No Information Rate : 0.6453
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.6341
##  Mcnemar's Test P-Value : 0.00005872
##
##             Sensitivity : 0.9084
##             Specificity : 0.7078
##          Pos Pred Value : 0.8497
##          Neg Pred Value : 0.8094
##              Prevalence : 0.6453
##          Detection Rate : 0.5861
##    Detection Prevalence : 0.6898
##       Balanced Accuracy : 0.8081
##
##        'Positive' Class : 0
##
```

```
conf_test <- table(test$Spam, test$prediction_class_50)
names(dimnames(conf_test)) <- c("Actual Test", "Predicted Test")
confusionMatrix(conf_test)
```

```
## Confusion Matrix and Statistics
##
##              Predicted Test
## Actual Test   0   1
##           0 791 146
##           1  97 336
##
##                Accuracy : 0.8226
##                  95% CI : (0.8014, 0.8425)
##     No Information Rate : 0.6482
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.6018
##  Mcnemar's Test P-Value : 0.002076
##
##             Sensitivity : 0.8908
##             Specificity : 0.6971
##          Pos Pred Value : 0.8442
##          Neg Pred Value : 0.7760
##              Prevalence : 0.6482
##          Detection Rate : 0.5774
##    Detection Prevalence : 0.6839
##       Balanced Accuracy : 0.7939
##
##        'Positive' Class : 0
##
```

Analysis: Distribution of the prediction score grouped by known outcome given that our model's final objective is to classify new instances into one of two categories (spam vs. non-spam). We will want the model to give high scores to positive instances (1: spam) and low scores (0 : not spam) otherwise.Ideally you want the distribution of scores to be separated, with the score of the negative instances to be on the left and the score of the positive instance to be on the right.

From the confusion matrix it is apparent that Accuracy on train and test dataset when cutoff=50% is about ~84% for train and ~82% for test, thus the misclassification rate is ~16 and ~18% for the train and test dataset.

1.3 Assessing the Fit on train dataset for 90%

```
#confusion table
conf_train1 <- table(train$Spam, train$prediction_class_90)
```

```r
names(dimnames(conf_train1)) <- c("Actual Train", "Predicted Train")
conf_train1
```

```
##               Predicted Train
## Actual Train    0   1
##              0 944   1
##              1 419   6
```

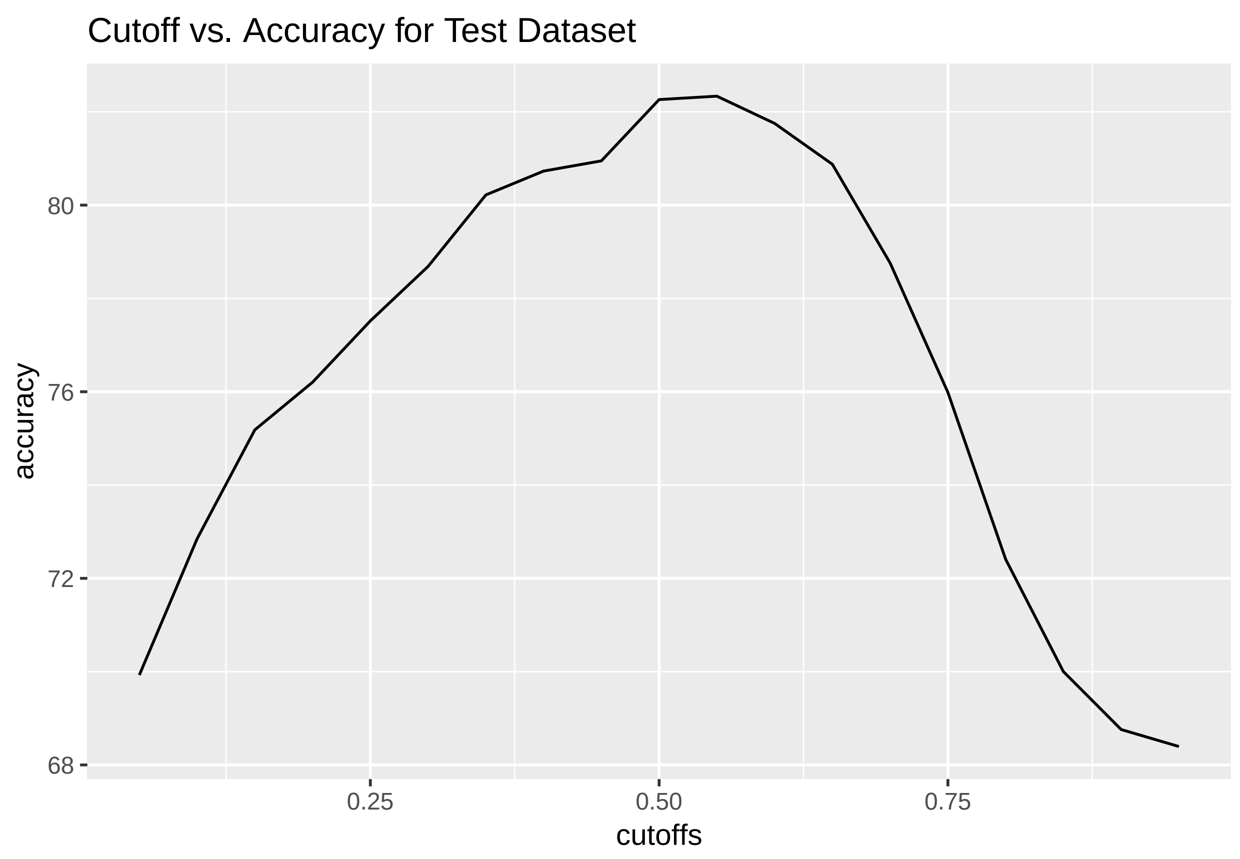
```r
conf_test1 <- table(test$Spam, test$prediction_class_90)
names(dimnames(conf_test1)) <- c("Actual Test", "Predicted Test")
conf_test1
```

```
##              Predicted Test
## Actual Test    0   1
##             0 936   1
##             1 427   6
```

Analysis: Strange, the model almost only predicts one class!! We know that the prediction of a logistic regression model is a probability, thus in order to use it as a classifier, we'll have to choose a cutoff value, or threshold (cutoff). Where scores above this value will classified as positive, those below as negative. Lets us find this optimum value.

Choosing the best cutoff for test

```r
cutoffs <- seq(from = 0.05, to = 0.95, by = 0.05)
accuracy <- NULL

for (i in seq_along(cutoffs)){
    prediction <- ifelse(test$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")
```



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%.

From the confusion matrix it is evident that the model becomes a trivial model(predicts all cases as one class) and thus the prediction is worse than just tossing a coin. This should be the absoutely the worst case that we should avoid.

The missclassication rate is about 31% for both the trainning dataset and test dataset.

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.

```
knn_model30 <- train.kknn(Spam ~., data = train, kmax = 30)

train$knn_prediction_class <- predict(knn_model30, train)
test$knn_prediction_class <- predict(knn_model30, test)

conf_train2 <- table(train$Spam, train$knn_prediction_class)
names(dimnames(conf_train2)) <- c("Actual Train", "Predicted Train")
confusionMatrix(conf_train2)
```

```
## Confusion Matrix and Statistics
##
##             Predicted Train
## Actual Train   0   1
##            0 845 100
##            1  91 334
##
##                Accuracy : 0.8606
##                  95% CI : (0.8411, 0.8785)
##     No Information Rate : 0.6832
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.6761
##  Mcnemar's Test P-Value : 0.5627
##
##             Sensitivity : 0.9028
##             Specificity : 0.7696
##          Pos Pred Value : 0.8942
##          Neg Pred Value : 0.7859
##              Prevalence : 0.6832
##          Detection Rate : 0.6168
##    Detection Prevalence : 0.6898
##       Balanced Accuracy : 0.8362
##
##        'Positive' Class : 0
##
```

```
conf_test2 <- table(test$Spam, test$knn_prediction_class)
names(dimnames(conf_test2)) <- c("Actual Test", "Predicted Test")
confusionMatrix(conf_test2)
```

```
## Confusion Matrix and Statistics
##
##            Predicted Test
## Actual Test   0   1
##           0 801 136
##           1 139 294
##
##                Accuracy : 0.7993
##                  95% CI : (0.7771, 0.8202)
##     No Information Rate : 0.6861
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.5348
##  Mcnemar's Test P-Value : 0.904
##
##             Sensitivity : 0.8521
##             Specificity : 0.6837
##          Pos Pred Value : 0.8549
##          Neg Pred Value : 0.6790
##              Prevalence : 0.6861
##          Detection Rate : 0.5847
##    Detection Prevalence : 0.6839
##       Balanced Accuracy : 0.7679
##
##        'Positive' Class : 0
##
```

Analysis: Using KKNN with K = 30, we increased our trainning accuracy to 86%, thus misclassification is 14%, however for the test dataset misclassification rate is about ~20%.

Thus compared to using logisitc model the misclassification error for the trainning dataset decreased by 2% to 14%, while for the test dataset the misclassification error increased by 2% to 20%.

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?

```r
knn_model1 <- train.kknn(Spam ~., data = train, kmax = 1)

train$knn_prediction_class <- predict(knn_model1, train)
test$knn_prediction_class <- predict(knn_model1, test)

conf_train2 <- table(train$Spam, train$knn_prediction_class)
names(dimnames(conf_train2)) <- c("Actual Train", "Predicted Train")
confusionMatrix(conf_train2)
```

```
## Confusion Matrix and Statistics
##
##             Predicted Train
## Actual Train   0   1
##            0 945   0
##            1   0 425
##
##                Accuracy : 1
##                  95% CI : (0.9973, 1)
##     No Information Rate : 0.6898
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6898
##          Detection Rate : 0.6898
##    Detection Prevalence : 0.6898
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

```r
conf_test2 <- table(test$Spam, test$knn_prediction_class)
names(dimnames(conf_test2)) <- c("Actual Test", "Predicted Test")
confusionMatrix(conf_test2)
```

```
## Confusion Matrix and Statistics
##
##            Predicted Test
## Actual Test   0   1
##           0 750 187
##           1 150 283
##
##                Accuracy : 0.754
##                  95% CI : (0.7303, 0.7766)
##     No Information Rate : 0.6569
##     P-Value [Acc > NIR] : 0.000000000000004691
##
##                   Kappa : 0.4438
##  Mcnemar's Test P-Value : 0.04987
##
##             Sensitivity : 0.8333
##             Specificity : 0.6021
##          Pos Pred Value : 0.8004
##          Neg Pred Value : 0.6536
##              Prevalence : 0.6569
##          Detection Rate : 0.5474
##    Detection Prevalence : 0.6839
##       Balanced Accuracy : 0.7177
##
##        'Positive' Class : 0
##
```

Analysis: Using KKNN with K = 1, we increased our trainning accuracy to 100%, thus misclassification is 0%, however for the test dataset accuracy is ~75% thus misclassification rate is about ~25%, thus we improved on the trainning accuracy but did bad on the test case, thus this is an example of overfitting leading to more varience.

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 2 Feature selection by cross-validation in a linear model

2.1 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)

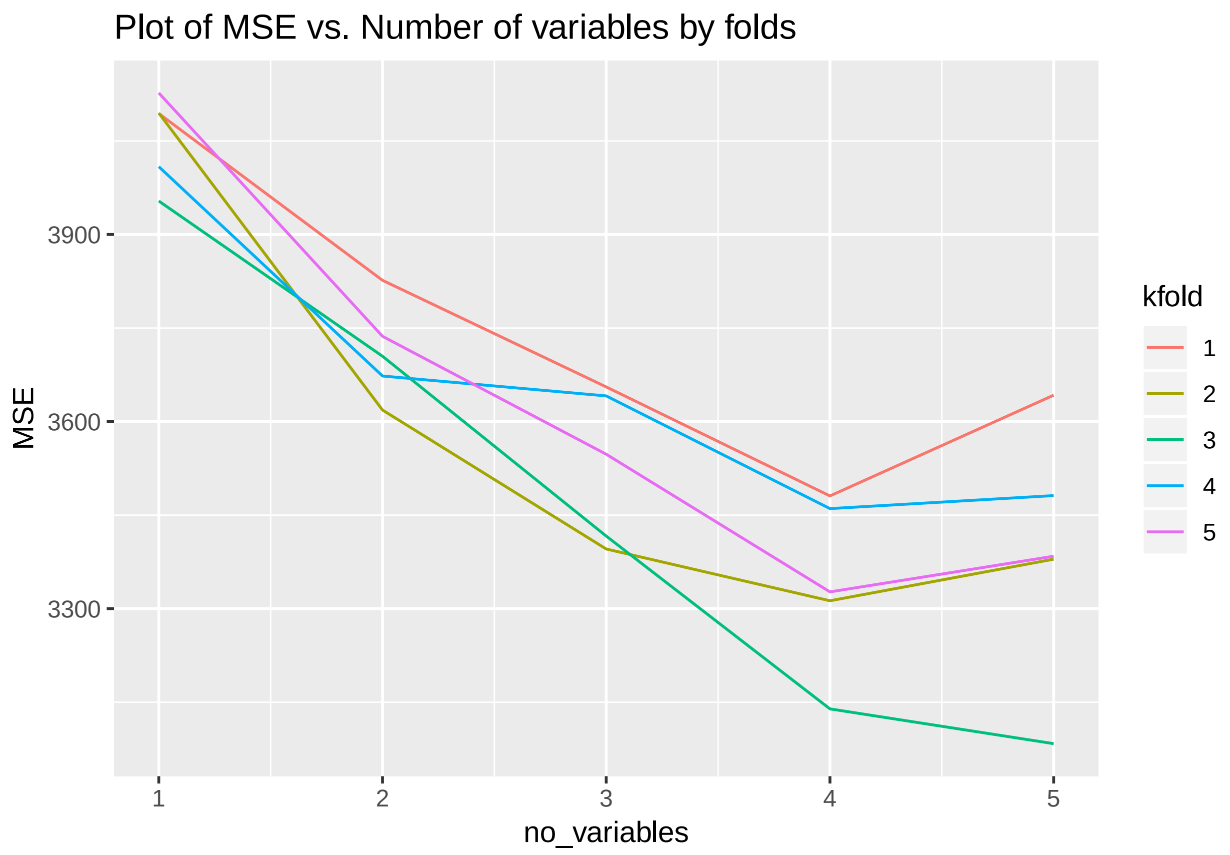
```
subset_function <- function(X,Y,N){

# X = swiss[,1:5]
# Y = swiss[,6:6]
# N = 5

df <- cbind(X,Y)
temp <- NULL
final <- NULL

for(i in 1:NCOL(X)){
combs <- as.data.frame(gtools::combinations(NCOL(X), r=i, v=colnames(X), repeats.allowed=FALSE))
combs <- tidyr::unite(combs, "formula", sep = ",")
temp <- rbind(combs, temp)
}

set.seed(12345)
df2 <- df[sample(nrow(df)),]
df2$k_fold <- sample(N, size = nrow(df), replace = TRUE)

result <- NULL

for (j in 1:NROW(temp))
{
  for(i in 1:N){

train = df2[df2$k_fold != i,]
test = df2[df2$k_fold == i,]

vec <- temp[j,]
train_trimmed = lapply(strsplit(as.character(vec), ","), function(x) train[x])[[1]]
test_trimmed = lapply(strsplit(as.character(vec), ","), function(x) test[x])[[1]]

y_train = train[,c("Y"), drop = FALSE]
y_test = test[,c("Y"), drop = FALSE]

train_trimmed = as.matrix(train_trimmed)
test_trimmed = as.matrix(test_trimmed)
y_test = as.matrix(y_test)
y_train = as.matrix(y_train)

t_train =  as.matrix(t(train_trimmed))
t_test =  as.matrix(t(test_trimmed))

betas = solve(t_train %*% train_trimmed) %*% (t_train %*% y_train)

train_trimmed = as.data.frame(train_trimmed)
test_trimmed = as.data.frame(test_trimmed)

train_trimmed$type = "train"
test_trimmed$type = "test"
final <- rbind(train_trimmed, test_trimmed)


y_hat_val = as.matrix(final[,1:(ncol(final)-1)]) %*% betas
mse = (Y - y_hat_val)^ 2

data <- cbind(i, vec, mse, type = final$type)
result <- rbind(data, result)

}
}

result <- as.data.frame(result)

colnames(result) <- c("kfold", "variables", "mse", "type")
```

```
result$mse <- as.numeric(result$mse)
result$no_variables <- nchar(as.character(result$variables)) - nchar(gsub('\\,', "", result$variables))

variable_performance <- result %>% group_by(kfold, no_variables) %>%
  summarise(MSE = mean(mse, na.rm = TRUE))

myplot <- ggplot(data = variable_performance, aes(x = no_variables, y = MSE, color=kfold)) +
geom_line() + ggtitle("Plot of MSE vs. Number of variables by folds")

myplot2 <- ggplot(data = result, aes(x = variables, y = mse, color=kfold)) +
geom_bar(stat="identity") + ggtitle("Plot of RMSE vs. Features by folds") + coord_flip()

return(list(myplot, myplot2))
}
```

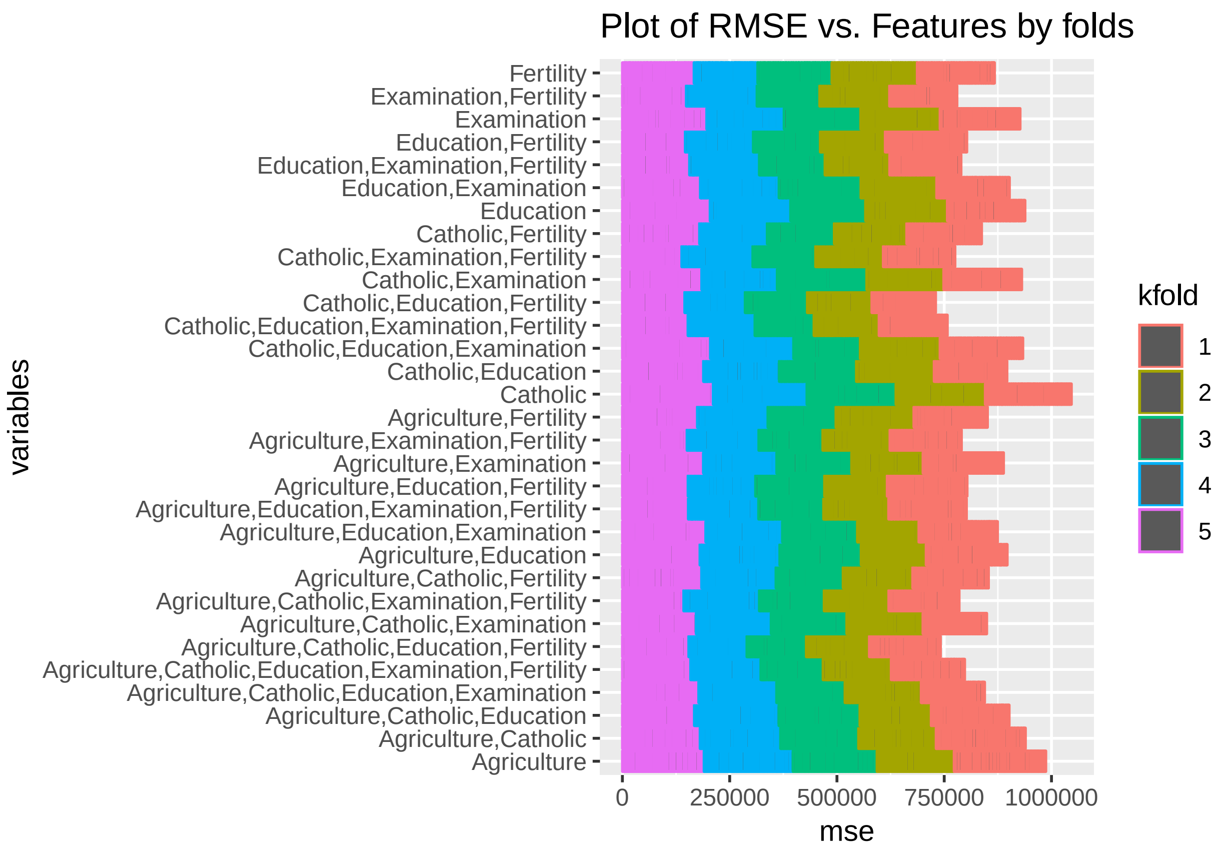
2.2 Test your function on data set swiss available in the standard R repository:

```
subset_function(X = swiss[,1:5], Y = swiss[,6], N = 5)
```

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



```
##
## [[2]]
```

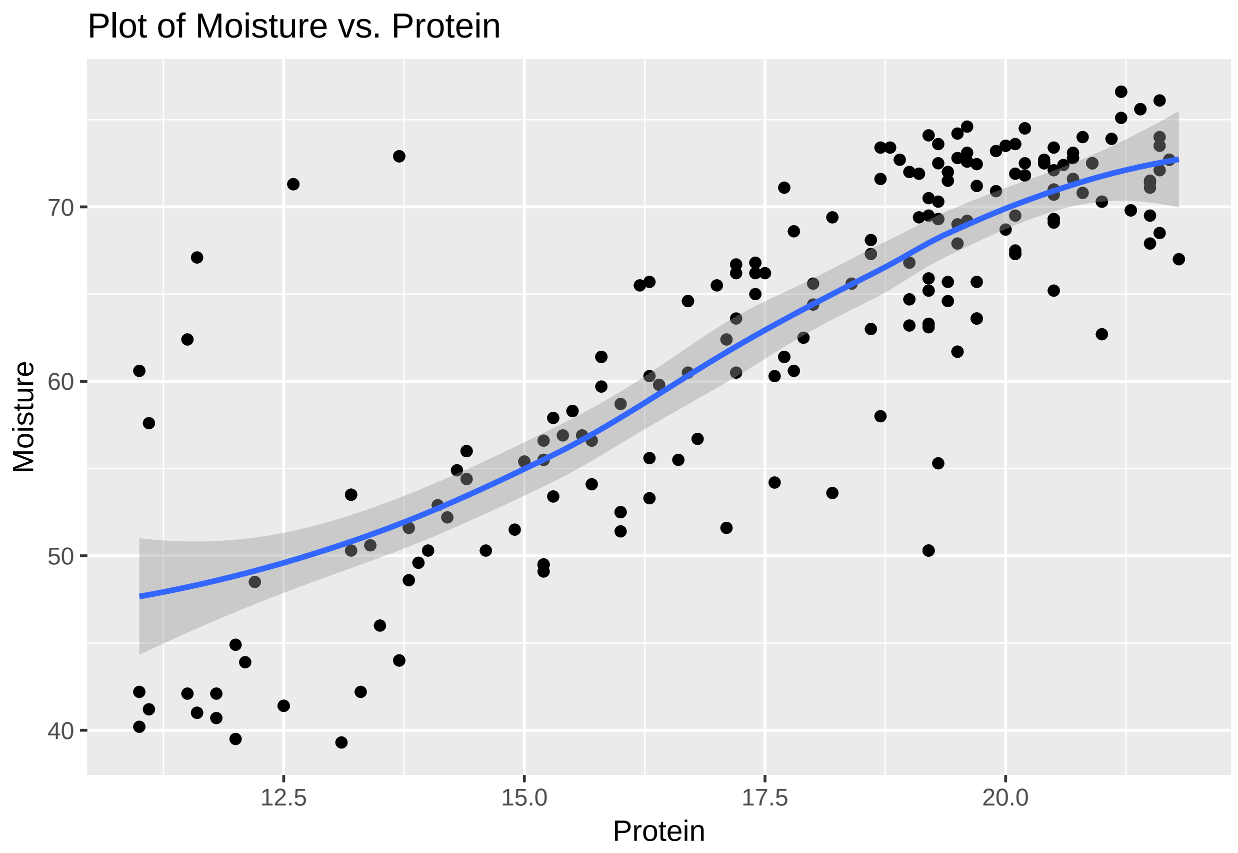


Analysis:

Assignment 3 Linear regression and regularization

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

```
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.

3.2 Multiple Models of varying degree.

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

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

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

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

or

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

$$Where,$$

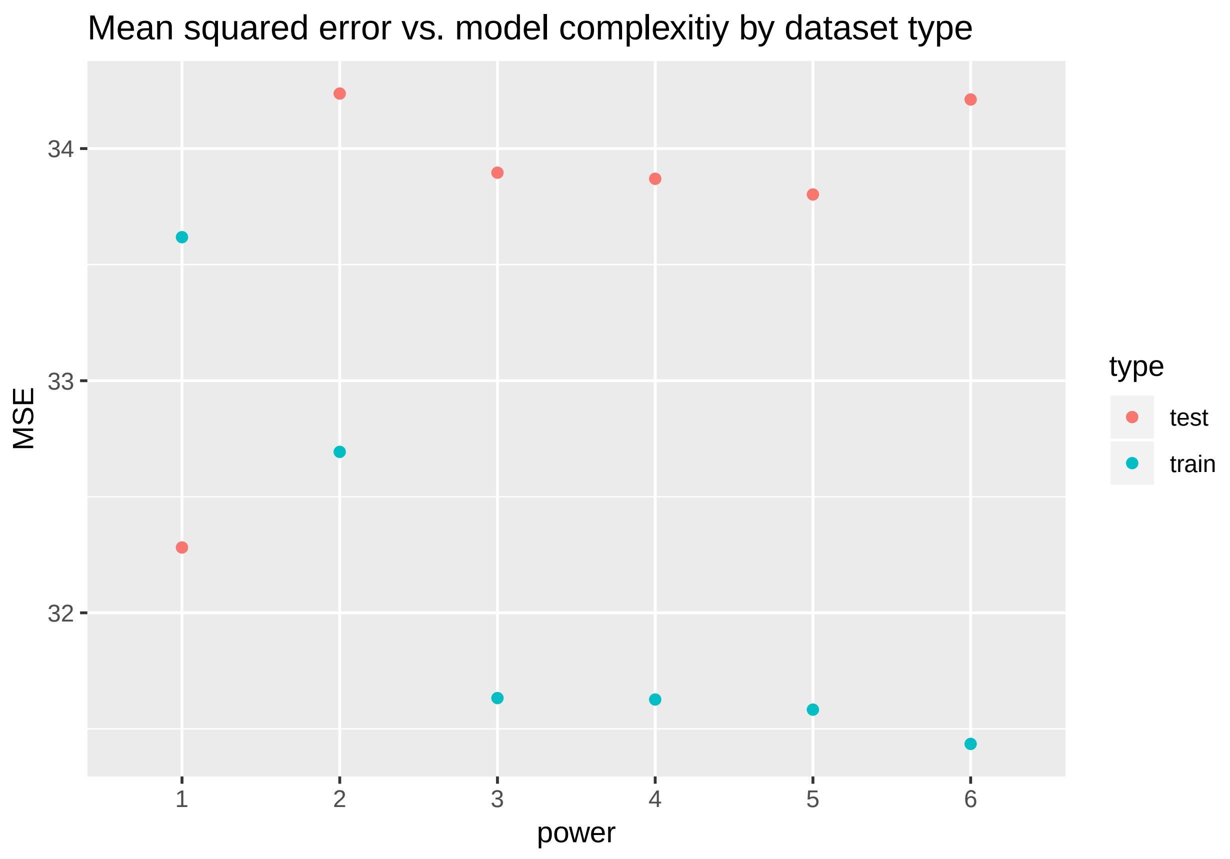
$$\sigma_M^2 : \text{variance of Moisture}$$

$$p : \text{degree of the polynomial}$$

3.3 Validation of the Model

```r
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.

3.4 Perform variable selection of a linear model in which Fat is response and Channel1:Channel100 are predicted by using stepAIC.

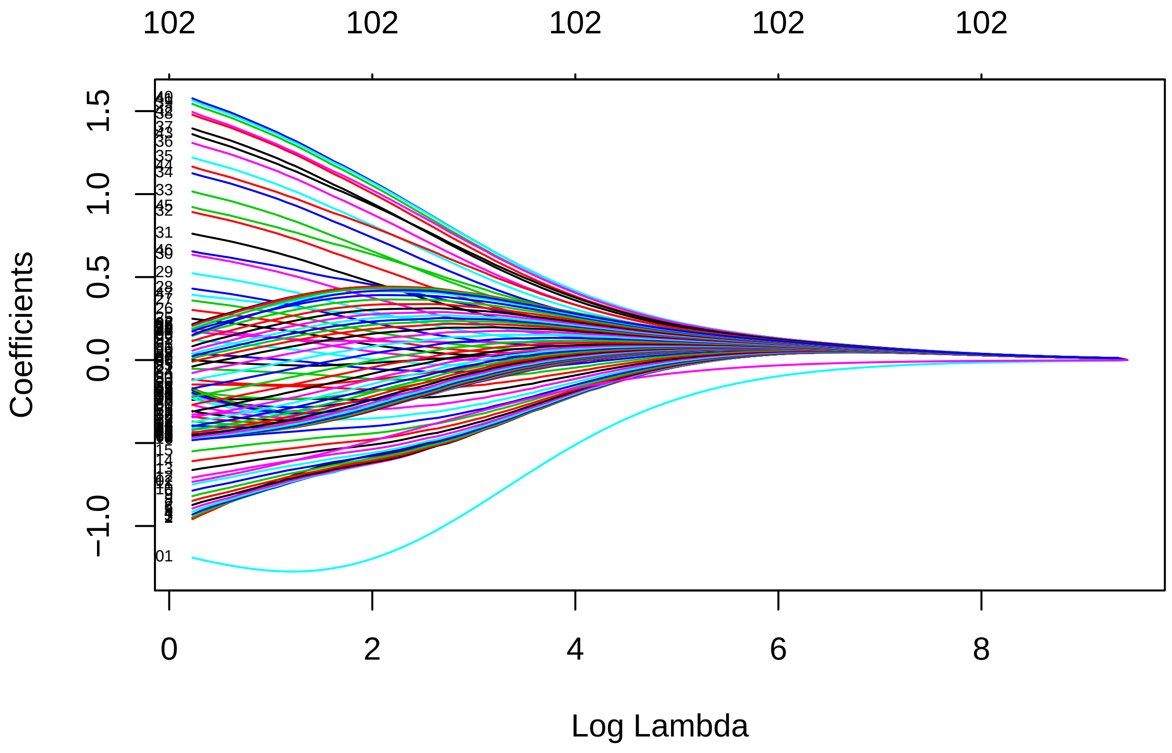
```
min.model1 = lm(Fat ~ 1, data=tecator_data[,-1])
biggest1 <- formula(lm(Fat ~.,  data=tecator_data[,-1]))

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 are above 0.0005 p values, thus the true variables may not even include these many.

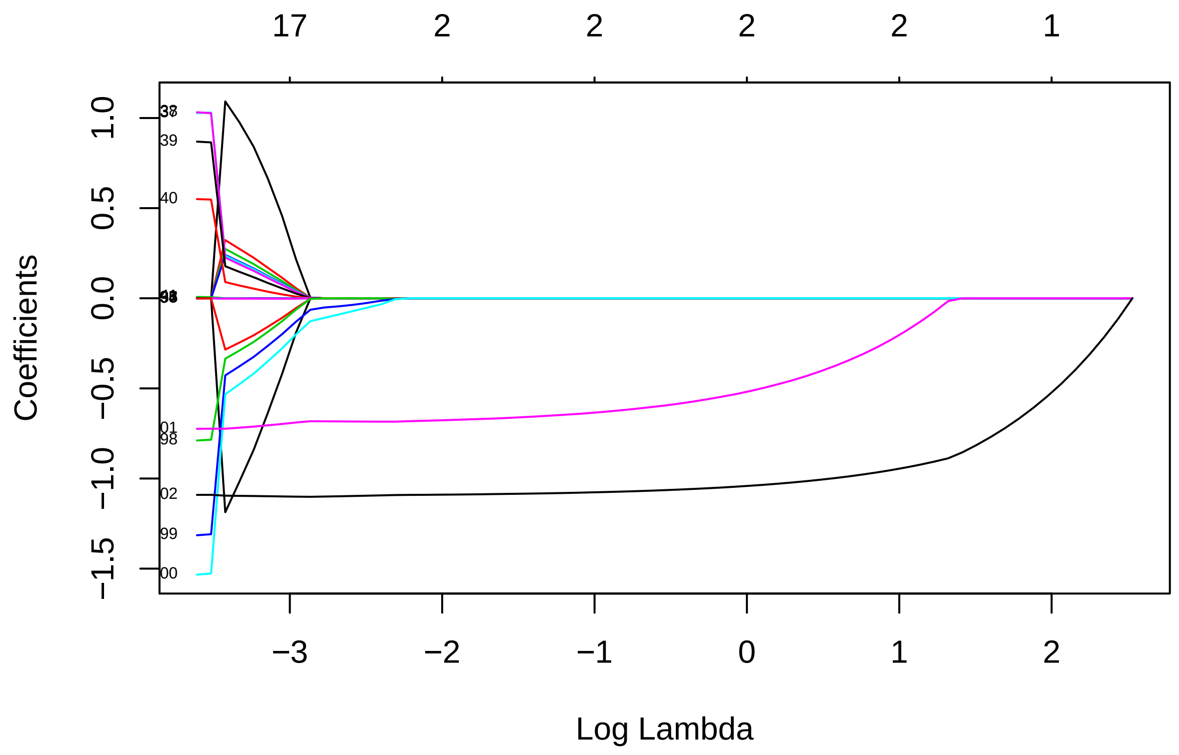
3.5 Fit a Ridge regression model with the same predictor and response

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

ridge_fit <- glmnet(x, y, alpha = 0, family = "gaussian")
plot(ridge_fit, xvar = "lambda", label = TRUE)
```



3.6 Fit a Lasso regression model with the same predictor and response

```r
lasso_fit <- glmnet(x, y, alpha = 1, family = "gaussian")
plot(lasso_fit, xvar = "lambda", label = TRUE)
```



Apendix

```r
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,
               kknn, tidyr, dplyr,reshape2, glmnet)

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

spam_data <- read.xlsx("spambase.xlsx", sheetName = "spambase_data")
spam_data$Spam <- as.factor(spam_data$Spam)

tecator_data <- read.xlsx("tecator.xlsx", sheetName = "data")
tecator_data <- tecator_data[,2:NCOL(tecator_data)] # removing sample column
set.seed(12345)

n =  NROW(spam_data)
id = sample(1:n, floor(n*0.5))
train = spam_data[id,]
test = spam_data[-id,]
best_model <- glm(formula = Spam ~., family = binomial, data = train)
summ(best_model)
```

```r
# prediction
train$prediction_prob <- predict(best_model, newdata = train, type = "response")
test$prediction_prob <- predict(best_model, newdata = test , type = "response")

train$prediction_class_50 <- ifelse(train$prediction_prob > 0.50, 1, 0)
test$prediction_class_50 <- ifelse(test$prediction_prob > 0.50, 1, 0)

train$prediction_class_90 <- ifelse(train$prediction_prob > 0.90, 1, 0)
test$prediction_class_90 <- ifelse(test$prediction_prob > 0.90, 1, 0)
# plots
ggplot(train, aes(prediction_prob, color = Spam)) +
geom_density(size = 1) + ggtitle("Training Set's Predicted Score for 50% cutoff") +
  scale_color_economist(name = "data", labels = c("negative", "positive")) +
  theme_economist()


ggplot(test, aes(prediction_prob, color = Spam)) +
geom_density(size = 1) + ggtitle("Test Set's Predicted Score for 50% cutoff") +
  scale_color_economist(name = "data", labels = c("negative", "positive")) +
  theme_economist()

#confusion table
conf_train <- table(train$Spam, train$prediction_class_50)
names(dimnames(conf_train)) <- c("Actual Train", "Predicted Train")
confusionMatrix(conf_train)

conf_test <- table(test$Spam, test$prediction_class_50)
names(dimnames(conf_test)) <- c("Actual Test", "Predicted Test")
confusionMatrix(conf_test)

#confusion table
conf_train1 <- table(train$Spam, train$prediction_class_90)
names(dimnames(conf_train1)) <- c("Actual Train", "Predicted Train")
conf_train1

conf_test1 <- table(test$Spam, test$prediction_class_90)
names(dimnames(conf_test1)) <- c("Actual Test", "Predicted Test")
conf_test1


cutoffs <- seq(from = 0.05, to = 0.95, by = 0.05)
accuracy <- NULL

for (i in seq_along(cutoffs)){
    prediction <- ifelse(test$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")
```

```r
knn_model30 <- train.kknn(Spam ~., data = train, kmax = 30)

train$knn_prediction_class <- predict(knn_model30, train)
test$knn_prediction_class <- predict(knn_model30, test)

conf_train2 <- table(train$Spam, train$knn_prediction_class)
names(dimnames(conf_train2)) <- c("Actual Train", "Predicted Train")
confusionMatrix(conf_train2)

conf_test2 <- table(test$Spam, test$knn_prediction_class)
names(dimnames(conf_test2)) <- c("Actual Test", "Predicted Test")
confusionMatrix(conf_test2)
knn_model1 <- train.kknn(Spam ~., data = train, kmax = 1)

train$knn_prediction_class <- predict(knn_model1, train)
test$knn_prediction_class <- predict(knn_model1, test)

conf_train2 <- table(train$Spam, train$knn_prediction_class)
names(dimnames(conf_train2)) <- c("Actual Train", "Predicted Train")
confusionMatrix(conf_train2)

conf_test2 <- table(test$Spam, test$knn_prediction_class)
names(dimnames(conf_test2)) <- c("Actual Test", "Predicted Test")
confusionMatrix(conf_test2)

subset_function <- function(X,Y,N){

# X = swiss[,1:5]
# Y = swiss[,6:6]
# N = 5

df <- cbind(X,Y)
temp <- NULL
final <- NULL

for(i in 1:NCOL(X)){
combs <- as.data.frame(gtools::combinations(NCOL(X), r=i, v=colnames(X), repeats.allowed=FALSE))
combs <- tidyr::unite(combs, "formula", sep = ",")
temp <- rbind(combs, temp)
}

set.seed(12345)
df2 <- df[sample(nrow(df)),]
df2$k_fold <- sample(N, size = nrow(df), replace = TRUE)

result <- NULL

for (j in 1:NROW(temp))
{
  for(i in 1:N){

train = df2[df2$k_fold != i,]
test = df2[df2$k_fold == i,]
vec <- temp[j,]
train_trimmed = lapply(strsplit(as.character(vec), ","), function(x) train[x])[[1]]
test_trimmed = lapply(strsplit(as.character(vec), ","), function(x) test[x])[[1]]

y_train = train[,c("Y"), drop = FALSE]
y_test = test[,c("Y"), drop = FALSE]

train_trimmed = as.matrix(train_trimmed)
test_trimmed = as.matrix(test_trimmed)
y_test = as.matrix(y_test)
y_train = as.matrix(y_train)

t_train =  as.matrix(t(train_trimmed))
t_test =  as.matrix(t(test_trimmed))

betas = solve(t_train %*% train_trimmed) %*% (t_train %*% y_train)

train_trimmed = as.data.frame(train_trimmed)
test_trimmed = as.data.frame(test_trimmed)

train_trimmed$type = "train"
test_trimmed$type = "test"
final <- rbind(train_trimmed, test_trimmed)


y_hat_val = as.matrix(final[,1:(ncol(final)-1)]) %*% betas
mse = (Y - y_hat_val)^ 2

data <- cbind(i, vec, mse, type = final$type)
result <- rbind(data, result)

}
}

result <- as.data.frame(result)

colnames(result) <- c("kfold", "variables", "mse", "type")

result$mse <- as.numeric(result$mse)
result$no_variables <- nchar(as.character(result$variables)) - nchar(gsub('\\,', "", result$variables))

variable_performance <- result %>% group_by(kfold, no_variables) %>%
  summarise(MSE = mean(mse, na.rm = TRUE))

myplot <- ggplot(data = variable_performance, aes(x = no_variables, y = MSE, color=kfold)) +
geom_line() + ggtitle("Plot of MSE vs. Number of variables by folds")

myplot2 <- ggplot(data = result, aes(x = variables, y = mse, color=kfold)) +
geom_bar(stat="identity") + ggtitle("Plot of RMSE vs. Features by folds") + coord_flip()

return(list(myplot, myplot2))
}
subset_function(X = swiss[,1:5], Y = swiss[,6], N = 5)
```

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

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)
```

```r
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()

ridge_fit <- glmnet(x, y, alpha = 0, family = "gaussian")
plot(ridge_fit, xvar = "lambda", label = TRUE)


lasso_fit <- glmnet(x, y, alpha = 1, family = "gaussian")
plot(lasso_fit, xvar = "lambda", label = TRUE)
```