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")
```

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

```
min.model = glm(Spam ~ 1, family=binomial, data=train)
biggest <- formula(glm(Spam ~., family=binomial, data=train))

step.model <- step(min.model, direction='forward', scope=biggest, trace = FALSE)
summary(step.model)
```

```
##
## Call:
## glm(formula = Spam ~ Word35 + Word46 + Word42 + Word44 + Word33 +
##     Word45 + Word39 + Word48 + Word30 + Word41 + Word43 + Word37 +
##     Word36 + Word31 + Word34 + Word27 + Word47 + Word13 + Word1 +
##     Word14 + Word22 + Word3 + Word25, family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8380  -0.4501  -0.0028   0.6678   3.5926
##
## Coefficients:
```

```
##                Estimate  Std. Error z value             Pr(>|z|)
## (Intercept)     1.61893     0.15717  10.300 < 0.0000000000000002 ***
## Word35         -7.21005     2.04613  -3.524             0.000426 ***
## Word46         -3.27987     0.51017  -6.429     0.00000000012848 ***
## Word42         -5.39752     1.29326  -4.174     0.00002998442908 ***
## Word44         -2.92415     0.64665  -4.522     0.00000612575895 ***
## Word33         -3.06853     0.67645  -4.536     0.00000572690173 ***
## Word45         -1.14418     0.16373  -6.988     0.00000000000278 ***
## Word39         -1.83922     0.41671  -4.414     0.00001016469695 ***
## Word48         -3.64684     1.10599  -3.297             0.000976 ***
## Word30         -1.80963     0.61159  -2.959             0.003087 **
## Word41       -320.20383 16770.81230  -0.019             0.984767
## Word43         -2.55804     0.72025  -3.552             0.000383 ***
## Word37         -0.85976     0.21258  -4.044     0.00005246958185 ***
## Word36         -1.64183     0.36746  -4.468     0.00000789257630 ***
## Word31         -4.63733     1.75043  -2.649             0.008067 **
## Word34         -6.18569     4.25451  -1.454             0.145970
## Word27         -0.20927     0.09473  -2.209             0.027163 *
## Word47         -3.66863     2.01705  -1.819             0.068941 .
## Word13          0.76472     0.34049   2.246             0.024705 *
## Word1          -0.79027     0.46161  -1.712             0.086900 .
## Word14         -0.53362     0.28254  -1.889             0.058941 .
## Word22         -0.45218     0.31199  -1.449             0.147242
## Word3          -0.31401     0.17278  -1.817             0.069157 .
## Word25         -0.08104     0.04980  -1.627             0.103670
## ---
## 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:  940.51  on 1346  degrees of freedom
## AIC: 988.51
##
## Number of Fisher Scoring iterations: 23
```

Manual Feature Selection

```
best_model <- glm(formula = Spam ~ Word35 + Word46 + Word42 + Word44 + Word33 +
    Word45 + Word39 + Word48 + Word30 + Word43 + Word37 +
    Word36 + Word31, family = binomial, data = train)
```

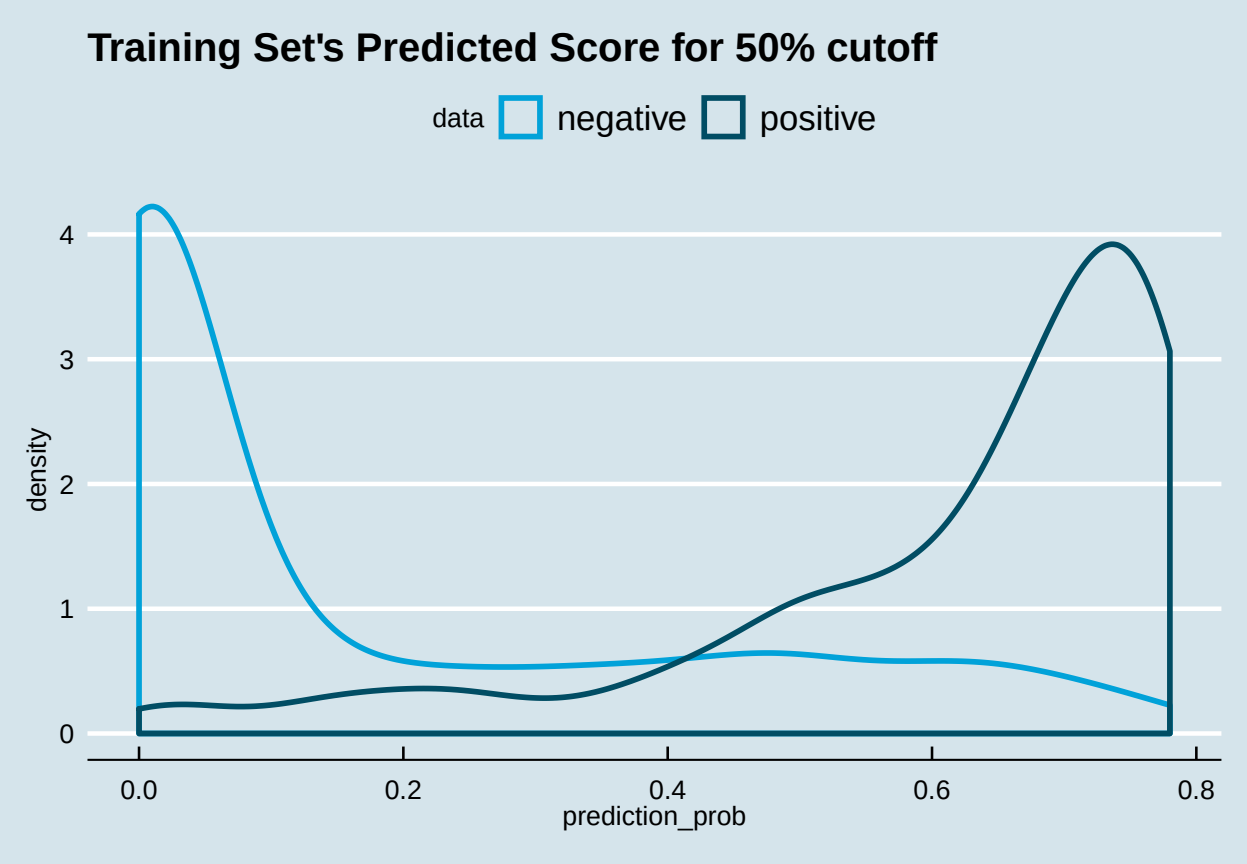
Prediction for probability greater than 50% and 90%

```
# 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)
```

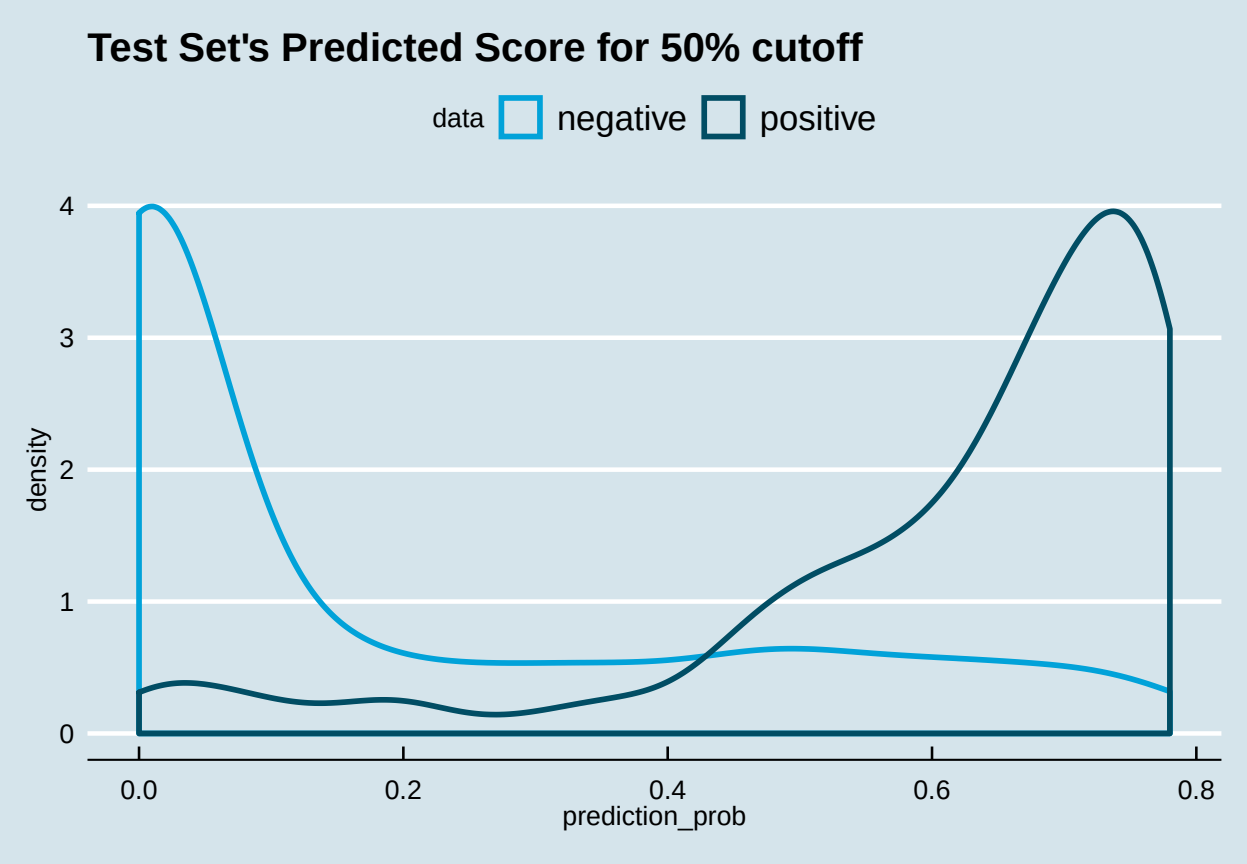
```
train$prediction_class_90 <- ifelse(train$prediction_prob > 0.90, 1, 0)
test$prediction_class_90 <- ifelse(test$prediction_prob > 0.90, 1, 0)
```

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%

```
#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
##            0 799 146
##            1  88 337
##
##                Accuracy : 0.8292
##                  95% CI : (0.8082, 0.8488)
##     No Information Rate : 0.6474
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.6153
##  Mcnemar's Test P-Value : 0.0001944
##
##             Sensitivity : 0.9008
##             Specificity : 0.6977
##          Pos Pred Value : 0.8455
```

```
##          Neg Pred Value : 0.7929
##              Prevalence : 0.6474
##          Detection Rate : 0.5832
##    Detection Prevalence : 0.6898
##       Balanced Accuracy : 0.7993
##
##        '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
##           0 785 152
##           1  80 353
##
##                Accuracy : 0.8307
##                  95% CI : (0.8097, 0.8502)
##     No Information Rate : 0.6314
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.6251
##  Mcnemar's Test P-Value : 0.000003141
##
##             Sensitivity : 0.9075
##             Specificity : 0.6990
##          Pos Pred Value : 0.8378
##          Neg Pred Value : 0.8152
##              Prevalence : 0.6314
##          Detection Rate : 0.5730
##    Detection Prevalence : 0.6839
##       Balanced Accuracy : 0.8033
##
##        '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 83%, thus the misclassification rate is 17% on 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)
names(dimnames(conf_train1)) <- c("Actual Train", "Predicted Train")
```

```
conf_train1
```

```
##              Predicted Train
## Actual Train   0
##            0 945
##            1 425
```

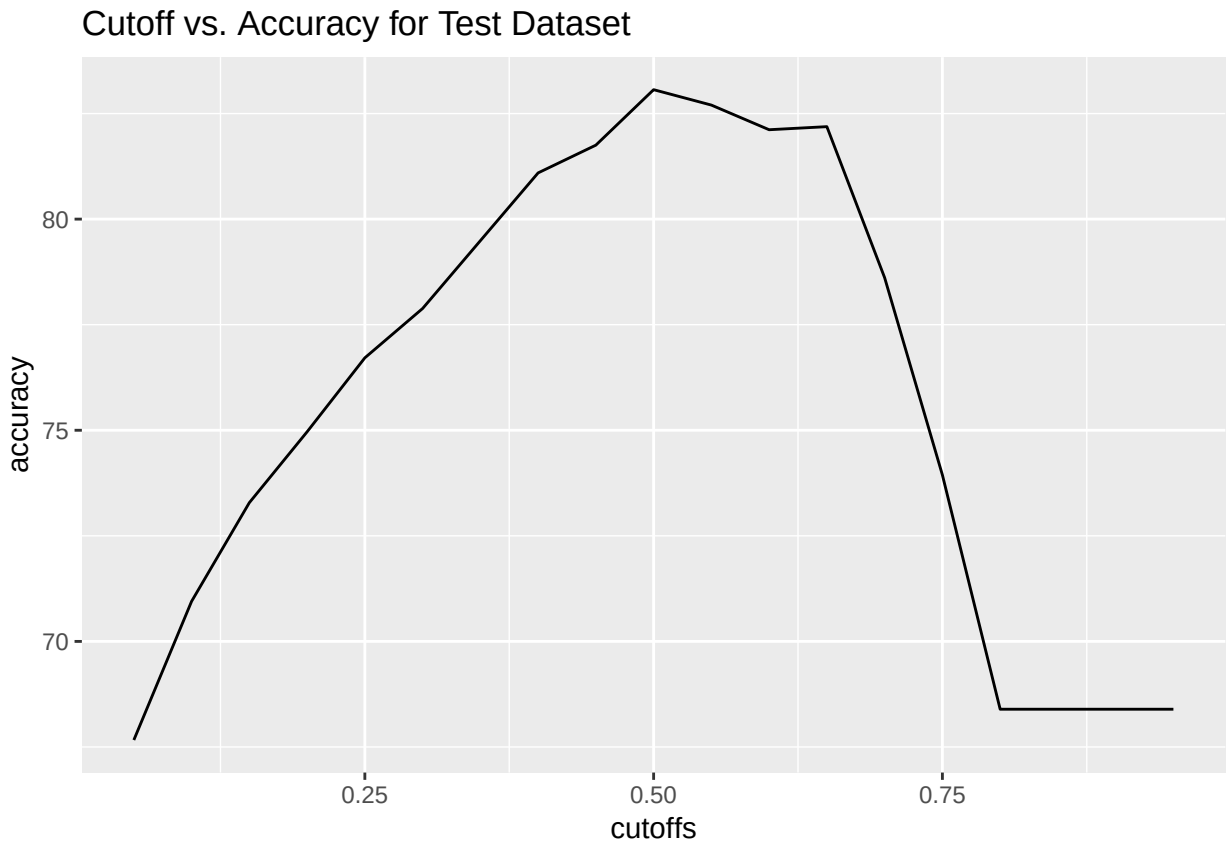
```
conf_test1 <- table(test$Spam, test$prediction_class_90)
names(dimnames(conf_test1)) <- c("Actual Test", "Predicted Test")
conf_test1
```

```
##             Predicted Test
## Actual Test   0
##           0 937
##           1 433
```

Analysis: Strange, the model 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

```
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 50% was the best for our purpose and going higher than this leads to worse results, at 0.8 and above the accuracy drastically reduces which is what we see when we make cutoff as 0.9.

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 no better than 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 ~ Word35 + Word46 + Word42 + Word44 + Word33 +
    Word45 + Word39 + Word48 + Word30 + Word43 + Word37 +
    Word36 + Word31, 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
##            0 869  76
##            1  48 377
##
##                Accuracy : 0.9095
##                  95% CI : (0.893, 0.9242)
##     No Information Rate : 0.6693
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.7923
##  Mcnemar's Test P-Value : 0.01532
##
##             Sensitivity : 0.9477
##             Specificity : 0.8322
##          Pos Pred Value : 0.9196
##          Neg Pred Value : 0.8871
##              Prevalence : 0.6693
##          Detection Rate : 0.6343
##    Detection Prevalence : 0.6898
##       Balanced Accuracy : 0.8899
##
##        '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
##           0 800 137
##           1  67 366
##
##                Accuracy : 0.8511
##                  95% CI : (0.8311, 0.8695)
##     No Information Rate : 0.6328
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.6699
##  Mcnemar's Test P-Value : 0.000001359
##
##             Sensitivity : 0.9227
##             Specificity : 0.7276
##          Pos Pred Value : 0.8538
##          Neg Pred Value : 0.8453
##              Prevalence : 0.6328
##          Detection Rate : 0.5839
##    Detection Prevalence : 0.6839
##       Balanced Accuracy : 0.8252
##
##        'Positive' Class : 0
```

##

Analysis: Using KKNN with K = 30, we increased our trainning accuracy to 90%, thus misclassification is 10%, however for the test dataset misclassification rate is about 15%.

Thus compared to using logisitc model the misclassification error for the trainning dataset decreased by 7% to just 10%, while for the test dataset the misclassification error decreased only by 2% to 15%.

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?

```
knn_model1 <- train.kknn(Spam ~ Word35 + Word46 + Word42 + Word44 + Word33 +
    Word45 + Word39 + Word48 + Word30 + Word43 + Word37 +
    Word36 + Word31, 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
##            0 912  33
##            1  18 407
##
##                Accuracy : 0.9628
##                  95% CI : (0.9513, 0.9722)
##     No Information Rate : 0.6788
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.9139
##  Mcnemar's Test P-Value : 0.04995
##
##             Sensitivity : 0.9806
##             Specificity : 0.9250
##          Pos Pred Value : 0.9651
##          Neg Pred Value : 0.9576
##              Prevalence : 0.6788
##          Detection Rate : 0.6657
##    Detection Prevalence : 0.6898
##       Balanced Accuracy : 0.9528
##
##        '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
##           0 782 155
##           1  77 356
##
##                Accuracy : 0.8307
##                  95% CI : (0.8097, 0.8502)
##     No Information Rate : 0.627
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.6264
##  Mcnemar's Test P-Value : 0.0000004297
##
##             Sensitivity : 0.9104
##             Specificity : 0.6967
##          Pos Pred Value : 0.8346
##          Neg Pred Value : 0.8222
##              Prevalence : 0.6270
##          Detection Rate : 0.5708
##    Detection Prevalence : 0.6839
##       Balanced Accuracy : 0.8035
##
##        'Positive' Class : 0
##
```

Analysis: Using KKNN with K = 1, we increased our trainning accuracy to 96%, thus misclassification is 4%, however for the test dataset accuracy is 83% thus misclassification rate is about 17%, thus we improved on the trainning accuracy but did bad on the test case, thus this is an example of overfitting more bias.

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
```

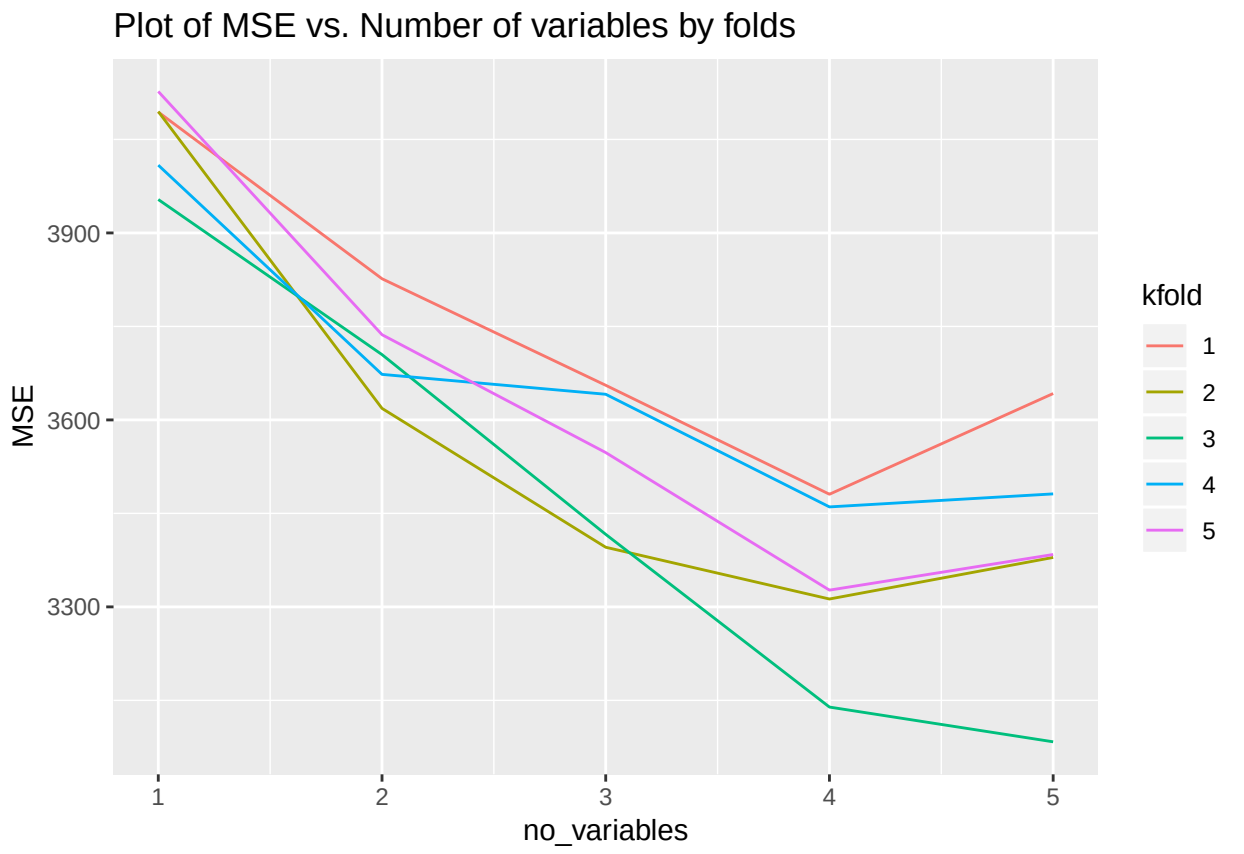
```
for(i in 1:NCOL(X)){
combs <- as.data.frame(gtools::combinations(NCOL(X), r=i, v=colnames(X), repeats.allowed=FALSE))
combs <- tidyr::unite(combs, "formula", sep = ",")
temp <- rbind(combs, temp)
}

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))
}
```

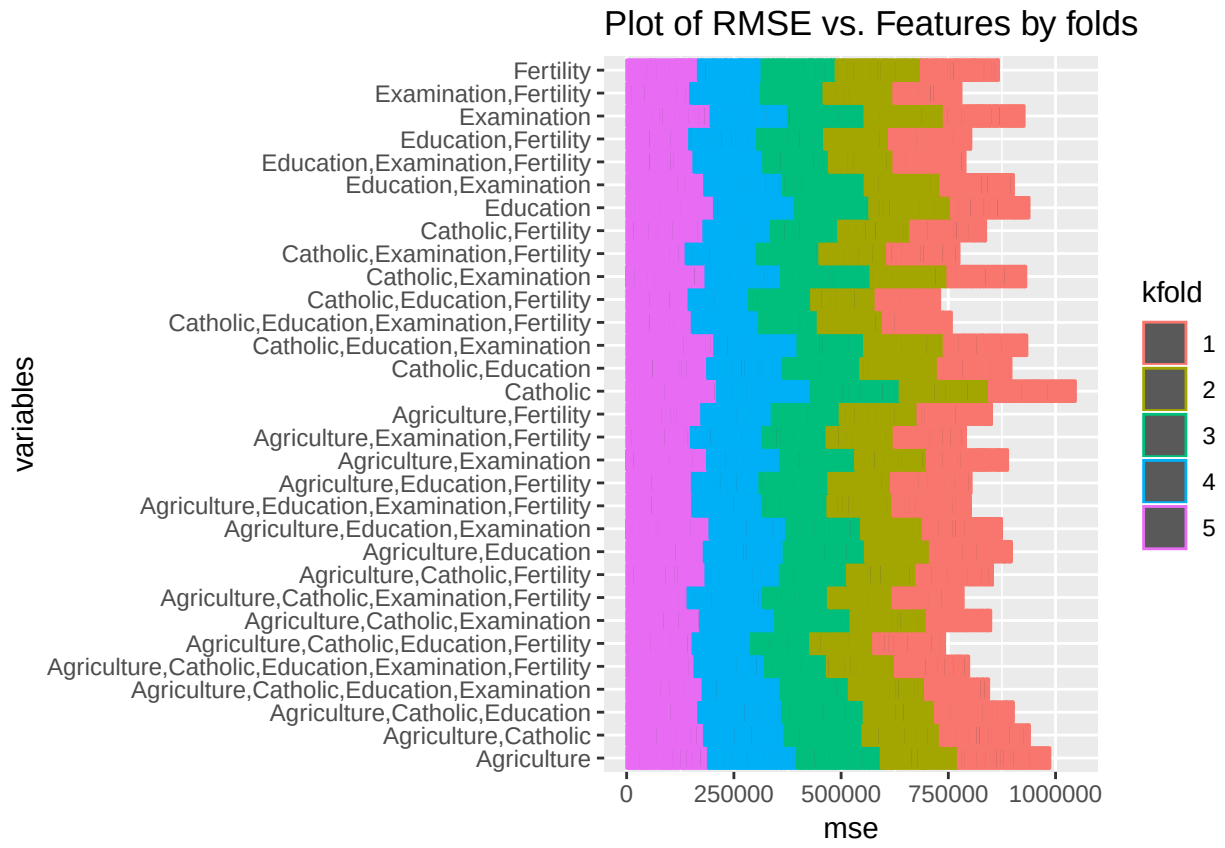
2.2 Test your function on data set swiss available in the standard R repository:

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

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



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



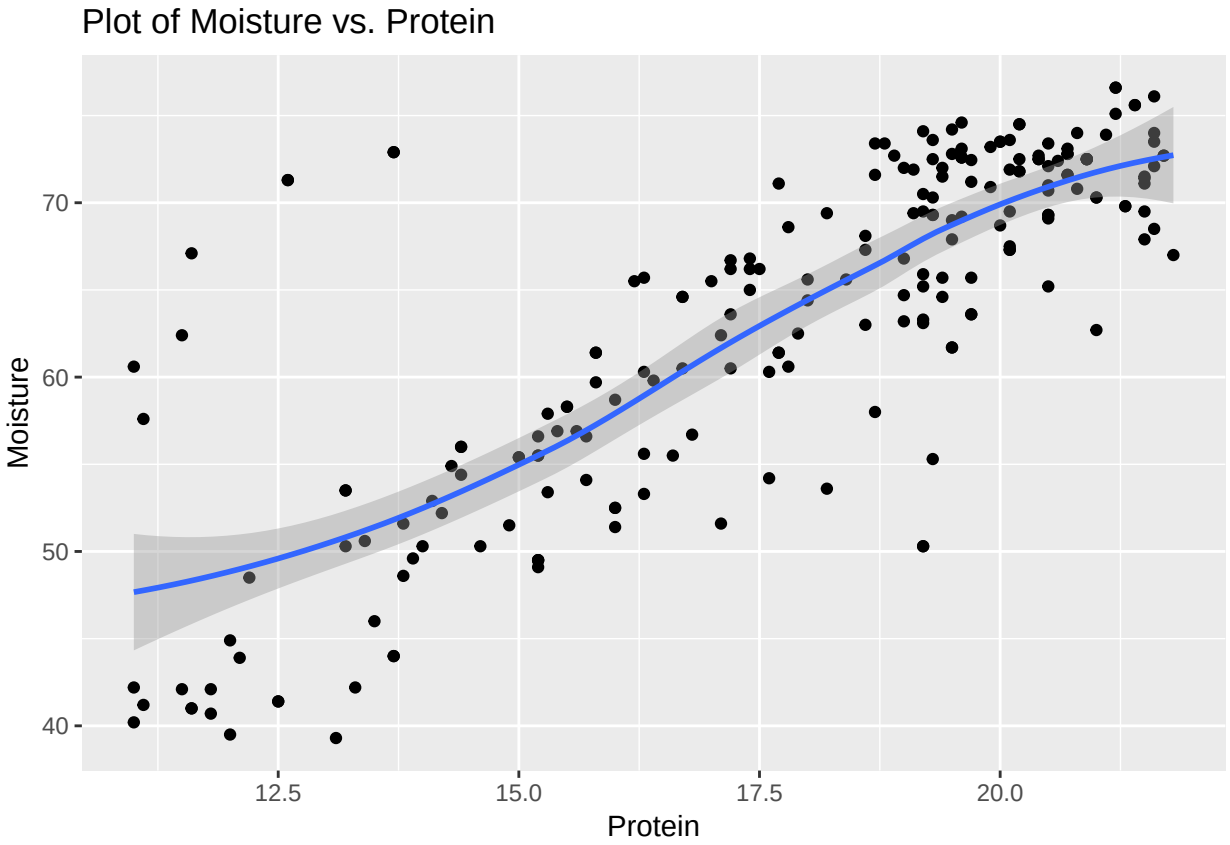
Analysis:

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() +
  ggtitle("Plot of Moisture vs. Protein")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



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

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

$$Where,$$

σ_M^2 : variance of Moisture

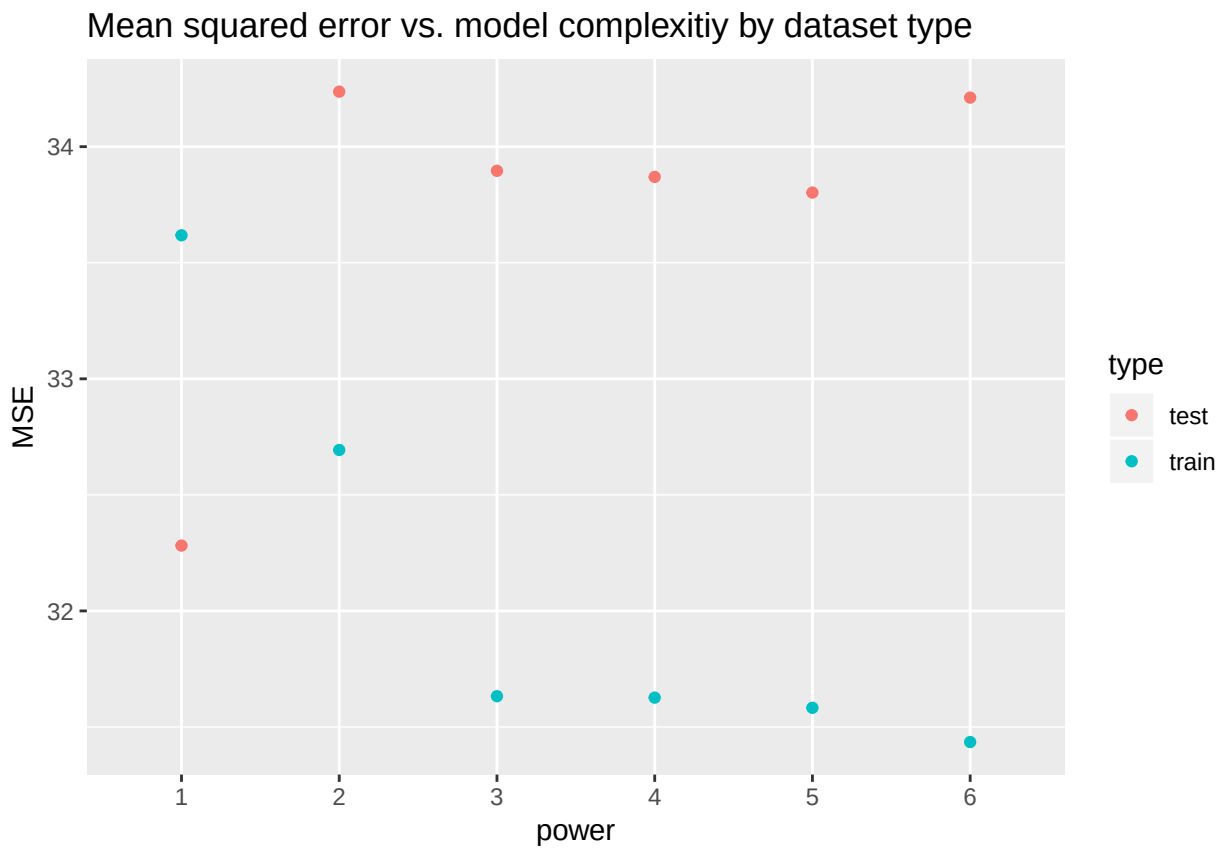
p : degree of the polynomial

Analysis: Thus the model that fits the Mositure is normally distributed with mean of 63.2044 and standard distribution of 9.87

3.3 Validation of the Model

```
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)
```

```r
# 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")
```



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)
```

```
summary(step.model1)
```

```
##
## Call:
## lm(formula = Fat ~ Moisture + Protein + Channel100 + Channel41 +
##     Channel7 + Channel48 + Channel42 + Channel50 + Channel45 +
##     Channel66 + Channel56 + Channel90 + Channel60 + Channel70 +
##     Channel67 + Channel59 + Channel65 + Channel58 + Channel44 +
##     Channel18 + Channel78 + Channel84 + Channel62 + Channel53 +
##     Channel75 + Channel57 + Channel63 + Channel24 + Channel37,
##     data = tecator_data[, -1])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.27136 -0.28488 -0.00599  0.33002  1.88817
##
## Coefficients:
##                Estimate  Std. Error t value             Pr(>|t|)
## (Intercept)    93.46223     1.58787  58.860 < 0.0000000000000002 ***
## Moisture       -1.03169     0.01902 -54.253 < 0.0000000000000002 ***
## Protein        -0.64377     0.05899 -10.914 < 0.0000000000000002 ***
## Channel100     66.56349    48.17557   1.382             0.168735
## Channel41   -3268.10600   826.91869  -3.952             0.000110 ***
## Channel7      -64.02598    20.79831  -3.078             0.002398 **
## Channel48   -2022.45968   254.45994  -7.948    0.000000000000181 ***
## Channel42    4934.22494  1124.96237   4.386    0.000019340406280 ***
## Channel50    1239.51753   236.09108   5.250    0.000000414456027 ***
## Channel45    4796.21682   783.38496   6.122    0.000000005394177 ***
## Channel66    2435.78706  1169.84707   2.082             0.038705 *
## Channel56    2372.99590   540.06095   4.394    0.000018721173105 ***
## Channel90    -258.26893   247.22053  -1.045             0.297529
## Channel60    -264.27434   708.11461  -0.373             0.709421
## Channel70      14.24897   327.11649   0.044             0.965303
## Channel67   -2015.91599   543.73686  -3.708             0.000276 ***
## Channel59     635.71013   996.30528   0.638             0.524219
## Channel65    -941.60761  1009.23045  -0.933             0.352038
```

```
## Channel58    1054.24379   927.95085   1.136            0.257385
## Channel44   -5733.84252  1079.18915  -5.313    0.000000307504880 ***
## Channel18     299.80050    88.43461   3.390            0.000854 ***
## Channel78    2371.11031   361.25352   6.564    0.000000000513410 ***
## Channel84    -428.99237   338.34806  -1.268            0.206426
## Channel62    3062.97254   769.58521   3.980    0.000098872806408 ***
## Channel53    -804.39127   203.44010  -3.954            0.000109 ***
## Channel75   -1461.42310   402.26061  -3.633            0.000363 ***
## Channel57   -3266.78970   876.70727  -3.726            0.000258 ***
## Channel63   -2844.66233   906.40307  -3.138            0.001977 **
## Channel24    -308.71263    97.87177  -3.154            0.001878 **
## Channel37     401.64118   151.75576   2.647            0.008830 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5005 on 185 degrees of freedom
## Multiple R-squared:  0.9987, Adjusted R-squared:  0.9985
## F-statistic:  4775 on 29 and 185 DF,  p-value: < 0.00000000000000022
```

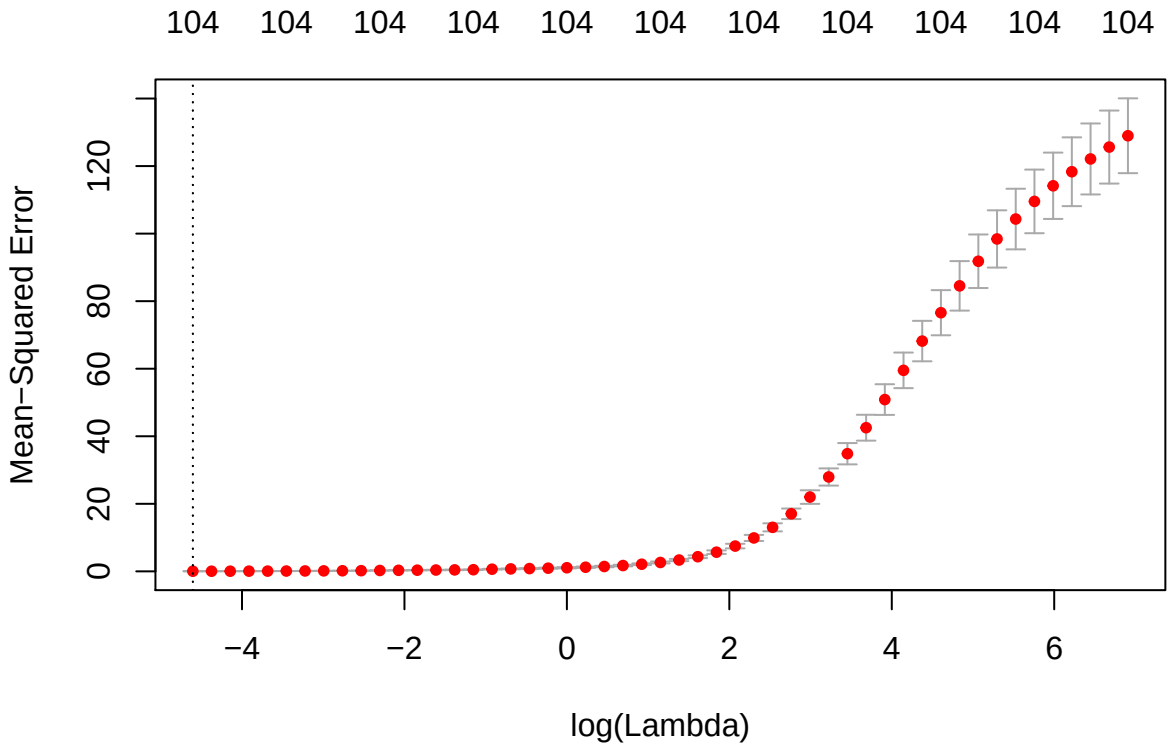
Analysis: 29 variables were choose out of 107. Even among these there are many which have very low p values thus statistically it is a practice to remove variables which are above 0.0005 p values, thus the true variables may not even include these many.

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

```
y <- tecator_data$Fat
x <- tecator_data %>% data.matrix()
lambdas <- 10^seq(3, -2, by = -.1)

ridge_fit <- cv.glmnet(x, y, alpha = 0, lambda = lambdas)
dim(coef(ridge_fit))
```

```
## [1] 105   1
```

```
plot(ridge_fit)
```




```
opt_lambda <- ridge_fit$lambda.min
best_ridge <- ridge_fit$glmnet.fit
```

Apendix

```
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)

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")
set.seed(12345)

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

```
min.model = glm(Spam ~ 1, family=binomial, data=train)
biggest <- formula(glm(Spam ~., family=binomial, data=train))

step.model <- step(min.model, direction='forward', scope=biggest, trace = FALSE)
summary(step.model)
best_model <- glm(formula = Spam ~ Word35 + Word46 + Word42 + Word44 + Word33 +
    Word45 + Word39 + Word48 + Word30 + Word43 + Word37 +
    Word36 + Word31, family = binomial, data = train)
# 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")

knn_model30 <- train.kknn(Spam ~ Word35 + Word46 + Word42 + Word44 + Word33 +
    Word45 + Word39 + Word48 + Word30 + Word43 + Word37 +
    Word36 + Word31, 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 ~ Word35 + Word46 + Word42 + Word44 + Word33 +
    Word45 + Word39 + Word48 + Word30 + Word43 + Word37 +
    Word36 + Word31, 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)
ggplot(data = tecator_data, aes(x = Protein, y = Moisture)) +
  geom_point() +
  geom_smooth() +
  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)
```

```r
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)
summary(step.model1)

y <- tecator_data$Fat
x <- tecator_data %>% data.matrix()
lambdas <- 10^seq(3, -2, by = -.1)

ridge_fit <- cv.glmnet(x, y, alpha = 0, lambda = lambdas)
dim(coef(ridge_fit))
plot(ridge_fit)


opt_lambda <- ridge_fit$lambda.min
best_ridge <- ridge_fit$glmnet.fit
```