machine learning(732A99) lab1

Anubhav Dikshit(anudi287)

26 November 2018

Contents

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 and report the confusion matrices (use table()) and the misclassification rates for training and test data. Analyse the obtained results.

Manual Feature Selection

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

```
##
## 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)
```

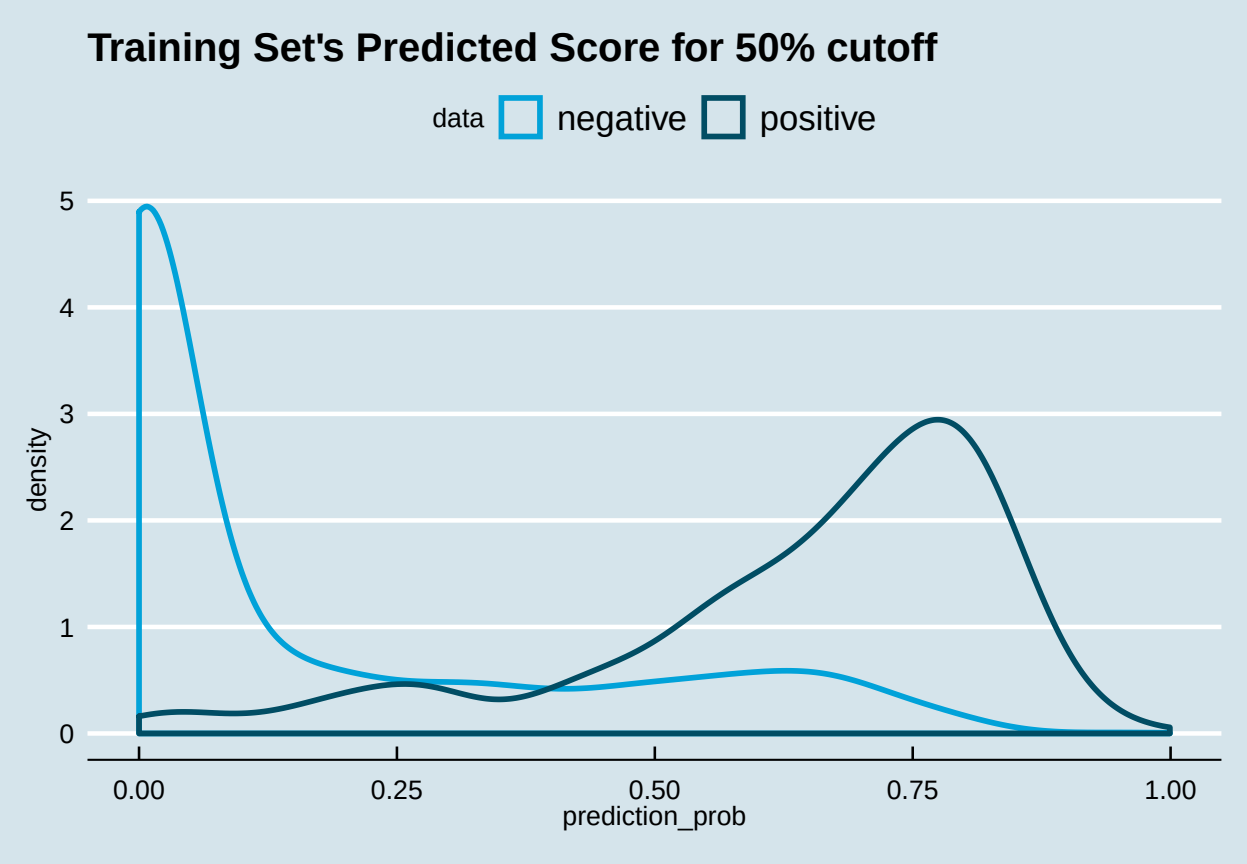
```
##
##     Null deviance: 1696.82  on 1369  degrees of freedom
## Residual deviance:  928.54  on 1321  degrees of freedom
## AIC: 1026.5
##
## Number of Fisher Scoring iterations: 23
```

Prediction for probability greater than 50% and 90%

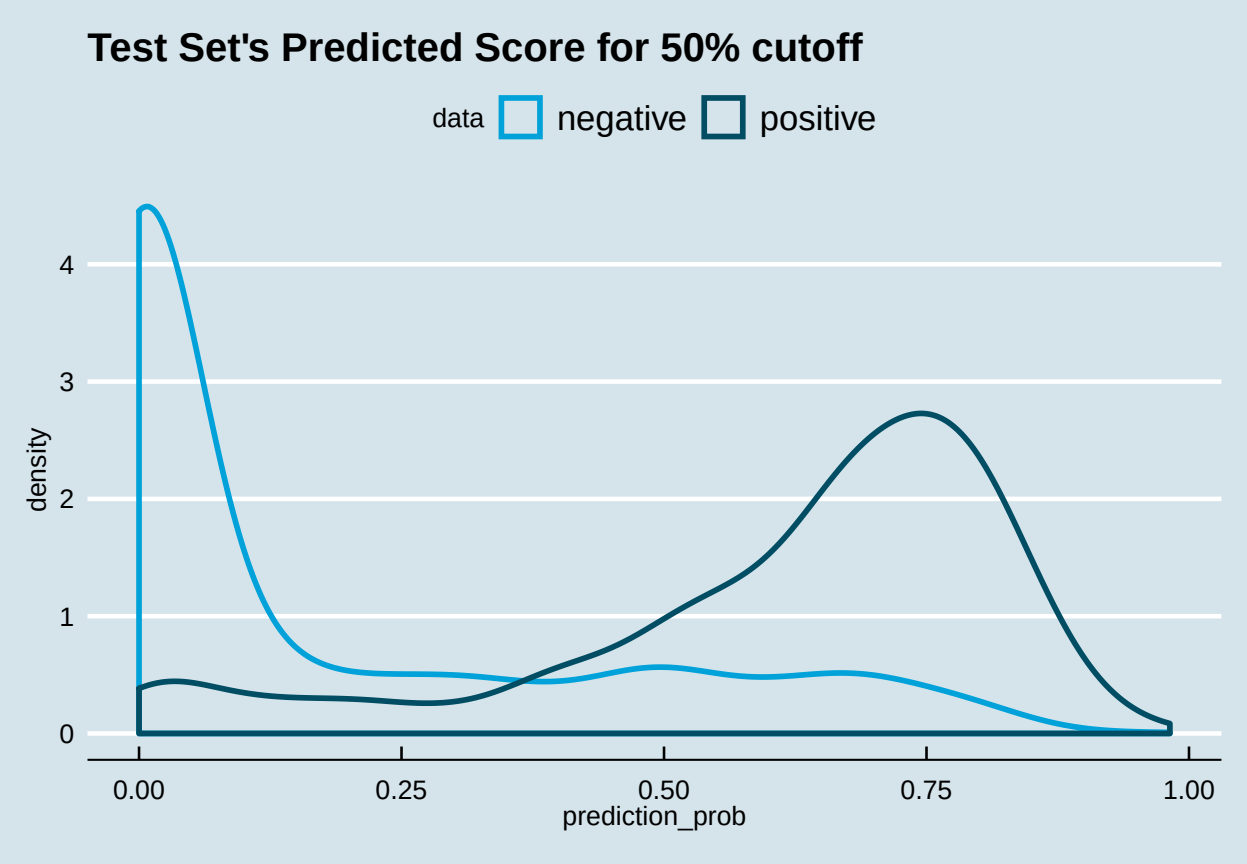
```
# prediction
train$prediction_prob <- predict(best_model, newdata = train, type = "response")
test$prediction_prob <- predict(best_model, newdata = test , type = "response")

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
##            0 803 142
##            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 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?

```
#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   1
##            0 944   1
##            1 419   6
```

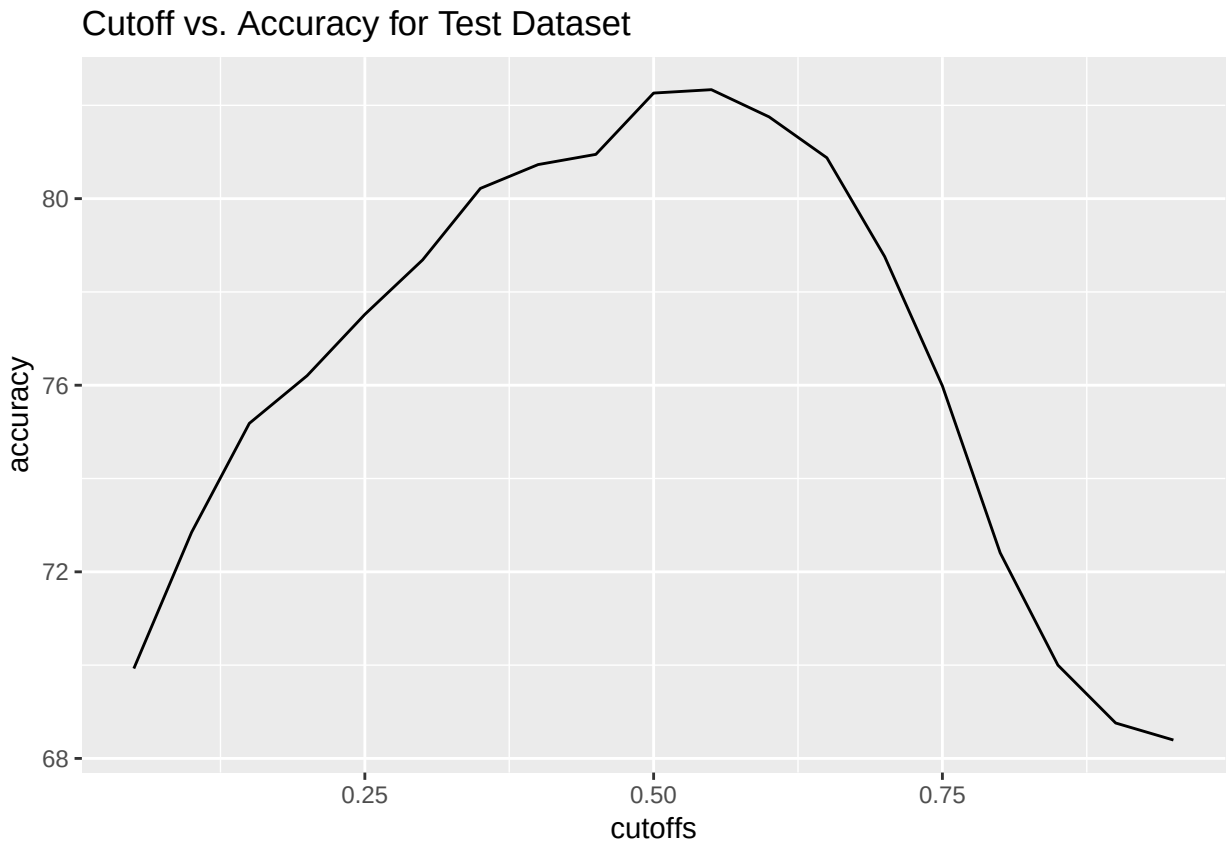
```
conf_test1 <- table(test$Spam, test$prediction_class_90)
names(dimnames(conf_test1)) <- c("Actual Test", "Predicted Test")
conf_test1
```

```
##             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

```
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
##
```

```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
##           0 801 136
##           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?

```
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
##
```

```
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)

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

# X = swiss[,2:6]
# Y = swiss[,1:1]
# 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 = ",")
```

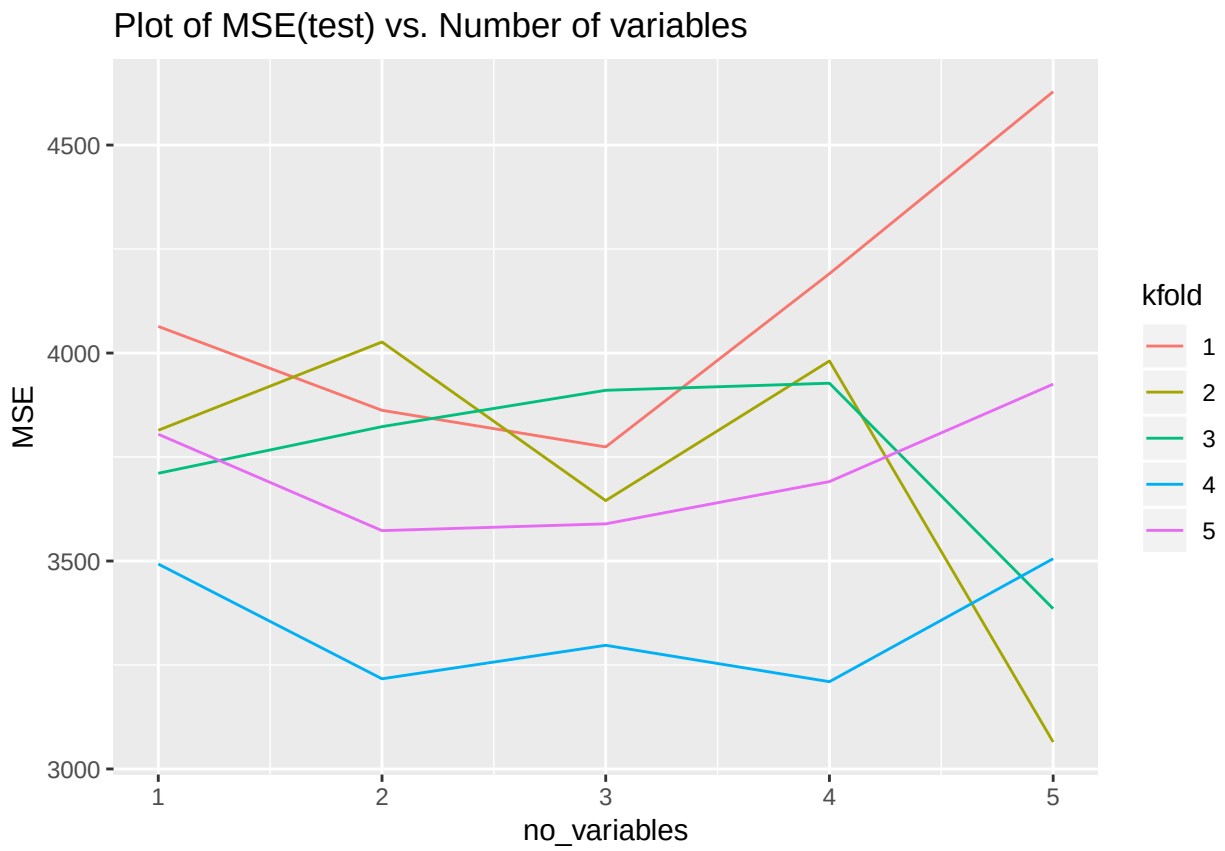
```
temp <- rbind(combs, temp)
}

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))

result_test <- result %>% filter(type == "test")

variable_performance <- result_test %>% 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(test) vs. Number of variables")

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

best_variables <- result_test %>% group_by(variables) %>%
  summarise(MSE = mean(mse, na.rm = TRUE)) %>% arrange(MSE) %>%
  select(variables) %>% slice(1) %>% as.vector()

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

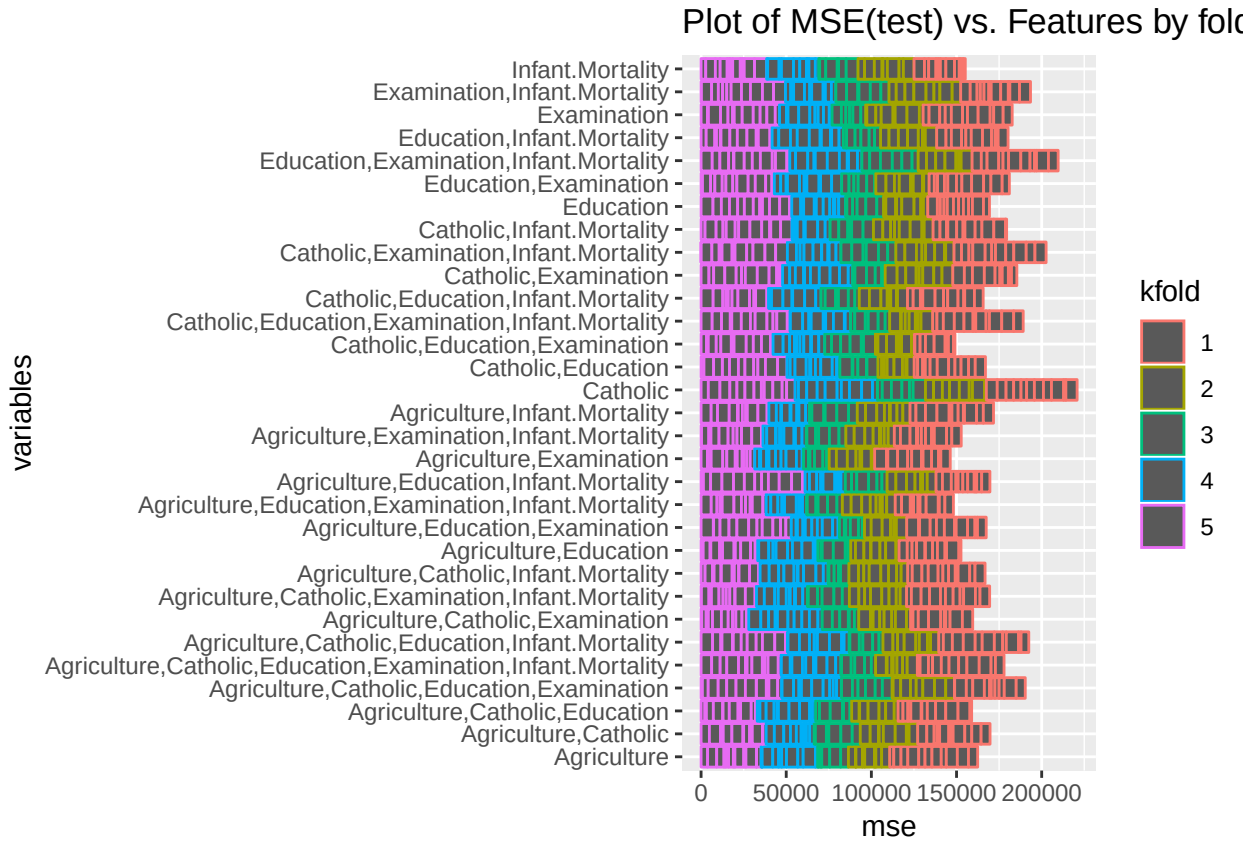
2.2 Test your function on data set swiss available in the standard R repository:

```
subset_function(X = swiss[,2:6], Y = swiss[,1:1], N = 5)
```

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

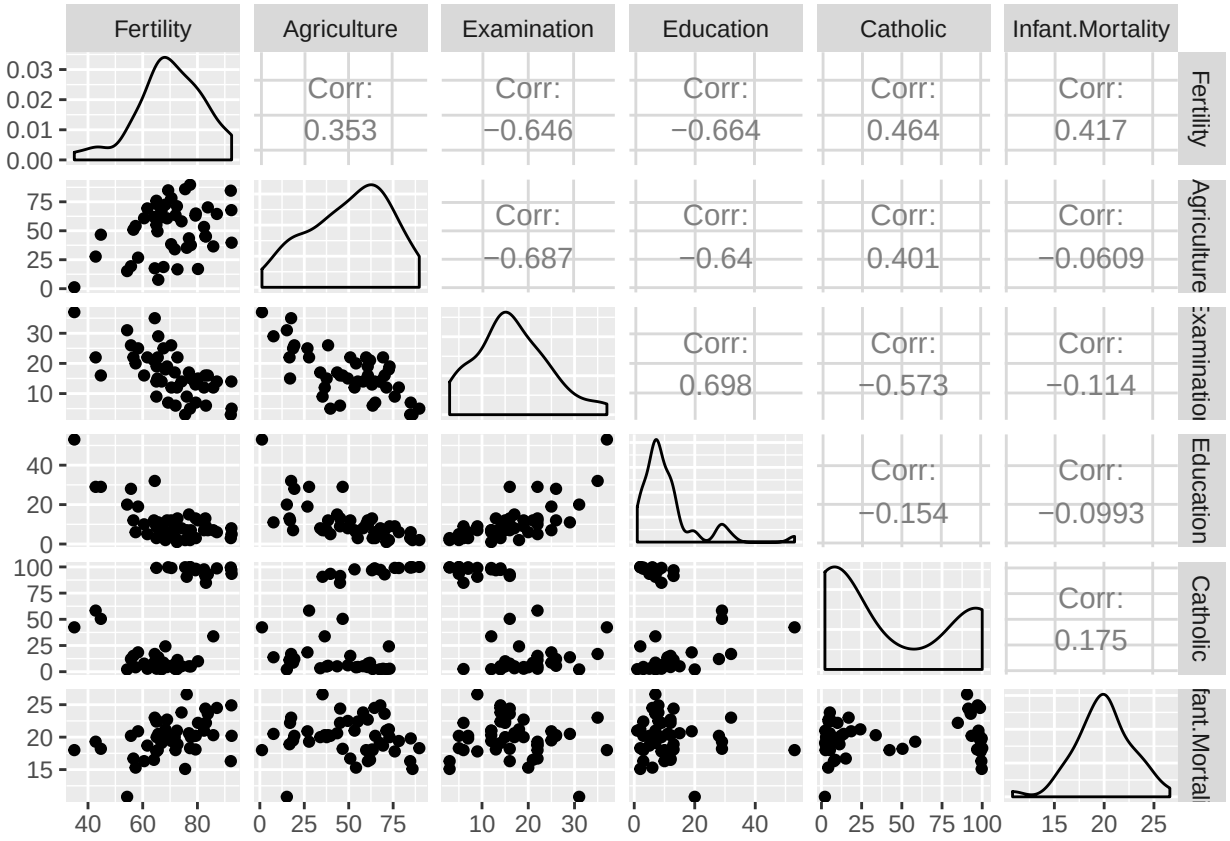

Plot of MSE(test) vs. Number of variables
MSE
4500
4000
3500
3000
1
2
3
4
5
no_variables
kfold
1
2
3
4
5

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




```
##
## [[3]]
## # A tibble: 1 x 1
##   variables
##   <fct>
## 1 Agriculture,Examination
```

```
# Using pairs on swiss dataset
GGally::ggpairs(swiss)
```



Analysis: Swiss dataset is a dataset which represents fertility measure and socio-economic indicators for each of 47 French speaking provinces of Switzerland.

From the dataset we have the following information:

Fertility -> 'common standardized fertility measure' Agriculture -> % of males involved in agriculture as occupation Examination -> % draftees receiving highest mark on army examination Education -> % education beyond primary school for draftees. Catholic -> % 'catholic' (as opposed to 'protestant'). Infant.Mortality -> live births who live less than 1 year.

From the graph we can see that there is correleation between Fertility and other variables, for variable 'Catholic'and 'Education', the relationship is not strong, while other variable do exhibit a reasonable correleation.

From our model we can see that the models with 2 features gives us the least mean squared error across most folds. Implies our best model is a function of 2 variables.

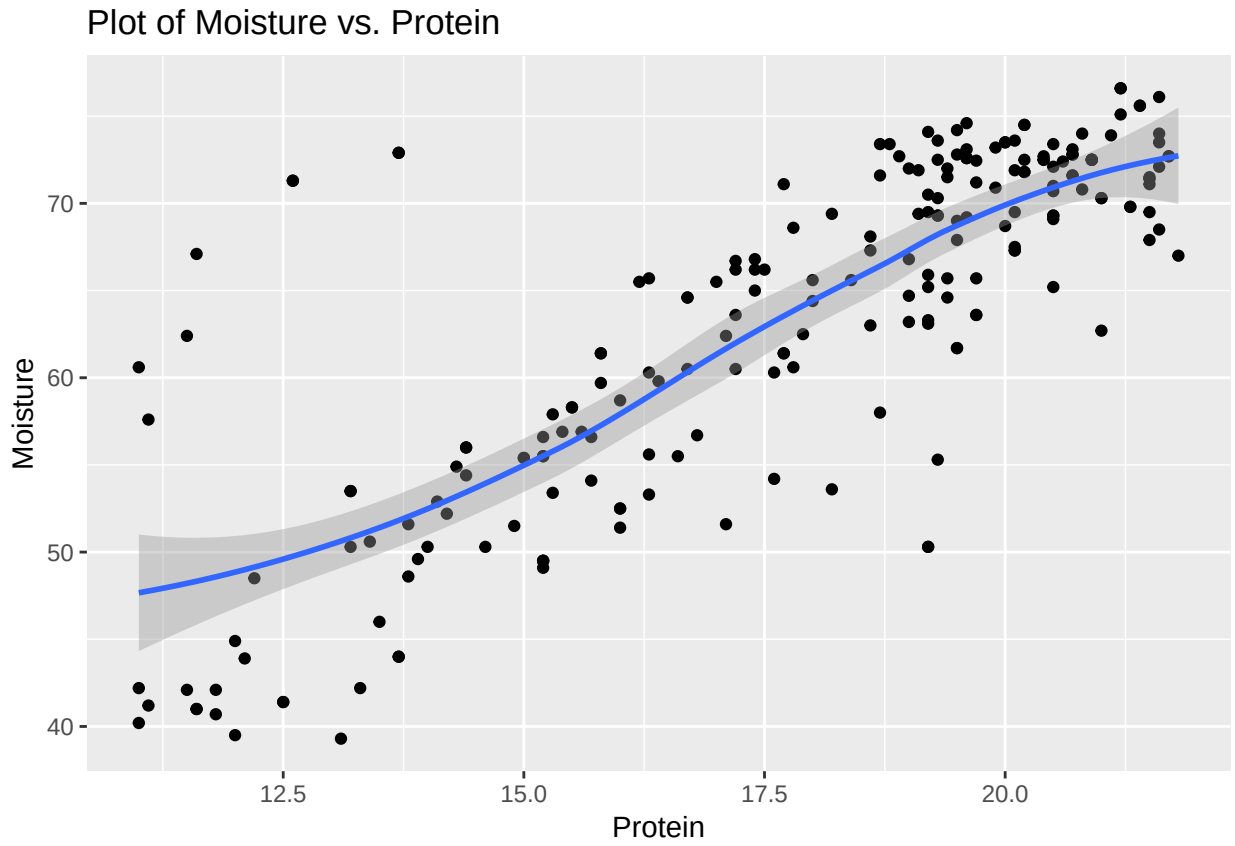
Our model predicts that variable 'Examination' and 'Agriculture' are the strongest predictors of 'Fertility', these results makes sense but from the graph it could be seen that even 'Infant.Mortality' might have a strong predictor.

Logically speaking infant mortality cannot be direct estimator of fertility, higher infant mortility could be responsible in population decrease but not in reduction in fertility.

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 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

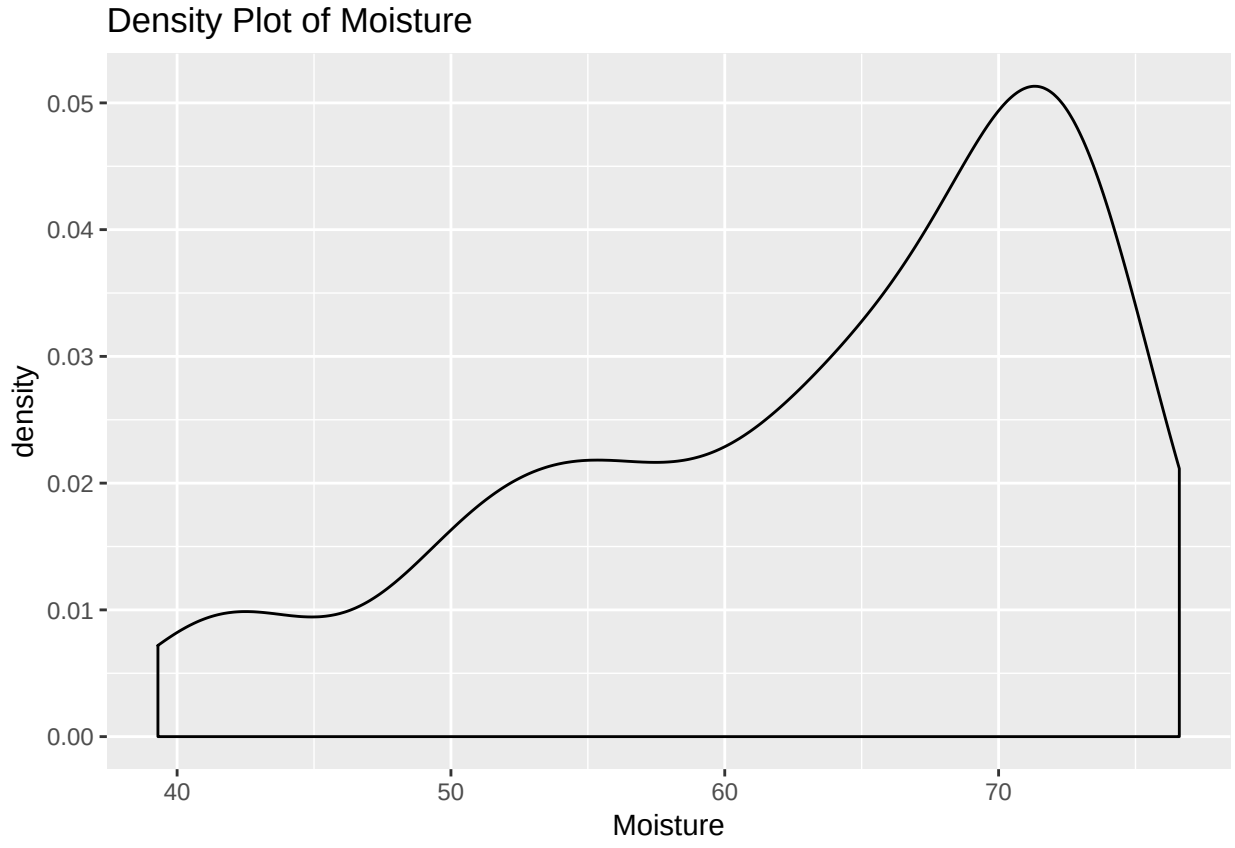
$$P\left(M_i | X_{Protein}, \vec{\beta}\right) = N\left(\sum_{i=0}^{p} X^i Protein * \beta i, \sigma_M^2\right)$$

$Where,$

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")
```

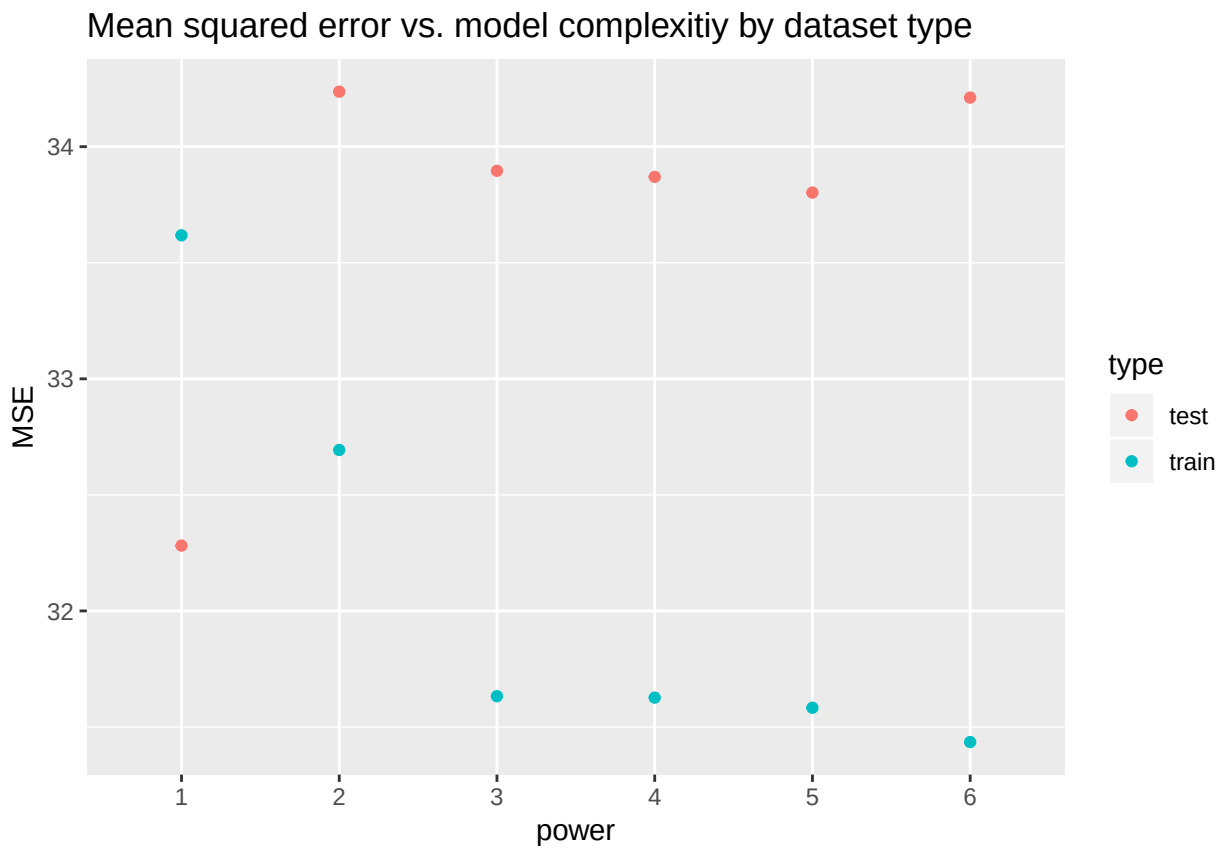


Analysis: In this case we are given that mositure is normally distributed, thus the loss function to minimize had to be (actual-predicted)^2, if we were to minimize the absoulte value, then this would assume a Laplace distribution.

3.3 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)
```

```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. 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)
```

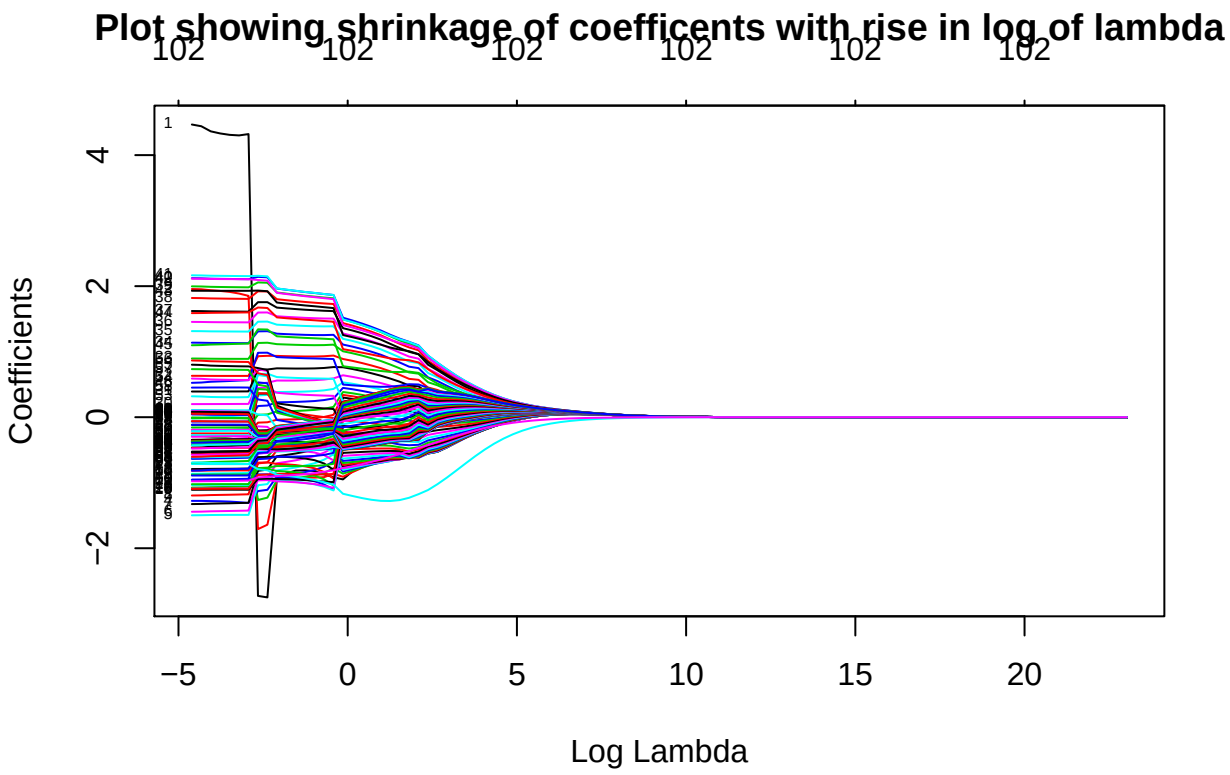
```
## MODEL INFO:
## Observations: 215
## Dependent Variable: Fat
## Type: OLS linear regression
##
## MODEL FIT:
```

```
## 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.

3.5 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

Channel1	Channel2	Channel84	Channel62	Channel53	Channel75	Channel57	Protein	lambda
-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

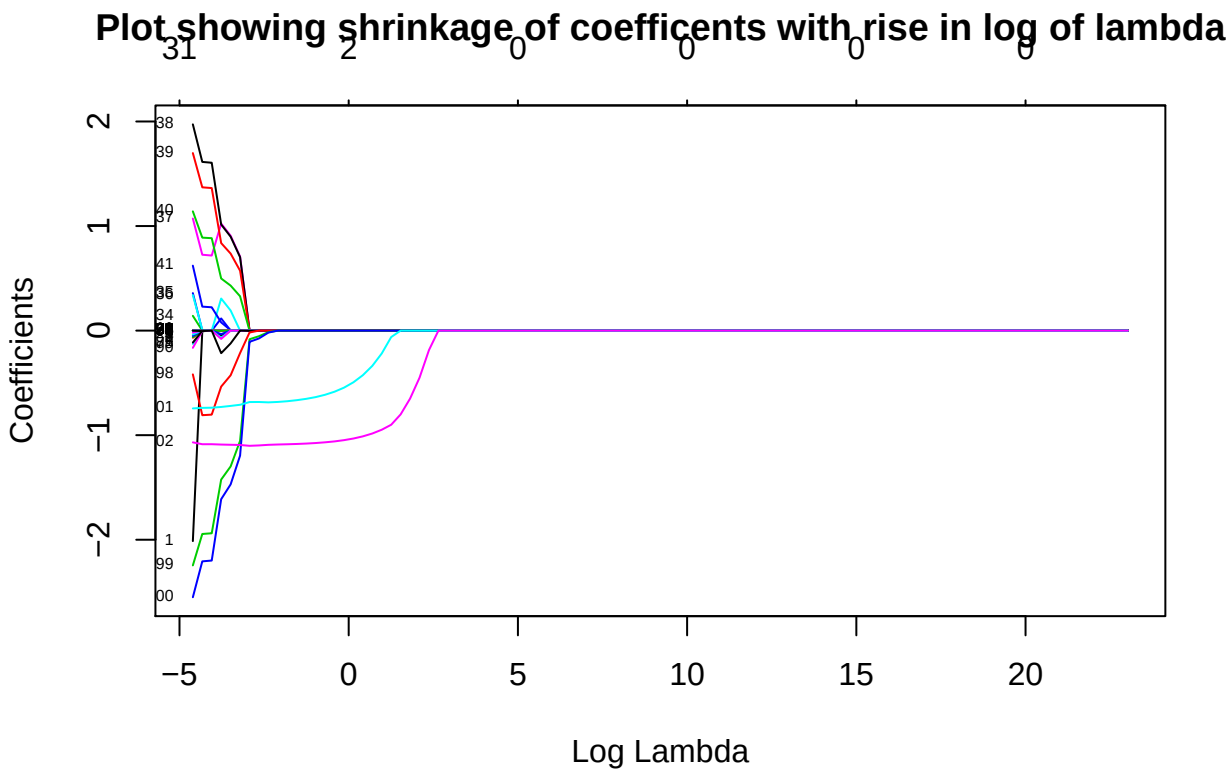
Analysis: The idea of lasso and ridge regression is to introduce bias to variables in order to reduce/account for multicollinearity. Introducing bias (lambda) to covarience matrix is done by multiplying the diagonal elements by lambda(often 1+lambda), this inflates the covarience of predictors compared 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.

3.6 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.

3.7 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)
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) 98.4429946
## 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    .
## Channel38    .
## Channel39    .
## Channel40    .
## Channel41    .
## Channel42    .
## Channel43    .
## Channel44    .
## Channel45    .
## Channel46    .
## Channel47    .
## Channel48    .
## Channel49    .
## Channel50    .
## Channel51    .
## Channel52    .
## Channel53    .
## Channel54    .
## Channel55    .
## Channel56    .
## Channel57    .
## Channel58    .
## Channel59    .
## Channel60    .
```

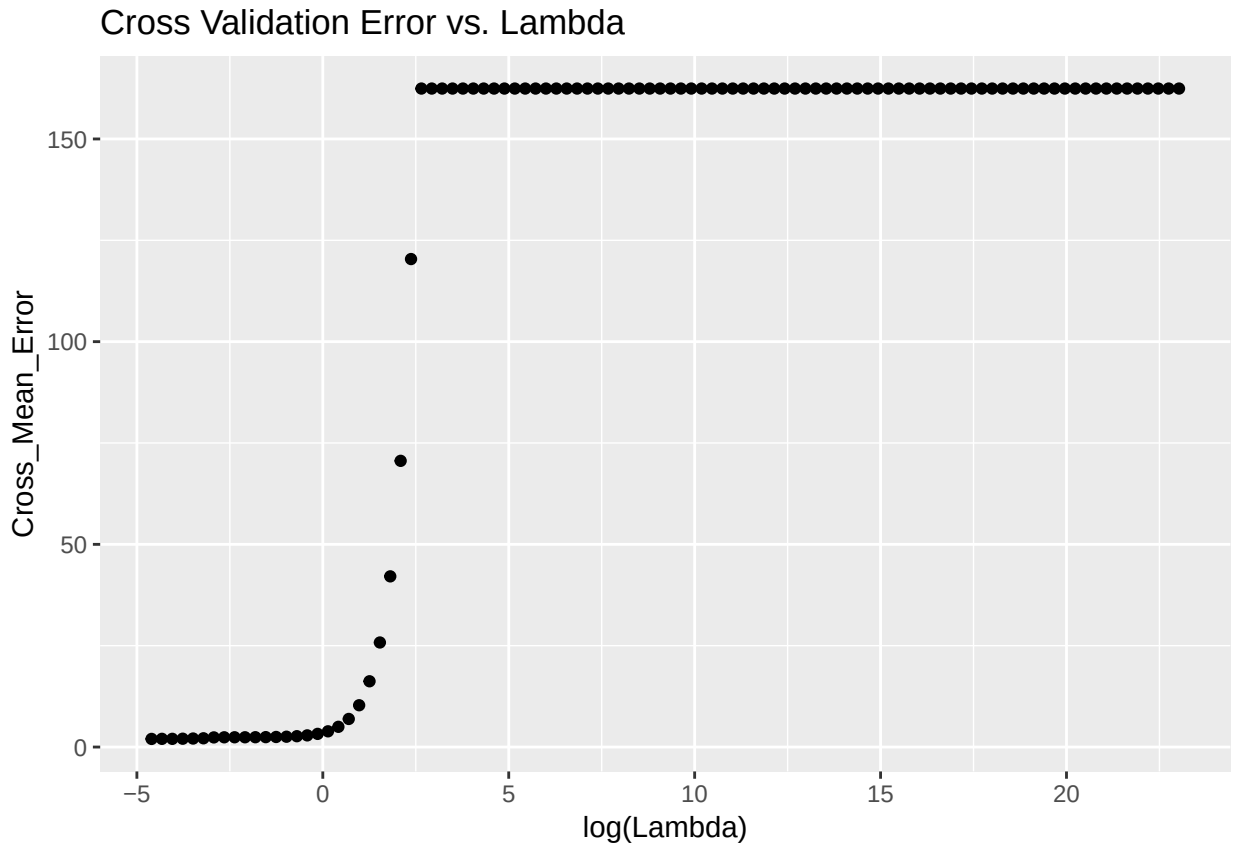
```
## Channel61    .
## Channel62    .
## Channel63    .
## Channel64    .
## Channel65    .
## Channel66    .
## Channel67    .
## Channel68    .
## Channel69    .
## Channel70    .
## Channel71    .
## Channel72    .
## Channel73    .
## Channel74    .
## Channel75    .
## Channel76    .
## Channel77    .
## Channel78    .
## Channel79    .
## Channel80    .
## Channel81    .
## Channel82    .
## Channel83    .
## Channel84    .
## Channel85    .
## Channel86    .
## Channel87    .
## Channel88    .
## Channel89    .
## Channel90    .
## Channel91    .
## Channel92    .
## Channel93    .
## Channel94    .
## Channel95    .
## Channel96    .
## Channel97    .
## Channel98    .
## Channel99    .
## Channel100   .
## Protein     -0.6655559
## Moisture    -1.0842878
```

```
lasso_cv$lambda.min
```

```
## [1] 0.01
```

```
## 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 0.1, implies almost 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 0.1 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.

3.8 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.

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")
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)
summary(best_model)
# 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)
```

```r
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 ~., 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)
```

```r
names(dimnames(conf_test2)) <- c("Actual Test", "Predicted Test")
confusionMatrix(conf_test2)

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

# X = swiss[,2:6]
# Y = swiss[,1:1]
# 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))

result_test <- result %>% filter(type == "test")

variable_performance <- result_test %>% 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(test) vs. Number of variables")

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

best_variables <- result_test %>% group_by(variables) %>%
  summarise(MSE = mean(mse, na.rm = TRUE)) %>% arrange(MSE) %>%
  select(variables) %>% slice(1) %>% as.vector()

return(list(myplot, myplot2, best_variables))
}
subset_function(X = swiss[,2:6], Y = swiss[,1:1], N = 5)

# Using pairs on swiss dataset
GGally::ggpairs(swiss)
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
```

```r
# 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)
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
```

```r
## 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")
```