Group_A11

Sridhar

26 November 2018

Contents

Assignment 1

1. Dividing the dataset into training and test

```
##  [1] "Word1"  "Word2"  "Word3"  "Word4"  "Word5"  "Word6"  "Word7"
##  [8] "Word8"  "Word9"  "Word10" "Word11" "Word12" "Word13" "Word14"
## [15] "Word15" "Word16" "Word17" "Word18" "Word19" "Word20" "Word21"
## [22] "Word22" "Word23" "Word24" "Word25" "Word26" "Word27" "Word28"
## [29] "Word29" "Word30" "Word31" "Word32" "Word33" "Word34" "Word35"
## [36] "Word36" "Word37" "Word38" "Word39" "Word40" "Word41" "Word42"
## [43] "Word43" "Word44" "Word45" "Word46" "Word47" "Word48" "Spam"
```

Read the data file spambase.xlsx into spam_data. I am then spliting the columns of spam_data into X(features) and Y(Labels). I am then dividing the data into train and test as 50% data into train and rest 50% test. The seed is set to 12345 so that we can get the same split every time we execute this block of code.

2. Logistic Regression with boundary at 0.5

```
## [1] "Train Results -"

## [1] "Train Misclassification Rate :  0.159124087591241"

##          not_spam spam
## not_spam      811  133
## spam           85  341

## [1] "Test Results -"

## [1] "Test Misclassification Rate :  0.186131386861314"

##          not_spam spam
## not_spam      789  149
## spam          106  326
```

In this task we had to classify an email as spam or not with probability (0.5). There is not much difference in the accuracy of the model on train and test datasets which shows that the model is not overfitting the training set. There are lot of emails that are not spam and are being classified as spam, which is bad. This can be bacause of the low probability we are using to classify an email as spam(0.5). The overall accuracy of the system is good, around 82%, and the precision of the model is (0.85). This is bad as with a probability of 0.15 a not spam email will be marked as spam. The recall of the model is (0.84). This preformance is consistent amoung the test and train datasets which were created using a random split to the original dataset.

3. Logistic Regression with boundary at 0.5

```
## [1] "Train Results -"

## [1] "Train Misclassification Rate :  0.283941605839416"

##          not_spam spam
## not_spam      944    0
## spam          389   37

## [1] "Test Results -"

## [1] "Test Misclassification Rate :  0.291240875912409"
##          not_spam spam
## not_spam      932    6
## spam          393   39
```

On increasing the probability of predicting an email as spam from 0.5 to 0.9 the number of not-spam emails being classified as spam is decreased subtancially, this increases the precision of the model greatly but decreases the accuracy and recall of the model. The precision of the model on train data was (1) and on the test data was (0.99). The number of emails being classified as spam is decreased. This decreases the recall of the model to 0.71 for train and 0.70 for test data. We can get a value for the probability of the classifier by trading off between precision and recall. If we want a system with high precision, the racall of the system goes down.

4. Weighted k-Nearest Neighbour with K=30

```
## [1] "Train Results -"
```

```
## [1] "Train Misclassification Rate :  0.173722627737226"
```

```
##          not_spam spam
## not_spam      809  135
## spam          103  323
```

```
## [1] "Test Results -"
```

```
## [1] "Test Misclassification Rate :  0.31970802919708"
```

```
##          not_spam spam
## not_spam      693  245
## spam          193  239
```

Using a K-Nearest neighbours model to classify emails with K=30 gives us these results. It performs good on the train data but there is a marginal difference in its perfoemance on test data, which shows clearly that the model is overfitting the training data. The train accuracy was 82% and the test accuracy was 68%. This is expected of the KNN algorithm as it stores all the training data and uses those to classify the new data, this is the reason it perfoems better on the train data. Using the train data as a measure of goodness for this algorithm would not be a fair measure for this reason. The performance of this model on the train data is similar to the linear regression model, but it is not as consistent and does not generalize as good as that model. The performance of the KNN model on the test data decreases to 68%.

Given the uneven distribution of the classes spam and not spam (70% of the data is not spam and 30% of it is spam), the performance of the KNN algorithm is bad. A classifier predicting all zeros would have had an accuracy of 70%. We could try changing the values of K and find a value that works well for this classifier.

5. Weighted k-Nearest Neighbour with K=1

```
## [1] "Train Results -"
```

```
## [1] "Train Misclassification Rate :  0"
```

```
##          not_spam spam
## not_spam      944    0
## spam            0  426
```

```
## [1] "Test Results -"
```

```
## [1] "Test Misclassification Rate :  0.360583941605839"
##          not_spam spam
## not_spam      649  289
## spam          205  227
```

Using a K-Nearest neighbours model to classify emails with K=1 gives us these results. The accuracy for this model on the train data is 100% which is because it has all the correct classes for the points it is predicting. It performs vary bad on the test data with accuracy of 63%. The precision for this system was 1 for train and 0.69 for the test data. Using K=1 we assign a new point the class of the closest point to it. This is not an accurate assumption, a larger sample would give a more accurate result. As we saw for the case of K=30, that model generalized better than the model with K=1. As in probability, learger the sample, better is it an estimate of the population. Same is the situation with this, but too large a value of K will also not give a good model. We can find a good model iteratively, trying different values of K.

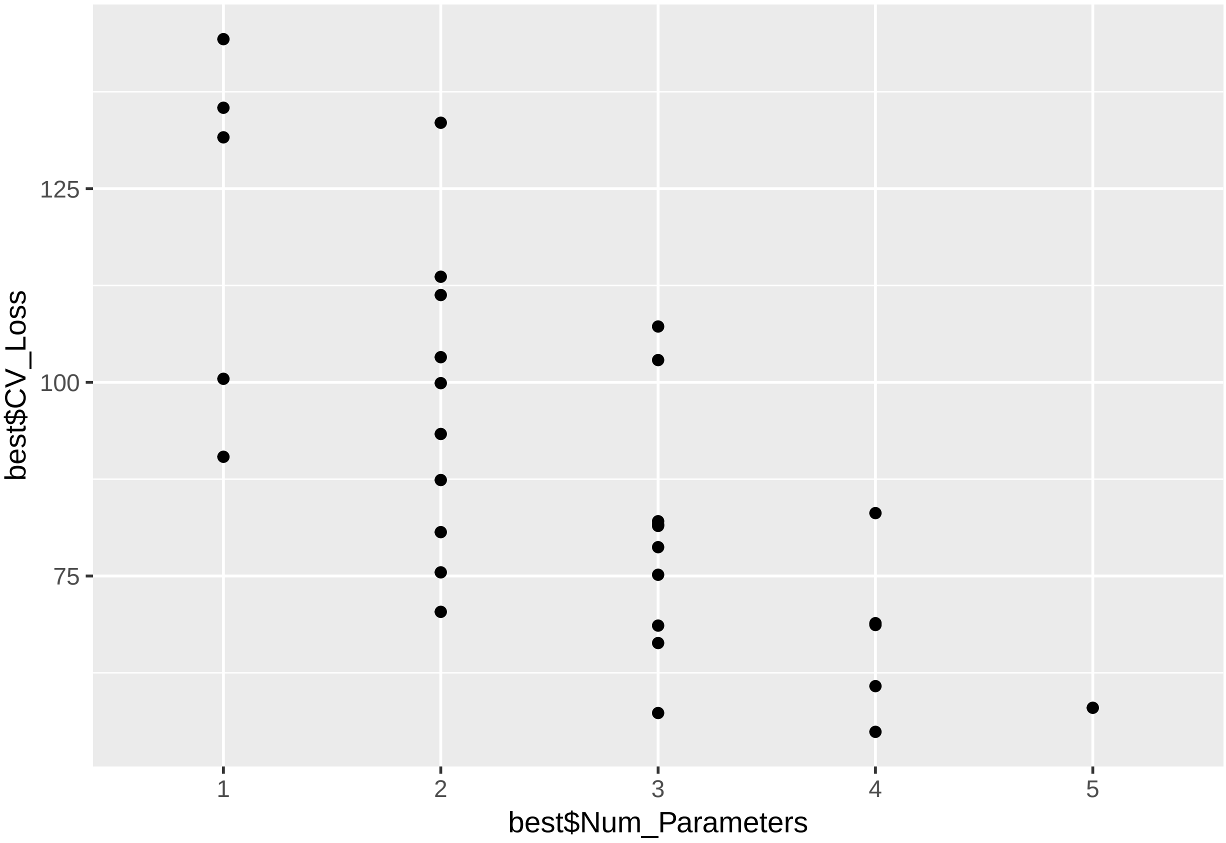
Assignment 3

1. Create function for best Subset Selection using K fold Cross Validation

In this question we made a function that takes in X(features) and Y(True Values) and finds the best subset of features using K-fold cross validation. It uses the helper functions my_lm and my_predict to calculate the weights and predict the outputs or calculate the loss on the new data using the weights calculated.

2. Testing the function on swiss dataset

```
##    Sequence  CV_Loss Num_Parameters
## 29  1,3,4,5 54.88725              4
```



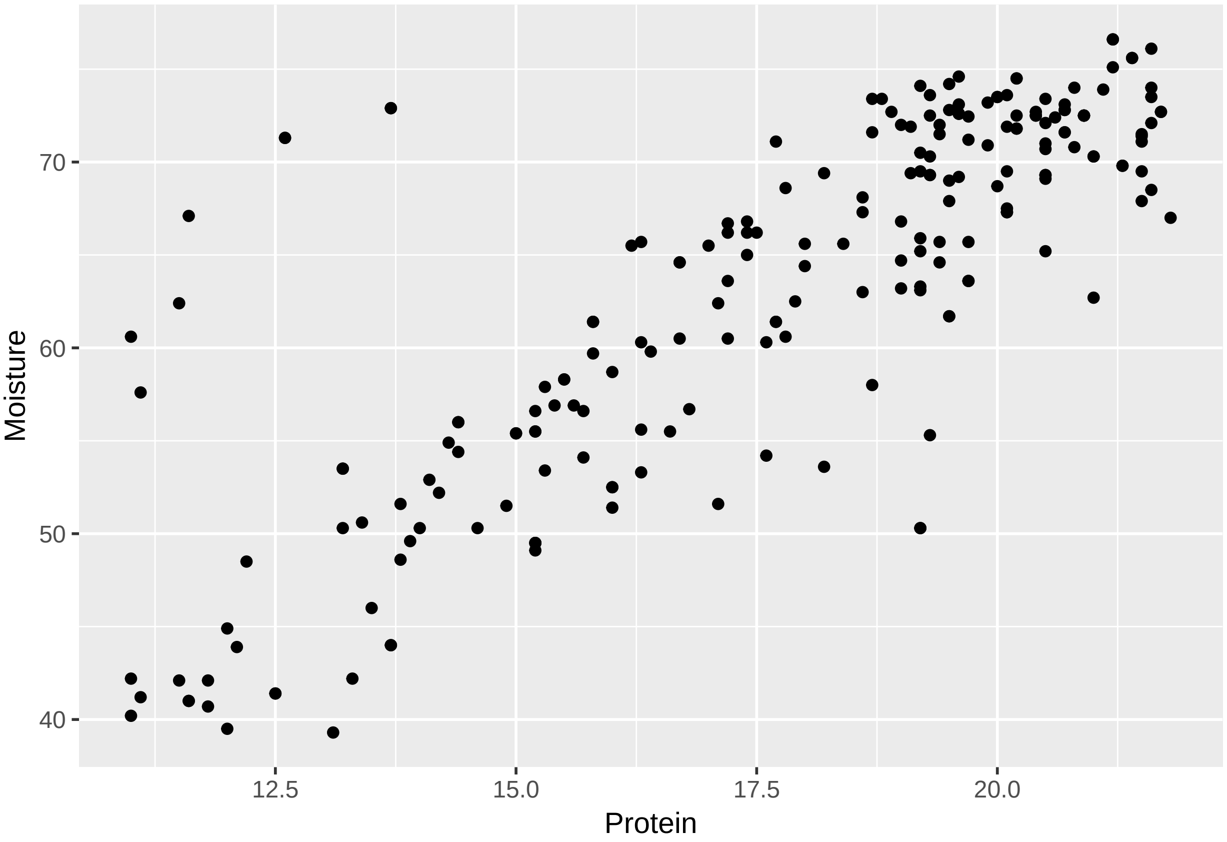
As we increase the complexcity of the model by introducting more power terms of a feature the loss goes down. The variance of the loss also decreases between the folds.The optimal subset are features 1,3,4 and 5 (Agriculture, Education, Catholic, Infant.Mortality) as they evaluate to lowest MSE of 54.88 than any other feature subset. Therefore, these features have major impact on the target. The feature left out was Examination, and I dont think highest marks on army examination will have a great impact on Fertility, so it was a reasonable thing to leave out that feature. Including that feature it did not make much of a difference to the MSE score, it just increased the MSE score by a small value(from 54.88 to 57).

On looking into the model further, and examining the dataset I found out that Agriculture and Education had a negative impact on fertility. They were assigned negative weights. This is correct as the person gets more educated they think of controlling the population and Agriculture has a negative impact as I assume farmers are not too rich to take care of many kids.

Catholic and Infant Mortality had positive impact on the Fertility as they were assigned positive weights. This is correct as the time this data was collected due to lack of good healthcare few infants never actually made it past 2 years, so to increase the chances of having a healthy child, may be this had a positive impact on the Fertility.

Assignment 4

1 Plotting the data between Moisture and Protein



The plot between moisture and protein is linear as evident from the graph barring some points which has large value of moisture for less value of protein which can be considered as outlier. There are more data points towards the higher moisture and higher protein end, but they are all clustered close together, so a linear fit would be a reasonable thing to do in this scenario.

2. Probabilistic Model

To fit the data using polynomial function we use below model:

y(x,w)=w_0 + w_1x +..+$w_M x^M$ = $\sum_{j=0}^{M}$ w_j x^j

In our example M is the order of the polynomial. Comparing to the model we want we can write the above equation as:

(P,w)= $\sum_{n=0}^{M}$ $w_j P^j$

Here P denotes the Protein which is and explanatory variable and w being the coeffecients. Let's assume that our independent observation are drawn independently from a Gaussian distribution. Because we need to find the response variable based on coeffecient(which is our parameter) and variables we propose as probabilistic model as:

P(Mo|(x,w)) = N(Mo|y(Protein,w))

Here Mo is the target variable which is Moisture as from our data set.

On using the training data (Protein,Moisture), we determine the values of the unknown parameters w by maximum likelyhood. If data are assumed to be iid from the distribution then the likelyhood function is given by:

P(Moisture|Protein,w) = $\prod_{n=1}^{N}$ N(Mo_n|y(P_n,w))

On maximizing the the likelyhood function with respect to **w** we obtain the below equation which is basically minimizing the mean sum of square error function:

E(w)=(1/N)($\sum_{n=1}^{N}${y(P_n,w)-$t_n^{2})

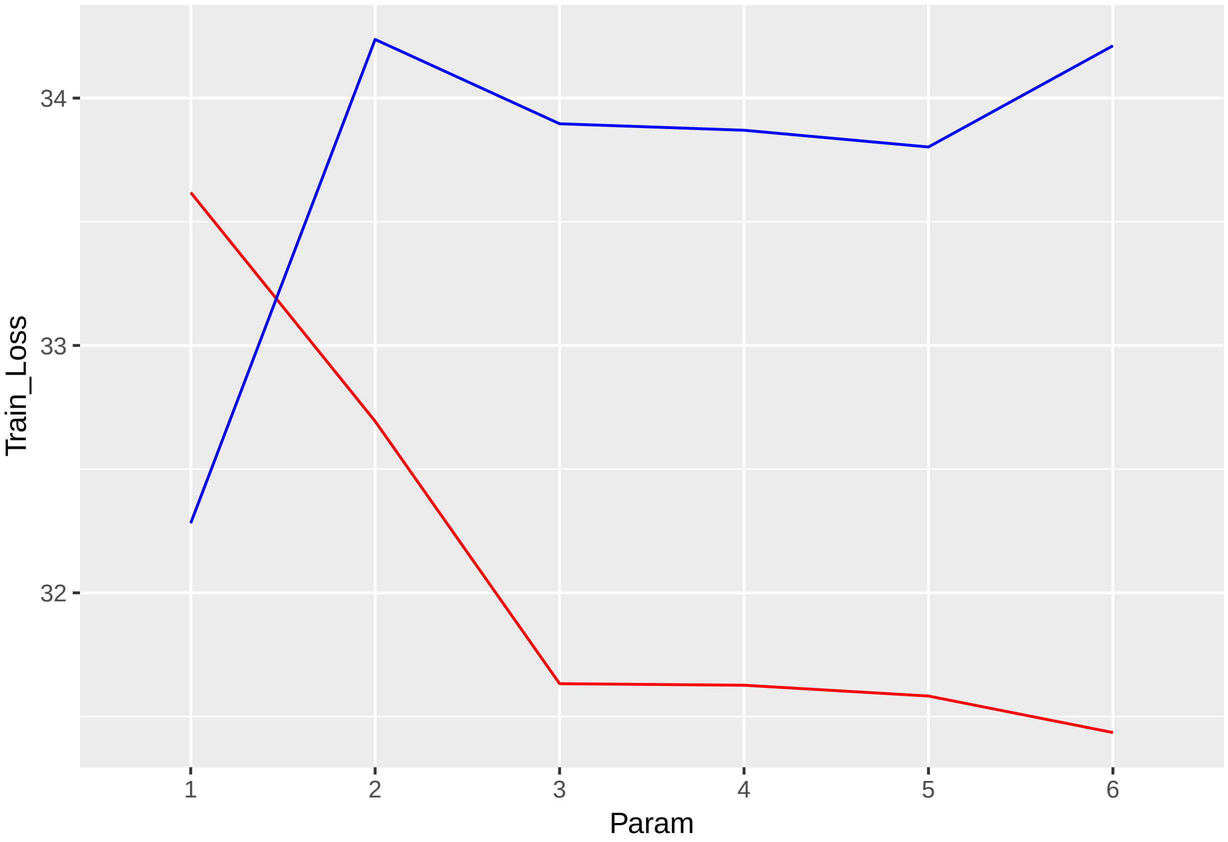
After determining the values of w, we can predict on the new values of protein to get the response variable

P(Mo|P,w_{ML})= N(Mo|y(Mo|P,w_{ML}))

MSE as a loss function maskes sense because according to probabilistic model, in order find the distribution of the data based on parameter we maximize the log likelyhood with respect to the parameter which leads us to the minimizing MSE equation. Which means, by minimizing the MSE, we can find the true distribution of data.

3. Polynomial Model

```
##   Train_Loss Test_Loss Param
## 1   33.61836  32.28154     1
## 2   32.69342  34.23708     2
## 3   31.63266  33.89615     3
## 4   31.62641  33.86992     4
## 5   31.58273  33.80234     5
## 6   31.43513  34.21152     6
```



As the model becomes more complex in terms of polynomial, the train error decreases as it predicts the data better but it performs poorly on the validation data set. As the model becomes more complex, its variance on training dataset increases while the bias decreases. On the other hand when the model predicts on test dataset, variances decreases and biases increases with model complexity. This is because more complex model overfit the training data which eventually perform poorly on test data set.

Because this dataset is very small, that is why we see a strange case of test loss lower than train loss at parameters=1. I think the model is just getting luckey in the first case, because as soon as we increase the model complexcity to 2(parameters), there is a sudden jump in test loss and sudden drop in test loss. This is the reason I am not selecting a model with single parameter.

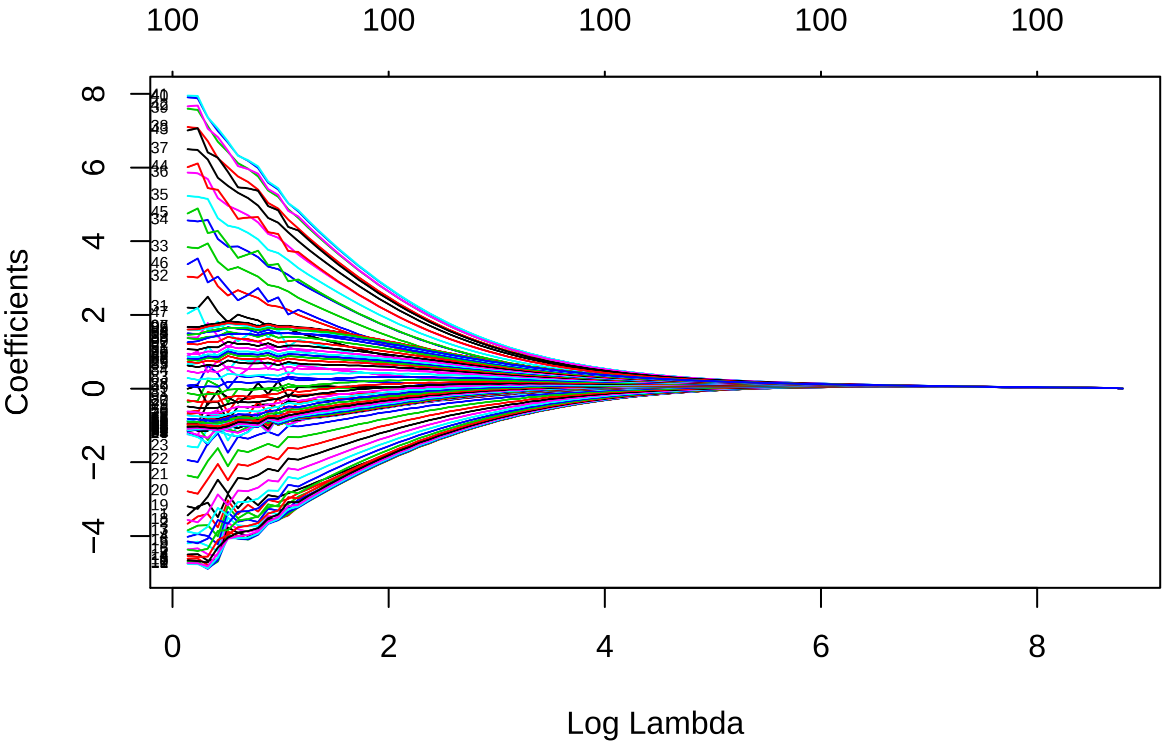
According to the plot a safe model would be with polynomial degree 5, because until then both the test and train loss show some decrease in loss, but there is a sudden increase in test loss after 5 as we can see in the plot. Both test and train start deviating after that. So I think a model with fifth degree polynomial would be a good fit to the data.

4. Step AIC

Among the 100 variables, the Step AIC function chooses 63 variable in a model which fits the data set better. Below are the variables chosen by stepAIC function.

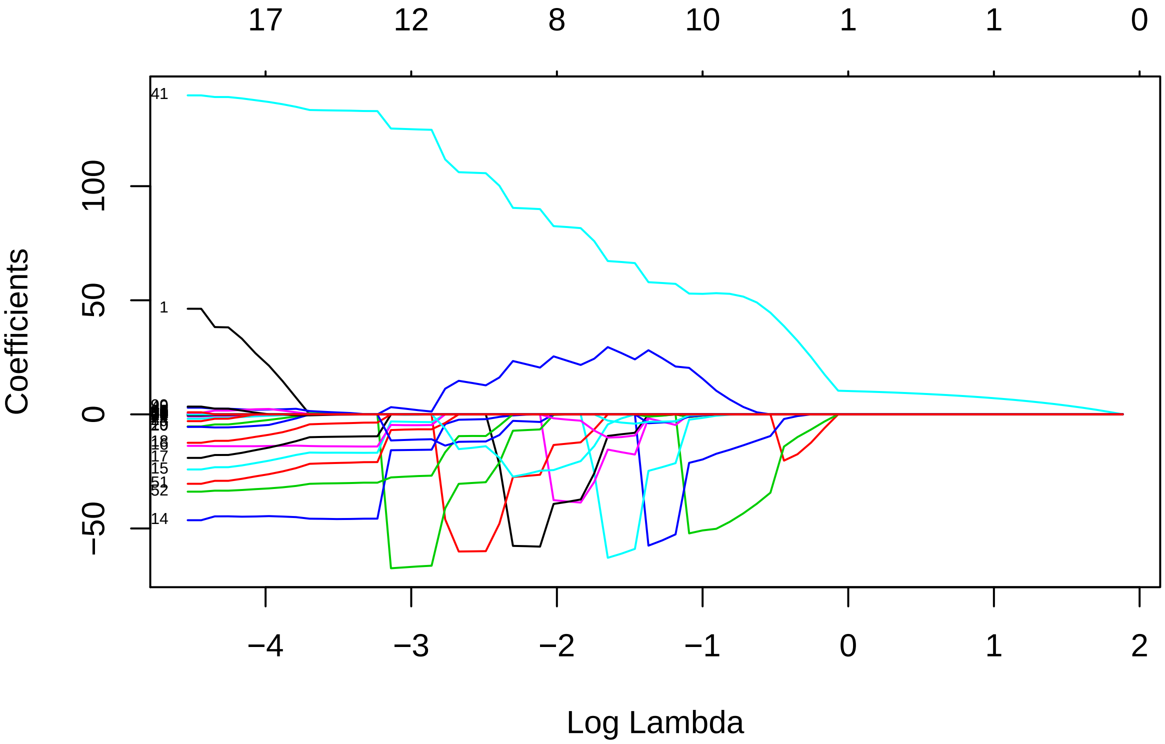
```
##  [1] "Channel1"  "Channel2"  "Channel4"  "Channel5"  "Channel7" 
##  [6] "Channel8"  "Channel11" "Channel12" "Channel13" "Channel14"
## [11] "Channel15" "Channel17" "Channel19" "Channel20" "Channel22"
## [16] "Channel24" "Channel25" "Channel26" "Channel28" "Channel29"
## [21] "Channel30" "Channel32" "Channel34" "Channel36" "Channel37"
## [26] "Channel39" "Channel40" "Channel41" "Channel42" "Channel45"
## [31] "Channel46" "Channel47" "Channel48" "Channel50" "Channel51"
## [36] "Channel52" "Channel54" "Channel55" "Channel56" "Channel59"
## [41] "Channel60" "Channel61" "Channel63" "Channel64" "Channel65"
## [46] "Channel67" "Channel68" "Channel69" "Channel71" "Channel73"
## [51] "Channel74" "Channel78" "Channel79" "Channel80" "Channel81"
## [56] "Channel84" "Channel85" "Channel87" "Channel88" "Channel92"
## [61] "Channel94" "Channel98" "Channel99"
```

5. Ridge Regression



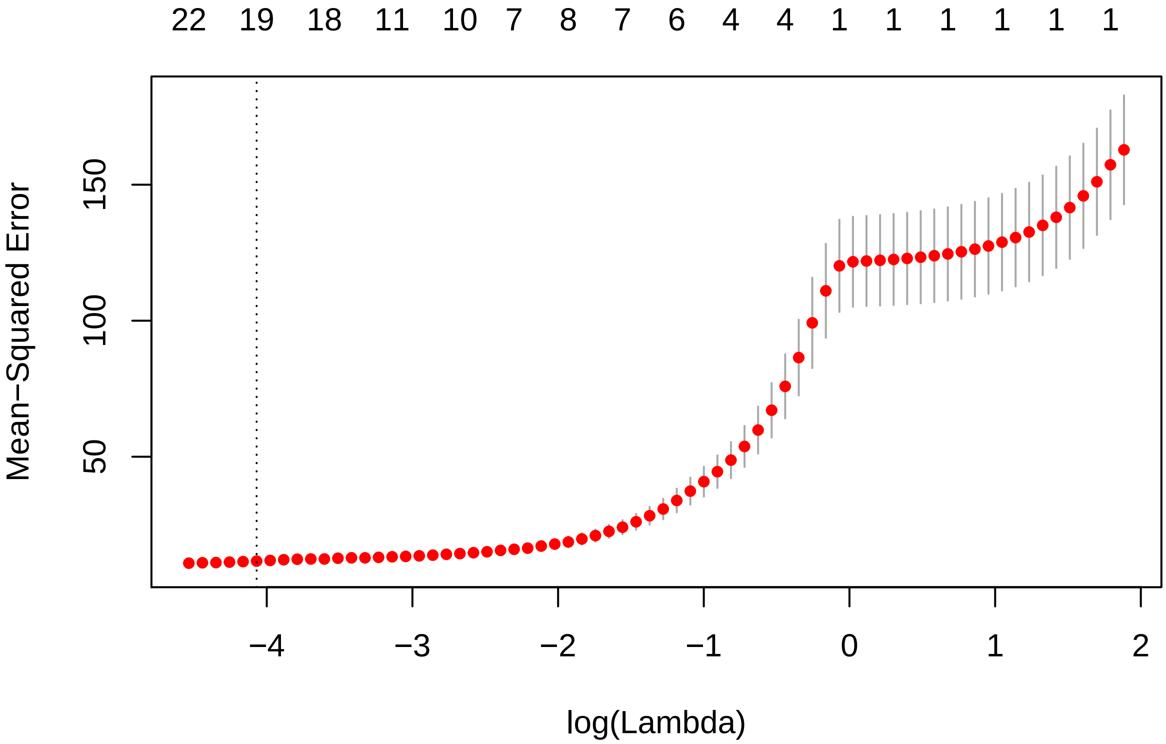
As the value of Λ increases, it keeps adding more penalty to the coeffecients which decreases the complexity of the model. So with the increase in lambda value the variance of the model decreases while the bias increases. In case of ridge regression as we can see from the plot all the coefficients are drawn close to zero, and are not equal to zero. As we increase the value of lambda the coefficients are drawn close to zero, but all of them still contribute in the model prediction.

6. Lasso Regression



Lasso regression performs both variable selection and shrinkage in order for the model to be less complex and thereby perform better. As the value of lambda increases it forces some of the coeffecient to zero hence removing the variables from the model and at the same time decreases the value of coeffecient to zero. Ridge regression performs parameter shrinkage while lasso does variable selection as well. Comparing the plot of lasso and ridge, the lasso makes some of the coeffecient to the zero suddenly whereas in the ridge the variables are shrinked slowly towards zero. With Lasso, it is easier to eliminate the variables which does not contribute to the output

7. Lasso Regression with CV



```
## [1] "Best lambda: 0.0170847299377344"
```

Lasso regression is performed with CV with 10 folds. Above plot shows the relationship between log λ with MSE. As the value of lambda increases the more variables are shrinked and selected. The number of variables selected by lasso regression is shown in the top x-axis. The value of lambda where the MSE is minimum is **0.01708473**. Minimum MSE is obtained by including 19 variables. Plot includes standard deviation around MSE for each lambda. The plot also shows that MSE error keep increasing with increase in λ but ineffect the lasso regression with CV eliminates high variance by including bias so the model does not overfit. But as the λ keeps increasing most of the important variables are eliminated so the model will not be useful in predicting data if variables does not exist. So very high λ value is not desirable.

8. Comparison of StepAIC vs Lasso vs Ridge vs CV Lasso

- StepAIC assume that there is no correlation between the selected variables and keeps selecting the variable one at a time only if it improves the model. Lasso and ridge tries to decreases the overfit by including the penalty to the coeffecient. Ridge regression performs parameter shrinkage but Lasso does variable selection as well. Lasso with CV finds the best lambda parameter by cross validation and returns the number of variables required by the model so that it won't overfit.
- StepAIC selected 63 variables. Lasso regression gave the list of lambda value and using this input in CV Lasso with 10 fold 19 variables are selected with the lambda value of 0.01708 where error was 1se from the minimum error.

Appendix

```
knitr::opts_chunk$set(
    echo = FALSE,
    message = FALSE,
    warning = FALSE
)
library(readxl)
library(ggplot2)
library(kknn)
library(MASS)
library(glmnet)
spam_data = read_xlsx("spambase.xlsx", sheet = "spambase_data")
colnames(spam_data)
X = spam_data[, 1:(ncol(spam_data)-1)]
Y = spam_data[, ncol(spam_data)]

## 50% of the sample size
smp_size <- floor(0.50 * nrow(spam_data))

## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(spam_data)), size = smp_size)

train = spam_data[train_ind, ]
X_train = X[train_ind,]
X_test = X[-train_ind,]
Y_train = Y[train_ind,]
Y_test = Y[-train_ind,]
confMat = function(pred, actual){
  pos = which(actual == 1)
  tn = sum(pred[pos])
  fn = length(pos) - tn
  neg = which(actual == 0)
  fp = sum(pred[neg])
  tp = length(neg) - fp
  tbl = data.frame('not_spam'= c(tp, fn), 'spam'=c(fp, tn),
                   row.names = c('not_spam', 'spam'))
  return(tbl)
}

model <- glm(Spam ~.,family=binomial(link='logit'),data=train)
#summary(model)

##Train predict
print("Train Results -")
fitted.results <- predict(model,newdata=X_train,type='response')
fitted.results <- ifelse(fitted.results > 0.5,1,0)
misClasificError <- mean(fitted.results != Y_train)
print(paste('Train Misclassification Rate : ',misClasificError))
tbl1_train = confMat(fitted.results, Y_train)
print(tbl1_train)

##Test predict
print("Test Results -")
```

```r
fitted.results <- predict(model,newdata=X_test,type='response')
fitted.results <- ifelse(fitted.results > 0.5,1,0)
misClasificError <- mean(fitted.results != Y_test)
print(paste('Test Misclassification Rate : ',misClasificError))
tbl1_test = confMat(fitted.results, Y_test)
print(tbl1_test)
##Train predict
print("Train Results -")
fitted.results <- predict(model,newdata=X_train,type='response')
fitted.results <- ifelse(fitted.results > 0.9,1,0)
misClasificError <- mean(fitted.results != Y_train)
print(paste('Train Misclassification Rate : ',misClasificError))
tbl2_train = confMat(fitted.results, Y_train)
print(tbl2_train)

##Test predict
print("Test Results -")
fitted.results <- predict(model,newdata=X_test,type='response')
fitted.results <- ifelse(fitted.results > 0.9,1,0)
misClasificError <- mean(fitted.results != Y_test)
print(paste('Test Misclassification Rate : ',misClasificError))
tbl2_test = confMat(fitted.results, Y_test)
print(tbl2_test)
##Train Results
print("Train Results -")
pred_kknn = kknn(Spam~., train, X_train, k= 30, kernel = "optimal")
fitted.results = fitted(pred_kknn)
fitted.results <- ifelse(fitted.results > 0.5,1,0)
misClasificError <- mean(fitted.results != Y_train)
print(paste('Train Misclassification Rate : ',misClasificError))
tbl3_train = confMat(fitted.results, Y_train)
print(tbl3_train)

##Test Results
print("Test Results -")
pred_kknn = kknn(Spam~., train, X_test, k= 30, kernel = "optimal")
fitted.results = fitted(pred_kknn)
fitted.results <- ifelse(fitted.results > 0.5,1,0)
misClasificError <- mean(fitted.results != Y_test)
print(paste('Test Misclassification Rate : ',misClasificError))
tbl3_test = confMat(fitted.results, Y_test)
print(tbl3_test)
print("Train Results -")
pred_kknn = kknn(Spam~., train, X_train, k= 1, kernel = "optimal")
fitted.results = fitted(pred_kknn)
fitted.results <- ifelse(fitted.results > 0.5,1,0)
misClasificError <- mean(fitted.results != Y_train)
print(paste('Train Misclassification Rate : ',misClasificError))
tbl4_train = confMat(fitted.results, Y_train)
print(tbl4_train)

##Test Results
print("Test Results -")
```

```
pred_kknn = kknn(Spam~., train, X_test, k= 1, kernel = "optimal")
fitted.results = fitted(pred_kknn)
fitted.results <- ifelse(fitted.results > 0.5,1,0)
misClasificError <- mean(fitted.results != Y_test)
print(paste('Test Misclassification Rate : ',misClasificError))
tbl4_test = confMat(fitted.results, Y_test)
print(tbl4_test)
#Assignment 3
my_lm = function(X, Y){#, X_cv, Y_cv){
  X = as.matrix(X)
  Y = as.matrix(Y)
  W = ginv(t(X)%*%X)%*%t(X)%*%Y
  return(W)
}

my_predict = function(W, X_test, Y_test=NULL){
  if(is.null(Y_test)){
    X_test = as.matrix(X_test)
    pred_Y = X_test%*%W
    return(pred_Y)
  }else{
    X_test = as.matrix(X_test)
    Y_test = as.matrix(Y_test)
    pred_Y = X_test%*%W
    sdif = (Y_test - pred_Y)^2
    loss = sum(sdif)/nrow(Y_test)
    return(loss)
  }
}

bsSel = function(X, Y, folds){
  set.seed(12345)
  seq_costs = matrix(0, nrow = 0, ncol = 3)
  nc = ncol(X)
  #Choosing best subset
  for(k in 1:nc){
    combs = combn(1:nc, k)
    for(j in 1:ncol(combs)){
      c = combs[, j]
      X_sub <- X[, c]
      X_sub <- cbind(X_sub, 1)

      #calculating the K folds cv loss
      set.seed(12345)
      n = nrow(X)
      fs = as.integer(n/folds)
      rnd_ind <- sample(seq_len(n), size = n)
      cvl = 0
      for(i in 1:folds){
        if(i==folds){
          l = length(rnd_ind)
          cv_ind = rnd_ind[((fs*(i-1))+1):l]
        }
        else{
          cv_ind = rnd_ind[((fs*(i-1))+1):(fs*i)]
        }
        X_train = X_sub[-cv_ind,]
        Y_train = Y[-cv_ind]
        X_cv = X_sub[cv_ind,]
        Y_cv = Y[cv_ind]
        W = my_lm(X_train, Y_train)
        cv_loss = my_predict(W, X_cv, Y_cv)
        cvl = cvl + cv_loss
      }
      loss = cvl/folds
      #end of calc

      seq = paste(c, collapse = ",")
      seq_costs = rbind(seq_costs, c(seq, loss, k))
    }
  }
  seq_costs = as.data.frame(seq_costs)
  colnames(seq_costs) = c("Sequence", "CV_Loss", "Num_Parameters")
  seq_costs$CV_Loss = as.numeric(as.character(seq_costs$CV_Loss))
  return(seq_costs)
}
data = swiss
Y = data$Fertility
X = data[,2:ncol(data)]

best = bsSel(X,Y, 5)
best_seq = best[which.min(best$CV_Loss),]
print(best_seq)

ggplot(best, aes(x=best$Num_Parameters, y=best$CV_Loss)) + geom_point()
tecator_data = read_xlsx("tecator.xlsx", sheet = "data")
data = tecator_data[, c("Protein", "Moisture")]
data$Intercept = 1

## 50% of the sample size
smp_size <- floor(0.50 * nrow(data))

## set the seed to make your partition reproducible
set.seed(12345)
train_ind <- sample(seq_len(nrow(data)), size = smp_size)

train <- data[train_ind, ]
X_train = data[train_ind, c("Protein", "Intercept")]
Y_train <- data[train_ind, "Moisture"]
X_test <- data[-train_ind, c("Protein", "Intercept")]
Y_test <- data[-train_ind, "Moisture"]

ggplot(tecator_data, aes(Protein, Moisture)) + geom_point()
train2 = train
X_test2 = X_test
losses = matrix(0, nrow = 0, ncol = 2)
```

```r
for(i in 1:6){
  if(i>1){
    train2[paste('P', i)] = sapply(train2['Protein'], function(x) x^i)
    X_test2[paste('P', i)] = sapply(X_test2['Protein'], function(x) x^i)
  }

  model <- glm(Moisture ~.,family=gaussian(link = "identity"),data=train2)
  summary(model)

  fitted.results <- predict(model,newdata=X_test2,type='response')
  diff = fitted.results - Y_test
  cost_test = sum(diff*diff)/nrow(Y_test)
  #print(paste('CV Cost',cost_test))

  x_tr = train2[ , !(names(train2) %in% "Moisture")]
  y_tr = train2['Moisture']
  fitted.results <- predict(model,newdata=x_tr,type='response')
  diff = fitted.results - y_tr
  cost_train = sum(diff*diff)/nrow(y_tr)
  #print(paste('Train Cost',cost_train))
  #print("")
  losses = rbind(losses, c(cost_train, cost_test))
}



#losses
los = data.frame(losses)
colnames(los)=c("Train_Loss", "Test_Loss")
los$Param = rownames(los)
print(los)

ggplot(los)  + geom_line(aes(Param, Train_Loss, group=1), col='red') + geom_line(aes(Param, Test_Loss, 
data = tecator_data
data$Moisture = NULL
data$Sample = NULL
data$Protein = NULL


tecatordata<-data[,c(1:101)]
lmmodel <- glm(Fat~., data = tecatordata,family=gaussian)
model_AIC<-stepAIC(lmmodel,trace=FALSE,direction = "both")
attr(terms(model_AIC),"term.labels")
Y = as.matrix(data["Fat"])
X = as.matrix(data[,-ncol(data)])

ridgereg<-glmnet(X,Y,alpha=0,family ="gaussian")

plot(ridgereg, "lambda", label=TRUE)

set.seed(12345)
lassoreg<-glmnet(X,Y,alpha=1,family ="gaussian")
```

```r
lam<-lassoreg$lambda
plot(lassoreg, "lambda" ,label=TRUE)

lam<-c(lam,0)
set.seed(12345)
lassreg_cv<-cv.glmnet(X,Y,alpha=1,family="gaussian",lambda = lam)
plot(lassreg_cv)
bestlam = lassreg_cv$lambda.1se

z<-as.matrix(coef(lassreg_cv,s="lambda.1se"))
z<-as.matrix(z[!rowSums(z==0),])

paste("Best lambda:" ,bestlam)
```