Computer Lab 1 Block 1

Zoe Patton, Rojan Karakaya, Suhani Ariga

12/11/2020

Contents

Contributions

For Lab 1, Zoe focused on assignment 1, Rojan focused on assignment 2 and Suhani focused on assignment 3.

Assignment 1. Handwritten digit recognition with K-means

1.

The optdigits data was imported into R and divided into training, validation and test sets using the partitioning ratios 50%, 25% and 25% respectively.

```
digits<-read.csv("optdigits.csv",header=FALSE)
n<-dim(digits)[1]
set.seed(12345)
id<-sample(1:n, floor(n*0.5))
train<-digits[id,1:65]

id1<-setdiff(1:n, id)
set.seed(12345)
id2<-sample(id1, floor(n*0.25))
validate<-digits[id2,1:65]

id3<-setdiff(id1,id2)
test<-digits[id3,1:65]
```

2.

A 30 nearest-neighbor classifier was fit to the training data using function kknn() from the kknn() package. The fitted values from the kknn object were identified and tabulated against the digits in the training data to create a confusion matrix. The resulting matrix represents the frequency of combinations occuring in the training data.The same process was done again with the test data to create its corresponding confusion matrix.

```
library(kknn)
#training data
num.knn1<-kknn(as.factor(V65)~ ., train, train,k=30, kernel="rectangular")
fit1<-fitted(num.knn1)
CM1<-table(train$V65,fit1)

#test data
num.knn2<-kknn(as.factor(V65)~ ., train, test,k=30, kernel="rectangular")
fit2<-fitted(num.knn2)
CM2<-table(test$V65,fit2)
```

As an additional check on the models, the accuracy for each confusion matrix was calculated with the first listed being training data:

```
## [1] 0.9549974
```

```
## [1] 0.9467085
```

Similarly, the to calculate the miss-classification rate, the fitted values of the test and training data were compared to true labelled digits found in the last column of each matrix. The average number of digits that were labeled incorrectly for the training and test data are listed accordingly:

```
## [1] 0.04500262
```

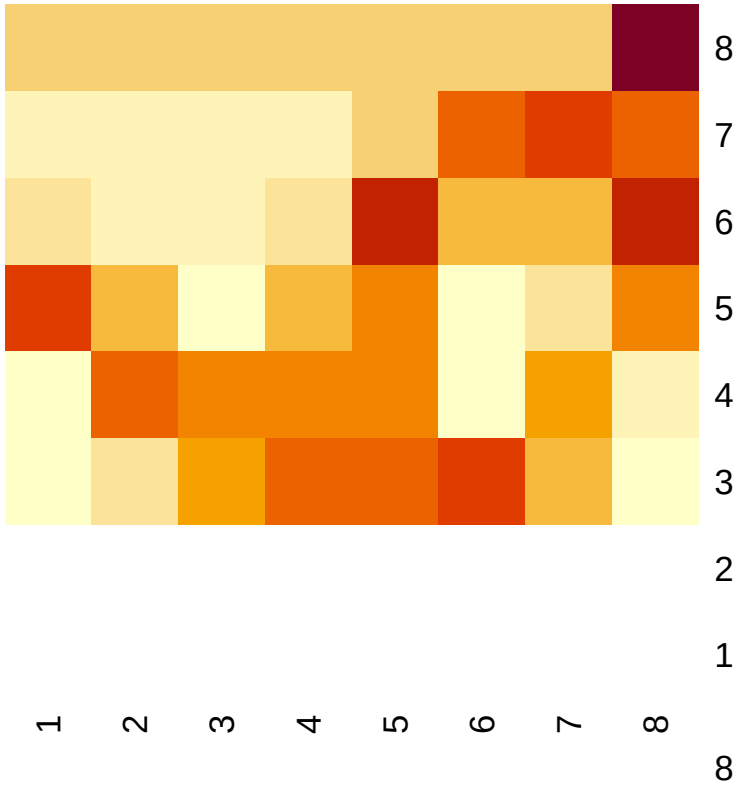
```
## [1] 0.05329154
```

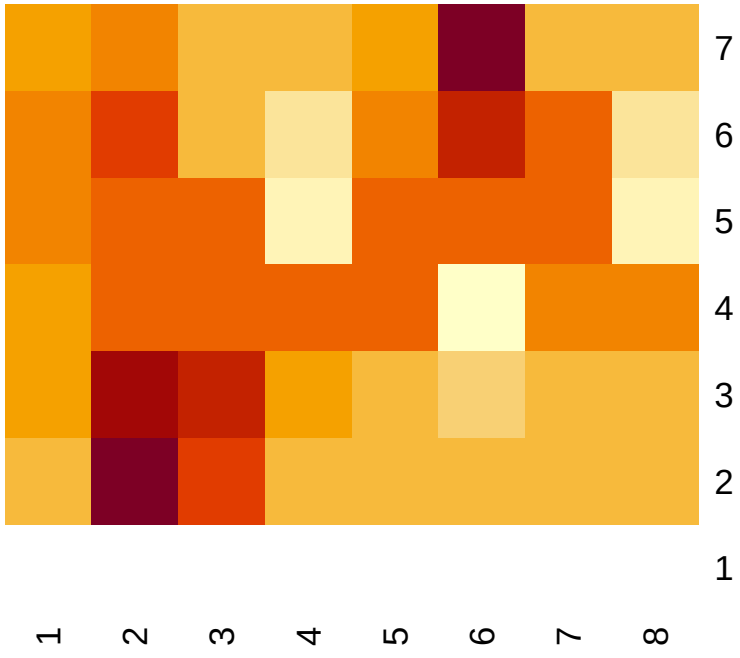
The high accuracies and low miss-classification rates show that the prediction quality is fairly decent for the two models. There is room for improvement since a 5% miss-classification rate can cause significant problems down the line but generally speaking the models predict the digits to a fair degree of certainty.

3. Using the probability matrix generated by the kknn object creating from the training data, two cases of digit "8" that were classified correctly, i.e. they had maximum probabilities, were located.

```
## [1] 129 195
```

The heat maps below shows the reshaped 8x8 matrices of the two easily classified cases, visualizing their corresponding digit "8". In all cases the heat maps are difficult to recognize visually. This may be because the matrices are only 8x8, making the color discrepancies very pronounced.

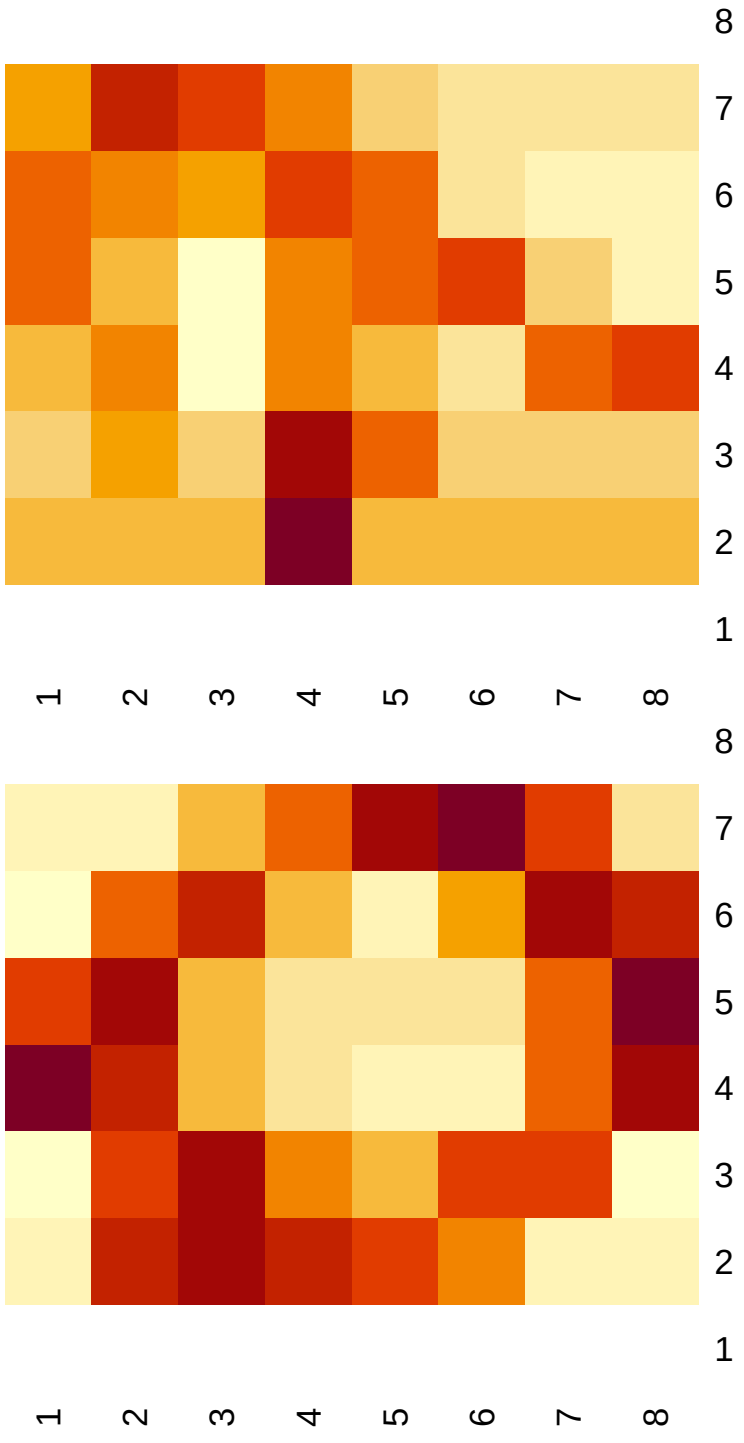


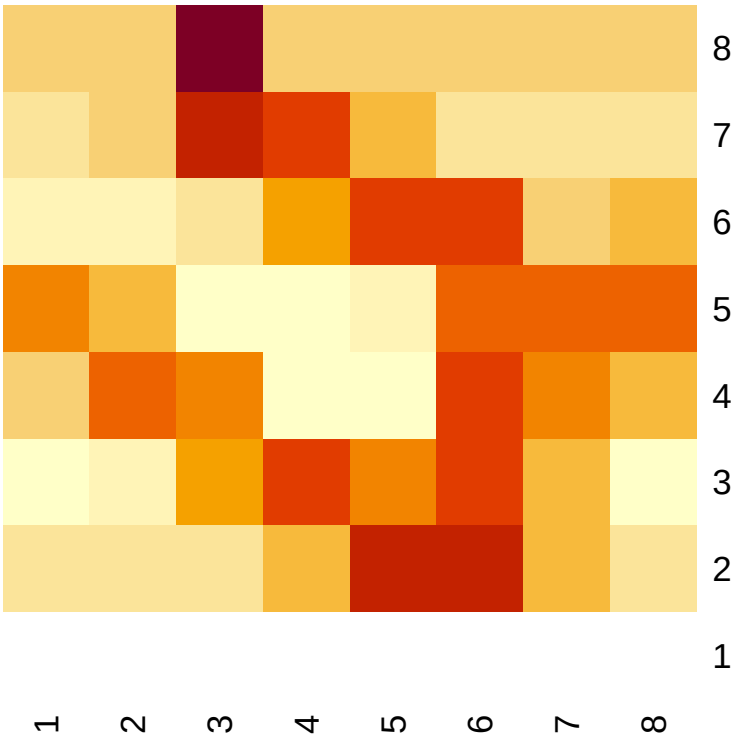


Similarly, three cases which were hard to classify, that had a minimum probability (not equal to zero) were identified:

```
## [1] 13 24 36
```

Their generated heatmaps are shown below:



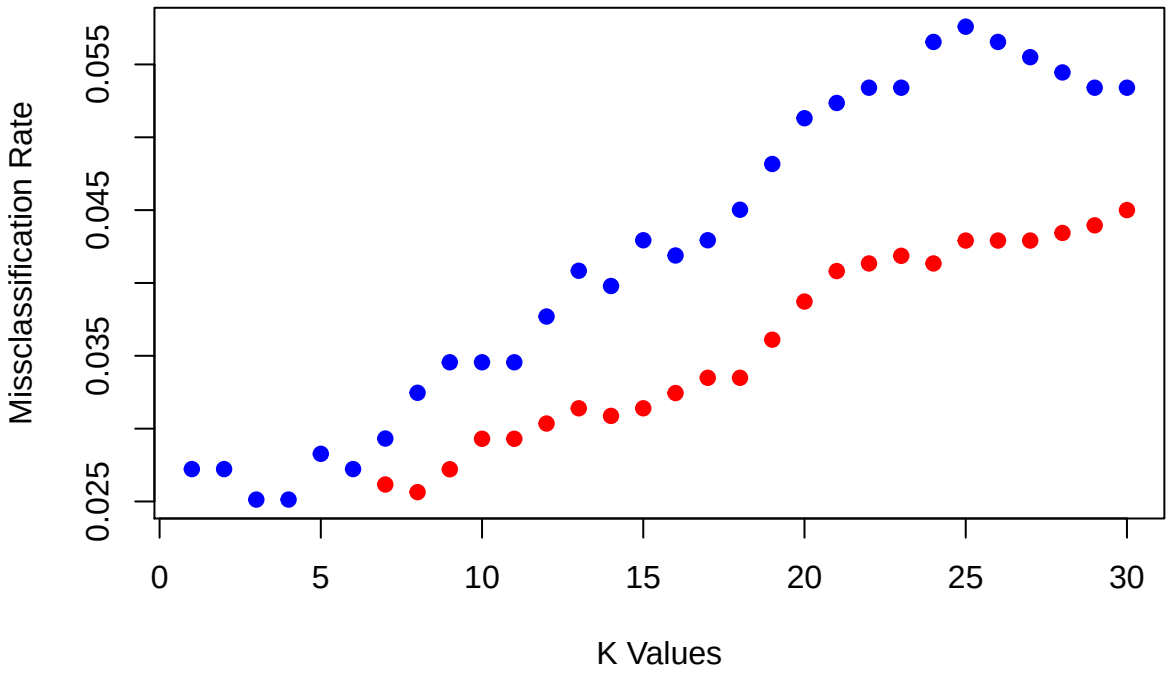


4.

Different values of K from 1 to 30 were used to fit classifiers to the training data and validation data using the functions below:

```
#knn missclassification rates for different values of k for training data
knn_train<-function(i){
  a<-kknn(as.factor(V65)~ ., train, train,k=i, kernel="rectangular")
  misscl<-mean(a$fitted.values != train$V65)
  return(misscl)
}
k_values<-c(1:30)
missclass_train<-unlist(lapply(k_values,knn_train))

#knn missclassification rates for different values of k for validation data
knn_validate<-function(i){
  a<-kknn(as.factor(V65)~ ., train, validate,k=i, kernel="rectangular")
  misscl<-mean(a$fitted.values != validate$V65)
  return(misscl)
}
k_values<-c(1:30)
missclass_validate<-unlist(lapply(k_values,knn_validate))
```

The dependence of the training and validation miss-classification errors were plotted on the value of K. As K increases the model becomes less complex, consequently the miss-classification rate increases. As model complexity decreases, bias is increased and variance is decreased. According to the plot (and using the which function to locate the minimum error) the optimal K value for both training and validation data is the one with the lowest risk, smallest miss-classification rate, which would be K=3.



Using K=3 the test error was calculated to be:

```
## [1] 0.02403344
```

Comparing this to the validation and training errors when K=3 which are listed below, one can see that the quality of the model is fairly high.

```
## [1] 0.02513089
```

```
## [1] 0.0115123
```

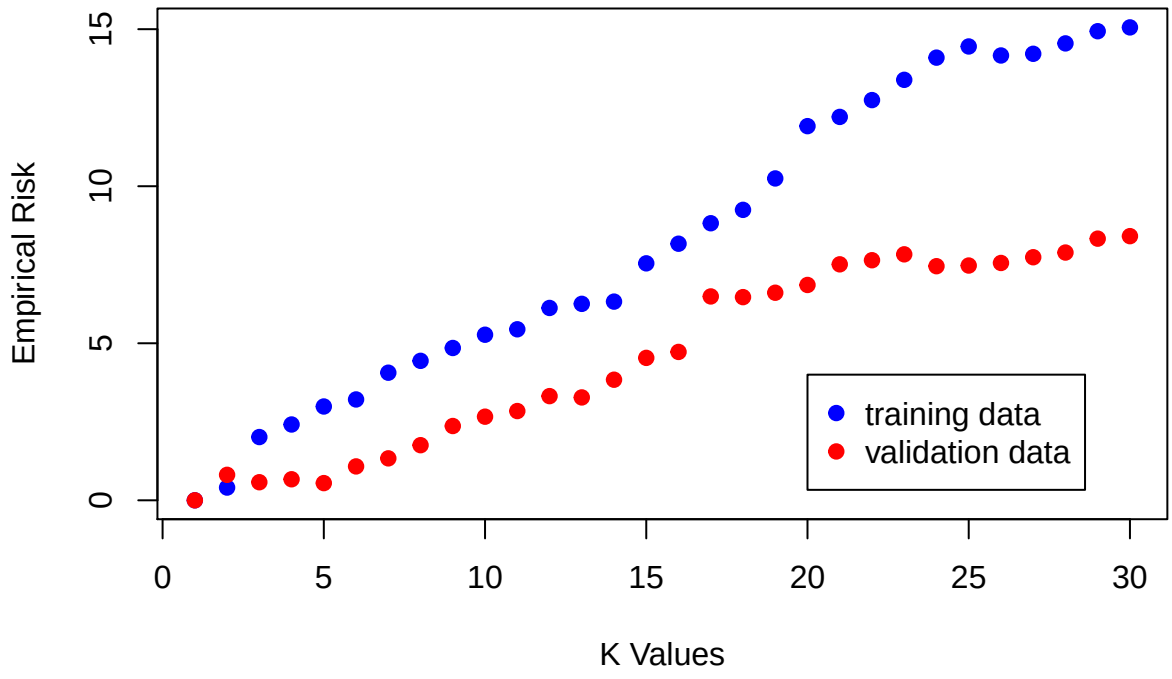
5.

The empirical risk for the training and validation data was computed using cross entropy. First the training and validation kknn objects were created for values of K=1:30, the fitted values and the prediction probability matrix were identified for each. Using the fitted values a target matrix was created in the form of a binary matrix with 1 representing the class digit of each row. This matrix was then multiplied by the logarithm of the prediction matrix to compute the cross entropy for each K value. The corresponing code used for the validation data is shown below.

```
CE_2<-function(i){
  valid_p<-kknn(as.factor(V65)~ ., train, validate,k=i, kernel="rectangular")
  target_vec<-valid_p$fitted.values
  pred_v<-valid_p$prob
  target_mat<-matrix(target_vec,nrow=length(target_vec),ncol=10)
  for (i in 1:length(target_vec)){
    pos<-as.integer(target_mat[i,1])
    target_mat[i,pos]<-1
    target_mat[i,-pos]<-0
  }
  target_mat<-as.numeric(target_mat)
  x<-c()
  for(i in 1:length(target_mat)){
    x[i]<-(target_mat[i]*log(pred_v[i]+1^-15))
  }
  s<-sum(x)
```

```
  return(s)
}
CE_validate<-unlist(lapply(k_values,CE_2))
k_values<-c(1:30)
```

The plot below shows the dependence of the empirical risk on the value of K for both the training and validation data. Cross entropy measures the difference between two probability distributions. In this case it is the probability distribution of the label target value and the model generated probability distribution. For both the validation data and the training data the optimal K is 1, where the empirical risk calculated via the cross entropy is zero. This indicates that the probability distributions for the real and predicted values are the same for each class, e.g zero loss when K=1. The cross-entropy increases as K increases, becoming less complex with the probability of the predicted distribution diverging farther from the target distribution. Using cross-entropy may be a more suitable choice to calculate empirical risk for the validation data because minimizing it can improve generalization, preventing overfitting.



Assignment 2. Ridge regression and model selection

1.

$$motor\_UPDRS =: y$$

For a single observation, we have the density as

$$p(y|\bar{x}) = \frac{1}{\sqrt{2\pi}\sigma} exp[-\frac{1}{2}\frac{(y - \Sigma_1^p x_j\theta_j)^2}{\sigma^2}]$$

thus the likelihood distribution is (after evaluating the product):

$$Likelihood(y_1, y_2, ..., y_n|\bar{x}_1, \bar{x}_2, ..., \bar{x}_n) = \frac{1}{[\sqrt{2\pi}\sigma]^n} exp[-\frac{1}{2}\frac{\Sigma_{i=1}^n(y_i - \Sigma_{j=1}^p x_{ij}\theta_j)^2}{\sigma^2}]$$

The implicit prior of the ridge-regression is $N(0, \frac{\lambda}{\sigma^2})$ for the squared sum of parameters (see p. 64 in Elements of Statistical learning for more detail), thus we obtain:

$$posterior(\theta) \propto \frac{1}{[\sqrt{2\pi}\sigma]^n} exp[-\frac{1}{2}\frac{\Sigma_{i=1}^{n}(y_i - \Sigma_{j=1}^{p} x_{ij}\theta_j)^2}{\sigma^2}] \frac{\sqrt{\lambda}}{\sqrt{2\pi}\sigma} exp[-\frac{\lambda}{2\sigma^2}\Sigma_{j=1}^{p}\theta_j^2]$$

$$= \frac{\sqrt{\lambda}}{[\sqrt{2\pi}\sigma]^{n+1}} exp[-\frac{1}{2}\frac{\Sigma_{i=1}^{n}(y_i - \Sigma_{j=1}^{p} x_{ij}\theta_j)^2 + \lambda\Sigma_{j=1}^{p}\theta_j^2}{\sigma^2}]$$

2. and 3.

```
df=read.csv("parkinsons.csv")

relevantColumns=c("motor_UPDRS","Jitter...", "Jitter.Abs.", "Jitter.RAP",
                  "Jitter.PPQ5", "Jitter.DDP", "Shimmer", "Shimmer.dB.",
                  "Shimmer.APQ3", "Shimmer.APQ5", "Shimmer.APQ11", "Shimmer.DDA",
                  "NHR", "HNR", "RPDE", "DFA", "PPE")

df2=df[,relevantColumns]
df3=scale(df2)
df3=data.frame(df3)


set.seed(12345)
id=sample(1:nrow(df3), floor(nrow(df3)*0.6))
train=df3[id,]
test=df3[-id,]

Loglikelihood=function(w, D){
  sigma=w[1]
  n=nrow(D)
  ys=D[,1]
  xs=data.matrix(D[,-1])
  yhats=xs %*% w[-1]
  constantFactor = (-n/2)*log(2*pi*(sigma^2))
  expFactor = - 0.5*sum((ys-yhats)^2)/sigma^2

  return(constantFactor+expFactor)
}

loglikelihood <- function(w, sigma){
  n <- dim(train)[1]
  part1 <- -(n/ 2) * log(2 * pi*(sigma^2))

  y <- train[, 1]
  x <- data.matrix(train[, -1])
  res <- sum((y - (t(w) %*% x))^2)

  return(part1 - (res/(2*((sigma)^2))))
}



Ridge=function(w, D, lambda){
```

```
  sigma=w[1]
  logL = Loglikelihood(w, D)
  ridgePenalty = 0.5*(log(2*pi*((sigma^2)/lambda)))+(lambda/(2*(sigma^2)))*sum(w[-1]^2)#((t(w[-1])%*%w[

  return(-logL + ridgePenalty)
}



RidgeOpt=function(w, D, sigma, lambda){
  w2=c(sigma,w)
  returnValue=optim(par=w2,fn=Ridge,D=D,lambda=lambda,method="BFGS")
  class(returnValue)="RidgeRegression"
  return(returnValue)
}


DF=function(D,lambda){
  xs=data.matrix(D[,-1])
  hatMatrix=xs%*%solve(t(xs)%*%xs + lambda*diag(ncol(xs)))%*%t(xs)
  dfs=sum(diag(hatMatrix))
  return(dfs)
}

AIC=function(ridgeModel,D,lambda){
  logL=Loglikelihood(ridgeModel$par, D)
  dof=DF(D,lambda)
  return(2*dof - 2*logL)
}

RidgeMSE=function(regrObject,D){
  ys=D[,1]
  xs=data.matrix(D[,-1])

  yhats=xs%*%regrObject$par[-1]
  MSE=mean((yhats-ys)^2)
  return(MSE)
}



for (lambda in c(1,100,1000)){
  w=rep(1,16)#initial weights for the optimization algorithm - can cause trouble
  sigma=1

  model=RidgeOpt(w, train, sigma, lambda)

  trainingMSE = RidgeMSE(model,train)
  testMSE = RidgeMSE(model,test)

  print(paste0("For lambda = ", lambda, ":"))
  print(paste0("MSE over training data is ", trainingMSE))
  print(paste0("and MSE over testing data is ", testMSE))
```

```
  print(paste0("and the AIC of the model is ", AIC(model,df3,lambda)))
  print("--------------------")
}
```

```
## [1] "For lambda = 1:"
## [1] "MSE over training data is 0.895531019497429"
## [1] "and MSE over testing data is 0.953448917108773"
## [1] "and the AIC of the model is 16228.9145960709"
## [1] "--------------------"
## [1] "For lambda = 100:"
## [1] "MSE over training data is 0.876906456893651"
## [1] "and MSE over testing data is 0.924258694507138"
## [1] "and the AIC of the model is 16048.7340088582"
## [1] "--------------------"
## [1] "For lambda = 1000:"
## [1] "MSE over training data is 1.07609819046199"
## [1] "and MSE over testing data is 1.11496556956011"
## [1] "and the AIC of the model is 17221.655289722"
## [1] "--------------------"
```

4.

$lambda = 100$ is preferable based on the MSE over the testing data. The reason that MSE is a fitting measure is because of the underlying normal distribution that is assumed in a ridge regression - MSE is proportional (with a factor $\frac{1}{n}$) to the loglikelihood.

5.

Based on the AIC-scores, the model that we obtain with $lambda = 100$ is preferable. The advantage is that AIC is the expected value of all possible cleavages of training/testing data, and thus should be more robust to sampling variance.

Assignment 3. Linear regression and LASSO

Loading the below required packages and data.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.0-2
```

1.

Importing R data. The data is divided into testing and training data sets. 50% of randomly selected data goes to training data set, and remaining 50% are used in the testing data set.

Probabilistic model:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_{100} x^{100} + \varepsilon.$$

$$\varepsilon \sim N(0, \sigma^2)$$

```
data=read.csv("tecator.csv")
set.seed(12345)
index = sample(1:nrow(data), 0.5*nrow(data))
train = data[index,]          # Create the training data
test = data[-index,]          # Create the test data
dim(train)
```

```
## [1] 107 104
```

```
dim(test)
```

```
## [1] 108 104
```

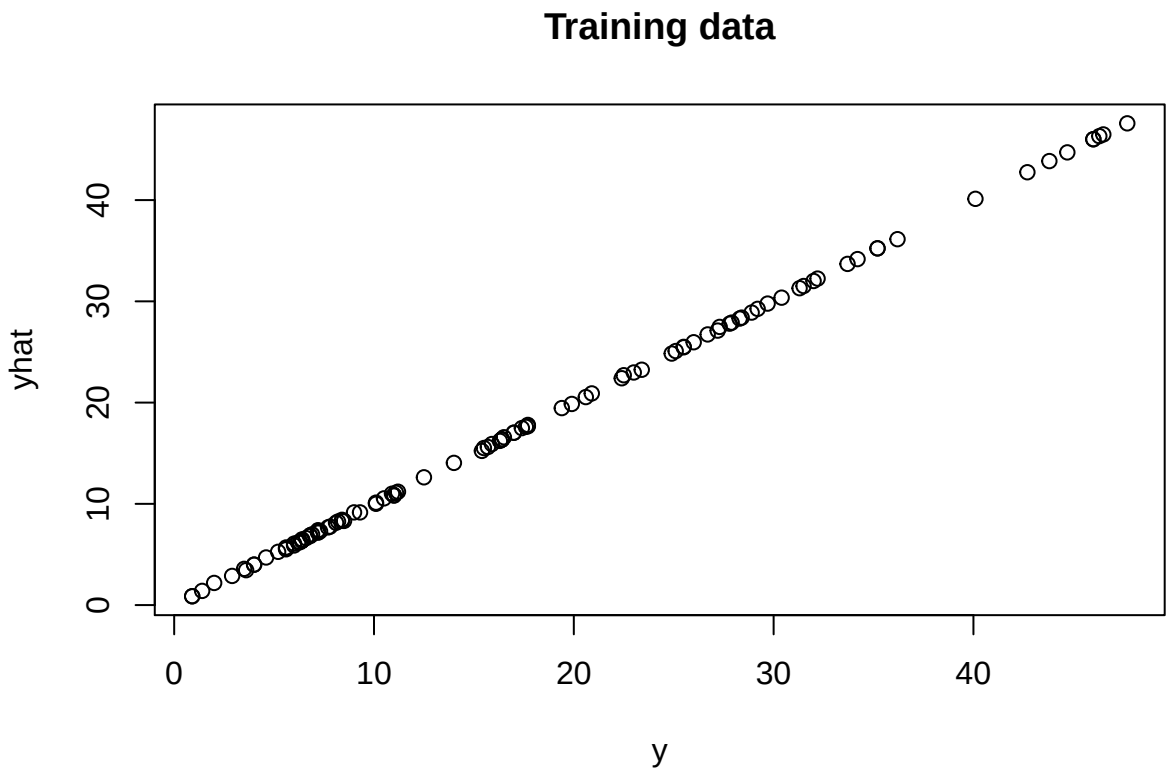
```
# Fit the linear regression to the training data

lmModel <- lm(Fat ~ ., data = train[,2:102])
#summary(lmModel)

# Estimating training and test errors for a simple linear model

y = as.matrix(train$Fat)
y0 = as.matrix(test$Fat)
yhat = predict(lmModel)
eval_metrics = function(obs,pred){
  mse = mean((obs-pred)^2)
  return(mse)
}
train.err = eval_metrics(y,yhat)
cat(paste("MSE for training data is ", train.err))
```

```
## MSE for training data is  0.00570911701090834
```

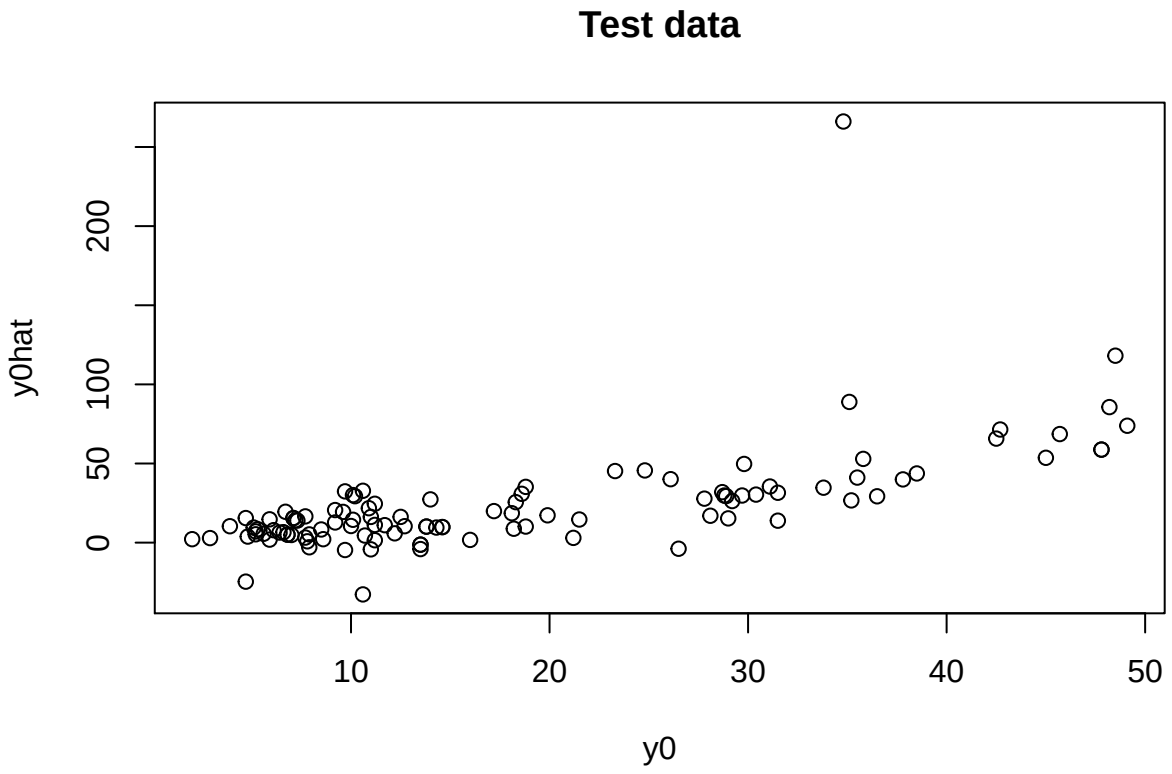
```
y0hat = predict(lmModel, test[,2:102])
test.err = eval_metrics(y0,y0hat)
cat(paste("\nMSE for testing data is ", test.err))
```

```
##
## MSE for testing data is  722.429419336971
```

```
plot(y, yhat, main="Training data")
```




```
plot(y0, y0hat, main="Test data")
```



MSE for training data is close to 0 but MSE for testing data is large hence it is not a good model. Due to the large number of features used, regularization is required.

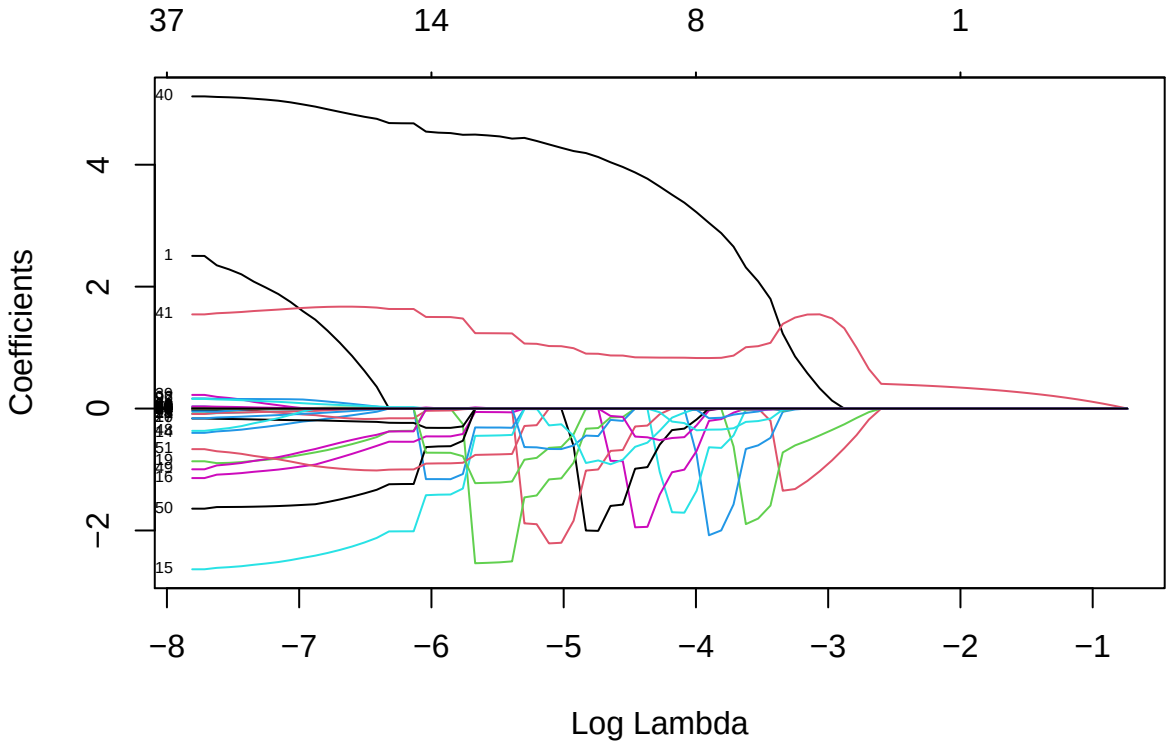
2.

Fat is modeled as a LASSO regression in which all Channels are used as features. Due to large number of features (channel1 to channel100) the model tends to over-fit. Lasso regression adds a penalty equivalent to the absolute magnitude of regression coefficients and tries to minimize them (i.e. shrink towards zero).Thus, it can avoid overfitting.

3.

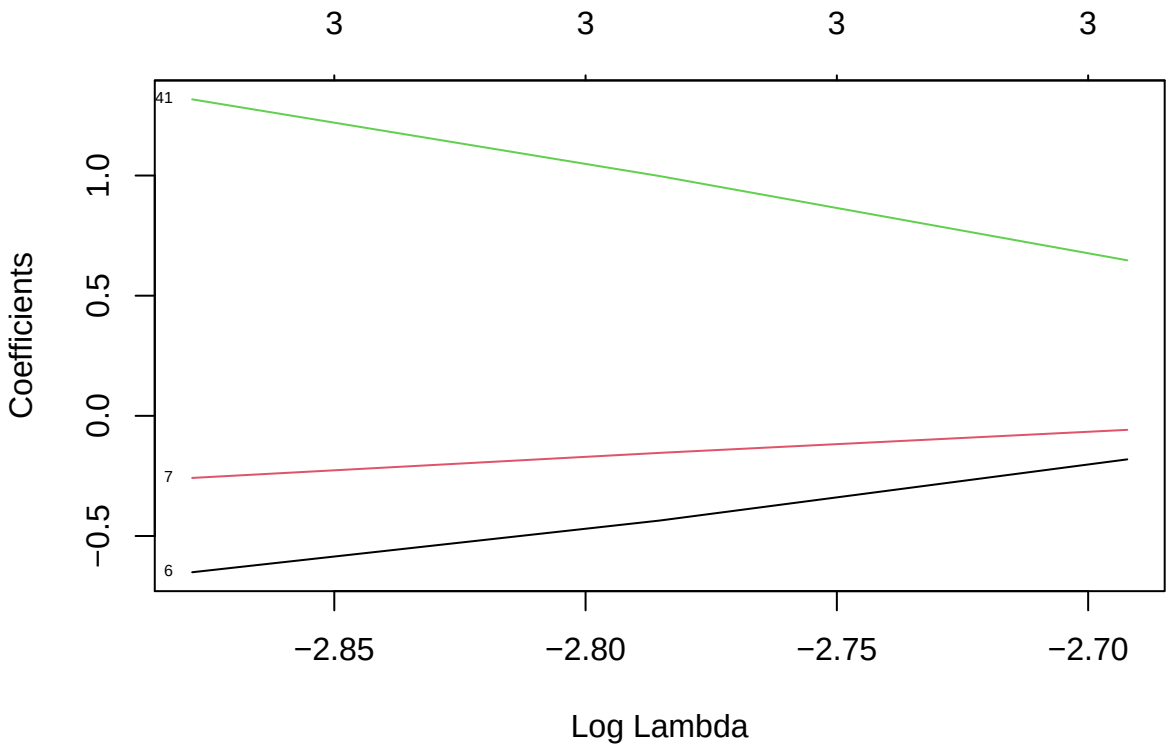
Fit the LASSO regression model to the training data.

```
dat=read.csv("tecator.csv")
covariates=scale(train[ ,2:101])
response=scale(train[ ,102])
model_lasso=glmnet(as.matrix(covariates),response, alpha=1,family="gaussian")
plot(model_lasso, xvar="lambda", label=TRUE)
```



In this plot, each line represent the channels(1 to 100). When lambda is very small, the LASSO solution should be very close to the OLS solution, and all the coefficients in the model. As lambda grows, the regularization term has greater effect and you will see fewer variables in your model (because more and more coefficients will be zero valued). One variable have a large impact regardless of log(lambda) value, the one with the highest values on coef (i.e. 41).

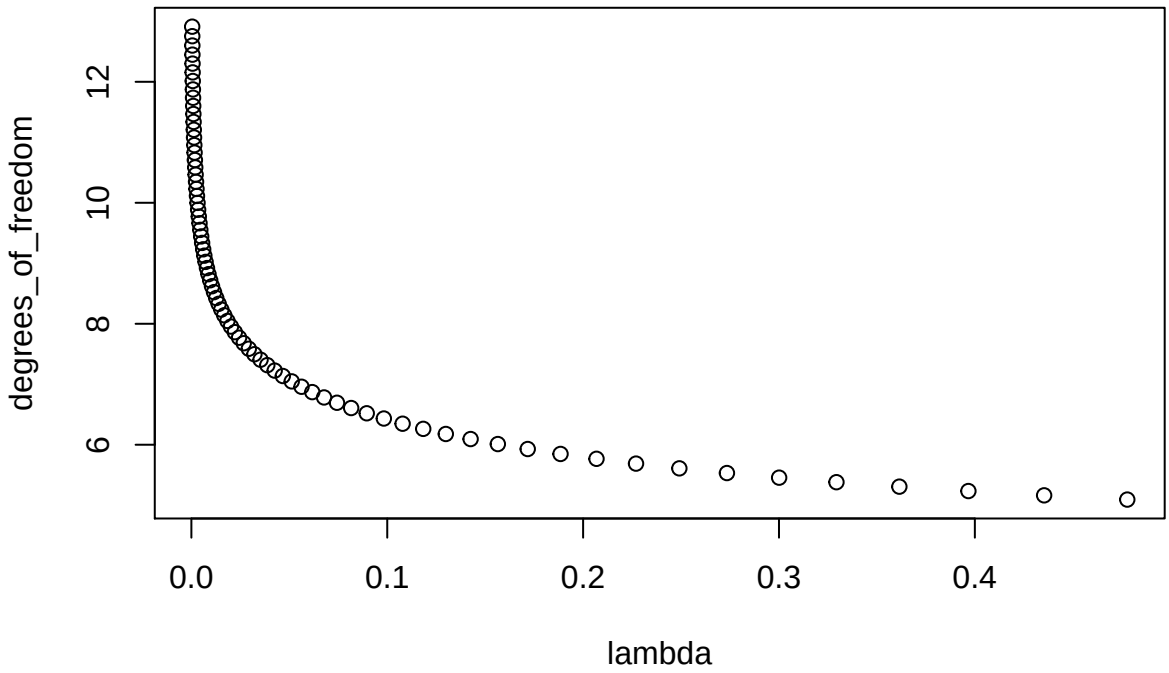
```
model=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian")
pen_fac = model$lambda[which(model$nzero == 3)]
model_lasso1 = glmnet(as.matrix(covariates), response, alpha = 1, family = "gaussian", lambda = pen_fac
plot(model_lasso1, xvar = "lambda", label = T)
```



4.

Degrees of freedom Vs penalty parameter

```
lambda = model_lasso$lambda
lasso.df<-function(lambda){
  x<-covariates
    df.ridge <- NULL
    for(l in lambda){
        df.ridge<-c(df.ridge,sum(diag(x%*% solve(t(x) %*% x + l*diag(ncol(x)))%*% t(x))))
    }
    return(df.ridge)
}
degrees_of_freedom = lasso.df(lambda)
plot(lambda, degrees_of_freedom)
```

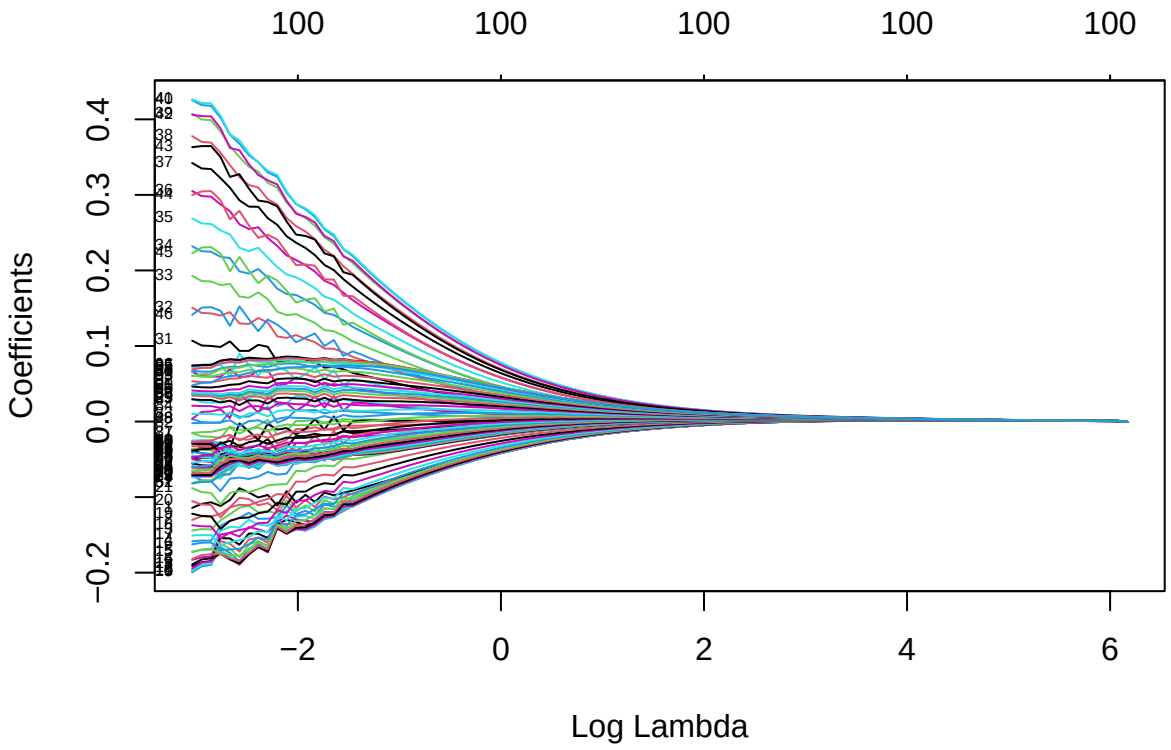


Degrees of freedom is inversely proportional to penalty parameter(lambda). Hence as shown in plot, degrees of freedom decreases with increase of penalty parameter.

5.

Fit the ridge regression model to the training data.

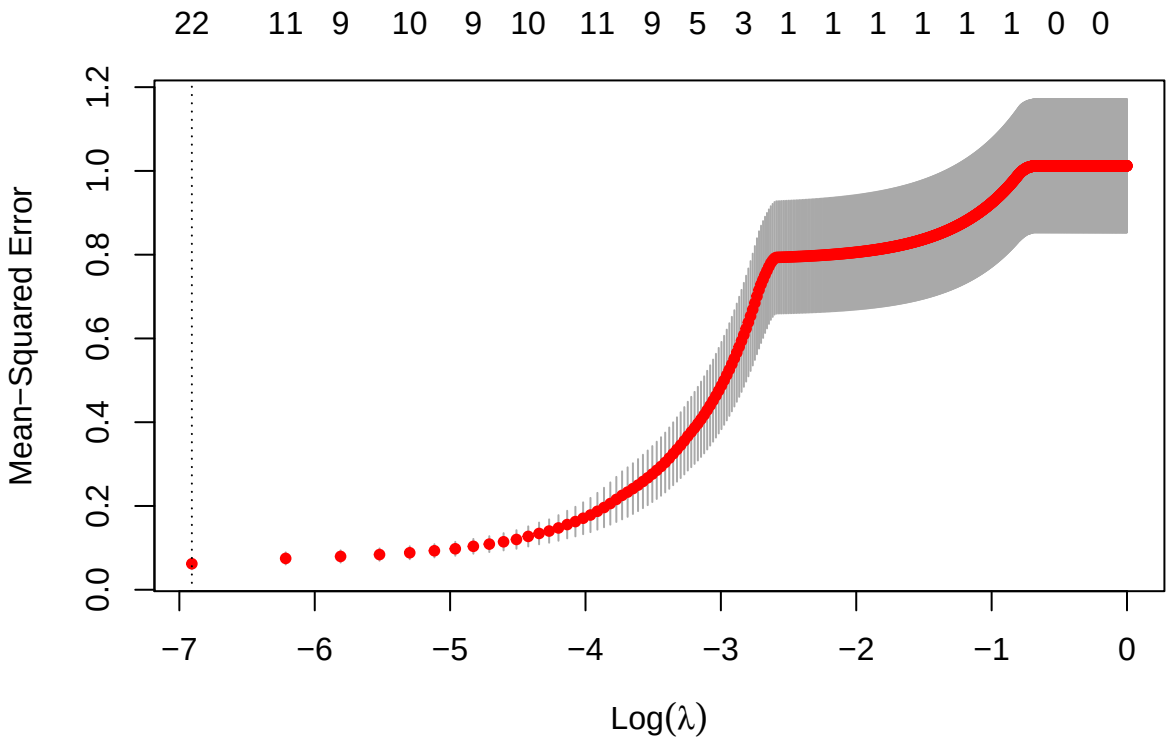
```
dat=read.csv("tecator.csv")
covariates=scale(train[,2:101])
response=scale(train[,102])
model_ridge=glmnet(as.matrix(covariates),
response, alpha=0,family="gaussian")
plot(model_ridge, xvar="lambda", label=TRUE)
```



In this plot, each line represent the channels(1 to 100) whose value is going to zero as you are penalizing more(increasing the lambda). When a line is further away from the line where y = 0 the more impact to the model it brings. Lasso regression is that it tends to make coefficients to absolute zero as compared to Ridge which never sets the value of coefficient to absolute zero.Ridge regression decreases the complexity of a model but does not reduce the number of variables since it never leads to a coefficient been zero rather only minimizes it.

6.

```
set.seed(12345)
lasso_cv = cv.glmnet(as.matrix(covariates),response, alpha=1, family="gaussian", lambda=seq(0,1,0.001))
plot(lasso_cv)
```



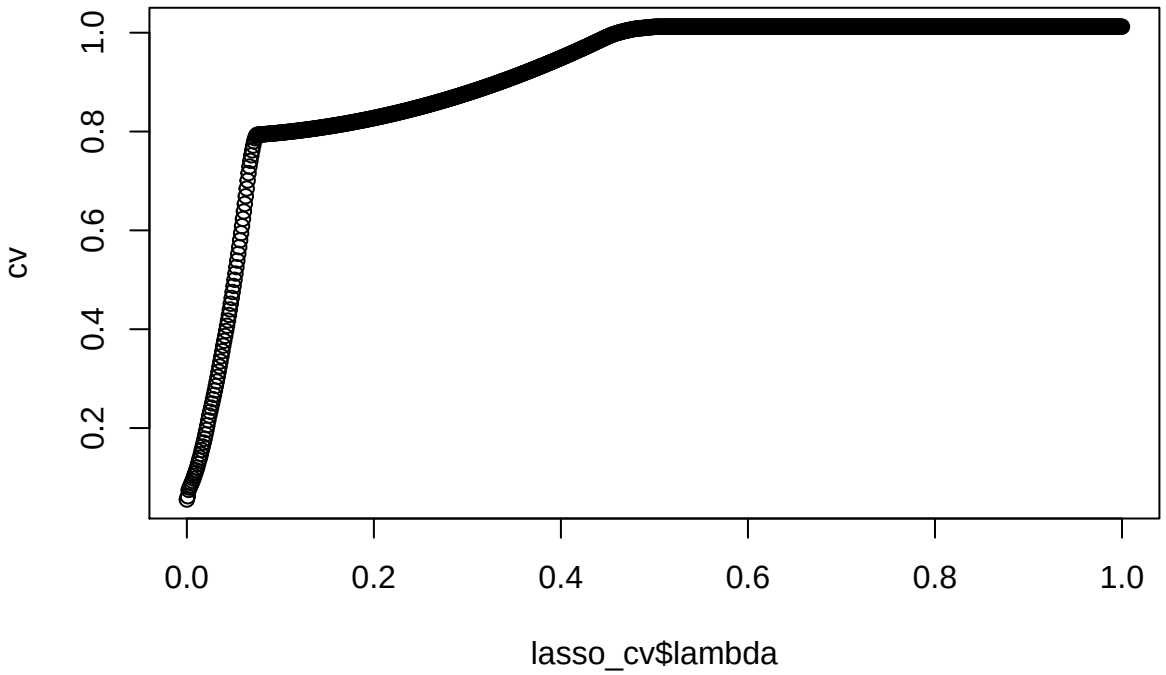

```
lambda_best <- lasso_cv$lambda.min
cat(paste("The lambda of the optimal model: ",lambda_best))
```

```
## The lambda of the optimal model:  0
```

```
cat(paste("\nNumber of variables chosen in optimal model: ",length(coef(lasso_cv, s="lambda.min"))))
```

```
##
## Number of variables chosen in optimal model:  101
```

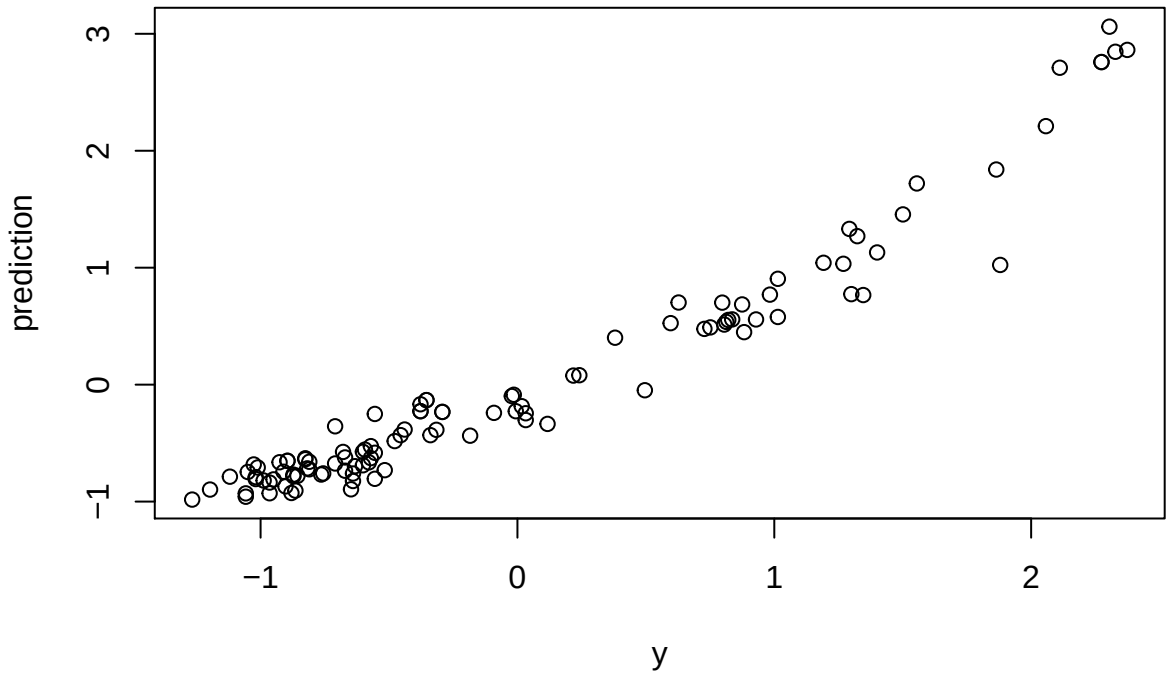
```
cv = lasso_cv$cvm
plot(lasso_cv$lambda,cv)
```



The CV score increases as the log(λ) increases. So when λ increases, MSE also increases which means the information can be removed as the λ increases. In this case optimal value of λ is 0. Number of variables chosen in optimal model is 101(100 channels + intercept). The plot (MSE Vs Log(λ)) starts with 22 coefficients and the value of log(λ) is almost -7. Till -1 at log(λ), MSE increases fastly. After -1, MSE becomes stagnant because there is no coefficients to penalize. From the plot, we can see that at log(λ) = -2, mean squared error is higher.

Original test vs predicted test

```
lambda_test =  glmnet(as.matrix(covariates), response, alpha = 1, family = "gaussian", lambda = lambda_
x = as.matrix(scale(test[, 2:101]))
y = as.matrix(scale(test[,102]))
prediction=predict(lambda_test, s = lambda_best, newx=x)
plot(y,prediction)
```

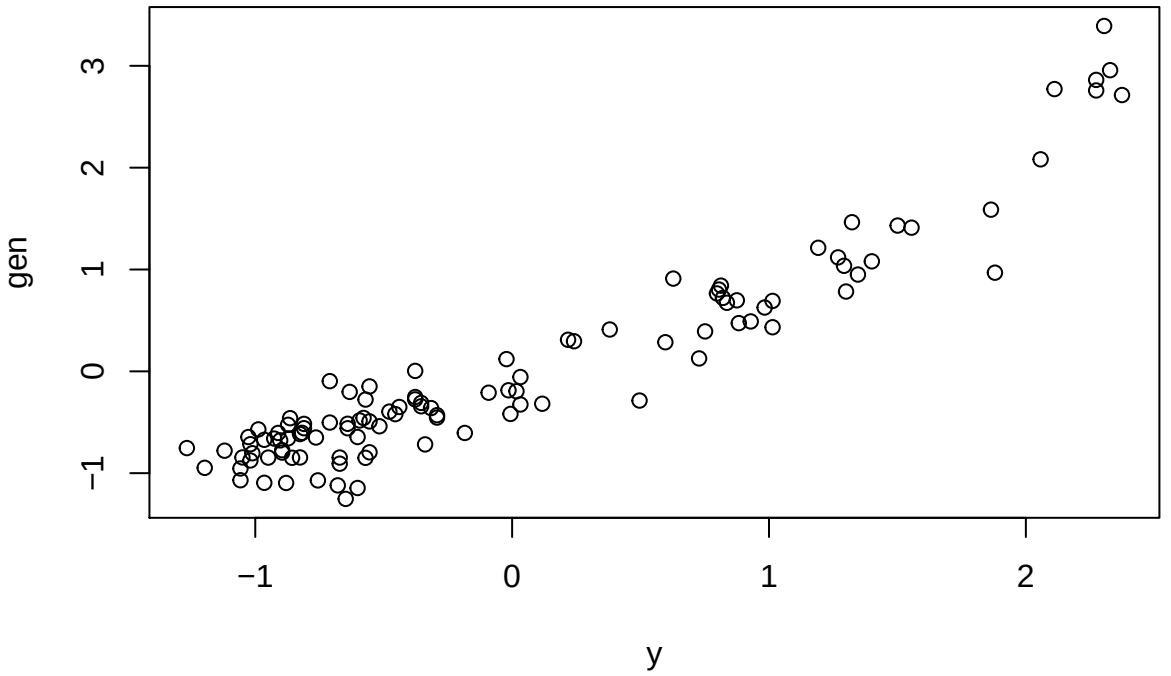


From the plot we can see that lasso model predictions are good for test data.

7.

Original Fat in test data Vs newly generated ones

```
target = as.matrix(coef(lambda_test))[-1]
resid = response - (covariates %*% target)
N = 108
std = sd(resid)
set.seed(123456)
gen = rnorm(N, prediction, std)
plot(y,gen)
```



```
cat(paste("MSE of data generation is :", eval_metrics(gen, y)))
```

```
## MSE of data generation is : 0.108944984375774
```

Here, MSE is low hence the quality of generated data fit well.

Apendix

```
knitr::opts_chunk$set(echo = TRUE)
digits<-read.csv("optdigits.csv",header=FALSE)
n<-dim(digits)[1]
set.seed(12345)
id<-sample(1:n, floor(n*0.5))
train<-digits[id,1:65]

id1<-setdiff(1:n, id)
set.seed(12345)
id2<-sample(id1, floor(n*0.25))
validate<-digits[id2,1:65]

id3<-setdiff(id1,id2)
test<-digits[id3,1:65]
library(kknn)
#training data
num.knn1<-kknn(as.factor(V65)~ ., train, train,k=30, kernel="rectangular")
fit1<-fitted(num.knn1)
CM1<-table(train$V65,fit1)

#test data
num.knn2<-kknn(as.factor(V65)~ ., train, test,k=30, kernel="rectangular")
fit2<-fitted(num.knn2)
CM2<-table(test$V65,fit2)
accuracy1<-(sum(diag(CM1)))/(sum(CM1))
accuracy1
accuracy2<-(sum(diag(CM2)))/(sum(CM2))
accuracy2
miss_train<-mean(fit1 != train$V65)
miss_train
miss_test<-mean(fit2 != test$V65)
miss_test
#prediction matrix
pred<-num.knn1$prob

#find locations in test data where probability is max
max<-which(pred[,9] == max(pred[,9]))
max_cases<-max[1:2]
max_cases
max1<-test[max_cases[1],]
max2<-test[max_cases[2],]
#reshape features and create heatmaps
mat_max1<-matrix(as.numeric(matrix(max1[1:64],ncol=8)),ncol=8)
heatmap(mat_max1, Colv=NA, Rowv=NA)
mat_max2<-matrix(as.numeric(matrix(max2[1:64],ncol=8)),ncol=8)
```

```
heatmap(mat_max2, Colv=NA, Rowv=NA)
#find locations in test data where probability is minimum
min<-which(pred[,9] != 0 & pred[,9] < 0.034)
min_cases<-min[1:3]
min_cases
min1<-test[min_cases[1],]
min2<-test[min_cases[2],]
min3<-test[min_cases[3],]
#reshape features and create heatmaps
mat_min1<-matrix(as.numeric(matrix(min1[1:64],ncol=8)),ncol=8)
heatmap(mat_min1, Colv=NA, Rowv=NA)
mat_min2<-matrix(as.numeric(matrix(min2[1:64],ncol=8)),ncol=8)
heatmap(mat_min2, Colv=NA, Rowv=NA)
mat_min3<-matrix(as.numeric(matrix(min3[1:64],ncol=8)),ncol=8)
heatmap(mat_min3, Colv=NA, Rowv=NA)
#knn missclassification rates for different values of k for training data
knn_train<-function(i){
  a<-kknn(as.factor(V65)~ ., train, train,k=i, kernel="rectangular")
  misscl<-mean(a$fitted.values != train$V65)
  return(misscl)
}
k_values<-c(1:30)
missclass_train<-unlist(lapply(k_values,knn_train))

#knn missclassification rates for different values of k for validation data
knn_validate<-function(i){
  a<-kknn(as.factor(V65)~ ., train, validate,k=i, kernel="rectangular")
  misscl<-mean(a$fitted.values != validate$V65)
  return(misscl)
}
k_values<-c(1:30)
missclass_validate<-unlist(lapply(k_values,knn_validate))
plot(k_values,missclass_validate, pch=19, col="blue", xlab="K Values", ylab="Missclassification Rate")
points(k_values,missclass_train,col="red",pch=19)
legend(20,0.01,legend=c("validation data","training data"), col=c("blue","red"),pch=19)
test_knn<-kknn(as.factor(V65)~ ., train, test,k=3, kernel="rectangular")
missclass_test<-mean(test_knn$fitted.values != test$V65)
missclass_test
valid_knn<-kknn(as.factor(V65)~ ., train, validate,k=3, kernel="rectangular")
missclass_valid<-mean(valid_knn$fitted.values != validate$V65)
missclass_valid

train_knn<-kknn(as.factor(V65)~ ., train, train,k=3, kernel="rectangular")
missclass_train<-mean(train_knn$fitted.values != train$V65)
missclass_train
CE_1<-function(i){
  train_p<-kknn(as.factor(V65)~ ., train, train,k=i, kernel="rectangular")
  target_vec<-train_p$fitted.values
  pred_v<-train_p$prob
  target_mat<-matrix(target_vec,nrow=length(target_vec),ncol=10)
  for (i in 1:length(target_vec)){
    pos<-as.integer(target_mat[i,1])
    target_mat[i,pos]<-1
```

```
    target_mat[i,-pos]<-0
  }
  target_mat<-as.numeric(target_mat)
  x<-c()
  for(i in 1:length(target_mat)){
    x[i]<-(target_mat[i]*log(pred_v[i]+1^-15))
  }
  s<-sum(x)
  return(s)
}
CE_train<-unlist(lapply(k_values,CE_1))
k_values<-c(1:30)
CE_2<-function(i){
  valid_p<-kknn(as.factor(V65)~ ., train, validate,k=i, kernel="rectangular")
  target_vec<-valid_p$fitted.values
  pred_v<-valid_p$prob
  target_mat<-matrix(target_vec,nrow=length(target_vec),ncol=10)
  for (i in 1:length(target_vec)){
    pos<-as.integer(target_mat[i,1])
    target_mat[i,pos]<-1
    target_mat[i,-pos]<-0
  }
  target_mat<-as.numeric(target_mat)
  x<-c()
  for(i in 1:length(target_mat)){
    x[i]<-(target_mat[i]*log(pred_v[i]+1^-15))
  }
  s<-sum(x)
  return(s)
}
CE_validate<-unlist(lapply(k_values,CE_2))
k_values<-c(1:30)
plot(k_values,CE_train, pch=19, col="blue", xlab="K Values", ylab="Empirical Risk")
points(k_values,CE_validate,col="red",pch=19)
legend(20,4,legend=c("training data","validation data"), col=c("blue","red"),pch=19)
df=read.csv("parkinsons.csv")

relevantColumns=c("motor_UPDRS","Jitter...", "Jitter.Abs.", "Jitter.RAP",
                  "Jitter.PPQ5", "Jitter.DDP", "Shimmer", "Shimmer.dB.",
                  "Shimmer.APQ3", "Shimmer.APQ5", "Shimmer.APQ11", "Shimmer.DDA",
                  "NHR", "HNR", "RPDE", "DFA", "PPE")

df2=df[,relevantColumns]
df3=scale(df2)
df3=data.frame(df3)


set.seed(12345)
id=sample(1:nrow(df3), floor(nrow(df3)*0.6))
train=df3[id,]
test=df3[-id,]

Loglikelihood=function(w, D){
```

```
  sigma=w[1]
  n=nrow(D)
  ys=D[,1]
  xs=data.matrix(D[,-1])
  yhats=xs %*% w[-1]
  constantFactor = (-n/2)*log(2*pi*(sigma^2))
  expFactor = - 0.5*sum((ys-yhats)^2)/sigma^2

  return(constantFactor+expFactor)
}

loglikelihood <- function(w, sigma){
  n <- dim(train)[1]
  part1 <- -(n/ 2) * log(2 * pi*(sigma^2))

  y <- train[, 1]
  x <- data.matrix(train[, -1])
  res <- sum((y - (t(w) %*% x))^2)

  return(part1 - (res/(2*((sigma)^2))))
}



Ridge=function(w, D, lambda){
  sigma=w[1]
  logL = Loglikelihood(w, D)
  ridgePenalty = 0.5*(log(2*pi*((sigma^2)/lambda)))+(lambda/(2*(sigma^2)))*sum(w[-1]^2)#((t(w[-1])%*%w[

  return(-logL + ridgePenalty)
}



RidgeOpt=function(w, D, sigma, lambda){
  w2=c(sigma,w)
  returnValue=optim(par=w2,fn=Ridge,D=D,lambda=lambda,method="BFGS")
  class(returnValue)="RidgeRegression"
  return(returnValue)
}


DF=function(D,lambda){
  xs=data.matrix(D[,-1])
  hatMatrix=xs%*%solve(t(xs)%*%xs + lambda*diag(ncol(xs)))%*%t(xs)
  dfs=sum(diag(hatMatrix))
  return(dfs)
}

AIC=function(ridgeModel,D,lambda){
  logL=Loglikelihood(ridgeModel$par, D)
  dof=DF(D,lambda)
  return(2*dof - 2*logL)
```

```
}

RidgeMSE=function(regrObject,D){
  ys=D[,1]
  xs=data.matrix(D[,-1])

  yhats=xs%*%regrObject$par[-1]
  MSE=mean((yhats-ys)^2)
  return(MSE)
}



for (lambda in c(1,100,1000)){
  w=rep(1,16)#initial weights for the optimization algorithm - can cause trouble
  sigma=1

  model=RidgeOpt(w, train, sigma, lambda)

  trainingMSE = RidgeMSE(model,train)
  testMSE = RidgeMSE(model,test)

  print(paste0("For lambda = ", lambda, ":"))
  print(paste0("MSE over training data is ", trainingMSE))
  print(paste0("and MSE over testing data is ", testMSE))

  print(paste0("and the AIC of the model is ", AIC(model,df3,lambda)))
  print("--------------------")
}
library(glmnet)
data=read.csv("tecator.csv")
set.seed(12345)
index = sample(1:nrow(data), 0.5*nrow(data))
train = data[index,]          # Create the training data
test = data[-index,]          # Create the test data
dim(train)
dim(test)

# Fit the linear regression to the training data

lmModel <- lm(Fat ~ ., data = train[,2:102])
#summary(lmModel)

# Estimating training and test errors for a simple linear model

y = as.matrix(train$Fat)
y0 = as.matrix(test$Fat)
yhat = predict(lmModel)
eval_metrics = function(obs,pred){
  mse = mean((obs-pred)^2)
  return(mse)
}
train.err = eval_metrics(y,yhat)
```

```
cat(paste("MSE for training data is ", train.err))
y0hat = predict(lmModel, test[,2:102])
test.err = eval_metrics(y0,y0hat)
cat(paste("\nMSE for testing data is ", test.err))

plot(y, yhat, main="Training data")
plot(y0, y0hat, main="Test data")
dat=read.csv("tecator.csv")
covariates=scale(train[ ,2:101])
response=scale(train[ ,102])
model_lasso=glmnet(as.matrix(covariates),response, alpha=1,family="gaussian")
plot(model_lasso, xvar="lambda", label=TRUE)
model=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian")
pen_fac = model$lambda[which(model$nzero == 3)]
model_lasso1 = glmnet(as.matrix(covariates), response, alpha = 1, family = "gaussian", lambda = pen_fac)
plot(model_lasso1, xvar = "lambda", label = T)

lambda = model_lasso$lambda
lasso.df<-function(lambda){
  x<-covariates
    df.ridge <- NULL
    for(l in lambda){
        df.ridge<-c(df.ridge,sum(diag(x%*% solve(t(x) %*% x + l*diag(ncol(x)))%*% t(x))))
    }
    return(df.ridge)
}
degrees_of_freedom = lasso.df(lambda)
plot(lambda, degrees_of_freedom)
dat=read.csv("tecator.csv")
covariates=scale(train[,2:101])
response=scale(train[,102])
model_ridge=glmnet(as.matrix(covariates),
response, alpha=0,family="gaussian")
plot(model_ridge, xvar="lambda", label=TRUE)
set.seed(12345)
lasso_cv = cv.glmnet(as.matrix(covariates),response, alpha=1, family="gaussian", lambda=seq(0,1,0.001))
plot(lasso_cv)
lambda_best <- lasso_cv$lambda.min
cat(paste("The lambda of the optimal model: ",lambda_best))
cat(paste("\nNumber of variables chosen in optimal model: ",length(coef(lasso_cv, s="lambda.min"))))
cv = lasso_cv$cvm
plot(lasso_cv$lambda,cv)
lambda_test =  glmnet(as.matrix(covariates), response, alpha = 1, family = "gaussian", lambda = lambda_l
x = as.matrix(scale(test[, 2:101]))
y = as.matrix(scale(test[,102]))
prediction=predict(lambda_test, s = lambda_best, newx=x)
plot(y,prediction)
target = as.matrix(coef(lambda_test))[-1]
resid = response - (covariates %*% target)
N = 108
std = sd(resid)
set.seed(123456)
gen = rnorm(N, prediction, std)
```

```r
plot(y,gen)
cat(paste("MSE of data generation is :", eval_metrics(gen, y)))
```