Computer Lab 1

Linköping University

Jakob Fjellström
Mowniesh Asokan
Tim Yuki Washio

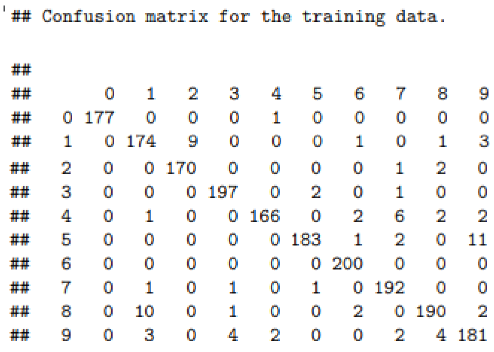
November 18, 2020

Assignment 1: Handwritten digit recognition with K-means

1.

```
#Split the data into train, validation and test
set.seed(12345)
sample_train<- sample(seq_len(nrow(dataset)),
size = floor(0.50*nrow(dataset)))
sample_valid<- sample(seq_len(nrow(dataset)),
size = floor(0.25*nrow(dataset)))
sample_test <- sample(seq_len(nrow(dataset)),
size = floor(0.25*nrow(dataset)))
train      <- dataset[sample_train, ]
validation<- dataset[sample_valid, ]
test       <- dataset[sample_test, ]
```

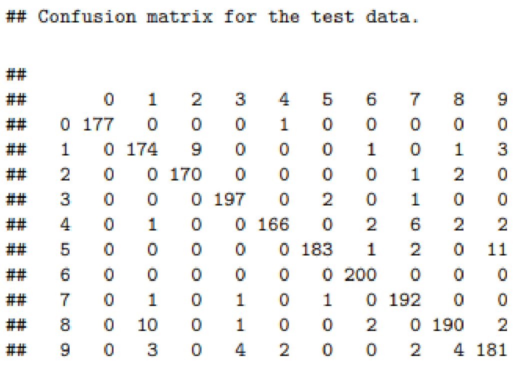
2.
Model is worked and trained by using the training data. In this nearest neighbor method estimate is formed over the average of K nearest data points and also the distance is calculated by using Euclidean distance formula. Below is the confusion matrix for the two data sets.



```
## Confusion matrix for the training data.

##
##        0   1   2   3   4   5   6   7   8   9
##    0 177   0   0   0   1   0   0   0   0   0
##    1   0 174   9   0   0   0   1   0   1   3
##    2   0   0 170   0   0   0   0   1   2   0
##    3   0   0   0 197   0   2   0   1   0   0
##    4   0   1   0   0 166   0   2   6   2   2
##    5   0   0   0   0   0 183   1   2   0  11
##    6   0   0   0   0   0   0 200   0   0   0
##    7   0   1   0   1   0   1   0 192   0   0
##    8   0  10   0   1   0   0   2   0 190   2
##    9   0   3   0   4   2   0   0   2   4 181
```

(a) Confusion matrix for the training data.



```
## Confusion matrix for the test data.

##
##        0   1   2   3   4   5   6   7   8   9
##    0 177   0   0   0   1   0   0   0   0   0
##    1   0 174   9   0   0   0   1   0   1   3
##    2   0   0 170   0   0   0   0   1   2   0
##    3   0   0   0 197   0   2   0   1   0   0
##    4   0   1   0   0 166   0   2   6   2   2
##    5   0   0   0   0   0 183   1   2   0  11
##    6   0   0   0   0   0   0 200   0   0   0
##    7   0   1   0   1   0   1   0 192   0   0
##    8   0  10   0   1   0   0   2   0 190   2
##    9   0   3   0   4   2   0   0   2   4 181
```

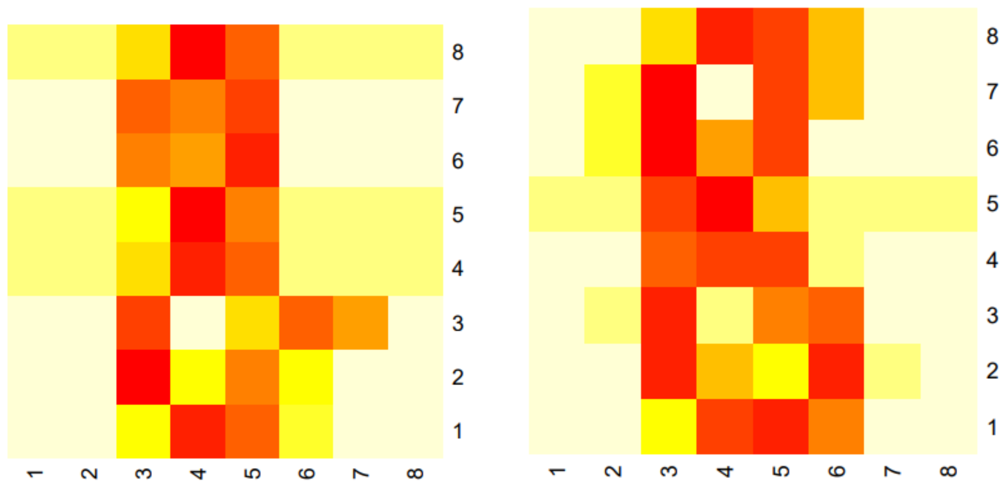
(b) Confusion matrix for the test data.

Figure 1: This figure shows the confusion matrices for both the training data and the test data.

The miss classification rate of each data set are the following:
i) 0.04238619
ii) 0.05968586

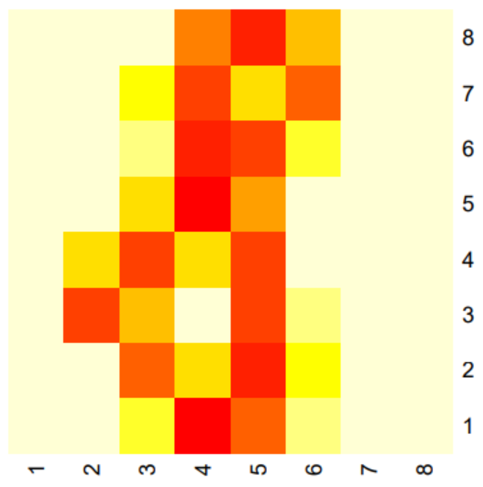
According to value We get through the confusion matrix, the miss classification of actual 3 and 4 are high in rate and somewhat 9 is predicted wrongly as 5 and that the miss classification rate is affected by test and validation data.

3.
In this exercise we were tasked to find any 2 cases of digit "8" in the training data which were easiest to classify and 3 cases that were hardest to classify. Below is the heatmaps.

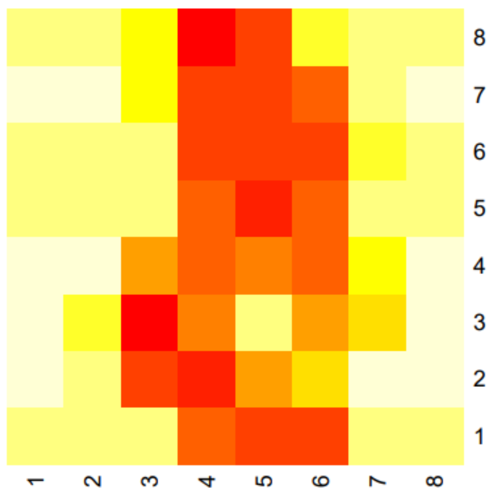


(a) (b)

Figure 2



(a)



(b)

Figure 3

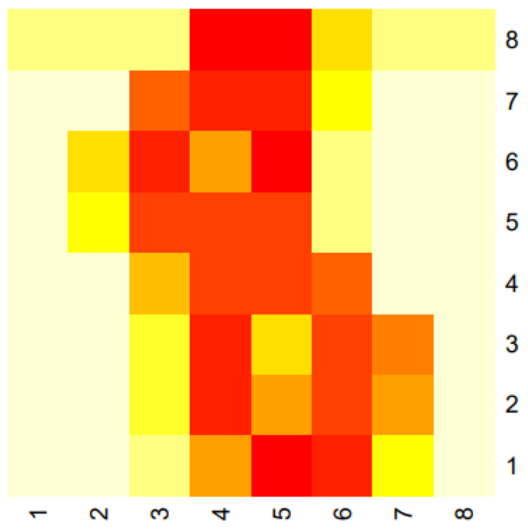


Figure 4

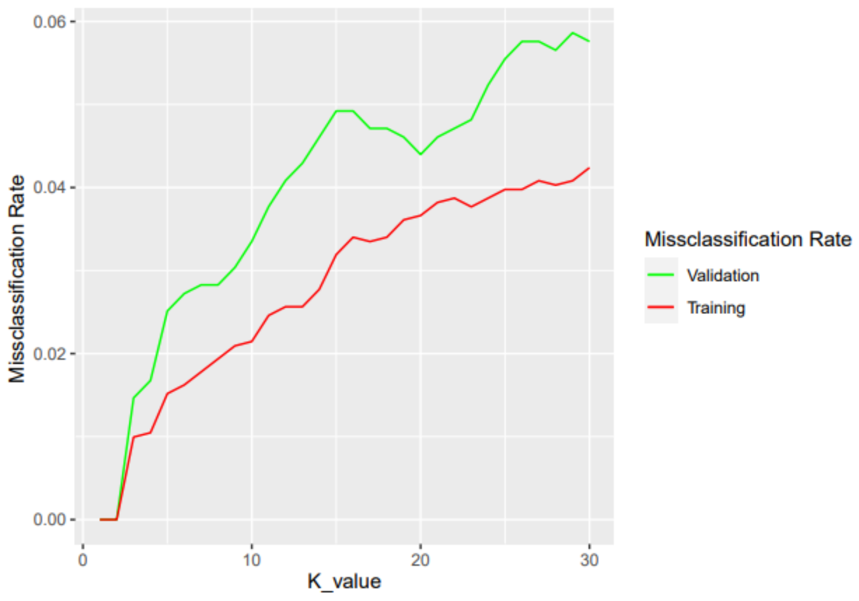
The easiest way to classify the 8 is 3 and the toughest way to classify the 8 is 9 and 4.

4. Find the optimal K value for KNN model for this dataset.(k=1 to 30)

```
##  [1] 0.000000000 0.000000000 0.009942439 0.010465725 0.015175301 0.016221873
##  [7] 0.017791732 0.019361591 0.020931450 0.021454736 0.024594453 0.025641026
## [13] 0.025641026 0.027734171 0.031920460 0.034013605 0.033490319 0.034013605
## [19] 0.036106750 0.036630037 0.038199895 0.038723182 0.037676609 0.038723182
## [25] 0.039769754 0.039769754 0.040816327 0.040293040 0.040816327 0.042386185

##  [1] 0.00000000 0.00000000 0.01465969 0.01675393 0.02513089 0.02722513
##  [7] 0.02827225 0.02827225 0.03036649 0.03350785 0.03769634 0.04083770
## [13] 0.04293194 0.04607330 0.04921466 0.04921466 0.04712042 0.04712042
## [19] 0.04607330 0.04397906 0.04607330 0.04712042 0.04816754 0.05235602
## [25] 0.05549738 0.05759162 0.05759162 0.05654450 0.05863874 0.05759162
```

Figure 5: Caption



```
## calculated optimum value of K is:  1
```

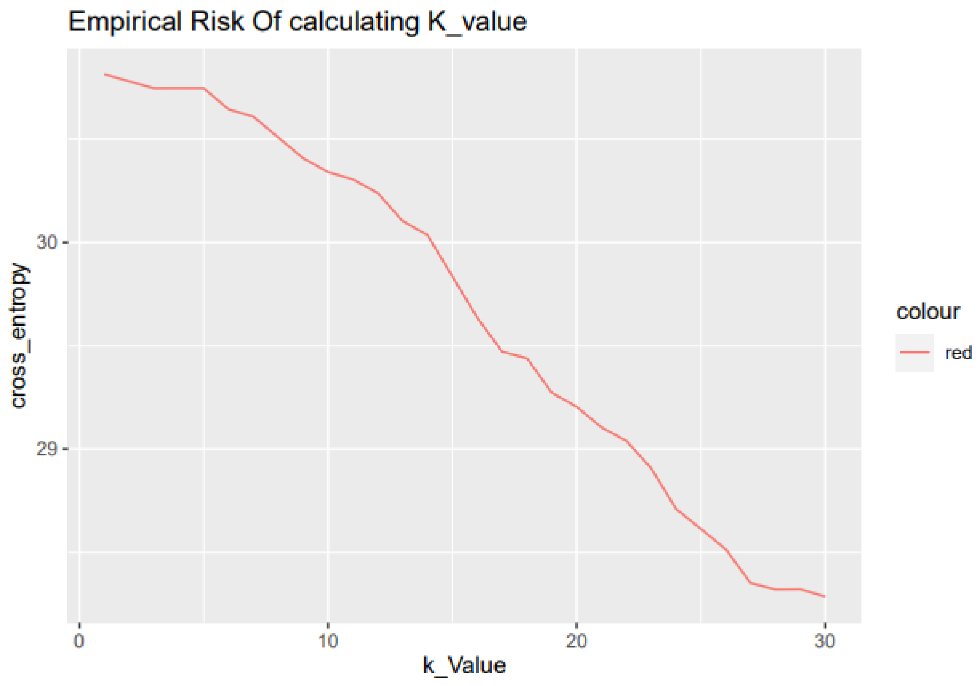
Figure 6

Based on the varying K value to the model the optimal k value for this dataset is 1. In this hand written digit recognition dataset bias between the data is low and the variance among the data points is high.Miss classification rate for validation and training set is not equal for all the value of k(1 to 30). But the Error rate is equal in the k= 1 and 2.

5. Empirical risk for the validation data as cross-entropy

```
##  [1] 30.81365 30.77966 30.74592 30.74640 30.74618 30.64378 30.60867 30.50763
##  [9] 30.40723 30.34040 30.30476 30.23907 30.10344 30.03662 29.83690 29.63724
## [17] 29.47151 29.43961 29.27262 29.20510 29.10504 29.04018 28.90677 28.71073
## [25] 28.61364 28.51576 28.35212 28.32167 28.32275 28.28716
```

Figure 7



```
## calculated optimum value of K by using the cross entropy function is:  28.28716
```

Figure 8

In this case the entropy function , Empirical risk of the entropy value get slightly decreased when the value of k increased.So the log loss of this function is decreased depend only on the k value.

Assignment 2: Ridge Regression and Model Selection

In the second assignment of this computer lab, we were supposed to employ a ridge regression on the data file **parkinson.csv**.
The assignment was composed of 5 smaller tasks, and this section will from

now be structured as in the lab PDF.

1. In this first sub-task, we were supposed to write down the Ridge regression probabilistic model as a Bayesian model. Below is our solution.

In the assignment PDF, we were given that our target value, motor_UPDRS is normally distributed, which makes us choose a normal prior for β. This results in the probability model y $\sim$ N($\mathbf{X}\beta, \sigma^2\mathbf{I}$), where $\beta \sim N(0, \tau\mathbf{I})$. So if we apply Bayes Theorem, we'll get the following expression for the posterior distribution of the coefficients, and since both components on the r.h.s. follows a normal distribution, the posterior will also be normally distributed.

$$p(\beta|y,\sigma^2) \propto p(\beta)p(y|\beta,\sigma^2)$$

2. Here, we were supposed to split the dataset into a training set and a test set with the ratio of 60/40. We did the following.

```
ind <- floor(nrow(data)*0.6)
train_ind <- sample(1:(nrow(data)), size = ind)

train <- data[train_ind,]
test <- data[-train_ind,]
```

3. In this sub-task, we were supposed to implement four functions, below - starting with the log-likelihood function - is our solution.

a) As mentioned above, we were supposed to write the log-likelihood function for the probability model in task (1). So what we are searching for is $P(Data|\beta,\sigma^2)$. We know from the question that the data is normally distributed, hence our Likelihood function is the following.

$$L(Data|\beta,\sigma^2) = \prod_{i=1}^{n} f(y|,\beta,\sigma^2)$$

$$\prod_{i=1}^{n} f(y|,\beta,\sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y_i-\beta^T X)^2} = (2\pi\sigma^2)^{-\frac{n}{2}} e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\beta^T X)^2}$$

Now, if we take the log of $L(Data|\beta, \sigma^2)$, we have the following.

$$logL(Data|\beta, \sigma^2) = -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta^T X)^2$$

In R, we implemented the log-likelihood as follows.

```
Loglikelihood <- function(i_w, input_data){

  beta <- as.matrix(i_w[1:16])
  sigma <- i_w[17]

  y <- as.matrix(input_data[ ,1])
  X <- as.matrix(input_data[,-1])
  n <- nrow(input_data)

  RSS <- sum((y - X %*% beta)^2)

  l_Lhood <- -(n/2) * log(2*pi*sigma^2) - (1/(2*sigma^2)) * RSS

  return(-l_Lhood)

}
```

b) Here, a Ridge function should be created. The function should use the log likelihood from above task, and add a ridge penalty term. Since the ridge regression penalty term is just a scalar λ multiplied with the sum of the squared coefficients. So if we continue from were we left off with the log likelihood function from above, we just need to add the mentioned penalty, i.e. we get the following expression.

$$logL(Data|\beta, \sigma^2) = -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta^T X)^2 + \lambda\sum_{j=1}^{P}\beta_j^2$$

In R, we implemented the above expression as following.

```
 Ridge <- function(i_w, input_data, lambda){
  beta <- as.matrix(i_w[1:16])
  sigma <- i_w[17]

  y <- as.matrix(input_data[1])
  X <- as.matrix(input_data[-1])
  n <- nrow(input_data)


  RSS <- sum((y - X %*% beta)^2)
  penalty <- lambda * sum(beta^2)

  l_Lhood <- -(n/2) * log(2*pi*sigma^2) - (1/(2*sigma^2)) * RSS +
  penalty

  return(-l_Lhood)
}
```

c) In this task, we were supposed to implement a function called *RidgeOpt* that uses the R base optim() function to find the optimal w and σ for a given λ. Below is the code we used to optimize those parameters.

```
RidgeOpt <- function(lambda){

  opt_estimate <- optim(par=c(i_w = 1:17), fn = Ridge,
  lambda = lambda, input_data = train , method = 'BFGS')

  return(opt_estimate)

}
```

d) In this final task, we we're supposed to write a function that for a given value of λ calculates the degrees of freedom of the ridge regression. Below is how it was implemented in R.

```
df <- function(lambda, input_data){
  X <- as.matrix(input_data[-1])
  d <- sum(diag( X %*% solve(t(X) %*% X + lambda * diag(ncol(X)))
  %*% t(X) ))

  return(d)
}
```

4. By using the RidgeOpt function from earlier, we were supposed to find the optimal values of the coefficients for the values $\lambda = 1$, 100 & 1000, and then use them to predict y. We prpopose the following solution by starting of with optimizing the parameters.

```
RidgeOpt <- function(lambda){

  opt_estimate <- optim(par=c(i_w = rep(1,17)), fn = Ridge,
  lambda = lambda, input_data = train, method = 'BFGS')

  beta_coef <- opt_estimate$par[1:16]    #since the 17th element
  is sigma
  return(beta_coef)
}
l_1 <- RidgeOpt(1)
l_100 <- RidgeOpt(100)
l_1000 <- RidgeOpt(1000)
```

With the optimized coefficients, we the created a function to predict y.

```
predicted_y <- function(coef_est, input_data){
  beta <- as.matrix(coef_est)
  X <- as.matrix(input_data[,-1])
  y_hat <- X %*% beta
  return(y_hat)
}
p1 <- predicted_y(coef_est = l_1, input_data = train)
p100 <- predicted_y(coef_est = l_100, input_data = train)
p1000 <- predicted_y(coef_est = l_1000, input_data = train)
```

Now, with the predicted values, we were supposed to find which λ values that gave the best prediction. The (atleast one of the) reason why MSE is an appropriate measure is because Ridge regression is *not* an unbiased estimator, and MSE captures both the variance and the bias of the estimate. This makes the MSE as a measure of quality, and hence it's a good measure to use when comparing. Below is the code used to compute the MSE for each value of λ for the training data.

```
MSE <- function(input_data, y_hat){
  n <- nrow(input_data)
  y <- input_data[,1]
  SSE <- sum( (y - y_hat)^2)

  M_S_E <- (1/n) * SSE
  return(M_S_E)
}
```

So each MSE estimates are the following, and as we can see, they are quite large.

	λ =1	λ = 100	λ = 1000
MSE [training set]	1.346055e+30	7.088023e+30	9.432514e+31
MSE [test set]	1.880297e+29	3.31029e+30	2.514068e+31

5. So as a complement to using MSE, we were here supposed to find the AIC-value of the Ridge regression model for every λ value. The AIC is defined as $2d - 2log(L)$ were d is the degrees of freedom, and L is the estimated likelihood function for the model. All we have to do is put together an AIC-function, from the functions we've defined above.

```
AIC <- function(input_data, lambda, i_w){

  beta <- as.matrix(i_w[1:16])
  sigma <- i_w[17]
  y <- as.matrix(input_data[1])
  n <- nrow(input_data)
  X <- as.matrix(input_data[-1])

  RSS <- sum((y - X %*% beta)^2)
  penalty <- lambda * sum(beta^2)

  l_Lhood <- -(n/2) * log(2*pi*sigma^2) - (1/(2*sigma^2)) *
  RSS + penalty
  df <- sum(diag( X %*% solve(t(X) %*% X + lambda *
  diag(ncol(X))) %*% t(X) ))

  A_I_C <- 2*df - 2*l_Lhood

  return(A_I_C)
}
```

The results from this function confirms the result from using the MSE metric, the optimal value for λ is 1, since it's returning the smallest AIC-value. So, finally, the reason to use the Akaike information criterion (AIC) is that it penalizes complex models. We want to minimize the AIC-value, but the more parameters the model incorporates, the higher the AIC. This is the reason it's used as a model selection metric.

Assignment 3: Linear Regression and the LASSO

```
# Load data
data = read.csv("data/tecator.csv")
# Split data into train and test set
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = data[id,]
```

```
test = data[-id ,]
```

1.

"*Assume that Fat can be modeled as a linear regression in which absorbance characteristics (Channels) are used as features. Report the underlying probabilistic model, fit the linear regression to the training data and estimate the training and test errors. Comment on the quality of fit and prediction and therefore on the quality of model.*"

Probabilistic Model:

$$\hat{y} = \beta_0 + \sum_{i=1}^{100} \beta_i x_i + \epsilon \sim N(\mu, \sigma^2)$$

This regression model return the following metrics.

	MSE	RMSE
Train Error	0.005709117	0.0755587
Test Error	722.4294	26.87805

Our loss-function (MSE) is showing big differences between the prediction of train and test set values. The MSE for the train set is very low while the MSE for the test set returns a very high value. Therefore, our model is overfitting on the training data.

2.

$$\underset{\beta}{\operatorname{argmin}} \left[\sum_{i=1}^{n} \left(Y_i - \beta_0 - \sum_{j=1}^{p} \beta_j X_{ji} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right]$$

3.

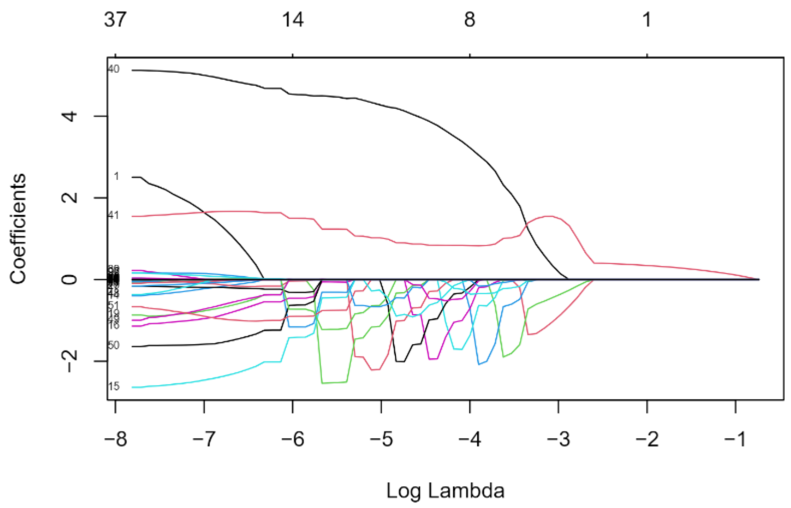


Figure 9: Plot shows the relationship between $log(\lambda)$ and value of coefficients in the $LASSO$ model.

We can see that a bigger $log(\lambda)$ evaluates to less non-zero coefficients. At $log(\lambda) = -8$ most of the coefficients are non-zero. Choosing $\lambda = 0$ returns the same results as using OLS for the model coefficients since no penalty is given. For each coefficient we can also see the influence (positive or negative) on our prediction depending on different $log(\lambda)$ values.

To find the λ value where our model contains only 3 non-zero coefficients we check the plot again. We see that the coefficient with the label '40' goes to 0 at around $log(\lambda) = -2.8$. We calculate $\lambda = e^{-2.8} \approx 0.06$ and get the approximate λ where we have 3 non-zero coefficients left.

4.

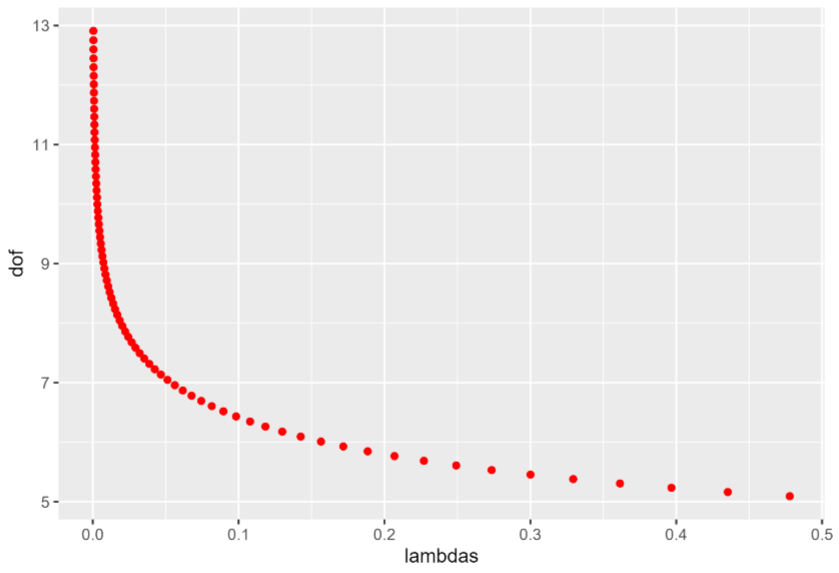


Figure 10: Plot shows the relationship between $log(\lambda)$ and the df.

The trend is as expected. If lambda is large, the parameters are heavily constrained and the degrees of freedom will effectively be lower, tending to 0 as $\lambda \to \infty$. For $\lambda \to 0$, we have p parameters. As in OLS all coefficients stay the same, no penalization to be found.

5.

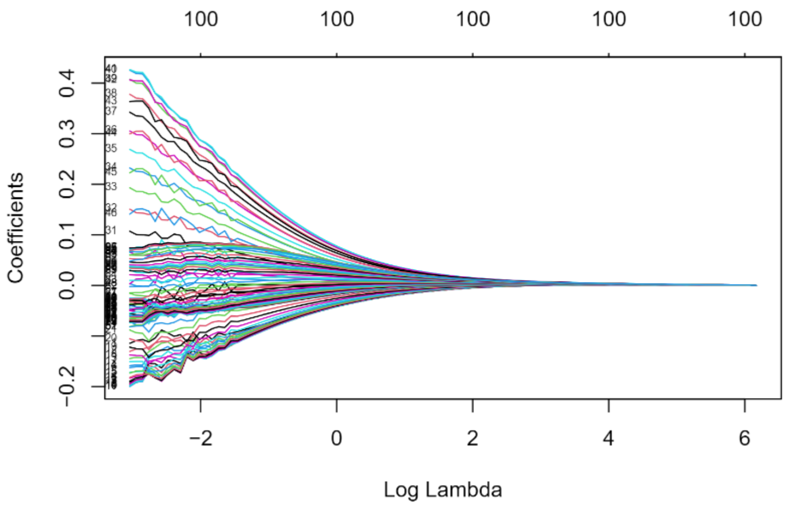


Figure 11

Both Ridge and LASSO regression are shrinkage methods that try to reduce the complexity of a model by shrinking or eliminating (LASSO only) its coefficients. For the Ridge regression the number of coefficients stays the same regardless of how big $log(\lambda)$ gets. For LASSO regression the number of coefficients decreases with increasing value of $log(\lambda)$. Looking at the LASSO plot it is easier to determine which coefficients are the most significant compared to looking at the Ridge plot.

6.

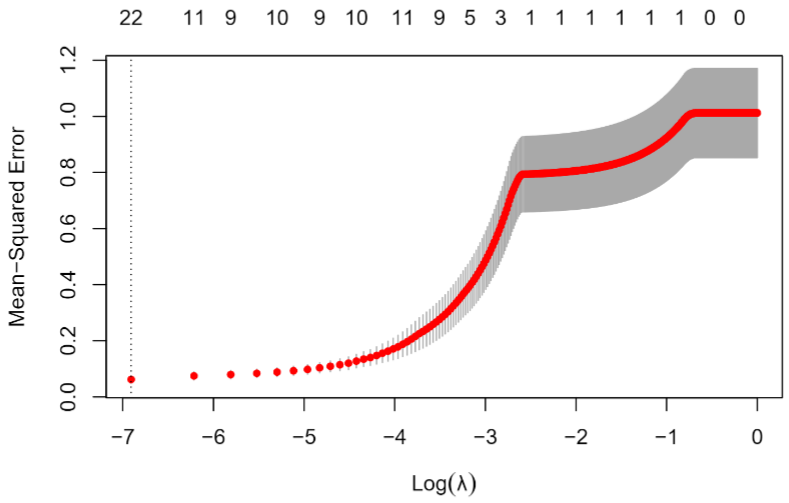


Figure 12

We can see a correlation between increasing $log(\lambda)$ and increasing MSE. Up until $log(\lambda) \approx -4$ we observe a flat rise of MSE. Afterwards the MSE grows very steeply until it reaches a plateau at $log(\lambda) \approx -2.5$. Then we observe a slow rise again until we reach $log(\lambda) \approx -1$. From this point onward the MSE stays at its maximum since there are no non-zero coefficients left.

```
## The optimal lambda:  0
## Number of variables choosen:  101
```

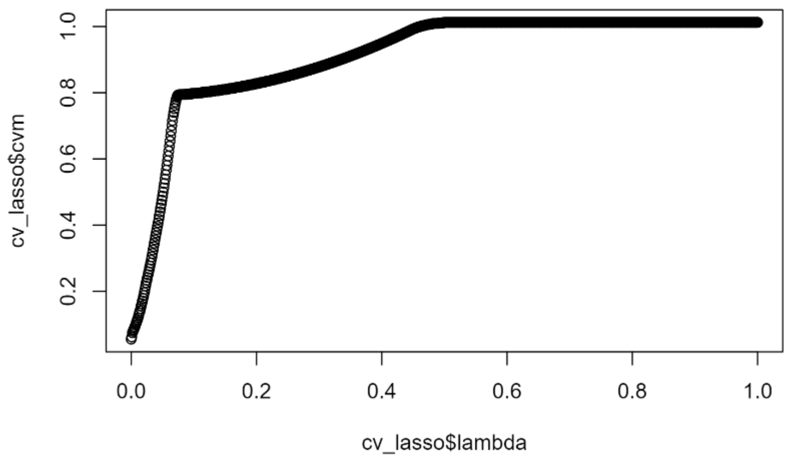


Figure 13

We check *cvm* for $log(\lambda) = -2$ and get a value of 0.8056608. $log(\lambda) = -2$ is significantly worse compared to our optimal λ with cv-score of 0.05526477.

```
## Coefficient of determination:  0.9889883
## MSE:  0.06737877
```

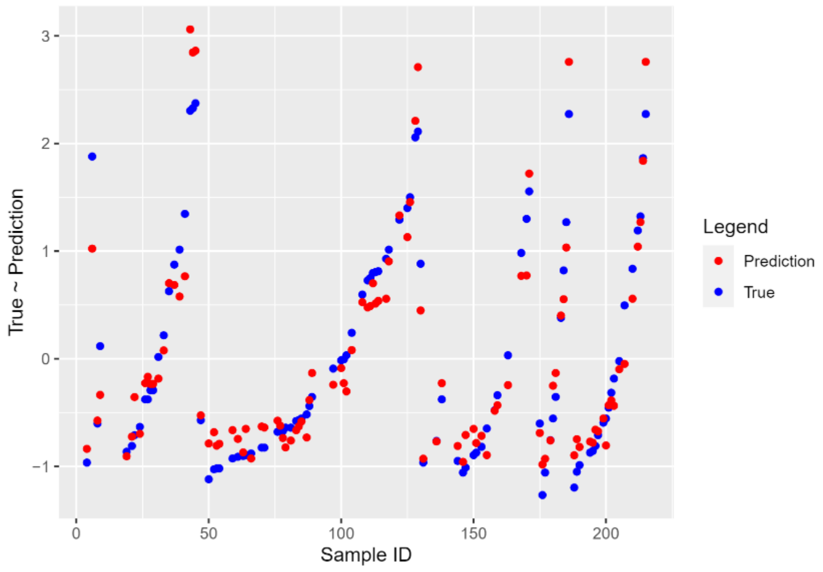


Figure 14

The MSE is better compared to the result of the overfitted linear model from exercise 1. Also, the coefficient of determination is near 1 which proves good predicting ability. Though, it can be seen from the plot that there are still outliers especially for larger *Fat* values.

7.

```
##
## MSE:  0.1231819
```

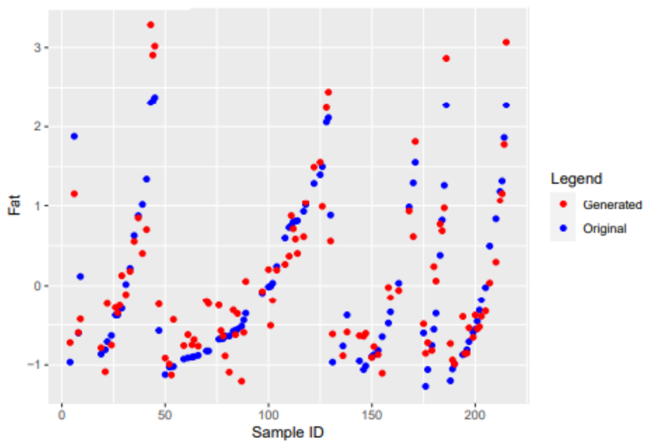


Figure 15

By looking at the plot we can see that the newly generated points are more or less in the same region as their original counterparts. Though, we can also observe bigger differences for some of the samples. The MSE between the original and generated target values is also bigger than what we saw for the test data in exercise 6.

Appendix: R Code

Assignment 1

```
setwd("C:/Users/Mounish/Desktop/ML/lab1")
dataset = read.csv("C:/Users/Mounish/Desktop/ML/Lab1/optdigits.csv")
dim(dataset)

set.seed(12345)
sample_train<- sample(seq_len(nrow(dataset)),
size = floor(0.50*nrow(dataset)))
sample_valid<- sample(seq_len(nrow(dataset)),
size = floor(0.25*nrow(dataset)))
sample_test <- sample(seq_len(nrow(dataset)),
size = floor(0.25*nrow(dataset)))

train      <- dataset[sample_train, ]
validation<- dataset[sample_valid, ]
test       <- dataset[sample_test, ]

dim(train)
dim(test)
dim(validation)

## Data is splitted into training, tests and validation

#Model kknn
library(kknn)

knn.fit <- kknn(as.factor(X0.26)~., train=train,test=train,k = 30,
                  kernel = "rectangular")

knn.fit_v <- kknn(as.factor(X0.26)~.,
train=validation,test=validation,k = 30,
                  kernel = "rectangular")

knn.fit_t <- kknn(as.factor(X0.26)~., train=test,test=test,k = 30,
                 kernel = "rectangular")
```

```
pred.knn <- fitted(knn.fit)
pred.knn_v <- fitted(knn.fit_v)
pred.knn_t <- fitted(knn.fit_t)


# Model is worked and trained by using the training data and also
there is no miss-classification rate while training


#Accuracy  of the model while using k value=30

cm=as.matrix(table(actual=validation$X0.26, Predicted = pred.knn_v))
accuracy=sum(diag(cm))/length(validation$X0.26)


#misscalculation
calc_class_err = function(actual, predicted) {
  mean(actual != predicted)
}
calc_class_err(validation$X0.26,pred.knn_v)

#Confusion Matrix

table(train$X0.26,as.factor(pred.knn)) # for train data

table(test$X0.26,as.factor(pred.knn_t)) # for test data

#Miss classification Rate

missclassrate=function(y,y_i)
  {
  n=length(y)
  v<-1-(sum(diag(table(y,y_i)))/n)
  return(v)
}


missclassrate(train$X0.26,as.factor(pred.knn)) # for training data
missclassrate(test$X0.26,as.factor(pred.knn_t)) # for test data

# Accuracy of the data
acc=function(x,y)
{
  n=length(x)
  ac=sum(diag(table(x,y)))/n
  return(ac)
}
acc(test$X0.26,as.factor(pred.knn_t))

# probability

#knn.fit$prob
v=data.frame(knn.fit$prob)
head(v)

estm_pb <- colnames(v)[apply(v, 1, which.max)]

v$y<-train$X0.26
head(v)
v$fit <- knn.fit$fitted

v$estm_pb <- estm_pb

###
y_8 <- v[v$y == 8,]

yhat_8 <- y_8[y_8$fit == 8,]

###
# Best
easy <- as.numeric(row.names(yhat_8[order(-yhat_8[,9]),][1:2,]))

# Worse
tougher <- as.numeric(row.names(yhat_8[order(yhat_8[,9]),][1:3,]))

#best
col=heat.colors(12)

heatmap(t(matrix(unlist(train[easy[1],-65]), nrow=8)),Colv = NA,
Rowv = NA, col=rev(heat.colors(12)))

heatmap(t(matrix(unlist(train[easy[2],-65]), nrow=8)), Colv = NA,
Rowv = NA, col=rev(heat.colors(12)))

#worst

heatmap(t(matrix(unlist(train[tougher[1],-65]), nrow=8)), Colv = NA,
Rowv = NA, col=rev(heat.colors(12)))

heatmap(t(matrix(unlist(train[tougher[2],-65]), nrow=8)), Colv = NA,
Rowv = NA, col=rev(heat.colors(12)))

heatmap(t(matrix(unlist(train[tougher[3],-65]), nrow=8)), Colv = NA,
Rowv = NA, col=rev(heat.colors(12)))

#u=as.vector(train$X0.26)
#train$X0.26
#n<-as.data.frame(cbind(v,u))
#dim(n)
#train$X0.26



#Find the optimal K value for KNN model for this dataset.(k=1 to 30)

k=1
k.optm=c()
y.optm=c()
for (i in 1:30){
  knn.fit <-kknn(as.factor(X0.26)~., train=train,test=train, k =
  i,kernel = "rectangular")

  knn.fit_v <-kknn(as.factor(X0.26)~.,
  train=validation,test=validation, k = i,kernel = "rectangular")

  ypred = fitted(knn.fit)

  vpred = fitted(knn.fit_v)

  k.optm[i] = 1-(sum(diag(as.matrix(table(Actual = train$X0.26,
  Predicted = ypred))))/nrow(train))

  y.optm[i] = 1-(sum(diag(as.matrix(table(Actual = validation$X0.26,
  Predicted = vpred))))/nrow(validation))

}
my.df  <- data.frame(K_Value = c(1:30), Training= c(k.optm),
Validation = c(y.optm))

plot3<-ggplot( ) +
 geom_line(aes(x=my.df$K_Value,y=my.df$Validation,colour="green")) +
  geom_line(aes(x=my.df$K_Value,y=my.df$Training,colour="red")) +
  ylab("Missclassification Rate ") + xlab("K_value") +
  scale_color_manual(name = "Missclassification Rate", labels =
  c("Validation ", "Training "), values =c("green", "red"))

optm_value_k <- which.min(y.optm - k.optm )

cat("calculated optimum value of K is: ", optm_value_k)

### Cross Entropy
rp <- function(i){
  n <- rep(0,10)
  n[i+1] <- 1
  return(I(n))
}
er<-c()
for (i in 1:30){
  knn.fit <-kknn(as.factor(X0.26)~., train=train,test=validation,
  k = i,kernel = "rectangular")

  x<- data.frame(knn.fit$prob)
  max_prob <- colnames(x)[apply(x ,1,which.max)]
  x$target <- validation$X0.26
  x$fit <- knn.fit$fitted
  x$max_prob <- max_prob
  x$binary <- (lapply(as.numeric(x$target)-1, rp))

  #cross entropy loss
  for (j in 1:nrow(x)){
    x[j, "cross_entropy"] <- -sum(log(x[j,1:10]+1e-15)* x[[j,
    "binary"]])
  }

  er[i] <- mean(x$cross_entropy)
}

er

df<-data.frame(cross_entropy=er,k_Value=c(1:30))

plot4<-ggplot(df,aes(x=k_Value,y=cross_entropy,col="red"  ))
 + geom_line()+ggtitle("Empirical Risk Of calculating K_value")

best_optm_k_value <-min(er)

cat("calculated optimum value of K by using the cross entropy
function is: ", best_optm_k_value)
```

R Code for Assignment 3

```
library(ggplot2)
library(glmnet)
library(psych)
library(tidyr)

#Load data
data = read.csv("data/tecator.csv")

# Split data into train and test set
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = data[id,]
test = data[-id,]


# Fitting linear regression model

predictors <- paste("Channel", 1:100, sep="")
fmla <- as.formula(paste("Fat ~ ", paste(predictors,
collapse = "+")))


fit <- lm(fmla, data=train)
summary(fit)

# Estimating training and test error

train_predictions_lm <- predict(fit, train)
train_mse <- mean((train$Fat - train_predictions_lm)^2)
train_rmse <- sqrt(mean((train$Fat - train_predictions_lm)^2))


test_predictions_lm <- predict(fit, test)
test_mse <- mean((test$Fat - test_predictions_lm)^2)
test_rmse <- sqrt(mean((test$Fat - test_predictions_lm)^2))


cat("Train error\nMSE : ", train_mse, "\nRMSE ", train_rmse,
"\n\nTest error\nMSE: ", test_mse, "\nRMSE: ", test_rmse)

covariates = scale(train[,2:101])
response = scale(train$Fat)


# alpha = 1 to choose 'Lasso'
set.seed(12345)
model_lasso = glmnet(as.matrix(covariates), response, alpha=1,
family="gaussian")
plot(model_lasso, xvar="lambda", label=T)

calc_dof <- function(lambda){
  ld <- lambda * diag(ncol(covariates))
  H <- covariates %*% solve(t(covariates) %*% covariates + ld) %*%
  t(covariates)
  dof <- tr(H)
}

dof = c()
lambdas = model_lasso$lambda
for(i in 1:length(lambdas)){
  dof[i]= calc_dof(lambdas[i])
}

ggplot(data=NULL, aes(lambdas, dof)) + geom_point(color="red")

# aplha = 0 to choose 'Ridge'
set.seed(12345)
model_ridge = glmnet(as.matrix(covariates), response, alpha=0,
family = "gaussian")
plot(model_ridge, xvar="lambda", label=T)



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

optimal_lambda = cv_lasso$lambda.min
cat("The optimal lambda: ", optimal_lambda, "\nNumber of variables
choose: ", length(coef(cv_lasso,)))
lambda_to_compre = round(exp(-2),3)
cvm = cv_lasso$cvm
lambda = cv_lasso$lambda
plot(cv_lasso$lambda, cv_lasso$cvm)



lambda_cvm = cbind(lambda, cvm)
ind_lambda_to_compare = which(lambda_cvm[,1]==lambda_to_compre,
arr.ind = T)
cvm_lambda_to_compare = lambda_cvm[ind_lambda_to_compare]
print(cvm_lambda_to_compare)
test_x = scale(test[,2:101])
test_y = scale(test$Fat)


model_lasso_optimal = glmnet(as.matrix(covariates), response,
alpha = 1, lambda = optimal_lambda)
test_predictions_lasso = predict(model_lasso_optimal, newx=test_x,
type="response")
```

```
determination_coefficient =
sum((test_predictions_lasso-mean(test_y))^2)/
sum((test_y-mean(test_y))^2)
test_mse_lasso_optimal = mean((test_predictions_lasso-test_y)^2)

cat("Coefficient of determination: ", determination_coefficient,
"\nMSE: ", test_mse_lasso_optimal)

true_pred_df = cbind(test[c(1)], scale(test[c(102)]),
test_predictions_lasso)
colnames(true_pred_df)[2] = "True"
colnames(true_pred_df)[3] = "Prediction"


true_pred_df %>%
  gather(key, value, -Sample) %>%
  ggplot(aes(Sample, value)) + geom_point(aes(color = key)) +
  ylab("True ~ Prediction") +
  xlab("Sample ID") +
  scale_colour_manual(name="Legend", values=c("red", "blue"))
calc_sigma <- function(model, covariates, response){
  betas = as.vector((coef(model))[-1,])
  residuals = response - (covariates %*% betas)
  sigma = sd(residuals)
  return(sigma)
}

sigma = calc_sigma(model_lasso_optimal, covariates, response)


set.seed(12345)
generated_data = rnorm(length(test_y), test_predictions_lasso,
sigma)

determination_coefficient = sum((generated_data-mean(test_y))^2)/
sum((generated_data-mean(test_y))^2)
test_mse_lasso_optimal = mean((generated_data-test_y)^2)

cat("Coefficient of determination: ", determination_coefficient,
"\nMSE: ", test_mse_lasso_optimal)
```

```
true_pred_df = cbind(test[c(1)], scale(test[c(1)]), generated_data)
colnames(true_pred_df)[2] = "True"
colnames(true_pred_df)[3] = "Generated"

true_pred_df %>%
  gather(key, value, -Sample) %>%
  ggplot(aes(Sample, value)) + geom_point(aes(color=key)) +
  ylab("True ~ Prediction") +
  xlab("Sample ID") +
  scale_colour_manual(name="Legend", values=c("red", "blue"))
```