Computer lab 1 block 1

Martynas Lukosevicius, Alejo Perez Gomez, Shwetha Vandagadde Chandramouly

16/11/2020

Statement of Contribution

- Assignment 1 - Alejo Perez Gomez
- Assignment 2 - Martynas Lukosevicius
- Assignment 3 - Shwetha Vandagadde Chandramouly

Assignment 1

1.

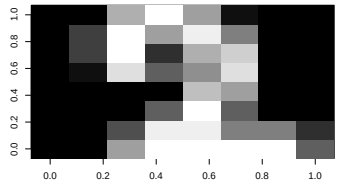
Import the data into R and divide it into training, validation and test sets (50%/25%/25%) by using the partitioning principle specified in the lecture slides.

```
digits <- read.csv(file = "optdigits.csv")
digits[, ncol(digits)] <- data.frame(sapply(digits[,
    ncol(digits)], as.character), stringsAsFactors = TRUE)
names(digits) <- c(seq.int(ncol(digits) -
    1), "target")
```

Train/Valation/Test Division

```
set.seed(12345)

n = dim(digits)[1]
set.seed(12345)

id = sample(1:n, floor(n * 0.5))
dfTraining = digits[id, ]

id1 = setdiff(1:n, id)
set.seed(12345)

id2 = sample(id1, floor(n * 0.25))
dfValidation = digits[id2, ]

id3 = setdiff(id1, id2)
dfTest = digits[id3, ]
```

Plotting one observation




```
## [1] 2
```

2.

Use training data to fit 30-nearest neighbor classifier with function kknn() and kernel=”rectangular” from package kknn and estimate - Confusion matrices for the training and test data (use table()) - Misclassification errors for the training and test data Comment on the quality of predictions for different digits and on the overall prediction quality.

Let’s fit the Model and predict results for Testing Set.

This will be our Confusion Matrix.

```
kknn_model <- kknn(dfTraining[["target"]] ~
    ., dfTraining, dfTest, k = 30, kernel = "rectangular")

fit <- kknn_model$fitted.values
CM_test <- table(dfTest$target, fit)
CM_test
```

```
##    fit
##      0  1  2  3  4  5  6  7  8  9
##   0 98  0  0  0  0  0  0  0  0  0
##   1  0 92  3  0  0  0  0  0  0  2
##   2  0  0 93  1  0  0  0  0  1  0
##   3  0  0  0 95  0  0  0  2  1  0
##   4  1  0  0  0 89  0  1  5  1  3
##   5  0  1  0  0  0 81  0  0  0  5
##   6  0  0  0  0  0  0 94  0  0  0
##   7  0  2  0  0  0  0  0 92  1  0
##   8  0  3  0  1  0  0  1  0 86  0
##   9  0  0  0  3  0  0  0  2  0 96
```

```
accuracy_test <- (sum(diag(CM_test)))/sum(CM_test)
cat("Misclassification error in testing set is",
    1 - accuracy_test)
```

```
## Misclassification error in testing set is 0.041841
```

Let’s fit the Model and predict results for Training Set. First we obtain the Confusion Matrix.

```
kknn_model <- kknn(dfTraining[["target"]] ~
    ., dfTraining, dfTraining, k = 30, kernel = "rectangular")
fit <- kknn_model$fitted.values
CM_train <- table(dfTraining$target, fit)
CM_train
```

```
##    fit
##       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   0 171   0   0   0   0   1   1   0
##   3   0   0   0 198   0   1   0   1   0   0
##   4   0   1   0   0 168   0   1   6   1   2
##   5   0   0   0   0   0 186   0   2   0   9
##   6   0   0   0   0   0   0 200   0   0   0
##   7   0   1   0   1   0   0   0 193   0   0
##   8   0   7   0   1   0   0   1   0 196   0
##   9   0   3   0   2   2   0   0   2   3 184
```

```
accuracy_train <- (sum(diag(CM_train)))/sum(CM_train)
cat("Misclassification error in training set is",
    1 - accuracy_train)
```

```
## Misclassification error in training set is 0.03349032
```

As seen in the results, misclassification test error surpasses training error which shows absence of underfitting. Classification behavior looks quite good since the error of misclassification doesn't exceed 0.05 in neither of the cases.

3.

Find any 2 cases of digit "8" in the training data which were easiest to classify and 3 cases that were hardest to classify (i.e. having highest and lowest probabilities of the correct class). Reshape features for each of these cases as matrix 8x8 and visualize the corresponding digits (by using e.g. heatmap() function with parameters Colv=NA and Rowv=NA) and comment on whether these cases seem to be hard or easy to recognize visually.

First we get the probabilities of predictions, joining them with the actual category and the prediction made.

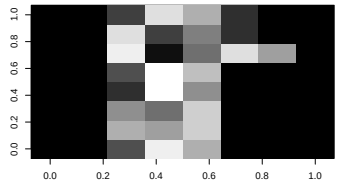
```
prob_df <- kknn_model$prob
max_prob <- colnames(prob_df)[apply(prob_df,
    1, which.max)]
train_probs <- data.frame(prob_df)
train_probs$actual <- dfTraining$target
train_probs$pred <- kknn_model$fitted.values
train_probs$max_prob <- max_prob
```

Let us subset the actual 8s and ordering by probabilities.

```
actual_8s <- train_probs[train_probs["actual"] ==
    8, ]
predicted_actual_8s <- actual_8s[actual_8s["pred"] ==
    8, ]
```

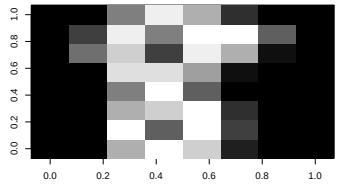
8 predicted correctly with high certainty and plots.

```
best_8_indexes <- as.numeric(row.names(predicted_actual_8s[order(-predicted_actual_8s[,
    9]), ][1:2, ]))

plot_observation(best_8_indexes[1], dfTraining)
```




```
## [1] 8
```

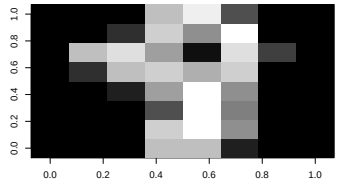
```
plot_observation(best_8_indexes[2], dfTraining)
```




```
## [1] 8
```

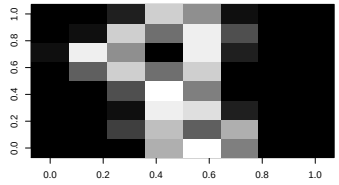
8 predicted correctly with low certainty and plots.

```
worst_8_indexes <- as.numeric(row.names(predicted_actual_8s[order(predicted_actual_8s[,
    9]), ][1:3, ]))
plot_observation(worst_8_indexes[1], dfTraining)
```



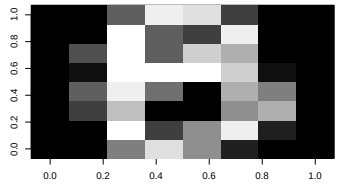

```
## [1] 8
```

```
plot_observation(worst_8_indexes[2], dfTraining)
```




```
## [1] 8
```

```
plot_observation(worst_8_indexes[3], dfTraining)
```




```
## [1] 8
```

Undoubtedly, the classified observations as 8s with less certainty are visually harder to tell apart from other numbers as the trace is pretty unclear. This doesn't happen to the correctly classified with high certainty.

4.

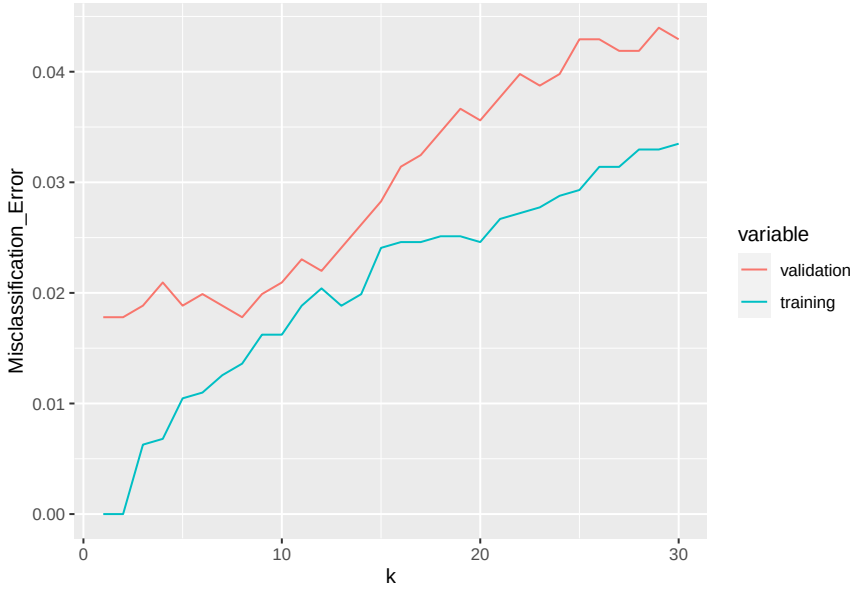
Fit a K-nearest neighbor classifiers to the training data for different values of $K = 1, 2, \ldots, 30$ and plot the dependence of the training and validation misclassification errors on the value of K (in the same plot). How does the model complexity change when K increases and how does it affect the training and validation errors? Report the optimal K according to this plot. Discuss this plot from the perspective of bias-variance tradeoff. Finally, estimate the test error for the model having the optimal K, compare it with the training and validation errors and make necessary conclusions regarding the model quality.

Let's fit a model for each k in the range.

```
KKNN_misclassification_training <- c()
KKNN_misclassification_validation <- c()

for (k in 1:30) {

    # Fit for dfValidation and dfTraining

    KKNN_model_valid <- kknn(dfTraining[["target"]] ~
        ., dfTraining, dfValidation, k = k,
        kernel = "rectangular")
    KKNN_model_train <- kknn(dfTraining[["target"]] ~
        ., dfTraining, dfTraining, k = k,
        kernel = "rectangular")

    # Calculation of Accuracies

    fit_train <- KKNN_model_train$fitted.values
    CM_train <- table(dfTraining$target,
        fit_train)
    KKNN_misclassification_training[k] <- 1 -
        (sum(diag(CM_train)))/sum(CM_train)

    fit_valid <- KKNN_model_valid$fitted.values
    CM_valid <- table(dfValidation$target,
        fit_valid)
    KKNN_misclassification_validation[k] <- 1 -
        (sum(diag(CM_valid)))/sum(CM_valid)


}
```

Plot of Accuracies.

```
metrics <- data.frame(KKNN_misclassification_validation,
    KKNN_misclassification_training, 1:30)
names(metrics) <- c("validation", "training",
    "k")


metricsMelted <- melt(metrics, id.var = "k")
names(metricsMelted)[3] <- "Misclassification_Error"
ggplot(metricsMelted, aes(x = k, y = Misclassification_Error,
    col = variable)) + geom_line()
```




```
best_K <- (which(KKNN_misclassification_validation ==
    min(KKNN_misclassification_validation)))
cat("The best performing K parameter are k =",
    as.character(best_K))
```

```
## The best performing K parameter are k = 1 2 8
```

The complexity of the model increases as K increases. For instance with $K = 1$ the error in training data is minimum resulting in minimum bias, whereas the variance is great as there are higher chances of getting error in testing set (generalizes worse, overfits). The training data is getting memorised at lowest K's, resulting in high variance. With higher K, training error increases (bias increases) but testing error decreases (decreasing variance) up to a point (optimal K, the sweet spot). Luckily, our optimal K (lowest error produced in validation set) coincides with a span where low variance and bias is attained. Before this spot, variance is decreasing and it starts increasing when it gets it passed. In regards to bias, the relationship with k is not that smooth, roughly it decreases until it reaches optimal K end then starts to increase again.

Testing of optimal K with Testing set. This is the Confusion Matrix for the optimal K in the Testing set.

```
kknn_model <- kknn(dfTraining[["target"]] ~
    ., dfTraining, dfTest, k = max(best_K),
    kernel = "rectangular")

fit <- fitted(kknn_model)
CM_test <- table(dfTest$target, fit)
```

```
CM_test
```

```
##    fit
##       0  1  2  3  4  5  6  7  8  9
##    0 97  0  0  0  0  0  1  0  0  0
##    1  0 93  3  0  0  0  0  0  0  1
##    2  0  1 93  0  0  0  0  1  0  0
##    3  0  0  0 95  0  0  0  2  1  0
##    4  1  0  0  0 92  0  1  3  0  3
##    5  0  0  0  0  0 85  0  0  0  2
##    6  0  0  0  0  0  0 94  0  0  0
##    7  0  1  0  0  0  0  0 92  1  1
##    8  0  3  0  0  0  0  0  0 88  0
##    9  0  0  0  3  1  0  0  0  0 97
```

```
accuracy_test <- 1 - (sum(diag(CM_test)))/sum(CM_test)
cat("Misclassification error is", accuracy_test)
```

```
## Misclassification error is 0.03138075
```

The error obtained in testing set classification is comparable to the error got among the best performing K at validation test classification. It leads us to thing the model is generalising properly.

5.

Fit K-nearest neighbor classifiers to training data for different values of $K = 1, 2, \ldots, 30$, compute the empirical risk for the validation data as cross-entropy (when computing log of probabilities add a small constant within log, e.g. 1e-15, to avoid numerical problems) and plot the dependence of the empirical risk on the value of K. What is the optimal K value here? Why might the cross-entropy be a more suitable choice of the empirical risk function than the misclassification error for this problem?

First we design our hot encode function to encode our target.

```
hot_encode <- function(i) {
    v <- rep(0, 10)
    v[i + 1] <- 1
    return(I(v))
}

empirical_risk <- c()
```

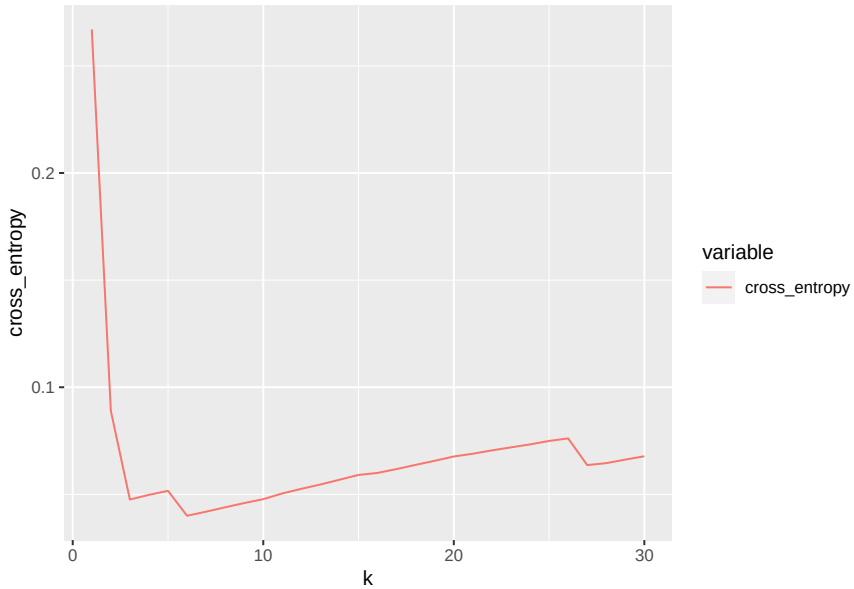
Let's fit a model for each k in the range. Then we will predict for the validation set and store the cross-entropy errors.

```
for (k in 1:30) {

    KKNN_model_valid <- kknn(dfTraining[["target"]] ~
        ., dfTraining, dfValidation, k = k,
        kernel = "rectangular")

    # Extraction of probabilities, we build a
    # df with the target hot encoded to add a
    # column of cross-entropy loss

    prob_df <- KKNN_model_valid$prob
    max_prob <- colnames(prob_df)[apply(prob_df,
        1, which.max)]
```

```
    valid_probs <- data.frame(prob_df)
    valid_probs$actual <- dfValidation$target
    valid_probs$pred <- KKNN_model_valid$fitted.values
    valid_probs$max_prob <- max_prob

    valid_probs$coded <- (lapply(as.numeric(valid_probs$actual) -
        1, hot_encode))

    # column of cross-entropy loss
    for (row in 1:nrow(valid_probs)) {
        valid_probs[row, "cross_entropy"] <- -sum(log(valid_probs[row,
            1:10] + 1e-15, base = 10) * valid_probs[[row,
            "coded"]])
    }

    empirical_risk[k] <- mean(valid_probs$cross_entropy)

}
```

Plot of Empirical Risk for each K.

```
metrics <- data.frame(empirical_risk, 1:30)
names(metrics) <- c("cross_entropy", "k")

metricsMelted <- melt(metrics, id.var = "k")

names(metricsMelted)[3] <- "cross_entropy"
ggplot(metricsMelted, aes(x = k, y = cross_entropy,
    col = variable)) + geom_line()
```




```
best_K <- which(empirical_risk == min(empirical_risk))
cat("The best performing K parameter is k =",
    as.character(best_K))
```

```
## The best performing K parameter is k = 6
```

As a conclusion, the choice of cross-entropy error is such a sensible procedure as there is a penalization involved as long as prediction probability towards the wrong class is given. Therefore, we are accounting for not only the actual target but also penalizing if uncertainty is present when predicting in some percentage for some classes.

Assignment 2

1.

Model:
$\hat{y} \sim N(w_0 + w^t X, \sigma^2)$ where $w \sim N(0, \frac{\sigma^2}{\lambda})$

- w - weights
- X - features
- λ - regularization penalty
- σ - standard deviation

Bayes' theorem:

$$p(w|d) \propto (D|w)p(w)$$

Probability of y:

$$p(y|X, w, \sigma) = N(w_0 + w^t X_i, \sigma^2)$$

Prior probability:

$$p(w) \sim N(0, \frac{\sigma^2}{\lambda}) = \frac{1}{\sqrt{2\pi\frac{\sigma^2}{\lambda}}} e^{-\frac{(w)^2}{2\frac{\sigma^2}{\lambda}}}$$

Likelihood:

$$p(D|w) = \prod_{i=1}^{n} N(w_0 + w^t X_i, \sigma^2)$$

$$p(D|w) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y - X_i w)^2}{2\sigma^2}} = \frac{1}{(\sqrt{2\pi\sigma^2})^n} e^{-\sum_{i=1}^{n} \frac{(y_i - w^T X_i)^2}{2\sigma^2}}$$

Model:

$$p(w|D) \propto \frac{1}{(\sqrt{2\pi\sigma^2})^n} e^{-\sum_{i=1}^{n} \frac{(y_i - w^T X_i)^2}{2\sigma^2}} * \frac{1}{(\sqrt{2\pi\frac{\sigma^2}{\lambda}})^j} e^{-\frac{\sum^j (w)^2}{2\frac{\sigma^2}{\lambda}}} = \frac{\sqrt{\lambda}}{(\sqrt{2\pi\sigma^2})^{n+1}} e^{-\frac{\sum_{i=1}^{n} (y_i - w^T X_i)^2 + \lambda \sum^j w^2}{2\sigma^2}}$$

2.

Scaling data:

```
library(readr)
parkinsons <- read_csv("parkinsons.csv")
cleaned <- parkinsons[c(-1:-4, -6)]
parkinsons.scaled <- scale(cleaned)
```

```r
set.seed(12345)
n <- dim(parkinsons.scaled)[1]
id = sample(1:n, floor(n * 0.6))
train = parkinsons.scaled[id, ]
test = parkinsons.scaled[-id, ]
```

3.

As we will be optimizing σ and w, likelihood and prior should contain all σ, even if data is scaled (so it means that $\sigma \sim 1$):

$$log(posterior) = log(likelihood * prior) = log(likelihood) + log(prior)$$

a)

loglikelihood:

$$log(p(D|w)) = -\frac{n}{2} log(2\pi\sigma^2) - \sum_{i=1}^{n} \frac{(y_i - w^T X_i)^2}{2\sigma^2}$$

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

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

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

b)

Ridge part $\sim$ log prior, where $\tau = \frac{\sigma^2}{\lambda}$, and p number of weights:

$$log(prior) = -\frac{p}{2} log(2\pi\tau) - \frac{\sum^{p} (w_i)^2}{2\tau}$$

function returns $-log(posterior)$

```r
ridge <- function(x, lambda) {
    w <- x[1:16]
    sigma <- x[17]
    p <- length(w)
    tau <- sigma^2/lambda
    part1 <- (-1/2) * log(2 * pi * tau)
    part2 <- (w^2)/(2 * tau)
    ridge <- part1 - part2
    # ridge <- lambda * (w %*% w)
    return(-(loglikelihood(w, sigma) + sum(ridge)))
    # return(sum(ridge))
}
```

c)

function to predict weights (w) and σ

```
ridgeOpt <- function(lambda) {
    x <- c(rep(1, 16), 1)
    a <- optim(x, ridge, method = "BFGS",
        lambda = lambda)
    w <- a$par[1:16]
    sigma <- a$par[17]
    return(a)
}
```

d)

function to calculate degrees of freedom

```
DF <- function(lambda) {
    m <- as.matrix(train[, -1])
    part1 <- t(m) %*% m + (lambda * diag(16))
    part2 <- m %*% solve(part1) %*% t(m)
    return(sum(diag(part2)))
}
```

4.

	MSE train	MSE test
lambda = 1	0.8747959	0.9332394
lambda = 100	0.8837090	0.9254054
lambda = 1000	0.8982641	0.9301081

$\lambda = 100$ is better than others because MSE for train set and for test set is lowest. MSE is good loss function because it comes from model's MLE.

5.

	AIC
lambda = 1	9559.988
lambda = 100	9634.181
lambda = 1000	10257.767

The optimal model is with lowest AIC score, in this case its a model with $\lambda = 1$. Hold out method requires to divide data into 3 parts: train,validation, test, which wont allow to use all data for training, its not the case with AIC, where its use on training + validation data

Assignment 3

1.

Here we are assuming that fat can be modeled as linear regression with channels as features.

Underlying probalistic model is :

$$\hat{y} = \beta_0 + \sum_{i=1}^{100} \beta_i * x_i + \epsilon, \epsilon \sim \mathsf{N}(0, \sigma^2)$$

y = *dependent variable (Fat)* x_i = features (channel1 to channel100) β_0 = *y-intercept* β_i = slope coefficient for each feature ϵ = model error term

Fitting linear regression model to training data

```
X_train = as.matrix(train[2:101])
Y_train = as.matrix(train$Fat)
fit_train = lm(Fat ~ ., data = train[2:102])  #fitting linear regression to training data
y_hat_train = predict(fit_train)
```

Error for training and test data :

```
## MSE for training data is  0.005709117
```

```
## MAE for training data is  0.05503807
```

```
##
## MSE for testing data  722.4294
```

```
## MAE for testing data is  12.20608
```

We can see that error(MSE and MAE) for the train data was low , but when we try to fit the model to test data error increases. This is because of the overfitting on training data due to large number of features in our model. Quality of fit : Overfit. Prediction quality is not good for test data , this is evident by observing increase in MSE and MAE in test data.

Over all quality of the model is not good , as there is large number of features here , regularization is required.

2.

In this case as the number of input features are high, high degree of polynomials lead to overfitting.

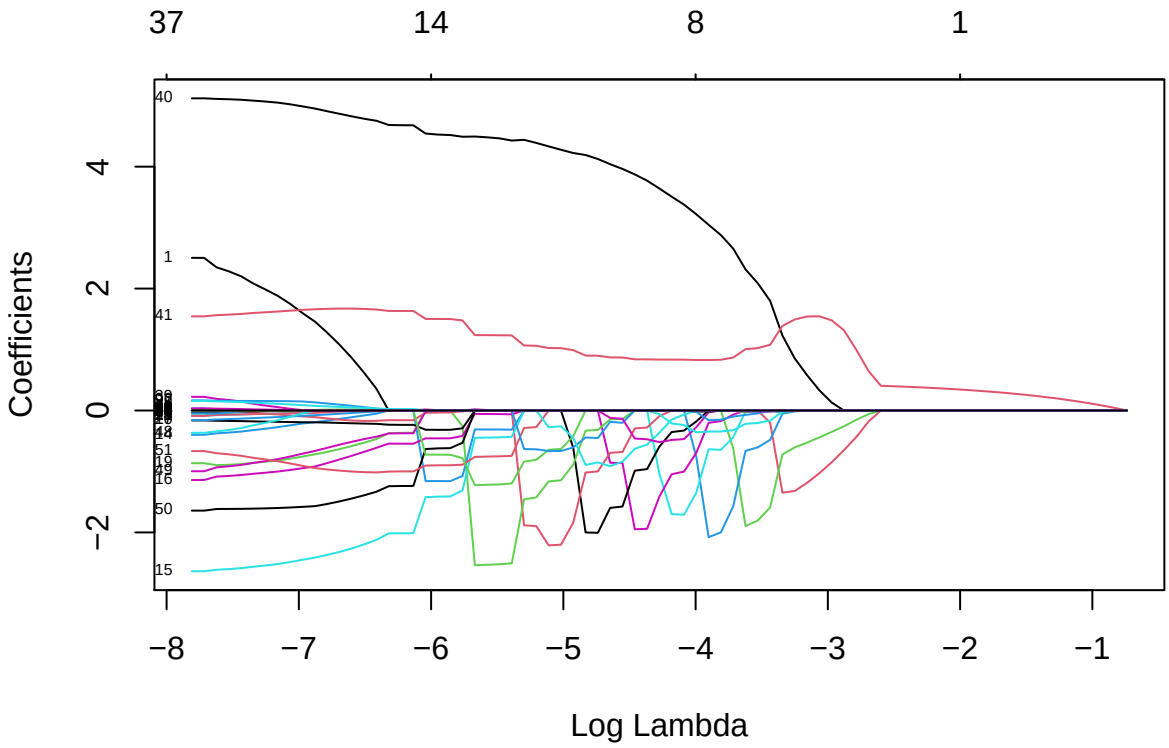
In LASSO , we try to choose the value of β such that the following loss function is minimized

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

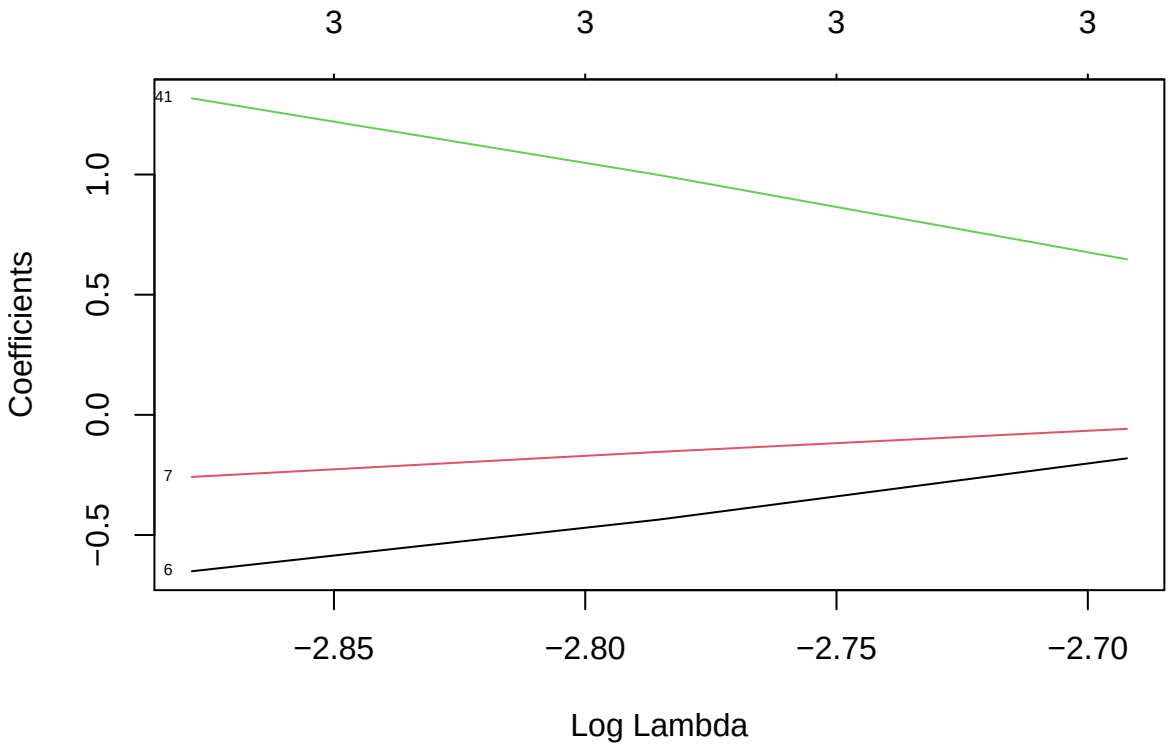
3.

Fitting LASSO regression model to training data

```
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 = T)
```



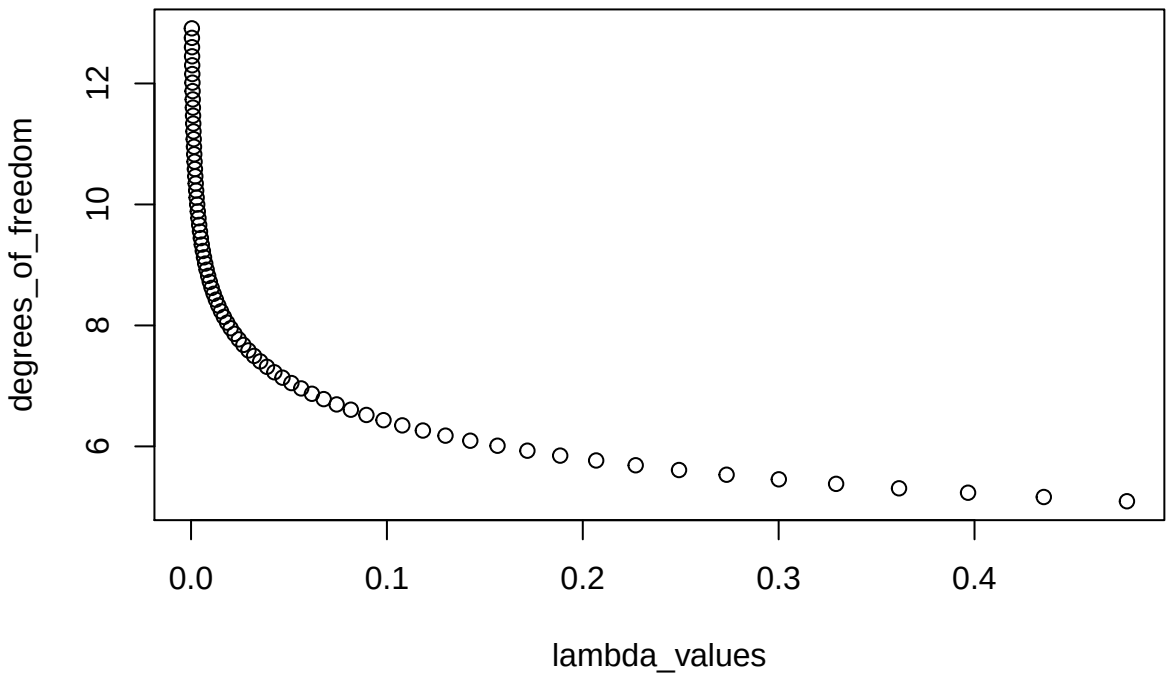
To choose lambda that has only 3 features , we can perform cross validation and check for lambda values which has only 3 variables.




```
## Lambda values that give 3 variables are : 0.06773096 0.06171393 0.05623144
```

4.

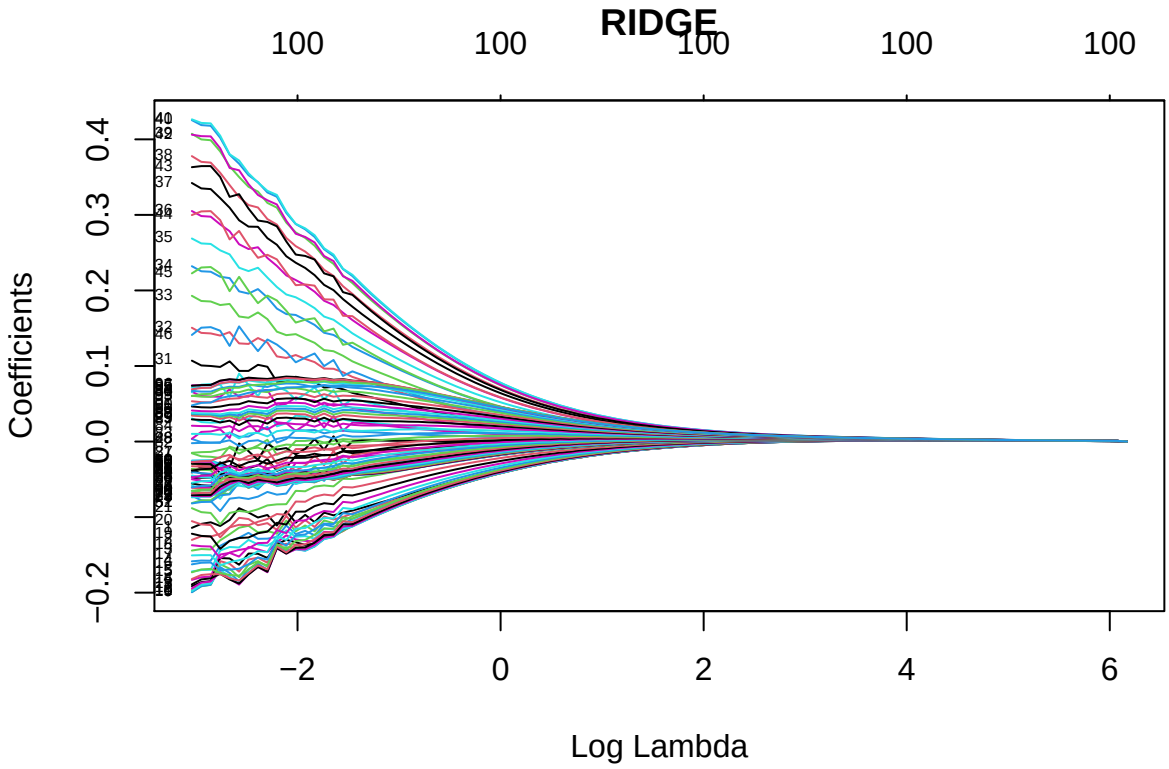
Dependence of degrees of freedom on panalty factor



Here as expected we can see that degrees of freedom decreases as penalty factor increases. When lambda is 0 , df is infinity and when df is 0, lambda is infinity.

5.

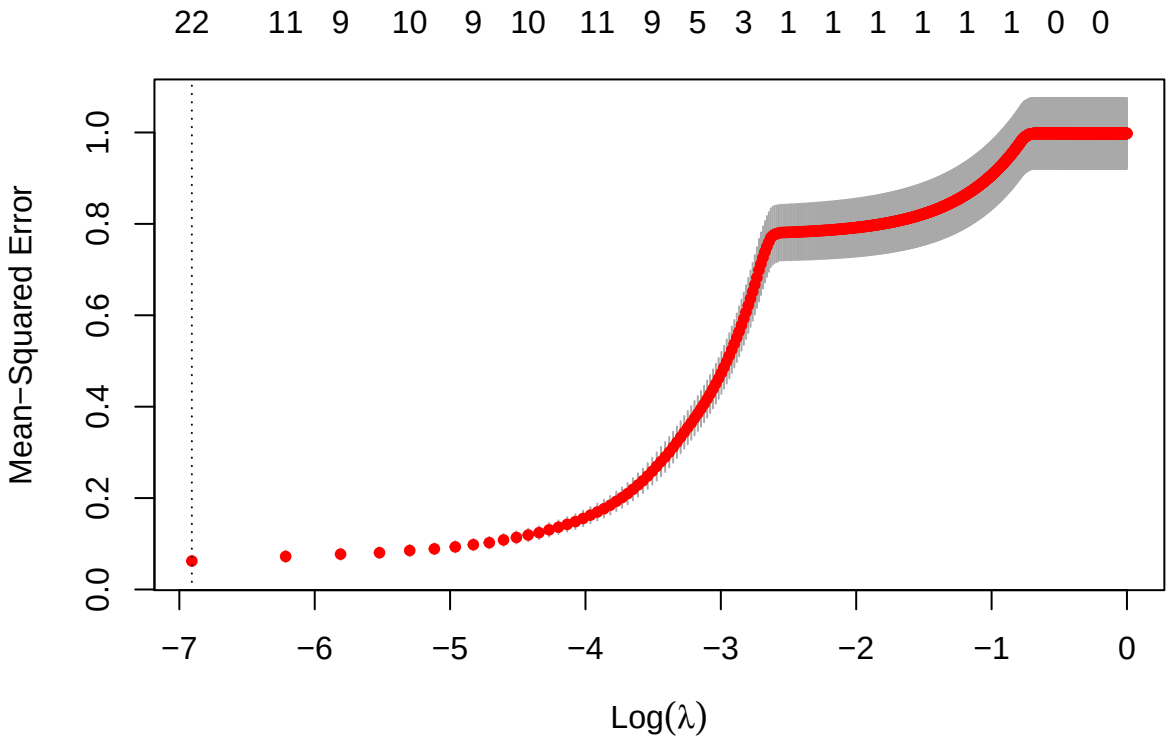
Fitting ridge model

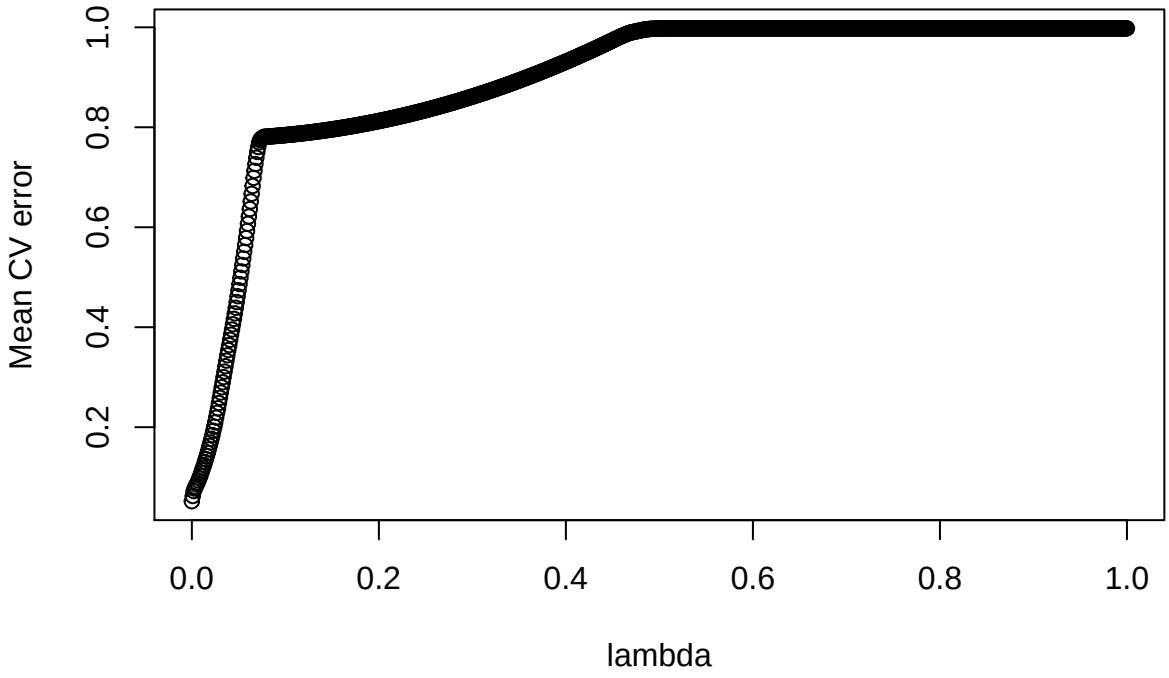


In both lasso and ridge we can see that the value of coefficents decreases with increase in lambda values, But only in case of lasso , coeffients converge to 0 with increase in lambda.

6.

To find optimal lambda, we use cross validation.


22 11 9 10 9 10 11 9 5 3 1 1 1 1 1 1 0 0
Mean−Squared Error
0.0
0.2
0.4
0.6
0.8
1.0
−7
−6
−5
−4
−3
−2
−1
0
Log(λ)




```
## Optimal lambda is : 0
```

Relationship between log(lambda) and MSE: As the lambda increases, the value of log(lambda) also increases, and we can see from the plot above that the MSE also increases with increasing lambda. Best lambda is given as zero for the above model , this makes sense as we can see from cvm(mean cross-validated error) vs lambda plot , the error is minimum when lambda is zero.

Fitting the model for 0 and log_lambda = -2 for comparison.

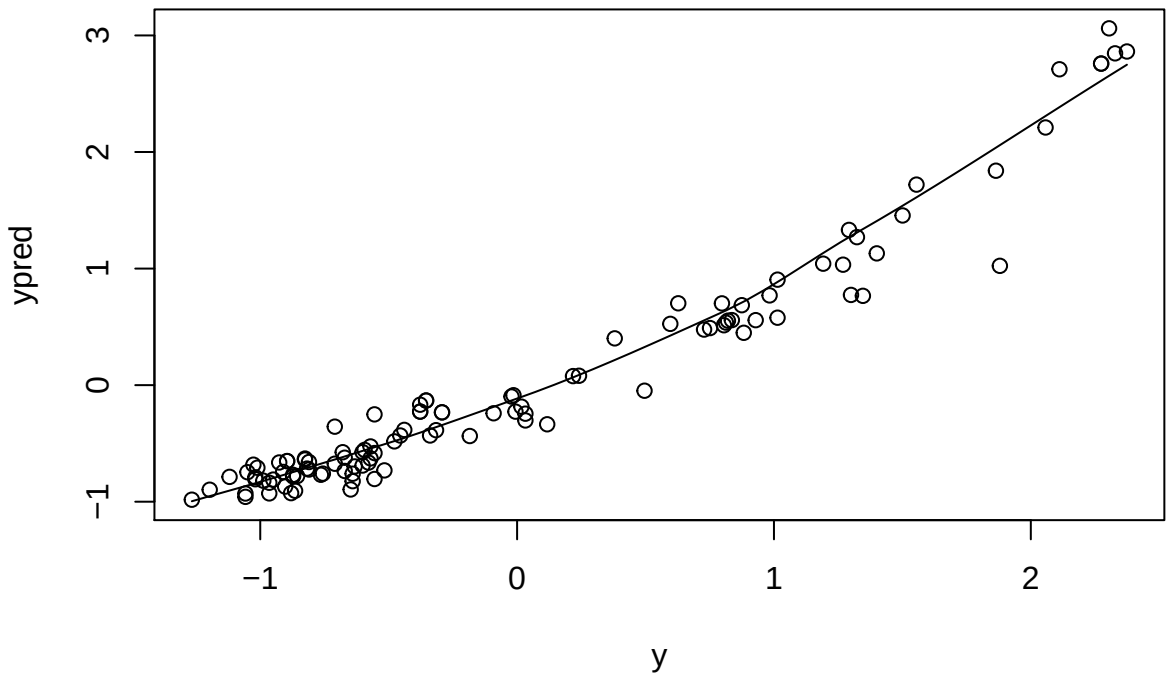
```
m = cv.glmnet(as.matrix(covariates), response,
    alpha = 1, family = "gaussian", lambda = c(model_l$lambda.min,
        0.1353353))
```

```
## CV Error when lambda is  0.1353353  = 0.8133424
```

```
##  and CV Error when lambda is  0  = 0.06861109
```

Log(bestlambda) ie log(0) is -Inf, from the plot of MSE vrs log(lambda) we can see that MSE is lowest for log(lambda) = -7(in the plot this is the lowest negetive number) for when compared to log(lambda) = -2. Also in lambda vrs Mean Cv error plot , we can see that for lambda = 0.1353353 (log(lambda)=-2 , so lambda = exp(-2) = 0.1353353) error is higher than error for lambda = 0.

Plot for Y vrs predicted Y for test data using optimal lambda

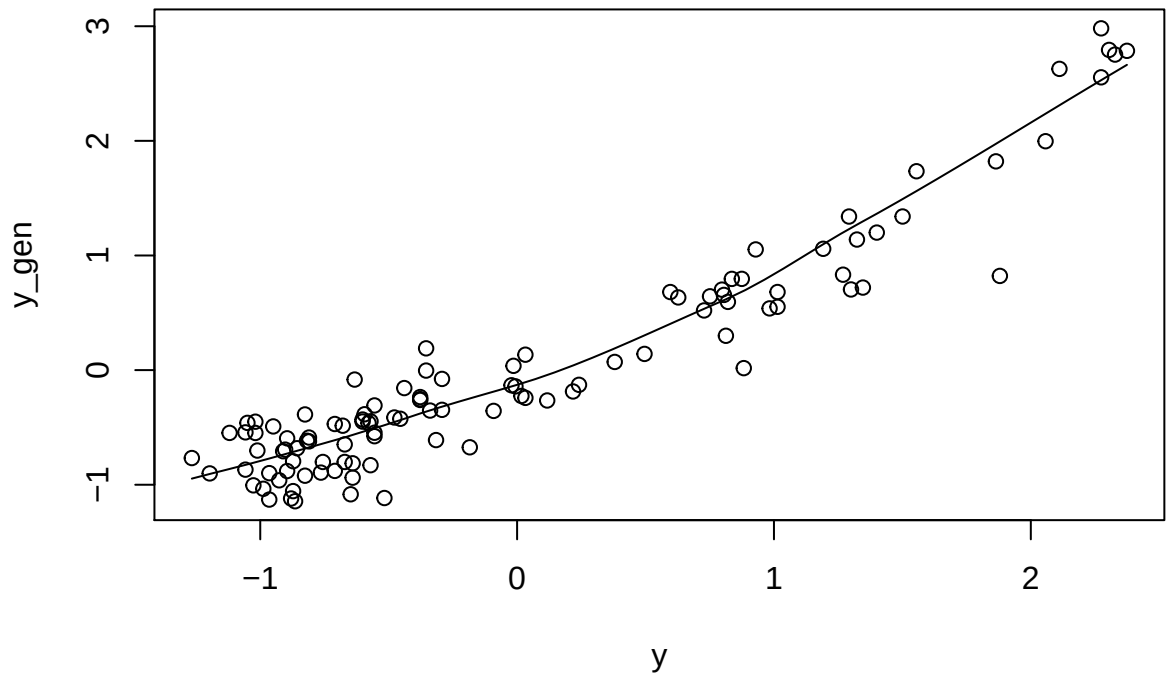



```
## coefficient of determination is  0.9889883 and MSE is  0.06737877
```

From the scatterplot we can see that y is similar to predited y and also the value of coefficient of determination is very close to 1 , this shows that lasso model is pretty good at predicting the Y for test data.

7.

Generating response using test data as features and optimal Lasso model




```
## coefficient of determination is  0.9572077  MSE is  0.1063286
```

The generated data seem to fit well as we can see value of coefficient of determination is close to 1 and MSE is also low.

Appendix

```
library(ggplot2)
library(kknn)
library(reshape2)
#### 1 ####

digits <- read.csv(file = 'optdigits.csv')
digits[,ncol(digits)] <- data.frame(sapply(digits[,ncol(digits)], as.character),
stringsAsFactors = TRUE)
names(digits) <- c(seq.int(ncol(digits)-1),"target")
#digits[1,]

### Plotting one observation ###

plot_observation <- function(id, M){
  number <- matrix(M[id, 1:(ncol(M)-1)], 8,8)
  mode(number) = "numeric"
  image(number[,nrow(number):1], col=grey(seq(0, 1, length = 256)))
  return(as.numeric(M[id, ncol(M)]))
}
```

```
plot_observation(5, digits)

##### 2 ######

### Train/Valation/Test Division ###

set.seed(12345)

n=dim(digits)[1]
set.seed(12345)

id=sample(1:n, floor(n*0.5))
dfTraining=digits[id,]

id1=setdiff(1:n, id)
set.seed(12345)

id2=sample(id1, floor(n*0.25))
dfValidation=digits[id2,]

id3=setdiff(id1,id2)
dfTest=digits[id3,]

### Let's fit the Model and predict results for Testing Set ###

kknn_model <- kknn(dfTraining[["target"]] ~ .,
                   dfTraining, dfTest,
                   k = 30,
                   kernel = "rectangular")

fit <- fitted(kknn_model)
CM_test <- table(dfTest$target, fit)
CM_test
accuracy_test <- (sum(diag(CM_test)))/sum(CM_test)
cat("Misclassification error in testing set is", 1 - accuracy_test)

### Let's fit the Model and predicting results for Testing Set ###

kknn_model <- kknn(dfTraining[["target"]] ~ .,
                    dfTraining, dfTraining,
                    k = 30,
                    kernel = "rectangular")
#summary(kknn_model)

fit <- fitted(kknn_model)
CM_train <- table(dfTraining$target, fit)
CM_train
accuracy_train <- (sum(diag(CM_train)))/sum(CM_train)
cat("Misclassification in training set is", 1 - accuracy_train)

##### 3 ######

# Getting the probabilities of prediction probabilities, joining with the actual
```

```
category and the prediction made

prob_df <- kknn_model$prob
max_prob <- colnames(prob_df)[apply(prob_df,1,which.max)]
train_probs <- data.frame(prob_df)
train_probs$actual <- dfTraining$target
train_probs$pred <- kknn_model$fitted.values
train_probs$max_prob <- max_prob

# Subsetting of the actual 8s and ordering by probabilities

actual_8s <- train_probs[train_probs["actual"]==8,]
predicted_actual_8s <- actual_8s[actual_8s["pred"]==8,]

# 8 predicted correctly with high certainty
best_8_indexes <- as.numeric(
row.names(predicted_actual_8s[order(-predicted_actual_8s[,9]),][1:2,]))

# 8 predicted wrongly with high certainty
worst_8_indexes <- as.numeric(row.names(predicted_actual_8s[order(
predicted_actual_8s[,9]),][1:3,]))


dfTraining[best_8_indexes,]
dfTraining[worst_8_indexes,]

# plot of 8 predicted correctly with high certainty

plot_observation(best_8_indexes[1], dfTraining)
plot_observation(best_8_indexes[2], dfTraining)

# plot of 8 predicted wrongly with high certainty

plot_observation(worst_8_indexes[1], dfTraining)
plot_observation(worst_8_indexes[2], dfTraining)
plot_observation(worst_8_indexes[3], dfTraining)

# plot of 8 predicted wrongly with high certainty

KKNN_misclassification_training <- c()
KKNN_misclassification_validation <- c()

##### 4 ######

#Let's fit a model for each k in the range

for (k in 1:30){

  # Fit for dfValidation and dfTraining

  KKNN_model_valid <- kknn(dfTraining[["target"]] ~ .,
  dfTraining, dfValidation, k = k, kernel = "rectangular")
  KKNN_model_train <- kknn(dfTraining[["target"]] ~ .,
  dfTraining, dfTraining, k = k, kernel = "rectangular")
```

```
  # Calculation of Accuracies

  fit_train <- fitted(KKNN_model_train)
  CM_train <- table(dfTraining$target, fit_train)
  KKNN_misclassification_training[k] <- 1 - (sum(diag(CM_train)))/sum(CM_train)

  fit_valid <- fitted(KKNN_model_valid)
  CM_valid <- table(dfValidation$target, fit_valid)
  KKNN_misclassification_validation[k] <- 1 -(sum(diag(CM_valid)))/sum(CM_valid)


}

# Plot of Accuracies

metrics <- data.frame(KKNN_misclassification_validation,
KKNN_misclassification_training, 1:30)
names(metrics) <- c("validation", "training", "k")


metricsMelted <- melt(metrics, id.var='k')
names(metricsMelted)[3] <- "Misclassification_Error"
head(metricsMelted)
ggplot(metricsMelted, aes(x=k, y=Misclassification_Error,
col=variable)) + geom_line()

best_K <- (which(
KKNN_misclassification_validation==min(KKNN_misclassification_validation)))
cat("The best performant K parameter are k =", as.character(best_K))

# Testing of optimal K with Testing set

kknn_model <- kknn(dfTraining[["target"]] ~ .,
dfTraining, dfTest, k = max(best_K), kernel = "rectangular")

fit <- fitted(kknn_model)
CM_test <- table(dfTest$target, fit)
CM_test
accuracy_test <- 1- (sum(diag(CM_test)))/sum(CM_test)
cat("Misclassification error is", accuracy_test)


### 4 ###

hot_encode <- function(i){
  v <- rep(0,10)
  v[i+1] <- 1
  return(I(v))
}

empirical_risk <- c()

# Let's fit a model for each k in the range
```

```
for (k in 1:30){

  KKNN_model_valid <- kknn(dfTraining[["target"]] ~ ., dfTraining,
  dfValidation, k = k, kernel = "rectangular")

  # Extraction of probabilities, we build a df with the target hot
  encoded to add a column of cross-entropy loss

  prob_df <- KKNN_model_valid$prob
  max_prob <- colnames(prob_df)[apply(prob_df,1,which.max)]
  valid_probs <- data.frame(prob_df)
  valid_probs$actual <- dfValidation$target
  valid_probs$pred <- KKNN_model_valid$fitted.values
  valid_probs$max_prob <- max_prob

  valid_probs$coded <- (lapply(as.numeric(valid_probs$actual )-1,  hot_encode))

  # column of cross-entropy loss
  for (row in 1:nrow(valid_probs)){
    valid_probs[row, "cross_entropy"] <- -sum(log(valid_probs[row, 1:10]+1e-15,
    base = 10)*valid_probs[[row,"coded"]] )
  }

  empirical_risk[k] <- mean(valid_probs$cross_entropy)

}
#### Differences of errors

# Plot of Empirical Risk for each K

metrics <- data.frame(empirical_risk, 1:30)
names(metrics) <- c("cross_entropy", "k")

metricsMelted <- melt(metrics, id.var='k')

names(metricsMelted)[3] <- "cross_entropy"
head(metricsMelted)
ggplot(metricsMelted, aes(x=k, y=cross_entropy, col=variable)) + geom_line()

best_K <- which(empirical_risk==min(empirical_risk))
cat("The best performant K parameter is k =", as.character(best_K))
```

```
##Assignment 2

library(readr)
parkinsons <- read_csv("parkinsons.csv")
cleaned <- parkinsons[c(-1:-4, -6)]
parkinsons.scaled <- scale(cleaned)
set.seed(12345)
n <- dim(parkinsons.scaled)[1]
id=sample(1:n, floor(n*0.6))
train=parkinsons.scaled[id,]
test=parkinsons.scaled[-id,]
```

```
loglikelihood <- function(w, sigma){
  n <- dim(train)[1]
  part1 <- -(n/ 2) * log(2 * pi*(sigma^2))
  y <- train[, 1]
  x <- train[, -1]
  res <- sum((y - (x %*% w))^2)
  return(part1 - (res/(2*(sigma^2))))
}

ridge <- function(x, lambda){
  w <- x[1:16]
  sigma <- x[17]
  tau <- sigma^2 / lambda
  part1 <-  (-1/2) * log(2* pi * tau)
  part2 <- (w %*% w) / (2* tau)
  ridge <- part1 - part2
  return( - (loglikelihood(w,sigma) + ridge))
}

ridgeOpt <- function(lambda){
  x <- rep(1,17)
  a <- optim(x ,ridge, method = "BFGS", lambda = lambda)
  w <- a$par[1:16]
  sigma <- a$par[17]
  return(a)
}

DF <- function(lambda){
  m <- as.matrix(train[ ,-1])
  part1 <- t(m) %*% m + (lambda * diag(16))
  part2 <- m %*% solve(part1) %*% t(m)
  return(sum(diag(part2)))
}

task4 <- function(lambda){
  a <- ridgeOpt(lambda)
  n <- dim(train)[1]

  w <- a$par[1:16]

  predict <- train[ , -1] %*% w
  y <- train[ ,1]

  MSEtrain <- (1/n) * t((y-predict)) %*% (y-predict)
  n <- dim(test)[1]
  predict <- test[ , -1] %*% w
  y <- test[ ,1]

  MSEtest <- (1/n) * t((y-predict)) %*% (y-predict)
  result <- list(w = w, sigma = a$par[17], testMSE = MSEtest, trainMSE = MSEtrain)
  return(result)

}
```

```
lambda1 <- task4(1)
lambda2 <- task4(100)
lambda3 <- task4(1000)
mm <- rbind(c(lambda1$trainMSE, lambda1$testMSE),
            c(lambda2$trainMSE, lambda2$testMSE),
            c(lambda3$trainMSE, lambda3$testMSE))
row.names(mm) <- c("lambda = 1", "lambda = 100", "lambda = 1000")
colnames(mm) <- c("MSE train", "MSE test")

DF <- function(lambda){
  m <- as.matrix(parkinsons.scaled[ ,-1])
  part1 <- t(m) %*% m + (lambda * diag(16))
  part2 <- m %*% solve(part1) %*% t(m)
  return(sum(diag(part2)))
}

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

y <- parkinsons.scaled[, 1]
x <- parkinsons.scaled[, -1]
res <- sum((y - (x %*% w))^2)

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

AIC1 <- -2 * loglikelihood(lambda1$w, lambda1$sigma) + 2 * DF(1)
AIC2 <- -2 * loglikelihood(lambda2$w, lambda2$sigma) + 2 * DF(100)
AIC3 <- -2 * loglikelihood(lambda3$w, lambda3$sigma) + 2 * DF(1000)
#print(paste("AIC1:", AIC1, "AIC2:", AIC2, "AIC3:", AIC3))

AIC <- c(AIC1 , AIC2 , AIC3)
names(AIC) <- c("lambda = 1", "lambda = 100", "lambda = 1000")
knitr::kable(as.matrix(AIC, nrow= 1), row.names = TRUE,  col.names = c("AIC") )
res <- which.min(AIC)
res <- ifelse(res == 1, "1", ifelse(res == 2, 100, 1000))
```

```
##Assignment 3

library(glmnet)
data = read.csv("tecator.csv", header = TRUE)
n = nrow(data)
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

#1
X_train = as.matrix(train[2:101])
Y_train = as.matrix(train$Fat)
fit_train = lm(Fat ~ . , data = train[2:102]) #fitting linear
#regression to training data
```

```
y_hat_train = predict(fit_train)
train_mse = mean((y_hat_train-Y_train)^2)
cat("MSE for training data is ",train_mse)
cat("MAE for training data is ",mean(abs(y_hat_train - Y_train)))
#scatter.smooth(Y_train,y_hat_train)
new_x=as.matrix((test[, 2:101]))
Y_test = as.matrix((test[,102]))
y_hat_test = predict(fit_train, newdata = as.data.frame(new_x))
#scatter.smooth(Y_test,y_hat_test)
test_mse = mean((y_hat_test-Y_test)^2)
cat("\nMSE for testing data ",test_mse)
cat("MAE for testing data is ",mean(abs(y_hat_test - Y_test)))

#3
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 = T)
model=cv.glmnet(as.matrix(covariates),
                response, alpha=1,family="gaussian")
model_lasso_3 = glmnet(as.matrix(covariates), response, alpha = 1,
family = "gaussian", lambda = model$lambda[which(model$nzero == 3)])
plot(model_lasso_3, xvar = "lambda", label = T)
cat("Lambda values that give 3 variables are :",
model$lambda[which(model$nzero == 3)])

#4
df = function(lambda){
  ld = lambda * diag(ncol(covariates))
  H = covariates %*% solve(t(covariates) %*% covariates + ld) %*% t(covariates)
  DOF = sum(diag(H))
  return(DOF)
}
lambda_values = model_lasso$lambda
degrees_of_freedom = c()
for(i in 1:length(lambda_values)){
  degrees_of_freedom[i] = df(lambda_values[i])
}
plot(lambda_values, degrees_of_freedom)

#5
covariates = scale(train[ ,2:101])
response = scale(train[ ,102])
model_ridge = glmnet(as.matrix(covariates), response, alpha = 0,
family = "gaussian")
plot(model_ridge, main = "RIDGE", xvar = "lambda", label = T)

#6
model_l=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian",
lambda=seq(0,1,0.001))
best_lambda = model_l$lambda.min
plot(model_l)
plot(model_l$lambda,model_l$cvm,xlab="lambda",ylab="Mean CV error")
```

```
cat("Optimal lambda is :",best_lambda)

m = cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian",
lambda=c(best_lambda,0.1353353))
cat("CV Error when lambda is ",m$lambda[1] ," =",m$cvm[1])
cat(" and CV Error when lambda is ",m$lambda[2] ," =",m$cvm[2])

model_optimallasso =  glmnet(as.matrix(covariates), response, alpha = 1,
family = "gaussian", lambda = best_lambda)
y = as.matrix(scale(test[,102]))
ypred=predict(model_optimallasso, newx=as.matrix(scale(test[, 2:101])),
type="response")
scatter.smooth(y,ypred)
cat("coefficient of determination is ",
    sum((ypred-mean(y))^2)/sum((y-mean(y))^2), "and MSE is ",
    mean((y-ypred)^2))

#7
betas = as.vector((as.matrix(coef(model_optimallasso))[-1, ])) # removing the
#first row for intercept
resid = response - (covariates %*% betas)
sigma = sd(resid)

ypred=predict(model_optimallasso, newx=as.matrix(scale(test[, 2:101])),
type="response")
set.seed(90000)
y_gen = rnorm(108,ypred,sigma)
scatter.smooth(y,y_gen)
cat("coefficient of determination is ",
sum((y_gen-mean(y))^2)/sum((y-mean(y))^2)," MSE is ",mean((y_gen - y)^2))
```