ML help file

Christophoros Spyretos

Lab01

Assignment 1

Reading Data

```
data <- read.csv("optdigits.csv", header = FALSE)
```

Task 1 Splitting the data into training, validation and test sets (50%/25%/25%).

```
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]

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

id2=sample(id1, floor(n*0.25))
valid=data[id2,]

id3=setdiff(id1,id2)
test=data[id3,]
```

Task 2 Training data to fit 30-nearest neighbor classifier.

```
library(kknn)
knn_train <- kknn(as.factor(V65)~., train = train, test = train, k = 30, kernel = "rectangular")
knn_valid <- kknn(as.factor(V65)~., train = train, test = valid, k = 30, kernel = "rectangular")
knn_test <- kknn(as.factor(V65)~., train = train, test = test, k = 30, kernel = "rectangular")
```

```
confusion_train <- table(train$V65, knn_train$fitted.values)
knitr::kable(confusion_train,caption = "Confusion matrix: training data")
```

Table 1: Confusion matrix: training data

	0	1	2	3	4	5	6	7	8	9
0	202	0	0	0	0	0	0	0	0	0
1	0	179	11	0	0	0	0	1	1	3
2	0	1	190	0	0	0	0	1	0	0
3	0	0	0	185	0	1	0	1	0	1
4	1	3	0	0	159	0	0	7	1	4
5	0	0	0	1	0	171	0	1	0	8

	0	1	2	3	4	5	6	7	8	9
6	0	2	0	0	0	0	190	0	0	0
7	0	3	0	0	0	0	0	178	1	0
8	0	10	0	2	0	0	2	0	188	2
9	1	3	0	5	2	0	0	3	3	183

```
confusion_test <- table(test$V65, knn_test$fitted.values)
knitr::kable(confusion_test,caption = "Confusion matrix: test data")
```

Table 2: Confusion matrix: test data

	0	1	2	3	4	5	6	7	8	9
0	77	0	0	0	1	0	0	0	0	0
1	0	81	2	0	0	0	0	0	0	3
2	0	0	98	0	0	0	0	0	3	0
3	0	0	0	107	0	2	0	0	1	1
4	0	0	0	0	94	0	2	6	2	5
5	0	1	1	0	0	93	2	1	0	5
6	0	0	0	0	0	0	90	0	0	0
7	0	0	0	1	0	0	0	111	0	0
8	0	7	0	1	0	0	0	0	70	0
9	0	1	1	1	0	0	0	1	0	85

Misclassification error

```
misclass <- function(actual_val,fitted_val){
  confusion_matrix <- table(actual_val,fitted_val)
  n <- length(actual_val)
  error <- 1 - (sum(diag(confusion_matrix))/n)
  return(error)
}
# misclass(train$V65, knn_train$fitted.values)
# misclass(test$V65, knn_test$fitted.values)
```

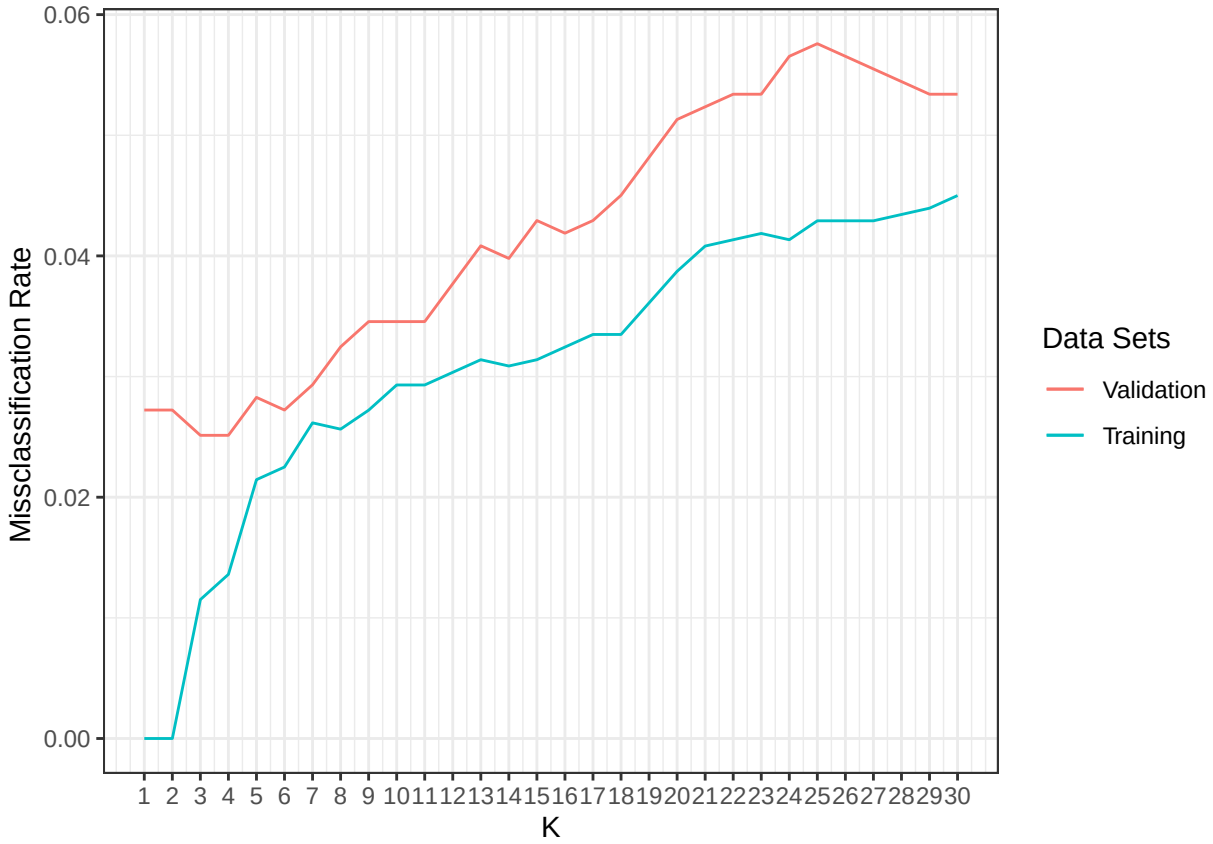
Task 3 Cases & Heatmap

```
index_8 = which(train$V65 == 8)
knn_train_prob = data.frame(knn_train$prob)
prob_8 = knn_train_prob$X8[index_8]
easiest_8 = index_8[tail(order(prob_8), n = 2)]
hardest_8 = index_8[head(order(prob_8), n = 3)]

easiest_8_1 <- matrix(as.matrix(train[easiest_8[1], 1:64]), nrow = 8, ncol = 8, byrow = TRUE)
#heatmap(easiest_8_1, Rowv = NA, Colv = NA)

easiest_8_2 <- matrix(as.matrix(train[easiest_8[2], 1:64]), nrow = 8, ncol = 8, byrow = TRUE)
#heatmap(easiest_8_2, Rowv = NA, Colv = NA)

hardest_8_1 <- matrix(as.matrix(train[hardest_8[1], 1:64]), nrow = 8, ncol = 8, byrow = TRUE)
#heatmap(hardest_8_1, Rowv = NA, Colv = NA)
```

```r
hardest_8_2 <- matrix(as.matrix(train[hardest_8[2], 1:64]), nrow = 8, ncol = 8, byrow = TRUE)
#heatmap(hardest_8_2, Rowv = NA, Colv = NA)

hardest_8_3 <- matrix(as.matrix(train[hardest_8[3], 1:64]), nrow = 8, ncol = 8, byrow = TRUE)
#heatmap(hardest_8_3, Rowv = NA, Colv = NA)
```

Task 4 Optimal K

```r
k_optmimal_train <- c()
k_optmimal_valid <- c()

for (i in 1:30){
  set.seed(12345)
  knn_train <-kknn(as.factor(V65)~., train=train,test=train, k = i,kernel = "rectangular")
  knn_valid <-kknn(as.factor(V65)~., train=train,test=valid, k = i,kernel = "rectangular")
  k_optmimal_train[i] = 1-(sum(diag(table(train$V65, knn_train$fitted.values)))/nrow(train))
  k_optmimal_valid[i] = 1-(sum(diag(table(valid$V65, knn_valid$fitted.values)))/nrow(valid))

}

rate <- data.frame( "K" <- c(1:30),
                    "k_optmimal_train" <- k_optmimal_train,
                    "k_optmimal_valid" <- k_optmimal_valid)

my_plot <- ggplot( data = rate) +
  geom_line(aes(x = K, y = k_optmimal_train, colour="#F8766D")) +
  geom_line(aes(x = K, y = k_optmimal_valid, colour="#00BFC4")) +
  ylab("Missclassification Rate ") + xlab("K") +
  scale_color_manual(name = "Data Sets",
                     labels = c("Validation ", "Training "),
                     values =c("#F8766D", "#00BFC4")) +
  theme_bw() +
  scale_x_continuous( breaks = 1:30)

my_plot
```



We can see that as K increases both the training and validation models, misclassification errors tend to increase. When K = 1 or K = 2, there is a zero misclassification error for the training data, yet the model generalizes worst to the testing data. After K = 4 we see that training and validation errors become closer to each other. Interestingly as the model becomes more complex, the classifier tends to get worse. An optimal K seem to be at either K = 3 or K = 4 because these models have the lowest misclassification error on the validation data and are relatively simple. We choose to continue with K = 3 since it is an odd number. When K = 4, ties are possible, so if two of the nearest neighbours are one class and two are another, there is an issue that is solved randomly, while with K = 3, there is always a clear decision.

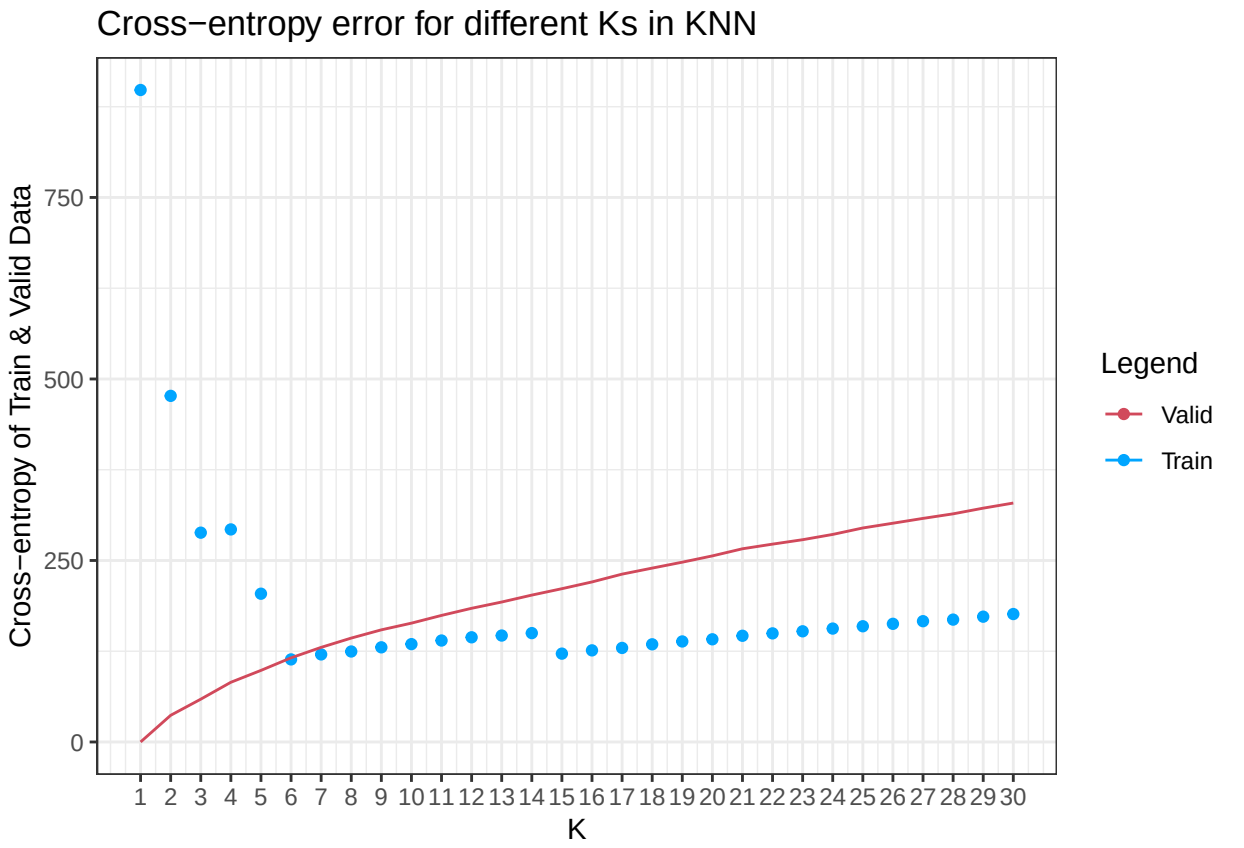
```
knn_train_K3 <- kknn(as.factor(V65)~., train = train, test = test, k = 3, kernel = "rectangular")
test_error <- misclass(test$V65, fitted(knn_train_K3))
errors_df <- data.frame(TrainingError = rate$X.k_optmimal_train.....k_optmimal_train[3],
                        ValidationError = rate$X.k_optmimal_valid.....k_optmimal_valid[3],
                        TestingError = test_error)
row.names(errors_df) <- "Misclassification error (in %)"

errors_df
```

```
##                                TrainingError ValidationError TestingError
## Misclassification error (in %)     0.0115123      0.02513089   0.02403344
```

For the KNN classifier, when $K = 3$, the model has a low misclassification rate for the training data. The classification rate increased by approximately 1% when comparing the model on the training data versus the validation and testing data. The error for the test set is low and even slightly less than for the validation set, indicating that the model generalizes well to new data. Moreover, when k increases the model complexity decreases and the model becomes more biased but has a lower variance.

Task 5 Cross-Entropy

```
cross_entropy_valid <- list()
for (K in 1:30) {
  fit = kknn(as.factor(V65)~., train, valid, k=K, kernel="rectangular")

  df <- data.frame(fit$prob)
  df$digit <- valid$V65

  entropy <- list()
  for (i in 1:nrow(df)) {
    for (n in 1:10) {
      if (df$digit[i] == n-1) {
        entropy[[i]] = -(log(df[i, n]+ 1e-15))
      }
    }
  }
  cross_entropy_valid[[K]] <- sum(unlist(entropy))
}

cross_entropy_train <- list()
for (K in 1:30) {
  fit = kknn(as.factor(V65)~., train, train, k=K, kernel="rectangular")

  df <- data.frame(fit$prob)
  df$digit <- train$V65

  entropy <- list()
  for (i in 1:nrow(df)) {
    for (n in 1:10) {
      if (df$digit[i] == n-1) {
        entropy[[i]] = -(log(df[i, n]+ 1e-15))
      }
    }
  }
  cross_entropy_train[[K]] <- sum(unlist(entropy))
}

cross_entropy <- data.frame(train = unlist(cross_entropy_train),
                            valid = unlist(cross_entropy_valid),
                            K = 1:30)

ggplot(cross_entropy) +
  geom_point(aes(x = K, y = valid, color = "#d1495b")) +
  geom_line(aes(x=K, y = train, color = "#00A5FF")) +
  theme_bw() +
  labs(title = "Cross-entropy error for different Ks in KNN", y = "Cross-entropy of Train & Valid Data"
  scale_x_continuous(breaks = 1:30) +
  theme(legend.position="right") +
  scale_color_manual(values=c('#d1495b','#00A5FF'),
                     name = "Legend",
                     labels = c("Valid","Train" ))
```



Using cross-entropy to evaluate which K is the best takes a more probability-based perspective than using the misclassification error. In cross-entropy, the predicted probabilities for the true label is what is relevant. Suppose the model predicts that the label is the actual label by a high probability. In that case, the cross-entropy is low, and vice versa if the probability for the true label is low, then the cross-entropy is high. Using this as a loss function would punish uncertain decisions. The model with the lowest cross-entropy is when $K = 6$. We can see here that the models with lower K, for example, $K = 3$, have higher cross-entropy. Meaning that even if there is a slightly lower misclassification rate when $K = 3$, the model estimates the probability for the truly labelled digit lower than a model with $K = 6$ does. It is reasonable to prefer $K = 6$ since the misclassification is slightly higher and has a lot lower cross-entropy.

Assignment 2

Task 1 (Scaling & splitting data into train & test set.)

We scale and split the data into 60% training data and 40% testing data. We standardise the data for consistency to have the same content and format. In addition, standardised values help track data that is not easy to compare otherwise. We calculate the mean and standard deviation of the variables in the training data and apply the same transformation to the testing data.

```
data <- read.csv("parkinsons.csv")
data <- data[,-c(1:4,6)]   # deleting undesirable variables

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train=data.frame(data[id,])
test=data.frame(data[-id,])
```

```
mean_train <- numeric(ncol(train))
sd_train <- numeric(ncol(train))

for(i in 1:ncol(train)){
  mean_train[i] <- mean(train[,i])
  sd_train[i] <- sd(train[,i])
}

train <- data.frame(scale(train))

test_new <- data.frame(matrix(nrow = nrow(test), ncol = ncol(test)))

for(i in 1:ncol(train)){
  test_new[,i] <- (test[,i] - mean_train[i])/sd_train[i]
}

test <- as.data.frame(test_new)
colnames(test) <- colnames(train)
```

Task 2 (Linear regression model & estimation of train and test MSE.)

```
lrm <- lm(motor_UPDRS ~. -1, data = train)
#summary(lrm)
train_MSE <- mean((train$motor_UPDRS - predict(lrm))^2)
#train_MSE
test_MSE <- mean((test$motor_UPDRS - predict(lrm,test))^2)
#test_MSE
```

Task 3 (Implementing 4 functions)

Log likelihood function

```
## 2.3
Loglikelihood <- function(theta, sigma, input_data){
  Y <- input_data[,1]
  X <- as.matrix(input_data[,-1])
  n <- nrow(input_data)
  logl <- -n*log(sqrt(2*pi)*abs(sigma)) - (1/(2*(sigma^2)))*sum((Y-X %*% theta)^2)
  return(logl)
}
```

Ridge function

```
Ridge <- function(param, input_data, lambda){
  theta <- param[-1]
  sigma <- param[1]
  logl <- -Loglikelihood(theta, sigma, input_data) + lambda*sum(theta^2)
  return(logl)
}
```

Ridge optimal

```
RidgeOpt <- function(lambda, input_data){
  optimal <- optim(par = rep(1, 17), fn = Ridge, input_data = input_data, lambda = lambda,
                   method = "BFGS")
```

```
  return(optimal$par)
}
```

Degrees of freedom

```
#lamda is 0 if we do not give a penalty
df <- function(input_data, lambda=0){
  X <- as.matrix(input_data[,-1])
  I <- diag(ncol(X))
  hat_matrix <- X %*% solve(t(X) %*% X + lambda*I) %*% t(X)
  degrees_of_freedom <- sum(diag(hat_matrix))
  return(degrees_of_freedom)
}
```

Task 4

Using function RidgeOpt to compute optimal theta parameters for lamda = 1, lamda = 100 and lamda = 1000.

```
theta_hat_1 <- RidgeOpt( lambda = 1, input_data = train)
theta_hat_100 <- RidgeOpt( lambda = 100, input_data = train)
theta_hat_1000 <- RidgeOpt( lambda = 1000, input_data = train)
# theta_hat_1[-1]
# theta_hat_100[-1]
# theta_hat_1000[-1]
```

Using the estimating parameters to predict the motor_UPDRS values for training and test data.

```
predicted_values <- function(theta_hat, input_data){
  X <- as.matrix(input_data[,-1])
  y_hat <- X %*% theta_hat[-1]
  return(y_hat)
}

y_hat_1_train <- predicted_values(theta_hat_1, train)
y_hat_100_train <- predicted_values(theta_hat_100, train)
y_hat_1000_train  <- predicted_values(theta_hat_1000, train)

y_hat_1_test <- predicted_values(theta_hat_1, test)
y_hat_100_test <- predicted_values(theta_hat_100, test)
y_hat_1000_test  <- predicted_values(theta_hat_1000, test)
```

Reporting the training and test MSE values.

```
MSE <- function(Y_hat, input_data){
  Y <- input_data[,1]
  n <- nrow(input_data)
  mse <- (1/n) * sum((Y-Y_hat)^2)
  return(mse)
}

MSE_1_train <- MSE(y_hat_1_train, train)
MSE_100_train <- MSE(y_hat_100_train, train)
MSE_1000_train <- MSE(y_hat_1000_train, train)

MSE_1_test <- MSE(y_hat_1_test, test)
```

```
MSE_100_test <- MSE(y_hat_100_test, test)
MSE_1000_test <- MSE(y_hat_1000_test, test)

errors_df <- data.frame("lambda=1" = c(MSE_1_train,
                                       MSE_1_test),
                        "lambda=100" = c(MSE_100_train,
                                         MSE_100_test),
                        "lambda=1000" = c(MSE_1000_train,
                                          MSE_1000_test))
row.names(errors_df) <- c("MSE (training data)", "MSE (test data)")
errors_df
```

```
##                       lambda.1 lambda.100 lambda.1000
## MSE (training data) 0.8786272  0.8844114   0.9211166
## MSE (test data)     0.9349976  0.9323317   0.9539456
```

The penalty that seems to be the most appropriate for the training dataset is the theta parameters for lambda = 1. The MSE of the predicted values with the above theta parameters is 0.8786272, which means that predicted values have the slightest error from the observed values compared to the other predicted values.

The theta parameters for lambda = 100 provide the best penalty for the test dataset with an MSE of 0.9323317. Thus, the predicted values based on these thetas are better estimators than the other predicted values.

Compute and compare the degrees of freedom of the training and test models.

Both the train and the test data have 16 degrees of freedom if you look at the normal linear model without penalty because we assume the variables to be independent. Using λ as a penalty term takes away some of the independence, and fewer degrees of freedom remain, as shown in the table below. So it makes sense that the higher the penalty term gets, the lower the degrees of freedom are.

```
freedom_df <- data.frame("no penalty" = c(df(input_data = train),
                                          df(input_data = test)),
                         "lamda=1" = c(df(input_data = train,
                                          lambda = 1),
                                       df(input_data = test,
                                          lambda = 1)),
                         "lamda=100" = c(df(input_data = train,
                                            lambda = 100),
                                         df(input_data = test,
                                            lambda = 100)),
                         "lamda=1000" = c(df(input_data = train,
                                             lambda = 1000),
                                          df(input_data = test,
                                             lambda = 1000)))
row.names(errors_df) <- c("df (training data)", "df (test data)")
freedom_df
```

```
##   no.penalty  lamda.1 lamda.100 lamda.1000
## 1         16 13.86074  9.924887   5.643925
## 2         16 13.77688  9.173182   4.822328
```

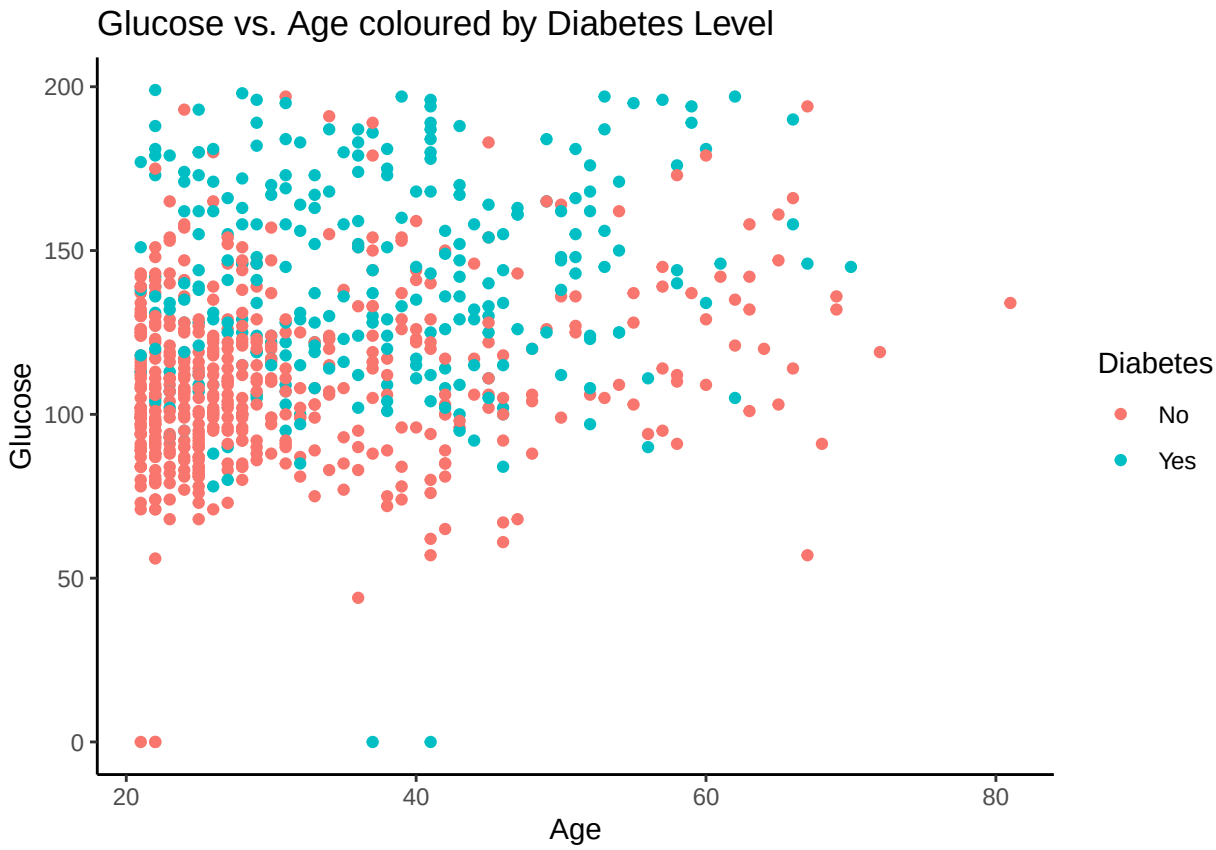
Assignment 3

Reading & Preparing Data

```
data <- read.csv("pima-indians-diabetes.csv", header = FALSE)
colnames(data) <- c("Pregnancies",
                    "Glucose",
                    "BloodPressure",
                    "SkinThickness",
                    "Insulin",
                    "BMI",
                    "DiabetesPedigreeFunction",
                    "Age",
                    "Diabetes")
data$Diabetes <- as.factor(data$Diabetes)
levels(data$Diabetes) <- c("No","Yes")
```

Task 1

```
scatterplot <- ggplot(data, aes(x = Age, y = Glucose, col = Diabetes)) +
  geom_point() +
  theme_classic() +
  ggtitle("Glucose vs. Age coloured by Diabetes Level")
scatterplot
```



Suppose one is only interested in a rough classification. In that case, these two variables are probably enough as one can definitely spot two different colour areas: Most people that have diabetes have a higher glucose concentration, and additionally, in the group of young people, fewer people have diabetes. Nevertheless, the classification will probably not work that well for all values in the area where both groups are bordering as

they have some overlapping areas there. So, in general, the groups are not very well separable, which will be a problem for the logistic regression.

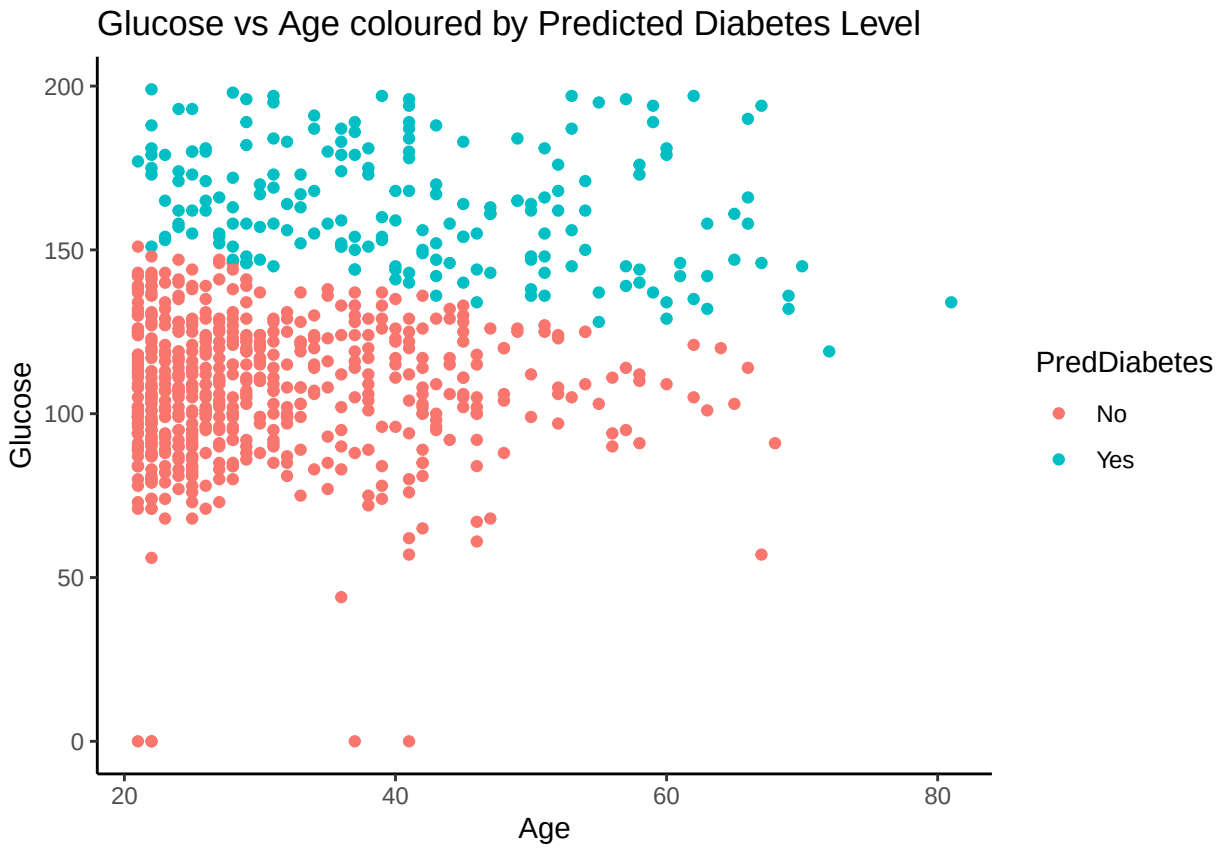
```
library(caret)
glm_fit <- train(Diabetes ~ Age + Glucose, data = data, method="glm")
data$PredDiabetes <- predict(glm_fit) # predict function uses 0.5 as default value
```

Task 2 $p(y = 1|w, x_1, x_2) = \frac{1}{1+e^{-w_0-w_1x_1-w_2x_2}}$ x_1 is the glucose concentration and x_2 is the age.

```
misclass <- function(actual_val,predicted_val){
  confusion_matrix <- table(actual_val,predicted_val)
  n <- length(actual_val)
  error <- 1 - (sum(diag(confusion_matrix))/n)
  return(error)
}
misclass(data$Diabetes,data$PredDiabetes)
```

```
## [1] 0.2630208
```

```
scatterplot2 <- ggplot(data, aes(x = Age, y = Glucose, col = PredDiabetes)) +
  geom_point() +
  theme_classic() +
  ggtitle("Glucose vs Age coloured by Predicted Diabetes Level")
scatterplot2
```

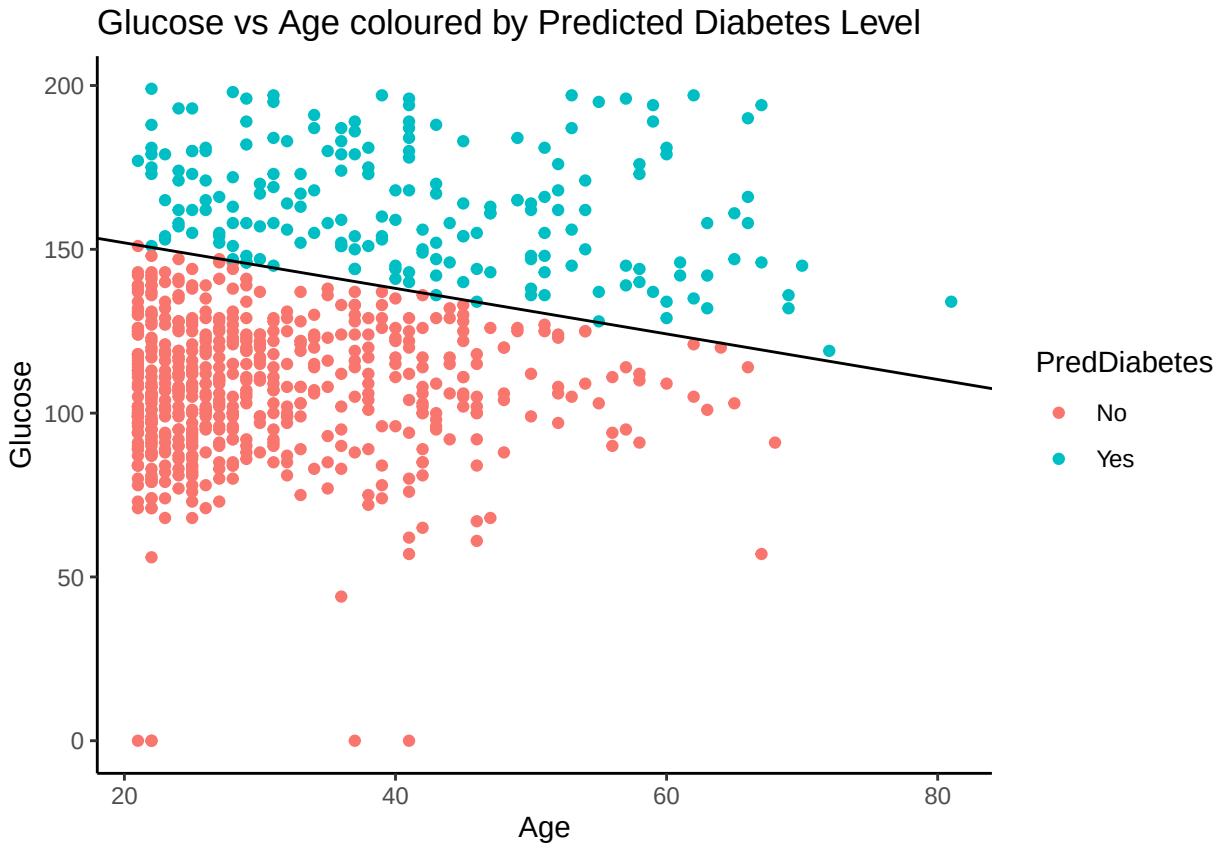


As the classification is only done using the two variables shown in the plot, we knew before that the decision boundary would be better visible after training a model. Therefore, the results are all right if one is only interested in a rough classification using these two variables. Furthermore, the misclassification rate is only

around 26%, so three out of four observations are classified right. Nevertheless, one should keep in mind that a higher classification success would definitely be desirable, so it might make sense to include some more variables in the model because there is no rigorous cut visible between these two variables, as mentioned earlier.

Task 3

```
intercept <- glm_fit$finalModel$coefficients[1]/(-glm_fit$finalModel$coefficients[3])
slope <- glm_fit$finalModel$coefficients[2]/(-glm_fit$finalModel$coefficients[3])

scatterplot3 <- scatterplot2 + geom_abline( intercept = intercept, slope = slope)
scatterplot3
```



In the plot above, one can see the decision boundary, which splits the predicted data as one would expect if you look at the predicted data. Obviously, the decision boundary catches the data distribution of the predicted data well.

Task 4

```
data$PredDiabetes2 <- as.factor(ifelse(predict(glm_fit,
                                               type="prob") < 0.2,
                                       "No","Yes")[,2])
misclass(data$Diabetes, data$PredDiabetes2)
```

```
## [1] 0.3723958
```

```
scatterplot3 <- ggplot(data, aes(x = Age, y = Glucose, col = PredDiabetes2)) +
  geom_point()
scatterplot3
```


```
data$PredDiabetes8 <- as.factor(ifelse(predict(glm_fit,
                                               type="prob") < 0.8,
                                       "No","Yes")[,2])
misclass(data$Diabetes, data$PredDiabetes8)
```

```
## [1] 0.3151042
```

```
scatterplot4 <- ggplot(data, aes(x = Age, y = Glucose, col = PredDiabetes8)) +
  geom_point()
scatterplot4
```

If you change the value for r to 0.2 or 0.8, the classification for the data points is estimated with values in the range between changes. So if you change the value from 0.5 to 0.8, fewer values are classified as Diabetes cases because all predicted values between 0.5 and 0.8 are now added to the non-Diabetes group as well. If you change the value to 0.2, many more people are classified as Diabetes cases because all predicted values above 0.2 are put into that group. The misclassification rate is higher than for the 0.5 thresholds, both for 0.2 and 0.8.

Task 5

```
data$z1 <- data$Glucose^4
data$z2 <- data$Glucose^3 * data$Age
data$z3 <- data$Glucose^2 * data$Age^2
data$z4 <- data$Glucose * data$Age^3
data$z5 <- data$Age^4

glm_fit_new <- train(Diabetes ~ Age + Glucose + z1 + z2 +
                       z3 + z4 + z5, data = data, method="glm")
data$PredDiabetesNew <- predict(glm_fit_new)

misclass(data$Diabetes, data$PredDiabetesNew)
```

```
## [1] 0.2447917
```

```
scatterplot5 <- ggplot( data = data, aes( x = Age, y = Glucose, colour = PredDiabetesNew)) +
  geom_point()
scatterplot5
```

What can you say about the quality of this model compared to the previous logistic regression model?

If one compares the misclassification rates of the two models, the new model performs a little bit better than the previous one. Before, the misclassification rate was 26% while it now is 24%. As our data set contains 768 observations, this means a difference of 14 right classified data points. So instead of 566 observations classified right before, we know we classify 580 data points correctly.

How have the basis expansion trick affected the shape of the decision boundary and the prediction accuracy?

The shape was affected by the basis expansion trick as we now do not have a straight line that cuts our data, but a parable shaped area that includes all the people predicted not to have diabetes. In contrast, the ones with predicted sickness are above (and one observation is below). What we think is particularly interesting is the blue point in the right bottom corner below the just mentioned red area, which is being classified wrong now but was being classified right before as it actually belongs to the group with people with no diabetes.

This example illustrates nicely that the fitting process obviously concentrates a lot on the more usual data. So we can assume now that if there would be more data points around the just mentioned data point that they would probably belong to the group that is classified as diabetes, even though when looking at the original data, there might be a higher chance that it actually belongs to the people that do not have diabetes.

Lab02

Assigmnent 1: Explict Regularization

Task 1: Splitting the data & Fitting the linear regression model.

```
data <- read.csv("tecator.csv")
n=dim(data)[1]
set.seed(12345)
```

```
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
```

The probabilistic model of linear regression is given by $Y = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + ... + b_{100}X_{100} + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and $\sigma^2 \geq 0$, also, $Y\ N(b_0 + b_1X_1 + b_2X_2 + b_3X_3 + ... + b_{100}X_{100}, \sigma^2)$ or $p(Y|X, \theta) = N(b_0 + b_1X_1 + b_2X_2 + b_3X_3 + ... + b_{100}X_{100}, \sigma^2))$

```
train <- train[,-c(1,103,104)]
test <- test[,-c(1,103,104)]

lrm <- lm(Fat ~., data = train)
#summary(lrm)

train_error <- mean((train$Fat - predict(lrm))^2)
#train_error
test_error<- mean((test$Fat - predict(lrm,test))^2)
#test_error
```

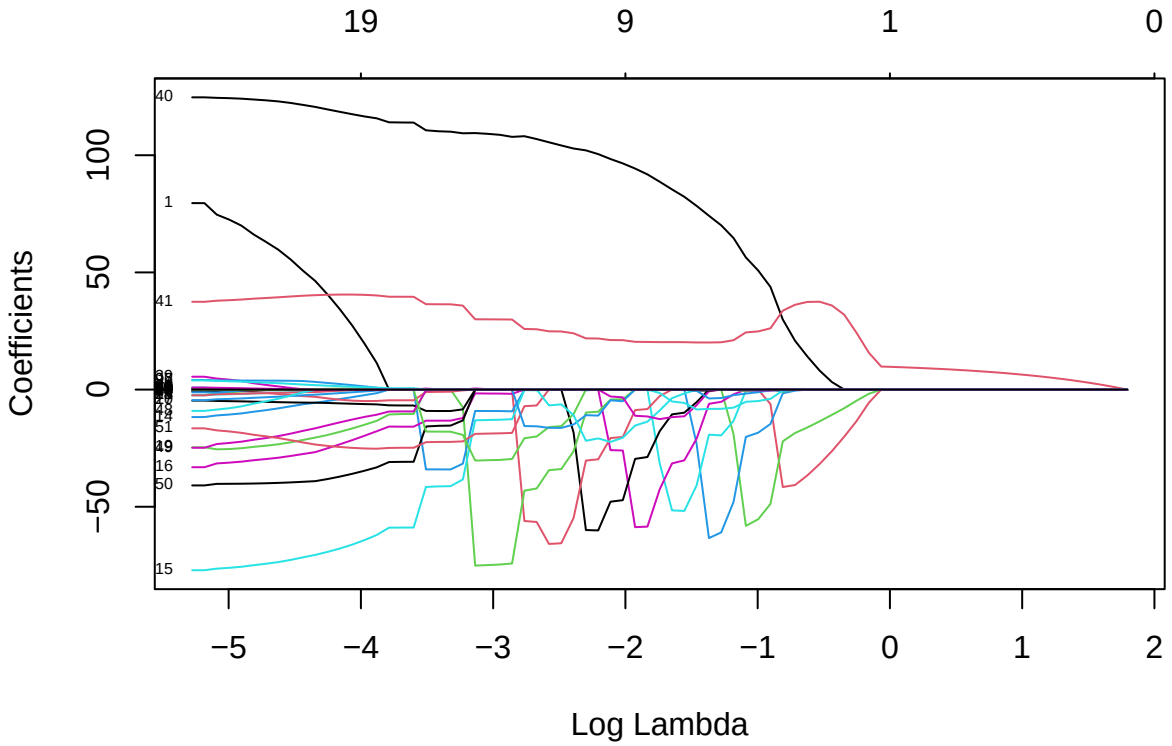
The model fits the training data well, but as the MSE of the test data is relatively high, it might have problems dealing with new data and is overfitting to the training data.

Task 2: Report the cost function of Lasso.

The cost function of the Lasso regression that it should be optimised is given by $\hat{\boldsymbol{\theta}} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n}||\boldsymbol{X\theta} - \boldsymbol{y}||_2^2 + \lambda||\boldsymbol{\theta}||$, as long as $||\theta||$ is Norm-L1.

Task 3: Fitting the lasso regression model.

```
library(glmnet)

X <- train[,-101]
Y <- data.frame(train[,101])

lasso <- glmnet(as.matrix(X), as.matrix(Y), alpha=1)

#summary(lasso)
#log(lasso$lambda)

plot(lasso,xvar="lambda",label=TRUE)
```



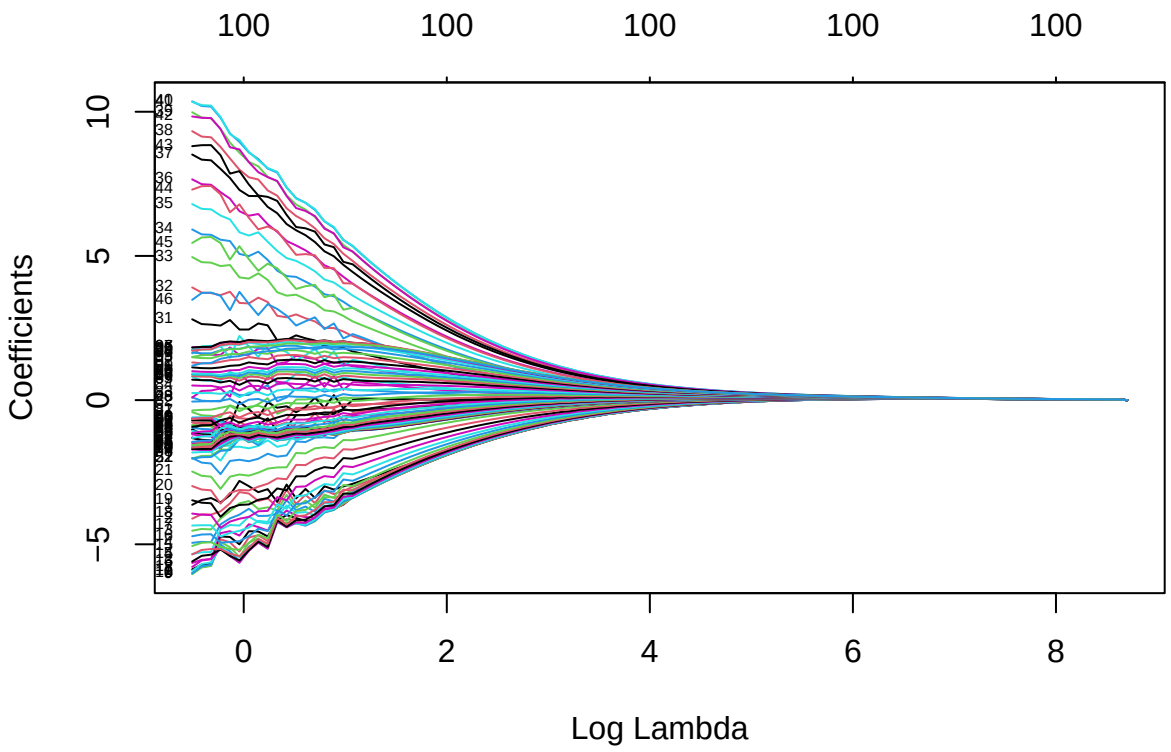

```
for (i in lasso$lambda) {
  nparam <- sum(coef(lasso, s = i)[,1] != 0)
  if(nparam == 4){
    lambda <- i
    break
  }
}
cat("Optimal lambda:",lambda)
```

```
## Optimal lambda: 0.8530452
```

The numbers above the plot represent the number of features that have coefficients that are not zero; thus, the larger value of lambda has, the fewer features with an impact on the model has. In order to select a model with only three features, several lambdas could be chosen, and we choose the smallest of these options, which is equal to 0.8530452.

Task 4: Fitting the ridge regression model.

```
ridge <- glmnet(as.matrix(X), as.matrix(Y), alpha=0)

plot(ridge,xvar="lambda",label=TRUE)
```



In ridge regression, when the value of lambda is increased, the features are never typically 0. It is not possible to reduce the number of features with coefficients with impact. Thus, it is better to use the lasso regression to select a model with specific features or reduce the number of features.

Task 5: Cross Validation for opimal lasso model.

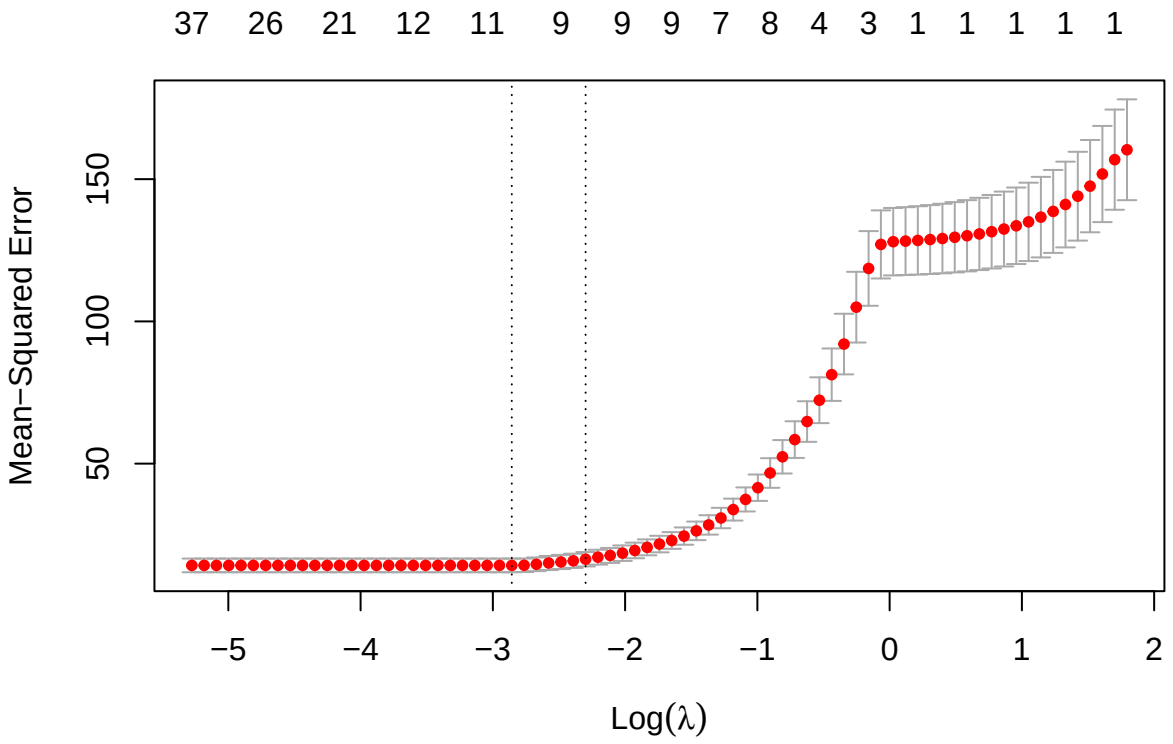
```
cv_lasso <- cv.glmnet(as.matrix(X), as.matrix(Y), alpha = 1)
cat("Optimal lambda:", cv_lasso$lambda.min)
```

```
## Optimal lambda: 0.05744535
```

```
sum(coef(cv_lasso, s = cv_lasso$lambda.min)[,1] != 0) - 1 #substract intercept
```

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

```
plot(cv_lasso)
```

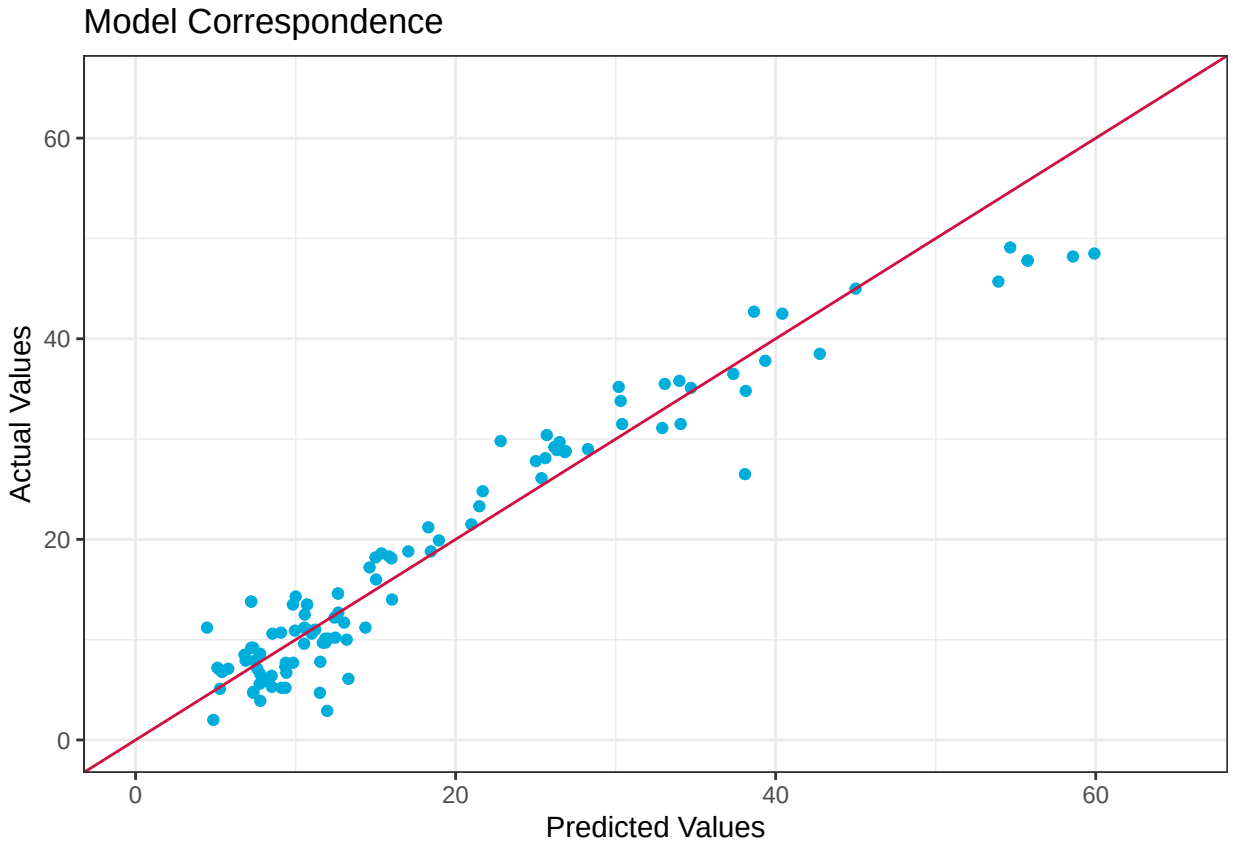


The confidence bounce is larger when the $\log(\lambda)$ is larger; thus, there is a confidence limit for each point, allowing how much better or worse each lambda is.

The optimal value for $log(\lambda)$ equals approximately -2.8, but the plot does not illustrate that this value is a statistical significant better prediction than the $\log(\lambda) = -4$. This can be seen by the fact that the confidence intervals overlap.

Choosing this optimal value for $log(\lambda)$, we get eight variables and the intercept that impact the model.

```
optimal.lambda <- predict(cv_lasso , as.matrix(test[,-101]) , s = cv_lasso$lambda.min)

optimal_df <- data.frame( "Actual"= test$Fat, "Predicted" = optimal.lambda)

library(ggplot2)

my_scatterplot <-ggplot(optimal_df,aes(x=s1, y= Actual)) +
  geom_point( color = "#00aedb") +
  lims(x=c(0,65), y=c(0,65)) +
  geom_abline(color = "#d11141") +
  labs(title = "Model Correspondence") +
  xlab("Predicted Values") +
  ylab("Actual Values") +
  theme_bw()

my_scatterplot
```



The plot illustrates that the model predictions corresponding to optimal lambda are relatively "good". Most observations are placed around the bisector, which is shown as the red line, but there is a prominent cluster of outliers when Fat is between 40 and 50, but predictions are between 50 and 60, which shows that for particular high values, the model is overestimating in all cases.

Assignment 2

Task 1: Reading & Preparing Data.

```
bank <- read.csv2("bank-full.csv", header = TRUE)
#remove duration
df <- bank[,-12]
# changing target to factor
df$y <- factor(df$y, levels=c("yes","no"))
#most other variables are supposed to be factor variables as well
for(i in c(2:5,7:11,15)){
  df[,i] <- as.factor(df[,i])
}

n=dim(df)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=df[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
```

```
valid=df[id2,]

id3=setdiff(id1,id2)
test=df[id3,]
```

Task 2: Fitting trees

```
library(tree)

misclass=function(X,X1){
  return(round(1-sum(diag(table(X,X1)))/length(X),4))
}

# Decision tree with default settings
tree1 <- tree(y~., data=train)
# summary(tree1)

# Decision tree with smallest allowed node size equal to 7000
tree2 <- tree(y~., data=train, control = tree.control(nobs = nrow(train), minsize = 7000))
# summary(tree2)

# Decision tree minimum deviance 0.0005
tree3 <- tree(y~., data=train, control = tree.control(nobs = nrow(train), mindev = 0.0005))
# summary(tree3)

train1 <- predict(tree1, type="class")
train2 <- predict(tree2, type="class")
train3 <- predict(tree3, type="class")

valid1 <- predict(tree1, newdata=valid, type="class")
valid2 <- predict(tree2, newdata=valid, type="class")
valid3 <- predict(tree3, newdata=valid, type="class")

errors_df <- data.frame('Training Error' = c(misclass(train$y, train1),
                                             misclass(train$y, train2),
                                             misclass(train$y, train3)),
                        'Validation Error' = c(misclass(valid$y, valid1),
                                               misclass(valid$y, valid2),
                                               misclass(valid$y, valid3)))

row.names(errors_df) <- c("Tree A","Tree B","Tree C")
errors_df
```

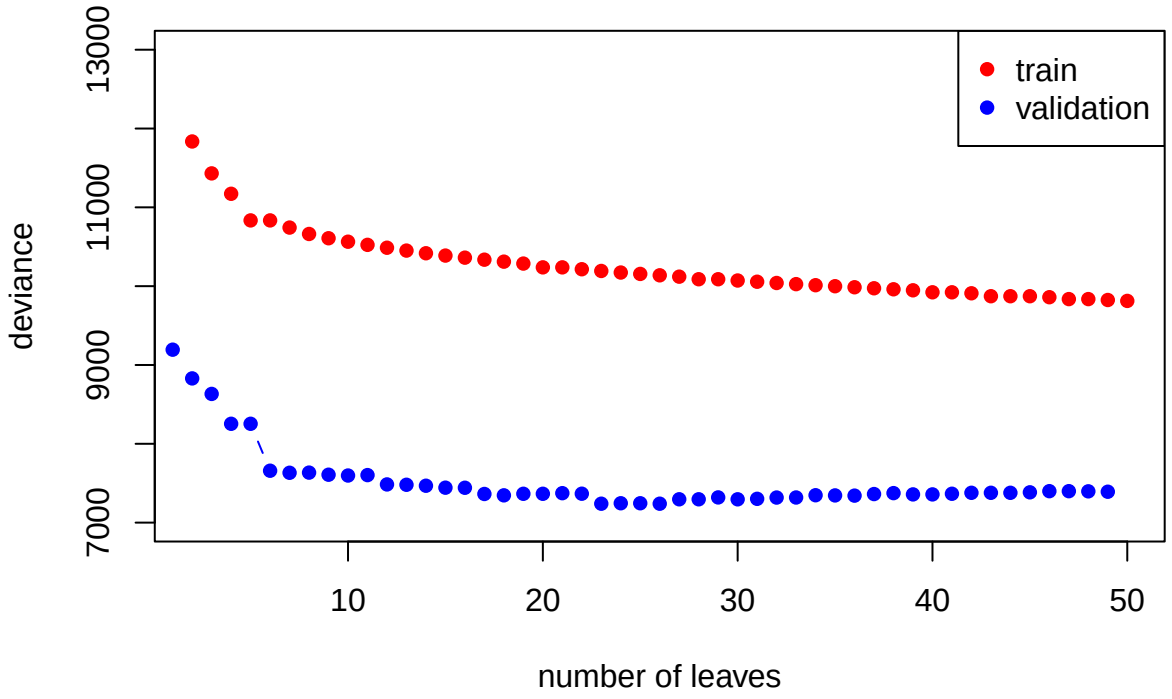
```
##        Training.Error Validation.Error
## Tree A         0.1048           0.1093
## Tree B         0.1048           0.1093
## Tree C         0.0912           0.1149
```

If one looks at the misclassification rates of the training data set, the last tree performs best. Nevertheless, taking the misclassification rates of the validation dataset into account, the last tree performs worst. This could show us that the last tree might overfit a bit as it is better on the training dataset. There is no difference visible between the first two trees. The second tree has one splitless; we would prefer that one because it is simpler and still achieves the same results.

Changing the minimal node size to 7000 (default value is 10) reduces the tree a little bit (one split less than before) while setting the minimum deviance to 0.0005 (default value is 0.01) builds a noticeable bigger tree. As we have a bigger value for the minimal node size, it makes sense that the tree stops growing earlier as the requirement is not fulfilled anymore afterwards, while reducing the minimum deviance lets the tree grow bigger. The value specified for the deviance shows how big the value for a specific node that is supposed to be split has to be compared to the root node for allowing another split. So the smaller this value is, the more likely it is that we can have another split, and hence the tree grows more significant with a smaller value.

Task 3: Optimal tree depth for decision tree with smallest allowed node size equal to 7000.

```
n <- 50
trainScore=rep(0,n)
validScore=rep(0,n)

for(i in 2:n) {
  prunedTree=prune.tree(tree3,best=i)
  pred=predict(prunedTree, newdata=valid, type="tree")
  trainScore[i]=deviance(prunedTree)
  validScore[i]=deviance(pred)
}

plot(2:n, trainScore[2:n], type="b", col="red", pch=16,
     xlab="number of leaves",ylab="deviance",ylim=c(7000,13000))
lines(validScore[2:n], type="b", pch=16, col="blue")
legend("topright", legend=c("train","validation"), pch=16, col=c("red","blue"))
```




```
# which.min(validScore[-1]) optimal number of leafs 21
prunedTree_opt <- prune.tree(tree3,best=which.min(validScore[-1])) #23 nodes
# summary(prunedTree_opt)

# plot(prunedTree_opt)
# text(prunedTree_opt,pretty = 0)
```

```
leafs <- list(prunedTree_opt$frame$var)

sort(table(leafs), decreasing = TRUE)
```

```
## leafs
##    <leaf>       day     month       age     pdays       job   housing   contact
##        23         8         5         2         2         1         1         1
##  campaign  poutcome   marital education   default   balance      loan  previous
##         1         1         0         0         0         0         0         0
```

The graph shows that the deviance is higher for very small numbers of leaves and decreases for the training dataset up to 50 trees. For the validation dataset, it decreases as well. However, it increases a tiny bit again after around 21 leaf nodes, which might lead to the conclusion that having more than 21 leaves might lead to an overfit to the training data.

The optimal amount of leaves seems to be 21 as it reduces the deviance of the validation dataset to a minimum and is the smallest value with this deviance. The simpler the model, the better.

One can see that the variable month has been used 7 times, and the variable pdays has been used 4 times within the tree building process. So both variables seem to be quite significant and helpful for splitting the data into the two desired groups. Nevertheless, one must also keep in mind at which point in the tree the respective variables are used as the more critical splits are done initially. Considering this, especially the variable poutcome has to be named, which is used in the first step and hence especially important for the splitting process. Other variables used for the splitting are age, day, balance, housing and contact. The rest of thevariables are not used in our optimal tree.

Task 4: Confusion Matrix, Accuracy, F1 Score

```
mis_test_opt <- predict(prunedTree_opt, newdata=test, type="class")
confusion_matrix <- table(test$y, mis_test_opt)

# Accuracy = TP+TN/TP+FP+FN+TN
accuracy <- ((confusion_matrix[1,1] + confusion_matrix[2,2])/ sum(confusion_matrix))*100

# Precision = TP/TP+FP
precision <- confusion_matrix[2,2]/sum(confusion_matrix[1,2]+confusion_matrix[2,2])

# Recall = TP/TP+FN
recall <- confusion_matrix[2,2]/sum(confusion_matrix[2,1]+confusion_matrix[2,2])

# F1 Score = 2*(Recall * Precision) / (Recall + Precision)
f1_score <- 2*(recall*precision)/(recall+precision)
```

Considering the accuracy of around 90%, the model seems to perform exceptionally well but not great. Looking at the confusion matrix, one can see that there are many more wrong classified cases to “no” that is “yes” in the real dataset than the other way around. If one thinks about the underlying problem, that would be a bad trade-off for the bank as they would have noticeable less “yes” with the prediction and would lose all the potential people that could have a deposit with them as obviously their desire is the receive as many deposits as possible. As we have a pretty imbalanced problem here, the F_1 score might, in general, be the better choice compared to the accuracy. However, the value for f_1 is quite bad; with the problem we just described, we should think about different weights for the different types of mistakes as one is a lot worse than the other: It might be okay to call a few people you do not get money from in the end while it might be pretty bad not to call the people potentially bringing lots of money with them. Hence, in the next part, this is given more weight.

Task 5: Perform a decision tree classification of the test data with a weighted loss matrix.

```
pred_loss <- as.data.frame(predict(prunedTree_opt, newdata=test, type="vector"))

pred_loss$ratio <- factor(ifelse(pred_loss$no / pred_loss$yes >= 5, "no", "yes"), levels = c("yes", "no

conf_matrix_loss <- table(test$y, pred_loss$ratio)
knitr::kable(conf_matrix_loss, caption = "Test data (rows=real values, cols=prediction)")
```

Table 3: Test data (rows=real values, cols=prediction)

	yes	no
yes	792	793
no	938	11041

Compared to the previous confusion matrix, a few changes can be observed. As we were punishing the false negatives more now, the number of false negatives has reduced noticeably because they were weighted a lot more (before 1255, now 793). On the other hand, false positives have increased quite a lot (before 196, now 938). Additionally, we have many more true positives (before 330, now 792) than before and only a few true negatives less than before (before 11783, now 11041).

Suppose one thinks about that in the context of the problem that would mean that there are a lot fewer people of the ones that will have the deposit with them that they "miss" because the prediction would not have recommended to call them. To achieve that, the number of people they would call after the prediction and be unsuccessful in the real world would increase quite a lot, and can be seen as the cost they need to pay to avoid losing the other people. Nevertheless, another positive effect is that many more people who will leave the deposit have also been predicted, so the actual positive rate is a lot higher.

```
paste("Accuracy for test data:", round((conf_matrix_loss[1,1] + conf_matrix_loss[2,2])/sum(conf_matrix_
```

```
## [1] "Accuracy for test data: 87.24 %"
```

```
prec <- conf_matrix_loss[1,1]/(conf_matrix_loss[2,1] +
                                 conf_matrix_loss[1,1])
recal <- conf_matrix_loss[1,1]/(conf_matrix_loss[1,2] +
                                  conf_matrix_loss[1,1])

f1 <- 2*prec*recal/(prec+recal) # not the best value
paste("F_1 score for test data:", round(f1, 4))
```

```
## [1] "F_1 score for test data: 0.4778"
```

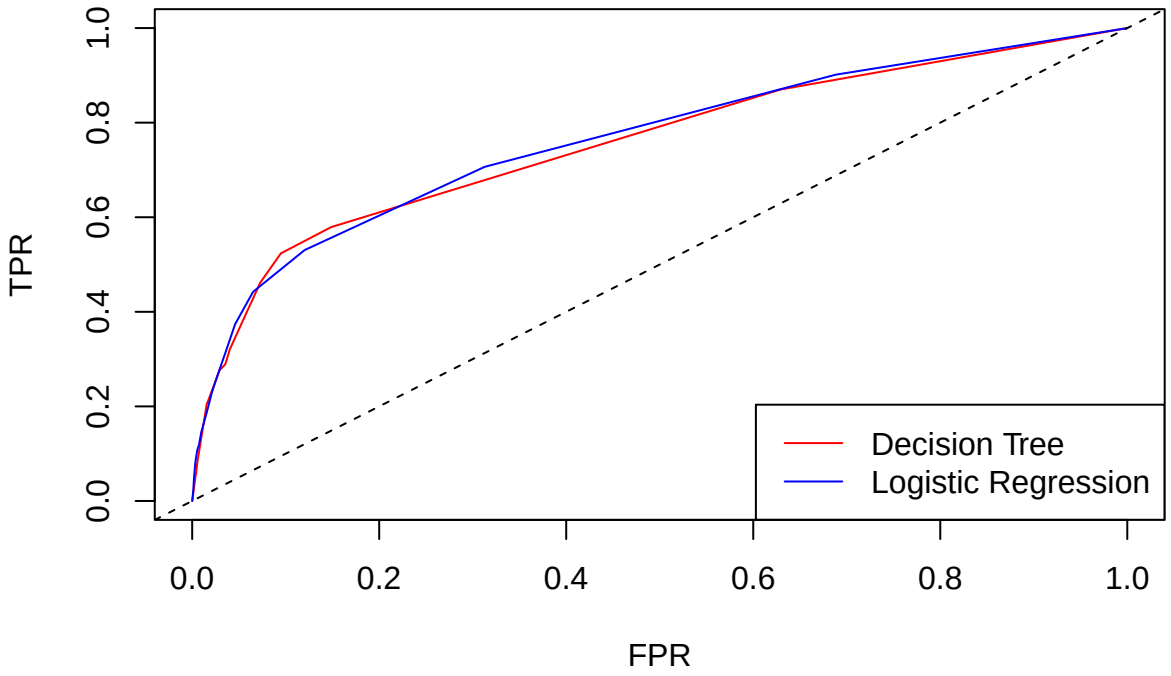
Comparing the F_1 score and accuracy, one can see that the accuracy is a bit smaller for this model, but the F_1 score increased a bit and is therefore better. As we said earlier the F_1 score is probably the better choice in this imbalanced problem. Together with the argumentation concerning the context of the problem above, we can be happy with this improvement and prefer this model to the previous one.

Task 6: Use the optimal tree and a logistic regression model to classify the test data by using different thresholds and plot the corresponding ROC curves.

As it is not completely clear whether the optimal tree is the one from step 3 or step 5, we decided to stick to the optimal tree from step 3 as it has the highest accuracy. One could have also picked the one from step 5 as it is more suitable for our problem.

```r
mySeq <- seq(0.05,0.95,by=0.05)
TPR <- numeric(length = length(mySeq))
FPR <- numeric(length = length(mySeq))

# for the first four sequences we can not use the following loop
# as everything is classified to the "no" group, so we couldn just set
# TPR and FPR to 0

TPR[1:4] <- 0
FPR[1:4] <- 0

# for the rest we do it step by step
for(i in 5:length(mySeq)){
  p <- as.factor(ifelse(predict(prunedTree_opt,
                                newdata=test,
                                type="vector")[,2] > mySeq[i],
                        "1","0"))

  conf <- table(test$y, p)

  TPR[i] <- conf[1,1]/(conf[1,1] + conf[1,2])
  FPR[i] <- conf[2,1]/(conf[2,1] + conf[2,2])
}

# connecting to 1 in the end as it would look weird if they end in different positions (default roc cur
TPR[length(mySeq) + 1] <- 1
FPR[length(mySeq) + 1] <- 1

# logistic regression
myGlm <- glm(y~.,data = train, family = "binomial")

TPR_l <- numeric(length = length(mySeq) + 1)
FPR_l <- numeric(length = length(mySeq) + 1)

for(i in 1:length(mySeq)){
  p <- as.factor(ifelse(predict(myGlm,
                                newdata=test,
                                type="response") > mySeq[i],
                        "1","0"))

  conf <- table(test$y, p)

  TPR_l[i] <- conf[1,1]/(conf[1,1] + conf[1,2])
  FPR_l[i] <- conf[2,1]/(conf[2,1] + conf[2,2])
}

TPR_l[length(mySeq) + 1] <- 1
FPR_l[length(mySeq) + 1] <- 1

plot(FPR, TPR, type = "l", col = "red", ylim = c(0,1), xlim = c(0,1),
     ylab="TPR",xlab="FPR")
lines(FPR_l, TPR_l, col = "blue")
abline(a = 0, b = 1, lty = 2, col = "black")
```

```
legend("bottomright", legend=c("Decision Tree", "Logistic Regression"),
       col=c("red","blue"), lty = 1)
```



Both curves for the Decision Tree and the Logistic Regression are pretty similar, and both of them are pretty bad. A good ROC curve would be close to the top of the plot, and both of them are closer to the angle bisector.

Why could the precision-recall curve be a better option here?

As we have already mentioned earlier, we have an imbalanced problem here. The precision-recall curve is usually more informative for imbalanced problems as the FPR in the roc-curve is prone to the significant size difference between the two classes.

Assignment 3

Task 1: Reading & Preparing Data

```
data <- read.csv("communities.csv")

data_scale <- data.frame(scale(data[,-101]))
data_scale$ViolentCrimesPerPop <- data$ViolentCrimesPerPop

cov_matrix <- cov(data_scale)
eig <- eigen(cov_matrix)

counter <- which(cumsum(eig$values/sum(eig$values) * 100) >= 95)
counter[1]
```

```
## [1] 35
```

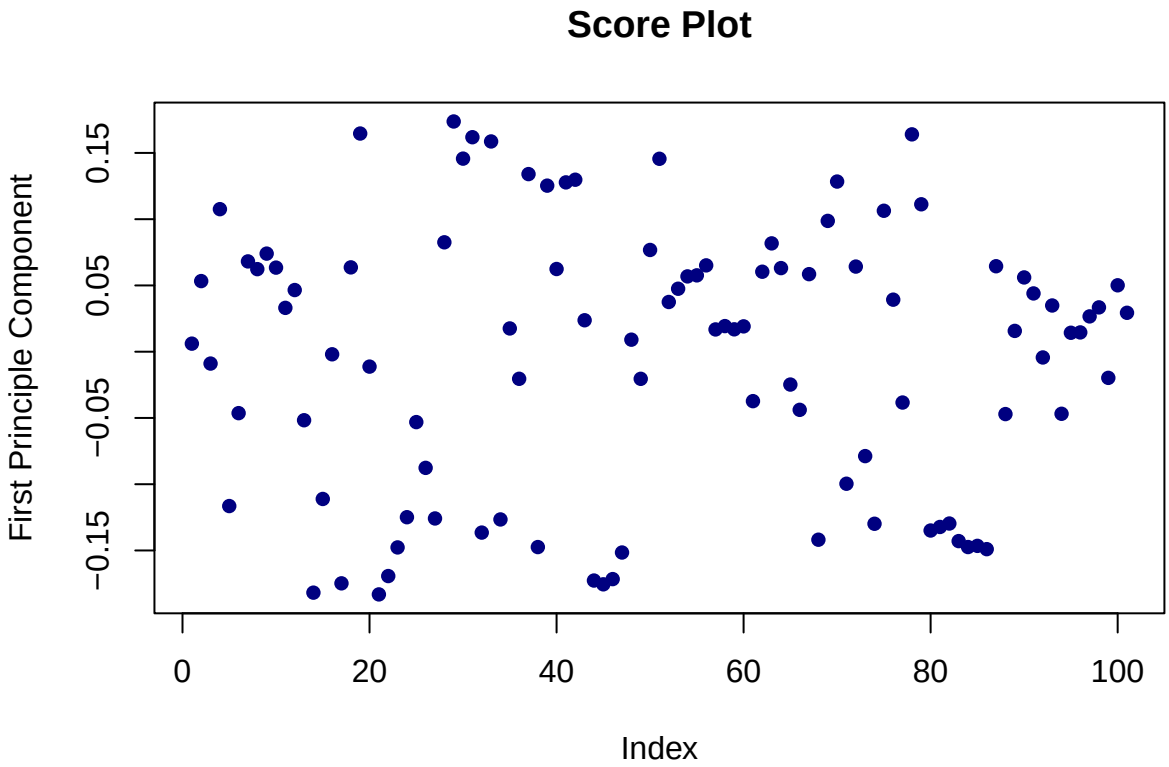
```
prop_var <- sprintf("%2.3f", eig$values/sum(eig$values)*100)
prop_var[c(1:2)]
```

```
## [1] "25.025" "16.931"
```

We need at least 35 components to account for 95% of the variance present in the original scaled data. The first component explains 25.02% of the variance, and the second explains 16.94% of the variance, giving them a total of 41.95% of variance explained together.

Task 2: Fitting PCA

```
pca <- princomp(data_scale)

plot(pca$loadings[,1], main = "Score Plot", pch = 16, col = "navy", ylab = "First Principle Component")
```



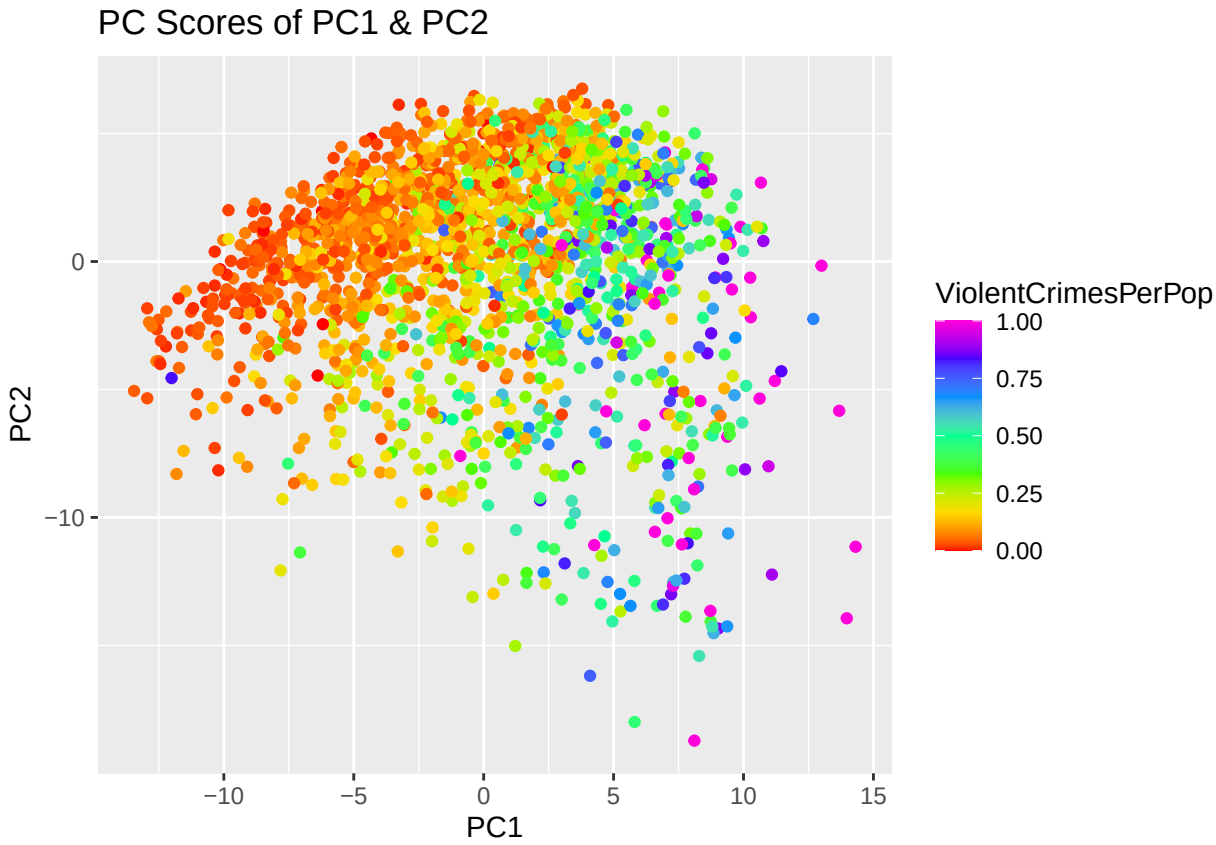
There are quite a lot of variables that contribute some information to the first principal component. There are, for instance, 40 variables that have a loading ≥ 0.1. The 5 variables that contribute the most can be seen below:

```
PC1_load <- sort(abs(pca$loadings[,1]), decreasing = T)
sum(PC1_load >= 0.1) # 40
head(PC1_load, 5)
```

They are the following: * medFamInc: median family income (differs from household income for non-family households) (**Loading: -0.183**) * medIncome: median household income (**Loading: -0.182**) * PctKids2Par: percentage of kids in family housing with two parents (**Loading: -0.176**) * pctWInvInc: percentage of households with investment / rent income in 1989 (**Loading: -0.175**) * PctPopUnderPov: percentage of people under the poverty level (**Loading: 0.174**)

These variables are related to household economic situations and poverty. Therefore, they can be seen as indicators of socio-economic levels in the different communities. The percentage of people under the poverty line has a logical relationship with increased levels of crime. Median family income and median household income are highly correlated and contain the same information (collinear). They negatively correlate with the principal component, meaning that higher median income results in a lower PCA score. The percentage of kids in family housing with two parents can be seen as an indicator of socio-economic levels in a community. In general, two parents can contribute more to the family economy. The percentage of households with investment income in 1989 is a direct indicator of the economic situation in the community.

```
df_scores <- data.frame(pca$scores[,1:2])
df_scores$ViolentCrimesPerPop <- data$ViolentCrimesPerPop

library(ggplot2)
ggplot(data = df_scores, aes(x=Comp.1, y=Comp.2, color = ViolentCrimesPerPop)) +
  geom_point() +
  labs(title = "PC Scores of PC1 & PC2") + xlab("PC1") + ylab("PC2") +
  scale_colour_gradientn(colors=rainbow(7))
```



It is possible to identify a trend in ViolentCrimesPerPop given the two principal components. Higher values on PC1 has a moderately strong positive relationship with ViolentCrimesPerPop. PC2 seem to have a negative relationship with ViolentCrimesPerPop, but it seems weaker than the first component. Finally, an interesting outlier has a low score on PC1 and a high value on ViolentCrimesPerPop.

Task 3: Scale the original data and split into training and test (50/50) and estimate a linear regression model.

In terms of predictions, the training and testing error is relatively low. It seems that using all of these variables, it is possible to create a model that can reasonably accurately predict ViolentCrimesPerPop. There are, however, many predictors in the model that are unnecessary since they cannot be significantly differentiated from 0. There is most likely multicollinearity in the model and low interpretability of the feature's effects. From a prediction perspective, the model can be deemed adequate. Having a higher MSE for the test data than for the training data shows a tendency to overfitting.

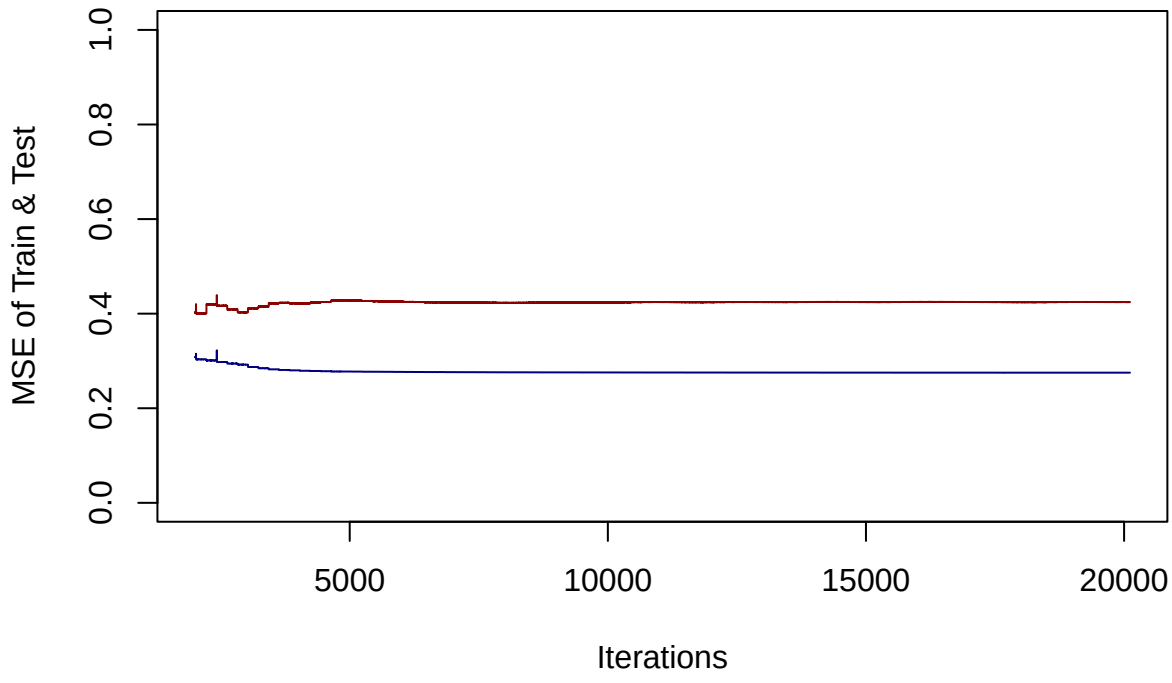
```
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data.frame(data[id,])
```

```r
test=data.frame(data[-id,])

mean_train <- numeric(ncol(train))
sd_train <- numeric(ncol(train))

for(i in 1:ncol(train)){
  mean_train[i] <- mean(train[,i])
  sd_train[i] <- sd(train[,i])
}

train <- data.frame(scale(train))

test_new <- data.frame(matrix(nrow = nrow(test), ncol = ncol(test)))

for(i in 1:ncol(train)){
  test_new[,i] <- (test[,i] - mean_train[i])/sd_train[i]
}

test <- as.data.frame(test_new)
colnames(test) <- colnames(train)

lrm <- lm(ViolentCrimesPerPop~., data = train)

#summary(lrm)

train_mse <- mean((train$ViolentCrimesPerPop-predict(lrm))^2)
test_mse <- mean((test$ViolentCrimesPerPop-predict(lrm,test))^2)
```

Task 4

```r
my_cost <- function(beta,input_data){
  X <- as.matrix(input_data[,101])
  Y <- as.matrix(input_data[,-101])

  cost <- mean((X - Y%*%beta)^2)

  return(cost)
}

params <- list()
mse_train <- list()
mse_test <- list()
k <- 0

my_function <- function(beta,train_data = train,test_data = test){
  .GlobalEnv$k = .GlobalEnv$k+1
  x <-  my_cost(beta,train_data)
  .GlobalEnv$mse_train[[k]] = x
  .GlobalEnv$mse_test[[k]] = my_cost(beta,test_data)
  return(x)
}

res <- optim( par = rep(0,100), fn = my_function, method = "BFGS")
```

```
plot(2000:k, mse_train[2000:k], col="navy", type="l", ylim = c(0,1),
     ylab = "MSE of Train & Test", xlab = "Iterations")
lines(2000:k, mse_test[2000:k], col="red4")
```



Task 4 (Theo Solution)

```
# Calculate cost given theta
cost_fun <- function(theta, data){
  X <- as.matrix(data[,-101])
  y <- data[,101]
  n <- length(y)
  stopifnot(length(theta) == ncol(X), length(y) == nrow(X))
  cost <- (1/n) * sum((X%*%theta - y)^2)
  return(cost)
}
```

Use BFGS optimization to optimize the cost and compute training and test errors for every iteration

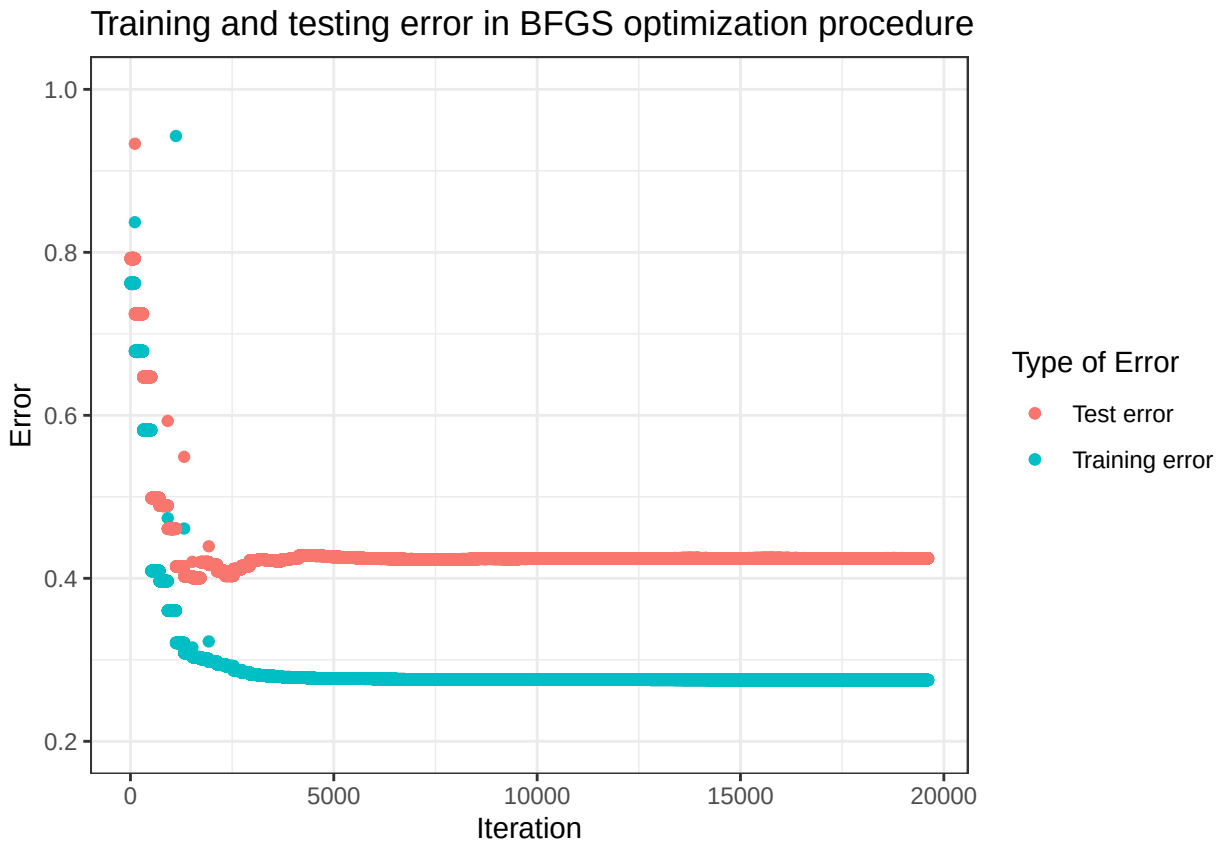
```
# Create new environment to store iterations
parameter_env <- new.env()
#parameter_env$iteration <- list()
# Create lists in new env to store data
parameter_env$cost_train <- list()
parameter_env$cost_test <- list()

# Wrapper function
cost.lm <- function(theta, training_data, testing_data){
  n <- length(parameter_env[["cost_train"]])
  #parameter_env[["iteration"]][[n+1]] <- theta
  cost_train <- cost_fun(theta, data = training_data) # Call cost on training
  parameter_env[["cost_train"]][[n+1]] <- cost_train # store cost train
  cost_test <- cost_fun(theta, data = testing_data) # call cost on testing
```

```
  parameter_env[["cost_test"]][[n+1]] <- cost_test # store cost test
  return(cost_train)
}

# Optimize train cost function through optimizing the wrapper function
opt <- optim(rep(0,100), fn = cost.lm,
             training_data = train,
             testing_data = test,
             gr = NULL, method = "BFGS")

# Gather errors and discard first 500 iterations.
train_errors <- as.numeric(do.call(rbind, parameter_env[["cost_train"]]))[-c(1:500)]
test_errors <- as.numeric(do.call(rbind, parameter_env[["cost_test"]]))[-c(1:500)]
```

Present a plot showing the dependence of both errors on the iteration number

The following plot shows the dependence of training and testing error in iterations in the BFGS procedure above. The y-axis has been limited to show only values between [0.2, 1] to better display the increase in test error after many iterations. As shown in the plot, implicit regularization seems to be a good idea for this problem, as the testing error increases after about 2000 iterations. Finding the minimum cost function for the training data leads to overfitting the training data. By stopping early, it is possible to achieve a model better at generalizing to the test data.

```
# Plot
ggplot() + geom_point(aes(x = 1:length(train_errors), y = train_errors, color = "Training error")) +
  geom_point(aes(x = 1:length(test_errors), y = test_errors, color = "Test error")) + theme_bw() +
  labs(title = "Training and testing error in BFGS optimization procedure",
       color = "Type of Error", y = "Error", x = "Iteration") + ylim(c(0.2,1))
```



Find the optimal iteration number according to early stopping criterion

```
# Find optimal set of parameters
optim_ind <- which.min(test_errors)
optimal_train <- train_errors[optim_ind]
optimal_test <- test_errors[optim_ind]

optim_errors <- data.frame("Training Error" = c(train_mse, optimal_train),
                           "Test Error" = c(test_mse, optimal_test))
row.names(optim_errors) <- c("Fully optimized model",
                             "Early stopping model")
optim_errors
```

```
##                        Training.Error Test.Error
## Fully optimized model       0.2752071  0.4248011
## Early stopping model        0.3032999  0.4002329
```

```
cat("Optimal iteration is", optim_ind + 500)
```

```
## Optimal iteration is 2182
```

The model when optimization of $\underset{\theta}{\text{argmin}} \frac{1}{n}||\boldsymbol{X\theta} - \boldsymbol{y}||_2^2$ is stopped early is better than the model when the optimization is run until convergence. The fact that the test error increases by the number of iterations are an indication of overfitting to the training data. The differences are, however, fairly low in this example. Still, implicit regularization here generates a model that is better at generalizing to new data.

Lab03

Assignment 1: Kernel Methods

```
library(geosphere)

set.seed(1234567890)
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")

#all data together
df <- merge(stations,temps,by="station_number")
```

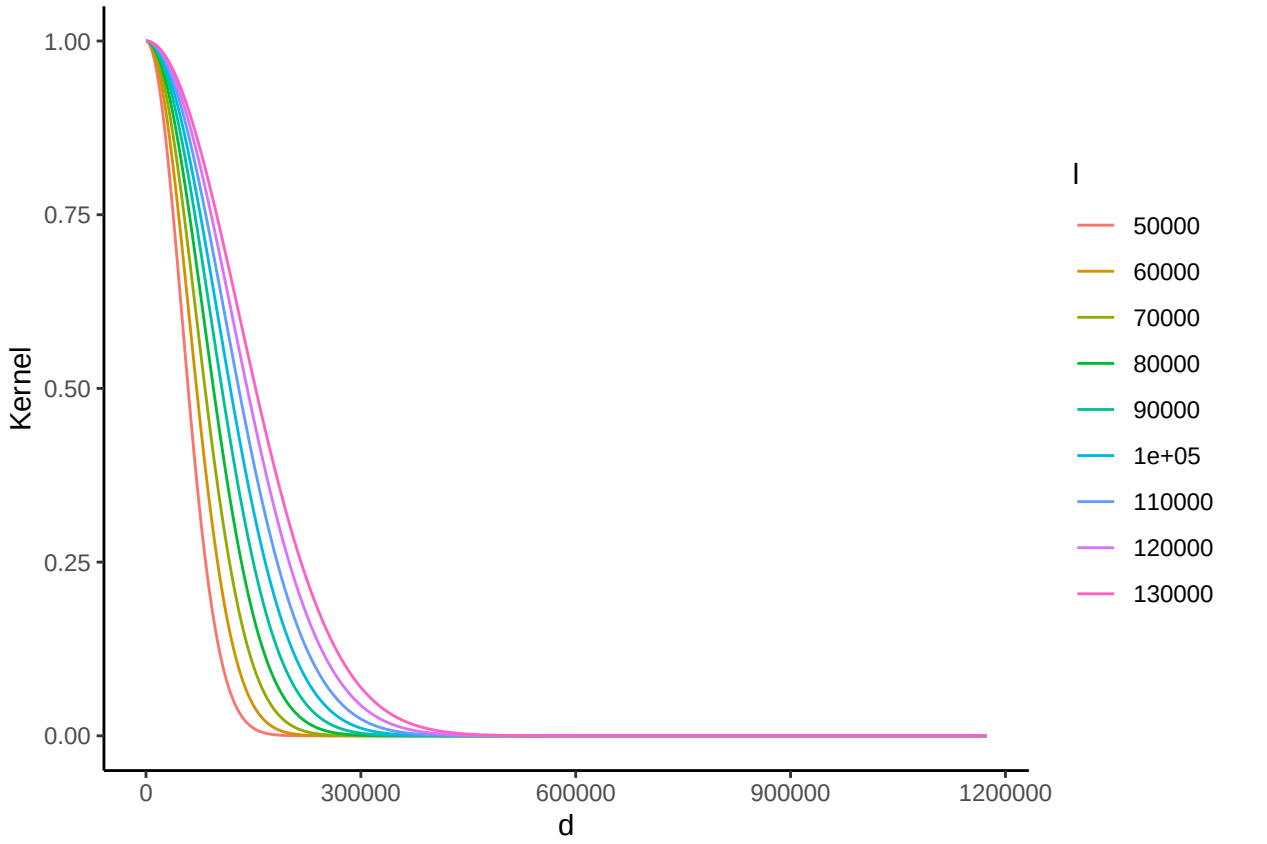
```
# date we are choosing (can make a comparison to Holma station
# later and we need to define the date now for filtering data)
date <- as.Date("1990-01-29")

# making date in the dataset a date.object to use difftime to filter out all
# data from this day and after
df$date <- as.Date(df$date)
oldDf <- df #later comparison
df <- df[-which(difftime(date, df$date) <= 0),]
```

The first kernel to account for the physical distance from a station to the point of interest.

First, we get our desired coordinates from google maps and then calculate the distance using the *distHaversine()* function. Next, we plot different width values between 50000 and 130000. We pick $l = 100000$ even though this is a quite random choice as many numbers around that area would have been a reasonable choice as well. We picked it for the reason that all points with a distance greater than 300km from our observation have

close to 0 weight now, and that seemed reasonable to us.

```
# place we are going to pick is Tyrislöt (points from google maps)
tyris <- c(16.89461,58.32633)

# calculate distance
d1 <- sapply(1:nrow(df), function(x) distHaversine(tyris,
                                                   c(df[x,]$longitude,
                                                     df[x,]$latitude)))

# plot different values for l
# x axis from zero up to maximal difference (max(d1))
h1 <- seq(0,max(d1),by=1000)

# how many different values do we wanna try and which sequence
m1 <- 9
mySeq1 <- seq(50000,130000,length.out=m1)

# calculate kernel for different values of l
results1 <- lapply(mySeq1, function(l) exp(-h1^2/(2*l^2)))
plot_df1 <- data.frame("Kernel"=unlist(results1),
                       "l"=rep(mySeq1,
                               each=length(h1)),
                       "d"=rep(h1, times=length(m1)))
plot_df1$l <- as.factor(plot_df1$l)
ggplot(plot_df1, aes(x=d,y=Kernel,col=l)) +
  geom_line() +
  theme_classic()
```




```
# pick 100000
# apply to observations
k1 <- exp(-d1^2/(2*100000^2))
```

The second kernel to account for the distance between the day a temperature measurement was made and the day of interest.

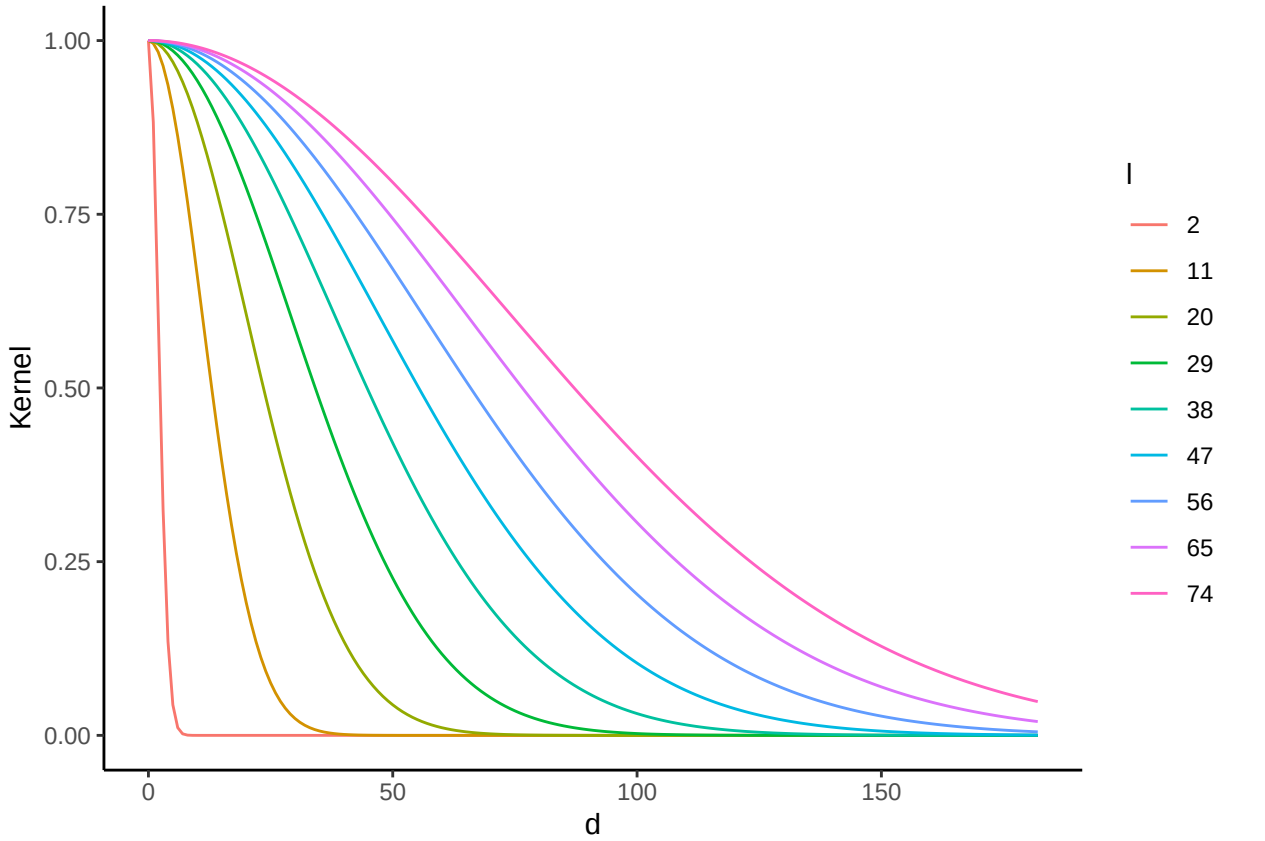
First, we have to decide whether we make a total difference of the day to previous measures or compare within one year and ignore which year the measure was taken. We decide that the second attempt would be more reasonable as it considers seasons in a better way. Doing this, we might ignore the climate change impact over the years, which would be a reason for considering the total day difference. However, as this probably has a quite small impact on the few measurements, we have decided that the difference within a year would be more reasonable.

Using modulo and dividing by the total number of days per year, we receive the difference in days within one year. Considering that we have leap years every four years, we use 365.25 instead of 365 to get a better approximation. We could also round the results we achieve, only having full days as differences but decide to stick to the not rounded numbers as we do not need exact numbers and are more precise using these numbers. Additionally, one must keep in mind that we have the situation that December and January are right after each other in the same way as December and November are. Hence, we need to consider that there are always two "directions" in which the time difference can be calculated. Whenever our differences exceed half a year (182.625 days), we go the other way, which is shorter then and subtract the difference we calculated from the total number of days.

For this kernel, we pick $l = 11$ as all observations that are more than a month away get around zero weight.

```
# date we are choosing is the 29th of January 1990

# calculate difference
```

```r
diff <- as.numeric(difftime(date, df$date)) %% 365.25
d2 <- ifelse(diff <= 182.625, diff, 365.25-diff)

# plot different values for l
# x axis from zero up to maximal difference (max(d2))
h2 <- seq(0,max(d2),by=1)

# how many different values do we wanna try and which sequence
m2 <- 9
mySeq2 <- seq(2,74,length.out=m2)

# calculate kernel for different values of l
results2 <- lapply(mySeq2, function(l) exp(-h2^2/(2*l^2)))
plot_df2 <- data.frame("Kernel"=unlist(results2),
                       "l"=rep(mySeq2,
                               each=length(h2)),
                       "d"=rep(h2, times=length(m2)))

plot_df2$l <- as.factor(plot_df2$l)
ggplot(plot_df2, aes(x=d,y=Kernel,col=l)) +
  geom_line() +
  theme_classic()
```



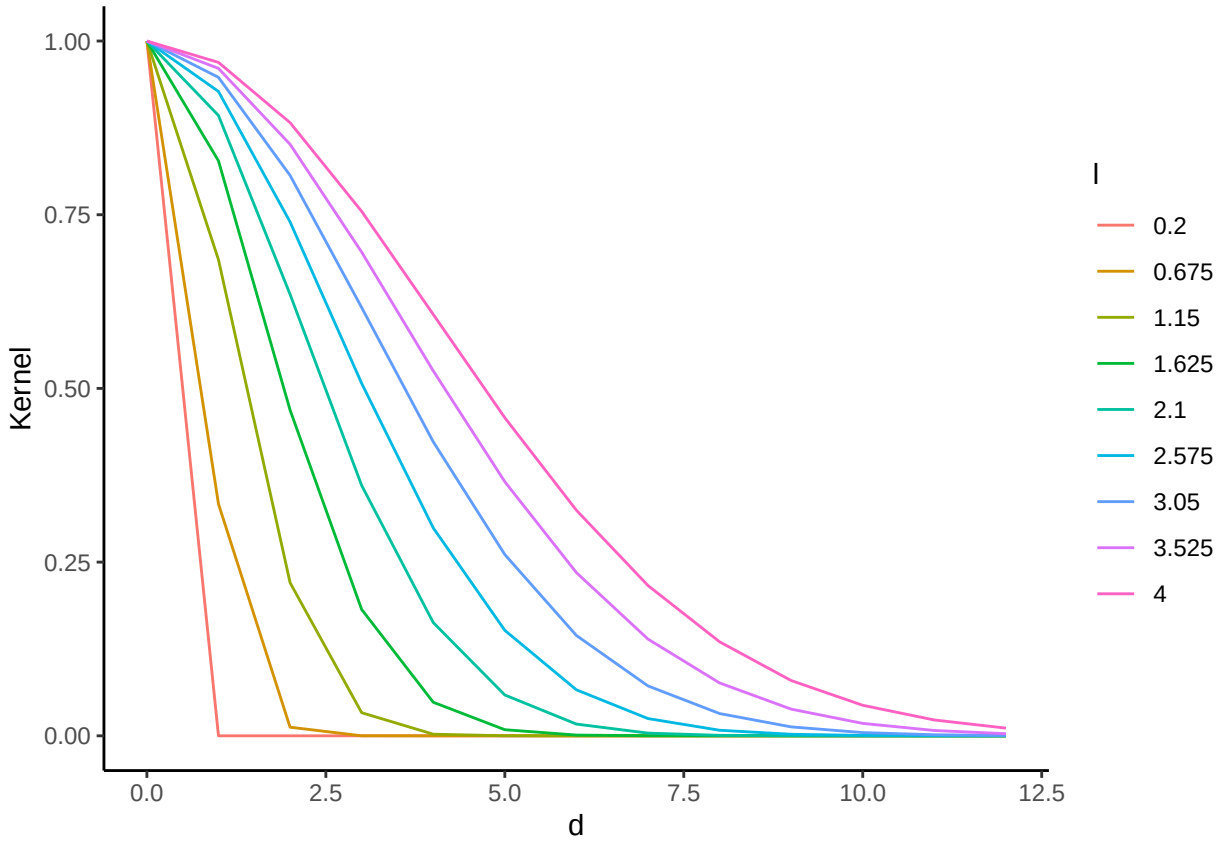

```r
# pick 11
# apply to observations
k2 <- exp(-d2^2/(2*11^2))
```

The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

For calculating the time difference during each day, we use a similar attempt to the previous one. However, this time, we check whether the difference is more than 12 hours to use the shorter difference if that is the case.

We pick $l = 1.15$ as a width value because it gives close to zero weight to everything that is more than three hours away.

```
# defining times we want estimations for
times <- c(paste0("0",seq(4,8,by=2),":00:00"),
           paste0(seq(10,24,by=2),":00:00"))
times <- strptime(times, format = "%H:%M:%S")

# change times in dataframe to time object
df$timeD <- strptime(df$time, format = "%H:%M:%S")
diff <- sapply(1:length(times), function(x){
  abs(as.numeric(difftime(times[x], df$timeD), units="hours"))
})
d3 <- ifelse(diff <= 12, diff, 24-diff)

# plot different values for l
# x axis from zero up to maximal difference (max(d1))
h3 <- seq(0,max(d3),by=1)

# how many different values do we wanna try and which sequence
m3 <- 9
mySeq3 <- seq(0.2,4,length.out=m3)

# calculate kernel for different values of l
results3 <- lapply(mySeq3, function(l) exp(-h3^2/(2*l^2)))
plot_df3 <- data.frame("Kernel"=unlist(results3),
                       "l"=rep(mySeq3,
                               each=length(h3)),
                       "d"=rep(h3, times=length(m3)))

plot_df3$l <- as.factor(plot_df3$l)
ggplot(plot_df3, aes(x=d,y=Kernel,col=l)) +
  geom_line() +
  theme_classic()
```



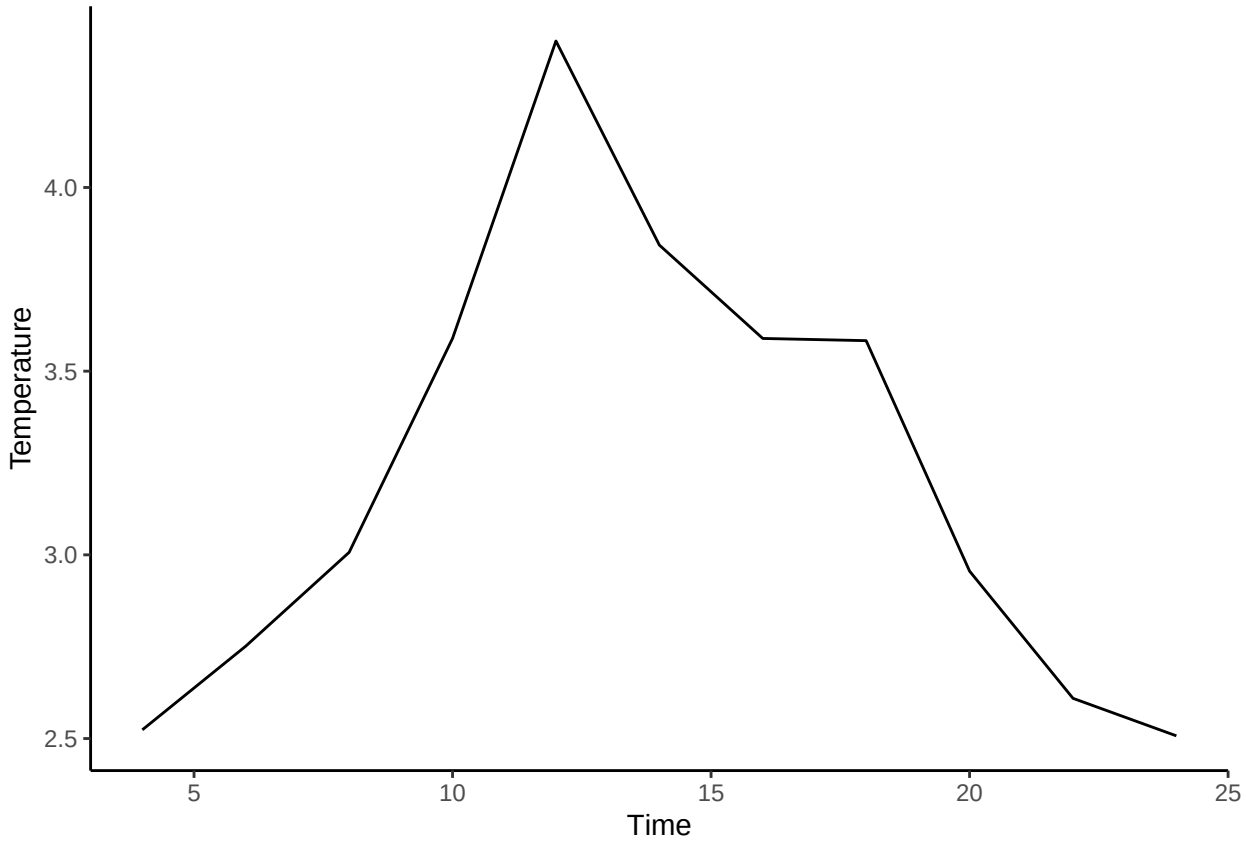

```
#pick 1.15
#apply to observations
k3 <- exp(-d3^2/(2*1.15^2))
```

Use a kernel that is the sum of three Gaussian kernels.

```
s <- sapply(1:length(times), function(x) sum((k1 + k2 + k3[,x])
                                             *df$air_temperature)/
               sum(k1,k2,k3[,x]))
(rs <- data.frame("Time"=seq(4,24,by=2),"Temperature"=s))
```

```
##    Time Temperature
## 1     4    2.523659
## 2     6    2.751415
## 3     8    3.006832
## 4    10    3.589137
## 5    12    4.398685
## 6    14    3.843309
## 7    16    3.589054
## 8    18    3.583031
## 9    20    2.955756
## 10   22    2.609468
## 11   24    2.507320
```

```
ggplot(rs, aes(y=Temperature,x=Time)) +
  geom_line() +
  theme_classic()
```

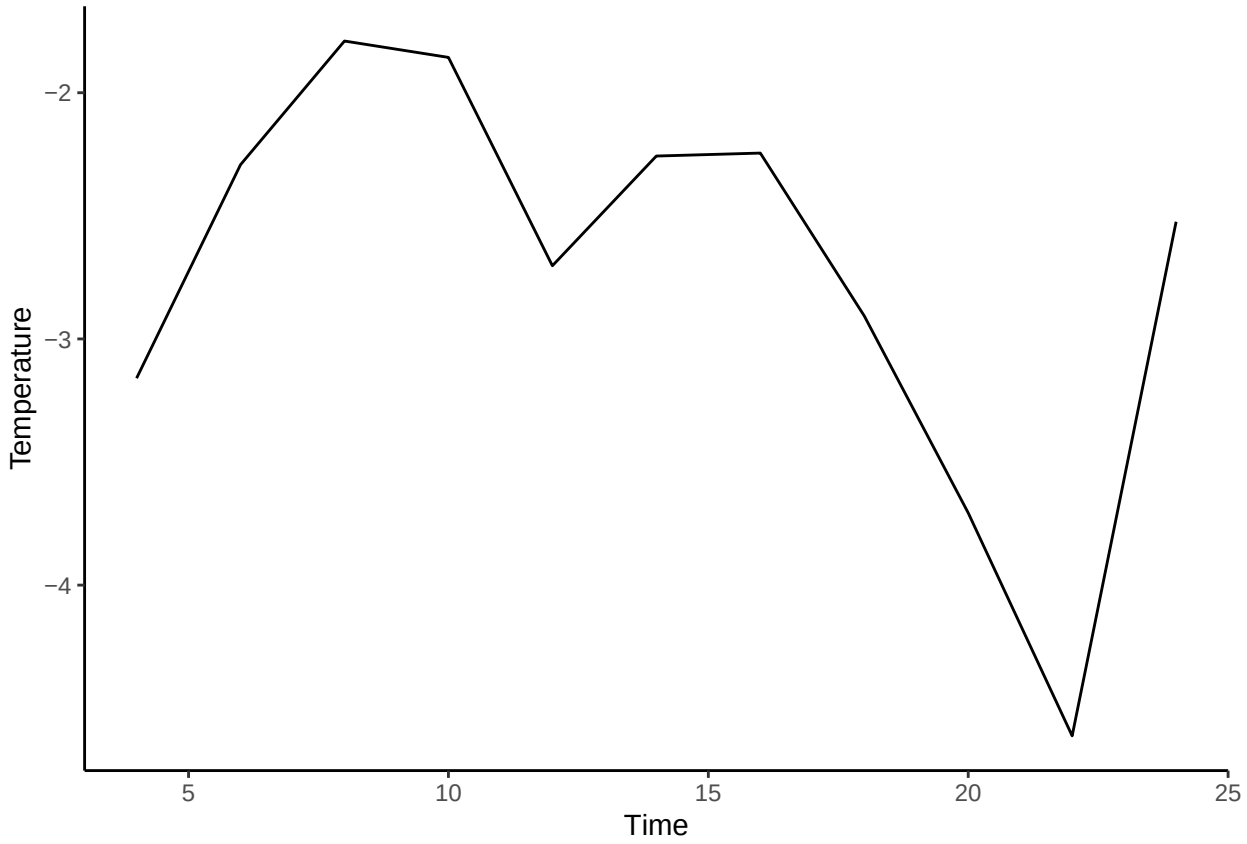


Combine the three kernels into one by multiplying them, instead of summing them up.

```
sM <- sapply(1:length(times), function(x) sum((k1 * k2 * k3[,x])
                                              *df$air_temperature)/
              sum(k1 * k2 * k3[,x]))

(rsM <- data.frame("Time"=seq(4,24,by=2),"Temperature"=sM))
```

```
##    Time Temperature
## 1     4   -3.159492
## 2     6   -2.292204
## 3     8   -1.789908
## 4    10   -1.856450
## 5    12   -2.702746
## 6    14   -2.257094
## 7    16   -2.245127
## 8    18   -2.906534
## 9    20   -3.706508
## 10   22   -4.612227
## 11   24   -2.523919
```

```
ggplot(rsM, aes(y=Temperature,x=Time)) +
  geom_line() +
  theme_classic()
```



Compare the results obtained in both cases and elaborate on why they may differ.

The results for both kernels differ noticeably in their sign as the prediction using multiplication only predicts negative values for the whole day. Of course, this could be right in January as well, so just looking at these values, we can not decide which prediction is more reasonable. However, we picked a date that has a good comparison value and considering this, and we can see that the summation works better in this case:

```
oldDf[which(oldDf$station_name=="Holma"&oldDf$date=="1990-01-29"),]
```

```
##       station_number station_name measurement_height latitude longitude
## 16770          86200        Holma                  2  58.3339   16.8157
##             readings_from         readings_to elevation       date     time
## 16770 1962-05-01 00:00:00 1996-11-30 23:59:59         5 1990-01-29 12:00:00
##       air_temperature quality
## 16770               5       G
```

Assignment 2: Support Vector Machines

```
set.seed(1234567890)
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
spam[,-58]<-scale(spam[,-58]) # Scale data except for response (type)
tr <- spam[1:3000, ] # Training data
va <- spam[3001:3800, ] # Validation data
trva <- spam[1:3800, ] # Training and validation data
te <- spam[3801:4601, ] # Testing data

by <- 0.3
# seq(0.3, 5, by = 0.3) -> 16 models with different values for the cost of
# constraints violation.
# C is the Its the regularization term in the Lagrange formulation
err_va <- NULL # Error on validation data
for(i in seq(by,5,by)){
  # Runs 16 different models with different C
  filter <- ksvm(type~.,data=tr,kernel="rbfdot",
                 kpar=list(sigma=0.05),C=i,scaled=FALSE)

  mailtype <- predict(filter,va[,-58]) # Predict on validation
  t <- table(mailtype,va[,58]) # confusion matrix
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t)) # error on validation data
}

# Some different models with C=3.9 (most optimal based on validation error)

### Training on training data
filter0 <- ksvm(type~.,data=tr,kernel="rbfdot",
                kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
## Predict validation
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)
err0 # Error on validation
```

```
## [1] 0.0675
```

```
### Training on training data
filter1 <- ksvm(type~.,data=tr,kernel="rbfdot",
                kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
## Predict testing
mailtype <- predict(filter1,te[,-58])
t <- table(mailtype,te[,58])
err1 <- (t[1,2]+t[2,1])/sum(t)
err1 # Error on testing
```

```
## [1] 0.08489388
```

```
### Training on training+validation data
filter2 <- ksvm(type~.,data=trva,kernel="rbfdot",
                kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
## Predict testing
mailtype <- predict(filter2,te[,-58])
t <- table(mailtype,te[,58])
err2 <- (t[1,2]+t[2,1])/sum(t)
```

```
err2 # Error on testing
```

```
## [1] 0.082397
```

```
### Training on all the data
filter3 <- ksvm(type~.,data=spam,kernel="rbfdot",
                kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
## Predict testing
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3 # Error on testing
```

```
## [1] 0.02122347
```

Questions

1. Which filter do we return to the user ? filter0, filter1, filter2 or filter3? Why?

2. What is the estimate of the generalization error of the filter returned to the user? err0, err1, err2 or err3? Why?

You would generally return filter3 as it uses all data for training and this would result in a better model. The estimated generalization error is given by err2. Note that err0 is optimistic (overfitting) since the validation data was used to select the model. So, err0 should not be used as estimate of the generalization error. On the other hand, err1 is a valid estimate since it is based on previously unseen data. However, it is pessimistic because the filter is trained on just tr. Finally, err3 is again optimistic because the filter's training data includes te. So, it shouldn't be used. In summary, a correct estimate is based on previously unseen data, and we should use the rest of the data available to train the filter (because only then the filter being evaluated will resemble the filter we return to the user).

3. Implementation of SVM predictions.

```
# 3. Implementation of SVM predictions.
sv<-alphaindex(filter3)[[1]] # indexes of the support vectors
co<-coef(filter3)[[1]] # the linear coefficients for the support vectors
inte<- - b(filter3) # the negative intercept of the linear combination

k <- NULL
rbfkernel <- rbfdot(sigma = 0.05)
for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset.
  k2<-inte
  for(j in 1:length(sv)){
    k2<- k2 + co[j] * rbfkernel(as.numeric(spam[i,-58]), as.numeric(spam[sv[j],-58]))
  }
  k<-c(k, k2)
}
k
```

```
##  [1] -1.998999  1.560584  1.000278 -1.756815 -2.669577  1.291312 -1.068444
##  [8] -1.312493  1.000184 -2.208639
```

```
# # Gaussian kernel
# Gkernel <- function(x, x_new, sigma = 0.05){
#   return(exp(-sigma*sum((x-x_new)^2)))
# }
```

```
#
# k<-numeric(10) # Vector to hold the predictions
# k2 <- matrix(0, ncol = length(sv), nrow = 10) # Matrix to hold the alpha_j * K(X_j, X_new)
# for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset.
#   for(j in 1:length(sv)){ # Using only the support vectors
#     # Take alpha_i times the kernel value of each support vector with the new observation
#     k2[i,j] <- co[j] * Gkernel(spam[sv[j],-58], spam[i,-58])
#   }
#   # Sum up all the kernelvalues scaled by alpha and add the intercept
#   k[i] <- sum(k2[i,]) + inte
# }

cat("Does our solution and predict() return the same function? Answer:",
    all.equal(k, as.numeric(predict(filter3,spam[1:10,-58], type = "decision"))))
```

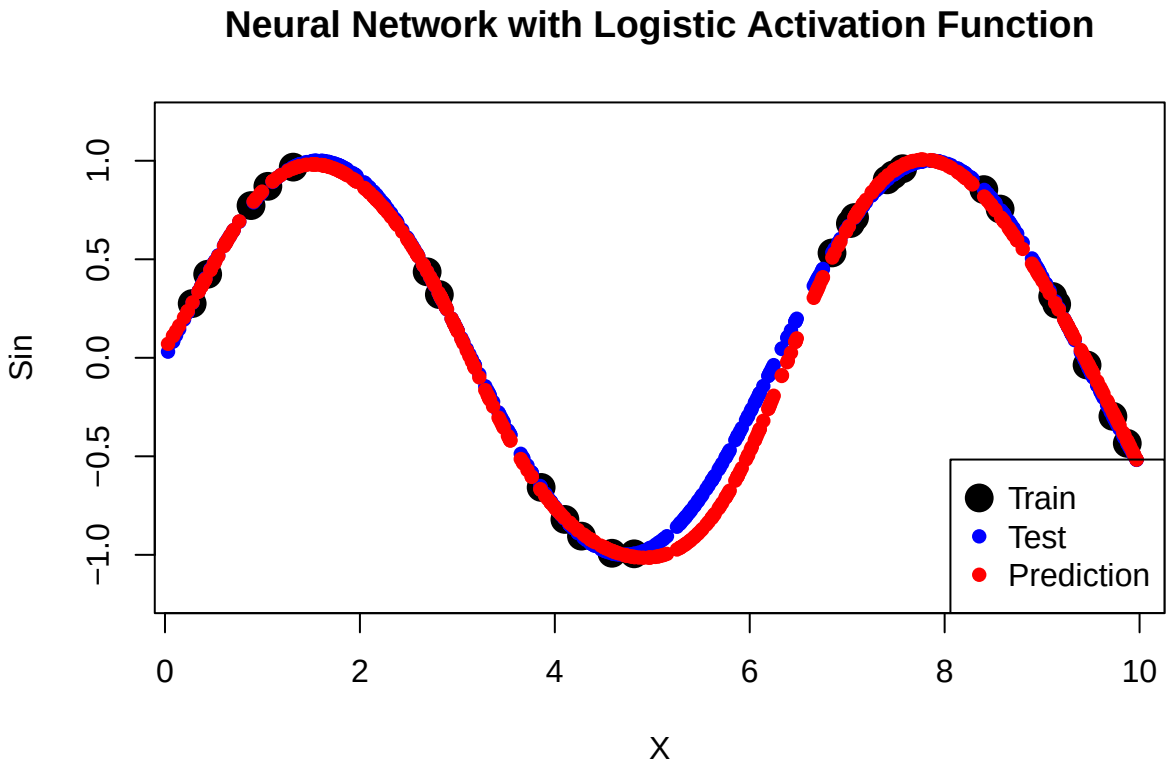
```
## Does our solution and predict() return the same function? Answer: TRUE
```

Assignment 3: Neural Networks

Task 1: Construct Data & Implement Neural Network

```
library(neuralnet)
set.seed(1234567890)
X <- runif(500, 0, 10)
data1 <- data.frame(X, Sin=sin(X))
train <- data1[1:25,] # Training
test <- data1[26:500,] # Test
```

```
nn <- neuralnet(Sin~X, hidden = 10, data = train) # default activation function logistic

pred_test <- predict(nn, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.2,1.2),
     main = "Neural Network with Logistic Activation Function")
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
```



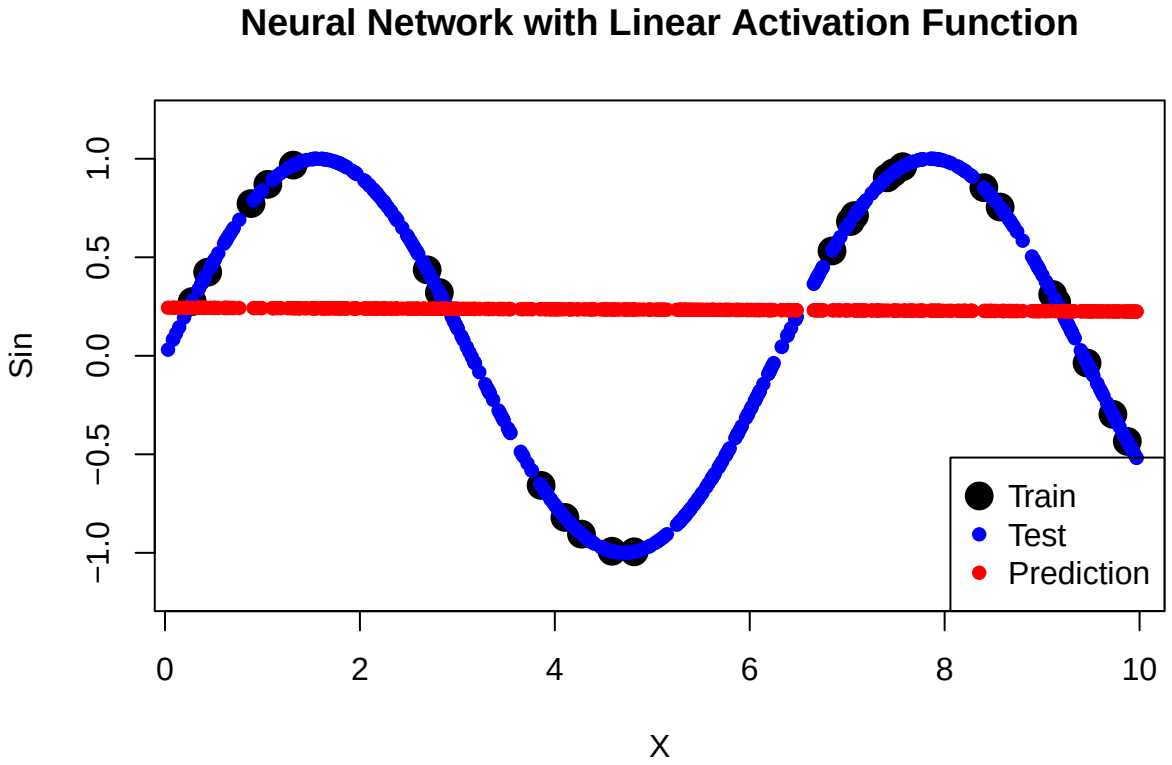
Using the logistic function, $f(x) = \frac{1}{1+e^x}$, as an activation function (default) the predictions of the learned model on the test data are relatively good. The only difference between the predicted points (red) and the actual point (black for train and blue for test) is when X has values between 5 and 6.5. This is due the fact that there are no training points in that interval.

The logistic activation function is the most commonly used for training neural network models where we have to predict the output as a probability. The sigmoid function is the most suitable choice because its range is between 0 and 1, the range of any probability. Moreover, the function is differentiable, and as a result, it provides a smooth gradient.

Task 2: Repeat question (1) with custom activation functions.

Linear activation function

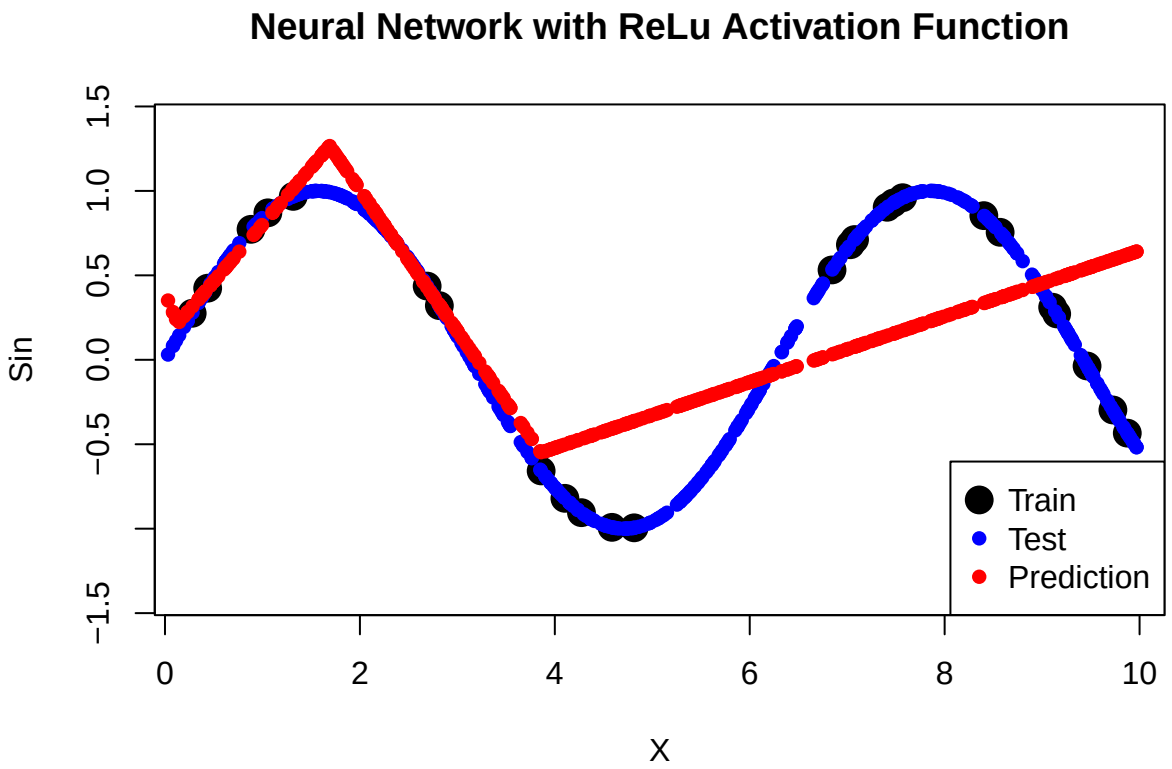
```
h1 <- function(x) {x}
nn1 <- neuralnet(Sin~X, hidden = 10, data = train, act.fct = h1)

pred_test_1 <- predict(nn1, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.2,1.2),
     main = "Neural Network with Linear Activation Function")
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test_1, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
```



The graph illustrates that the predictions of the learned neural networks on the test data are a straight line (red points), compared to the train (black points) and test (blue points) values, which correctly demonstrate the plot of sine. That happens because it is not possible to use backpropagation as the derivative of the linear function, $f(x) = x$, is a constant and has no relation to the input x. Thus, all neural network layers will collapse into one if a linear activation function is used. No matter the number of layers, the linear activation function turns the neural network into one layer.

ReLU activation function

```
h2 <- function(x){ifelse(x>=0,x,0)}
# h2 <- function(x){max(0,x)} # does not work.

nn2 <- neuralnet(Sin~X, hidden = 10, data = train, act.fct = h2)

pred_test_2 <- predict(nn2, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.4,1.4),
     main = "Neural Network with ReLu Activation Function")
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test_2, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
```

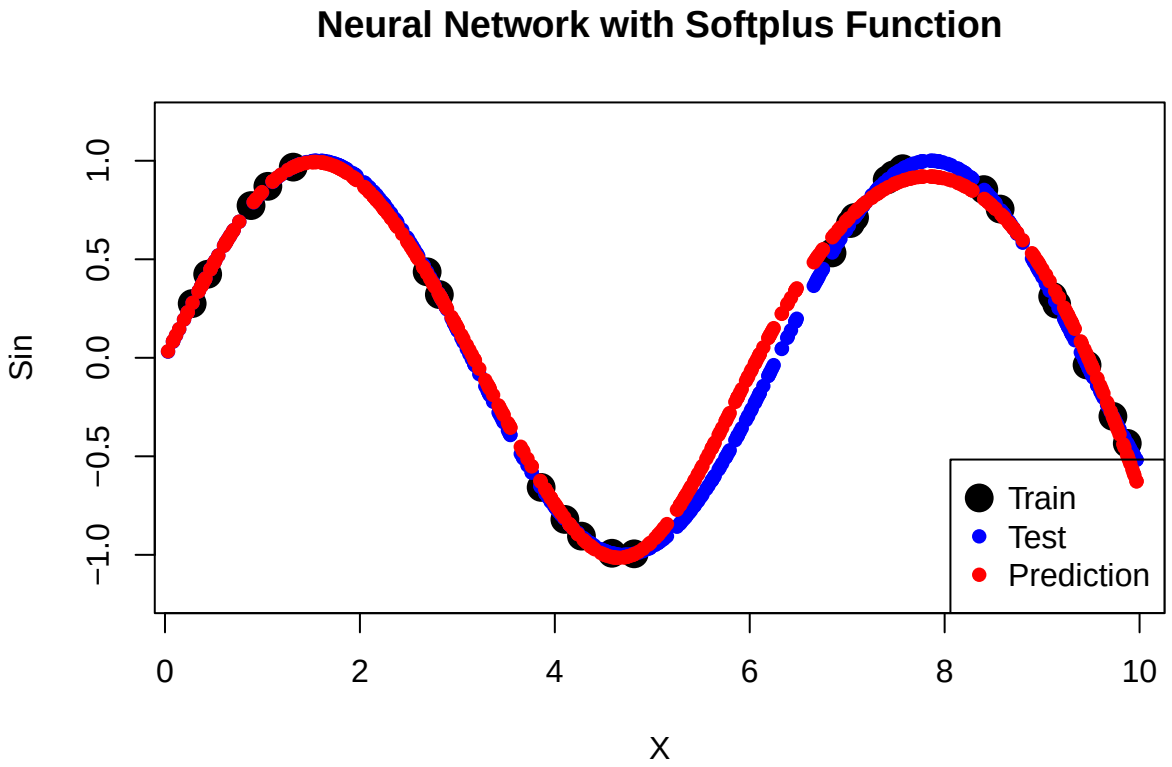


Using the ReLU function as an activation function the predictions of the learned model on the test data do not provide good results. The only predictions that illustrate the sine plot correctly are when X has values between 0.5 and 1.5 and from 2.2 to 3.5. The only problem with this activation function is that when x is smaller than zero, the function returns zero; thus, during the backpropagation process, the weights and biases for some neurons are not updated, which could create dead neurons that never get activated.

Moreover, defining the ReLU function in R as $f(x) = max(0, x)$, does not work and it returns an error because it is not differentiable when x equals zero. The most common solution is to set x equal 1, where the derivative of 1 is zero.

Softplus activation function

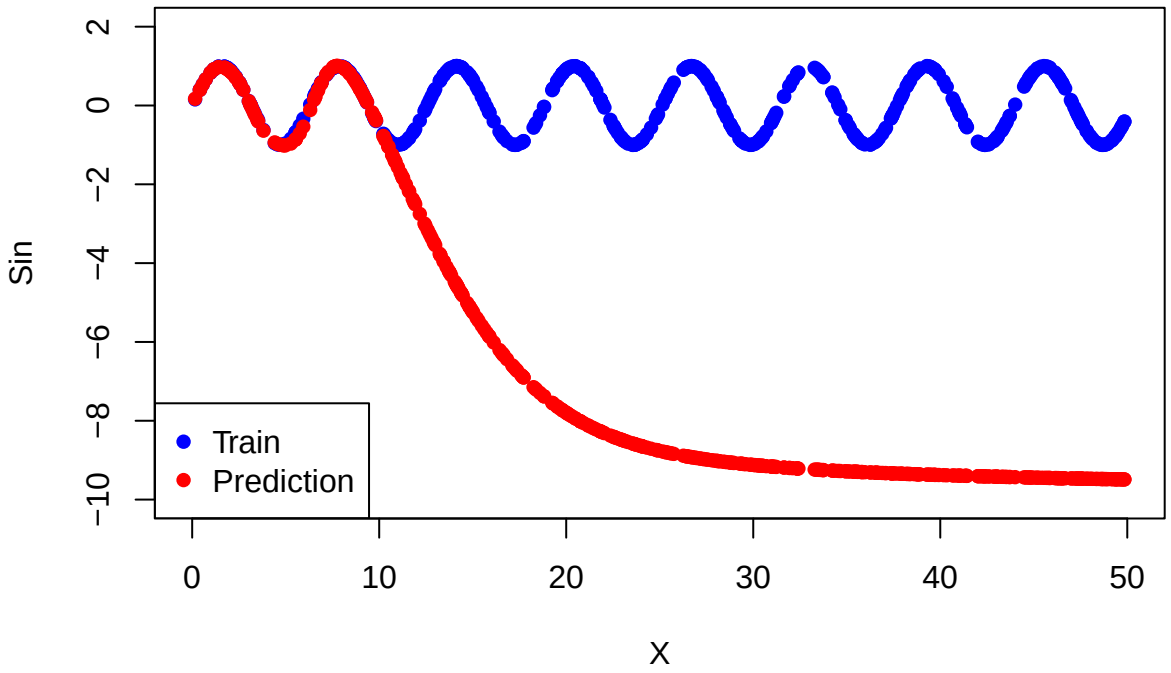
```
h3 <- function(x) log(1 + exp(x))

nn3 <- neuralnet(Sin~X, hidden = 10, data = train, act.fct = h3)

pred_test_3 <- predict(nn3, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.2,1.2),
     main = "Neural Network with Softplus Function")
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test_3, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
```



The above plot shows that the predictions of the learned neural network, which uses the softplus function $f(x) = \log(1 + e^x)$ as an activation function, on the test data provide the best results compared to the previous two examples. The softplus activation function is a smooth approximation of the ReLU activation function.

Task 3: New data and predictions using the initial NN model.

```
set.seed(1234567890)
X <- runif(500, 0, 50)
data2 <- data.frame(X, Sin=sin(X))

pred <- predict(nn, data2)

plot(data2, cex=1, col = "blue", pch = 16, xlim = c(0,50), ylim = c(-10,2))
points(data2[,1],pred, col="red", cex=1, pch = 16)
legend("bottomleft", legend = c("Train", "Prediction"),
       col = c("blue", "red"), pch = 16, pt.cex = c(1,1))
```



The predictions that the trained model returns illustrate sine correctly when X has values between 0 and 10, which makes sense because the initialized neural network model is trained with values between 0 and 10.

Task 4: Role of the weights.

Our model has:

$$\boldsymbol{W^{(1)}} = \begin{bmatrix} -1.84849659 \\ -1.10815997 \\ 0.26168253 \\ -2.29815442 \\ -0.05618884 \\ -1.96773214 \\ -2.51025218 \\ -1.10418652 \\ -0.76154882 \\ 1.75588529 \end{bmatrix}, \qquad \boldsymbol{b^{(1)}} = \begin{bmatrix} 3.9654060 \\ -1.2313394 \\ -2.7255077 \\ 7.3872163 \\ -0.4392372 \\ 0.8143266 \\ 16.9497283 \\ -0.8458926 \\ 2.4703284 \\ -11.6184909 \end{bmatrix}$$

and

$$\boldsymbol{W^{(2)}} = \begin{bmatrix} 0.6261725 \\ -1.3652234 \\ -15.2546738 \\ 1.4724743 \\ 4.0194898 \\ -2.3705710 \\ 1.5225230 \\ 1.3549643 \\ -1.9815383 \\ 6.4412888 \end{bmatrix}, \qquad \boldsymbol{b^{(2)}} = \begin{bmatrix} -0.8272835 \end{bmatrix}$$

As can be seen in $\boldsymbol{W^{(1)}}$, there are only two weights that are positive: W_3 and W_{10}. If we calculate $\boldsymbol{W^{(1)}X} + \boldsymbol{b^{(1)}}$, where $\boldsymbol{X}$ is simply a large scalar value then the resulting matrix will have large negative values in the rows where the weight is negative and large positive values where the weight is positive. Typically the

weights makes the input even larger. There are two exceptions, W_5 and W_2 of $\boldsymbol{W^{(1)}}$ which have values in the interval $(-1, 0)$ meaning that the input gets scaled down. If the input is very large this wont however make any difference for the calculation but has an effect before the prediction converges. Now consider when the input is large, say 100, then applying the sigmoid function to $\boldsymbol{W^{(1)}X + b^{(1)}}$. When the values in $\boldsymbol{W^{(1)}X + b^{(1)}}$ are large negative numbers then the sigmoid function will transform them to be closer and closer to zero as the input becomes larger and larger. When the values are large positive number, applying the sigmoid function will make them closer and closer to one. As the input increases, $\boldsymbol{q}$ will thus approach:

$$\boldsymbol{q} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Then doing next step in the neural network calculations:

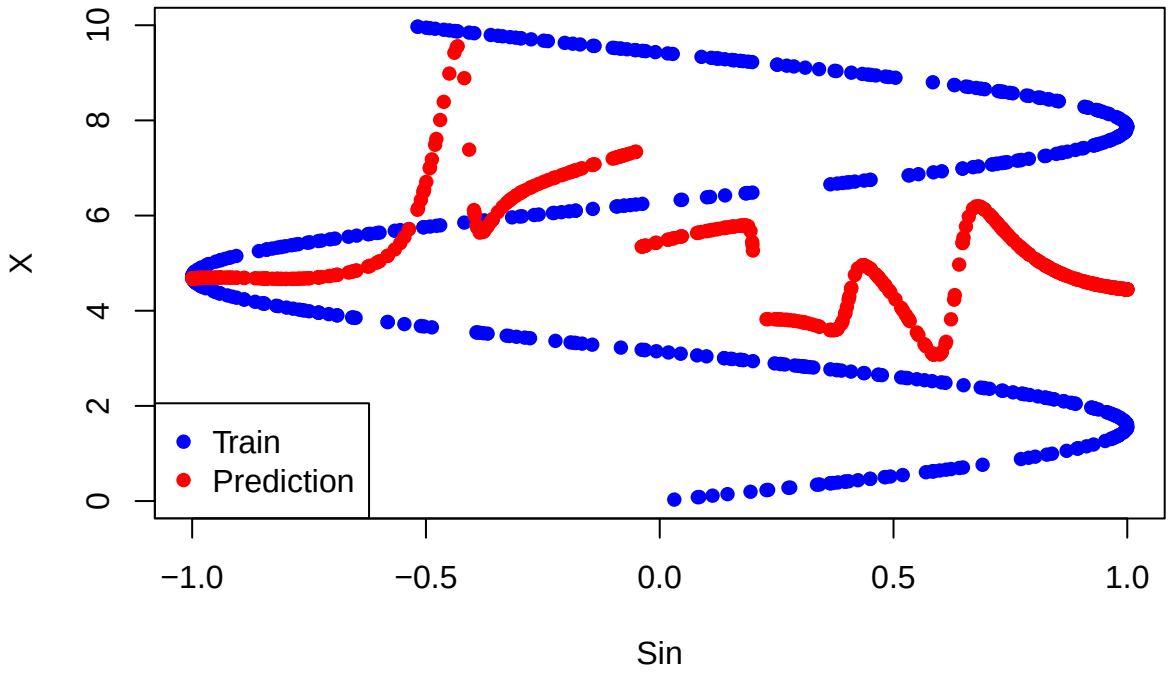
$$\hat{y} = \boldsymbol{W^{(2)}q + b^{(2)}}$$

```
# Sigmoid function
sigmoid <- function(x){
  return(1 / (1 + exp(-x)))
}
# Define weight matrix for layer 1
W1 <- nn$weights[[1]][[1]][2,]
# Define offset vector for layer 1
b1 <- nn$weights[[1]][[1]][1,]
# Get q: Test with input 100
q <- sigmoid(W1 * 100 + b1)
#
b2 <- nn$weights[[1]][[2]][1,]
W2 <- nn$weights[[1]][[2]][2:11,]
cat("The value will converge to:", W2%*%q + b2, "as the input increases")
```

```
## The value will converge to: -9.63129 as the input increases
```

Task 5: New NN model to predict x from sin(x)

```
nn5 <- neuralnet(X~Sin, hidden = 10, data = data1, threshold = 0.1)

pred_data <- predict(nn5, data1)

plot(data1$Sin, data1$X, col="blue", cex=1, pch = 16, xlab = "Sin", ylab = "X")
points(data1[,2],pred_data, col="red", cex=1, pch = 16)
legend("bottomleft", legend = c("Train", "Prediction"),
       col = c("blue", "red"), pch = 16, pt.cex = c(1,1))
```



The predicted results do not present the actual values; the reason is that the neural network model is confused because sine returns the same results for several x values.

Machine Learning Exam 16 January 2020

Assignment 1

```
# import data
glass <- read.csv("glass.csv")

# preparing data
glass$Class <- as.factor(glass$Class)
glass <- glass[,-1]

#splitting data
n=dim(glass)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=glass[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=glass[id2,]
id3=setdiff(id1,id2)
test=glass[id3,]

# combine train & valid data
trval=rbind(train,valid)
```

```
# fitting the logistic regression model
log_reg <- glm(Class~., data=trval, family=binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
# predict test data
# type = response for numerical values between 0 and 1
pred_test <- predict(log_reg, newdata = test, type = "response")

# determine the threshold value
pred_test=as.numeric(pred_test>0.5)

# confusion matrix
cm <- table(test$Class, pred_test)
```

The prediction of the model is good with only 8 misclassifications out of 65.

```
knitr::kable(cm)
```

	0	1
0	18	5
1	3	39

The fitted probabilistic model is given by the formula:

$$p(y = Class|w, x) = \frac{1}{1 + exp(-w_0 - \sum_1^9 w_i x_i)}$$

Where:

$$-w_0 - \sum_1^9 w_i x_i = 12081.97 - 3915.57RI - 50.77Na - 33.88Mg - 94.59Al - 66.46Si - 40K - 40.48Ca - 52.4Ba + 88.57Fe$$

The decision boundary is:

$$12081.97 - 3915.57RI - 50.77Na - 33.88Mg - 94.59Al - 66.46Si - 40K - 40.48Ca - 52.4Ba + 88.57Fe$$

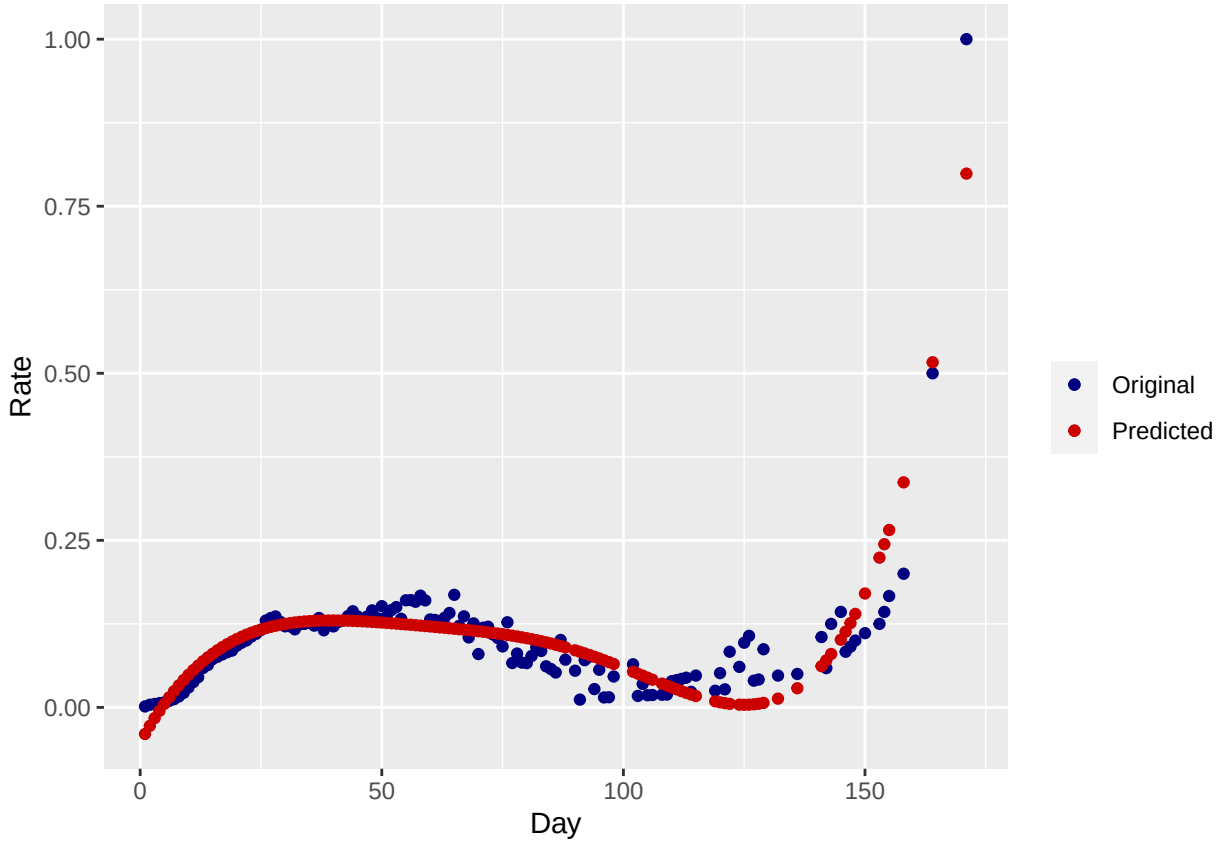
Assignment 2

```
# import data
mortality_rate <- read.csv2("mortality_rate.csv")

# preparing data
X  <- mortality_rate$Day
data <-data.frame(X1=X,
                  X2=X^2,
                  X3=X^3,
                  X4=X^4,
                  X5=ifelse(X-75>0, X-75,0)^4,
                  Y=mortality_rate$Rate)
```

```
#fitting the linear regression model
lin_reg <- lm(Y~., data = data)
pred <- predict(lin_reg)
```

```
# preparing data for plot
df_plot <- cbind(mortality_rate,pred)
```

```
# plot original & predicted data
ggplot() +
  geom_point(data = df_plot, aes(x=Day,y=Rate, color = "navy")) +
  geom_point(data = df_plot, aes(x=Day,y=pred, color = "red3")) +
  theme(legend.position="right") +
  scale_color_manual(values=c("navy","red3"),
                     name = "",
                     labels = c("Original","Predicted"))
```



From the above plot the model seems to be underfitted. The reasons that this might happen is because of the either the 5-spline order or the value of the knot.

```
# summary(lin_reg)
```

The third and fourth degree terms of the model were necessary with the significant level of 0.001.

```
df <- function(input_data){
  X <- as.matrix(input_data[,-6])
  I <- diag(ncol(X))
  hat_matrix <- X %*% solve(t(X) %*% X) %*% t(X)
  degrees_of_freedom <- sum(diag(hat_matrix))
  return(degrees_of_freedom)
}

df(data)
```

```
## [1] 5
```

The degrees of freedom are 6 (5 + the intercept), which is the amount of the parameters of the fitted linear

regression model. Therefore the model is parametric, because whatever data we take the degrees of freedom will always be 6 and thus the complexity of the data will remain the same.

Assignment 3

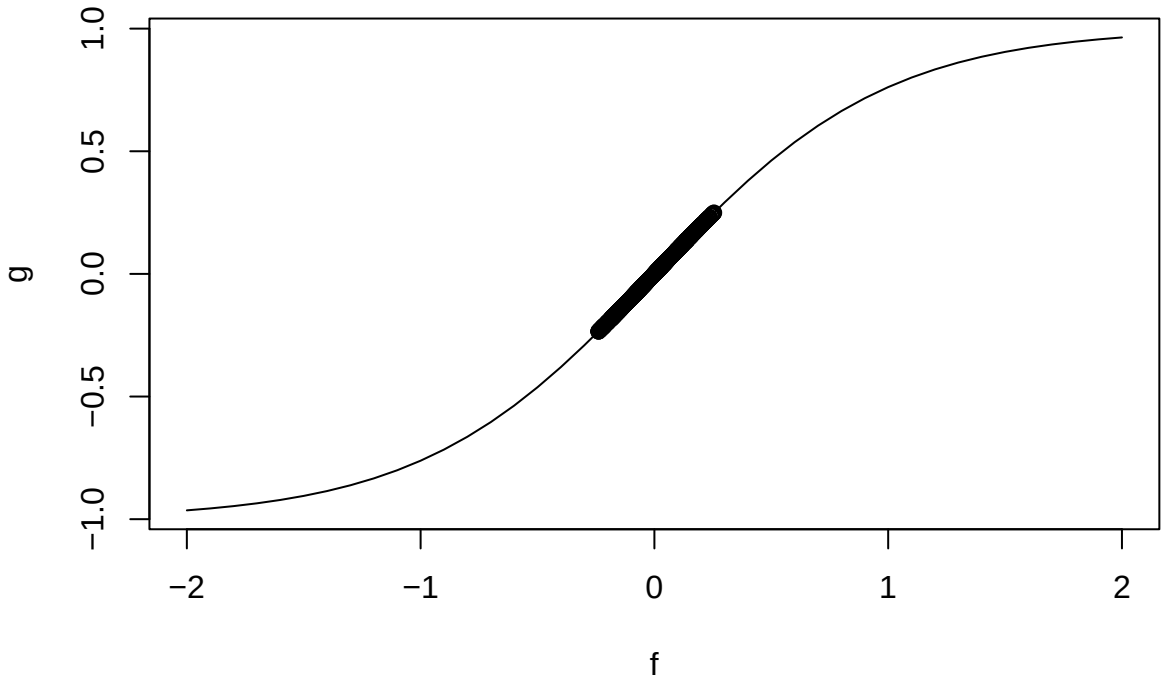
Assignment 4

Assignment 5

```
library(neuralnet)
set.seed(1234567890)
x1 <- runif(1000, -1, 1)
x2 <- runif(1000, -1, 1)
tr <- data.frame(x1,x2, y=x1 + x2)

nn<-neuralnet(formula = y ~ x1 + x2, data = tr, hidden = c(1), act.fct = "tanh")
# plot(nn)
```

y=x1+x2 implis that y=7.75(x1/7.75+x2/7.75)=7.75(0.13*x1+0.13*x2). Therefore, by choosing the weights as the NN does, the hidden unit takes the value resulting from passing 0.13*x1+0.13*x2 through tanh. However, 0.13*x1+0.13*x2 is small for the training data at hand and, thus, tanh behaves linearly (run the code below to appreciate this). Therefore, the hidden unit equals x1/7.75+x2/7.75 and, now, it only remains to weighten this with 7.75 to produce the output x1+x2.

```
f <- seq(-2,2,.1)
g <- tanh(f)
plot(f,g,type="l")
v <- tanh(0.13*tr$x1+0.13*tr$x2)
points(0.13*tr$x1+0.13*tr$x2,v) # The hidden unit only takes values in the linear part of the tanh.
```



Machine Learning Exam March 2020

Assignment 1

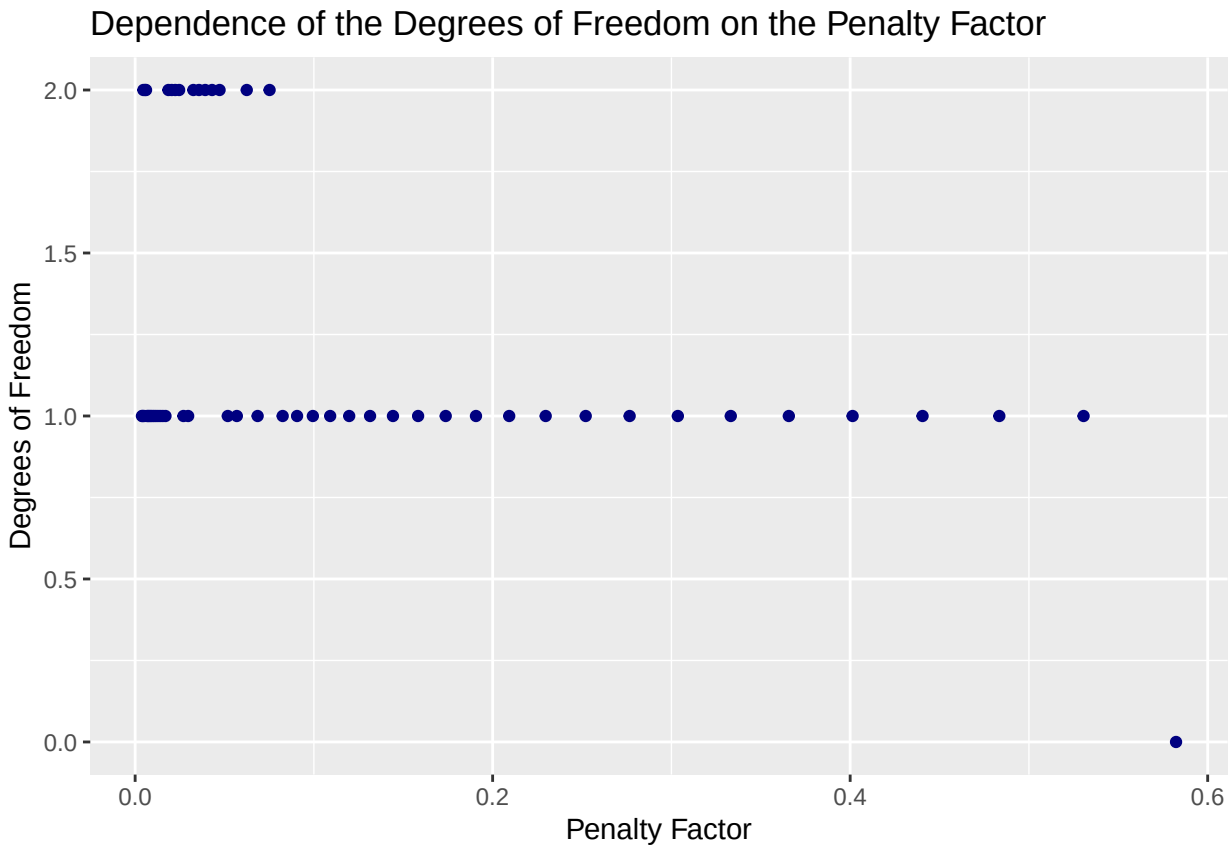
```
RNGversion("3.5.2")
```

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-uniform
## 'Rounding' sampler used
```

```
# import data
temperature <- read.csv2("Dailytemperature.csv")

# creating new features
x <- temperature$Day

phi1 <- NULL
phi2 <- NULL
k <- seq(-50,50,1)

for (i in k){
  phi1 <- sin(0.5^(i)*x)
  phi2 <- cos(0.5^(i)*x)
}

temperature$Phi1 <- phi1
temperature$Phi2 <- phi2
```

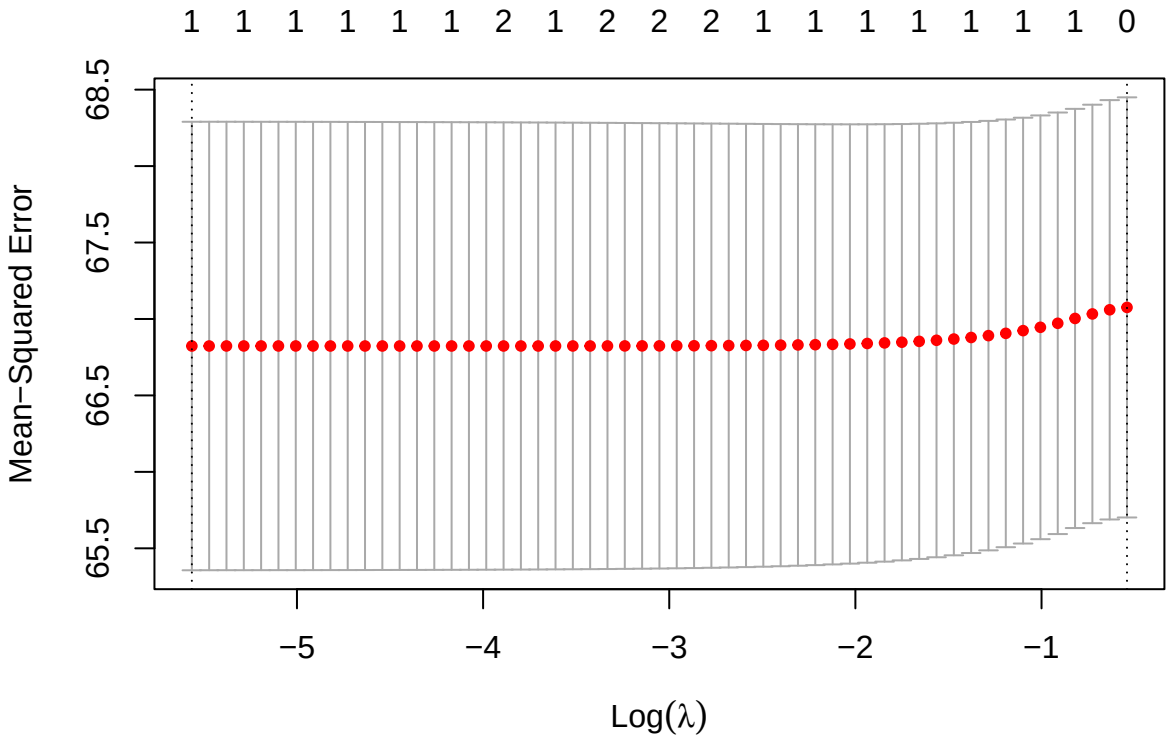
```
library(glmnet)
# fitting lasso regression model
X <- temperature[,-2]
Y <- temperature[,2]

lasso <- glmnet(as.matrix(X), as.matrix(Y), alpha=1)

# summary(lasso)
```

```
# plot of the dependence of the degrees of freedom on the value of the penalty factor
df_plot <- data.frame( "df" <- lasso$df,
                       "lambda" <- lasso$lambda)

ggplot( data = df_plot, aes(x=lambda,y=df)) +
  geom_point( color = "navy") +
  labs(x="Penalty Factor",
       y= "Degrees of Freedom",
       title = "Dependence of the Degrees of Freedom on the Penalty Factor")
```



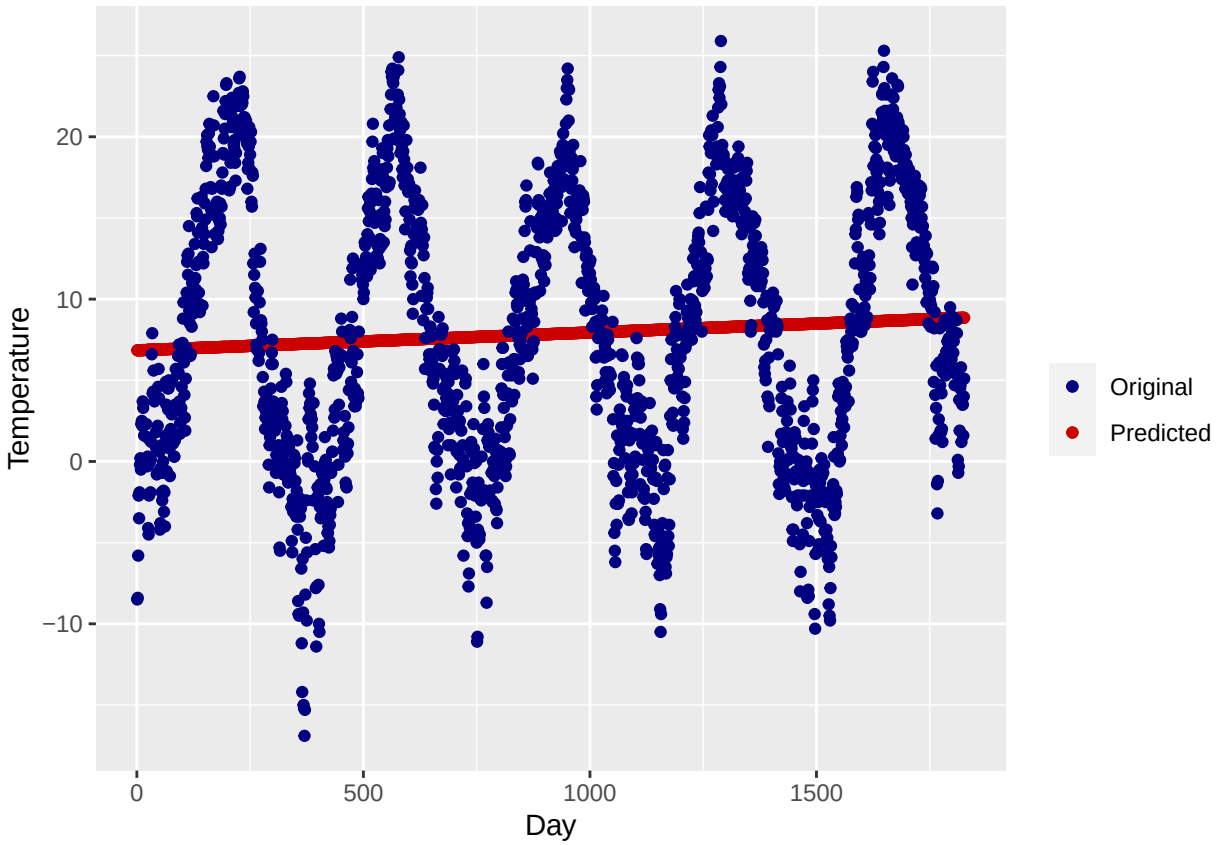
From the above plot it could be assumed that there is a downward tendency between the *Degrees of Freedom* and the *Penalty Factor*.

```
# cross-validation
cv_lasso <- cv.glmnet(as.matrix(X), as.matrix(Y), alpha = 1)

plot(cv_lasso)
```



The optimal lambda , which equals 0.003831144, and the $\log(\lambda) = -4$ have similar MSE values. The optimal lambda has a shred higher MSE, but due to the similar confidence bounce, there is no evident assumption if it is statistically significant.

```
# number of non-zero features corresponding to the optimal penalty factor
amount <- sum(coef(lasso, s = cv_lasso$lambda.min)[,1] != 0)

# type = "link" returns the fitted values
# type = "response" gives the same output "link" for "gaussian" family
# type = "coefficients" returns the model codfficients
# type = "nonzero' retuns a list of the indices of the nonzero coefficients for each value of s.

# # calculating fitted data corresponding to the optimal penalty factor
fitted_values <- predict(cv_lasso , as.matrix(temperature[,-2]) , type = "link",
                         s=cv_lasso$lambda.min)

# data for the plot
optimal_df <- data.frame( "Original"= temperature$Temperature,
                          "Predicted" = fitted_values,
                          "Day" = temperature$Day)

# time series plot of the original and the fitted data corresponding to the optimal penalty factor
my_scatterplot <-ggplot(optimal_df,aes(x=s1, y= Original)) +
  geom_point(aes(x=Day, y= s1, color = "red3")) +
  geom_point(aes(x=Day, y= Original, color = "navy")) +
  labs(x = "Day",y = "Temperature") +
  scale_color_manual(values=c("navy","red3"),
                     name = "",
                     labels = c("Original","Predicted"))

my_scatterplot
```



From the above plot, it could be assumed that the model's predictions are badly. It could be seen that the predicted values are only positive numbers, whereas the actual values are both positive and negative. However, it could be noticed that as the days pass, the winter seasons have become less cold and the summer seasons hotter.

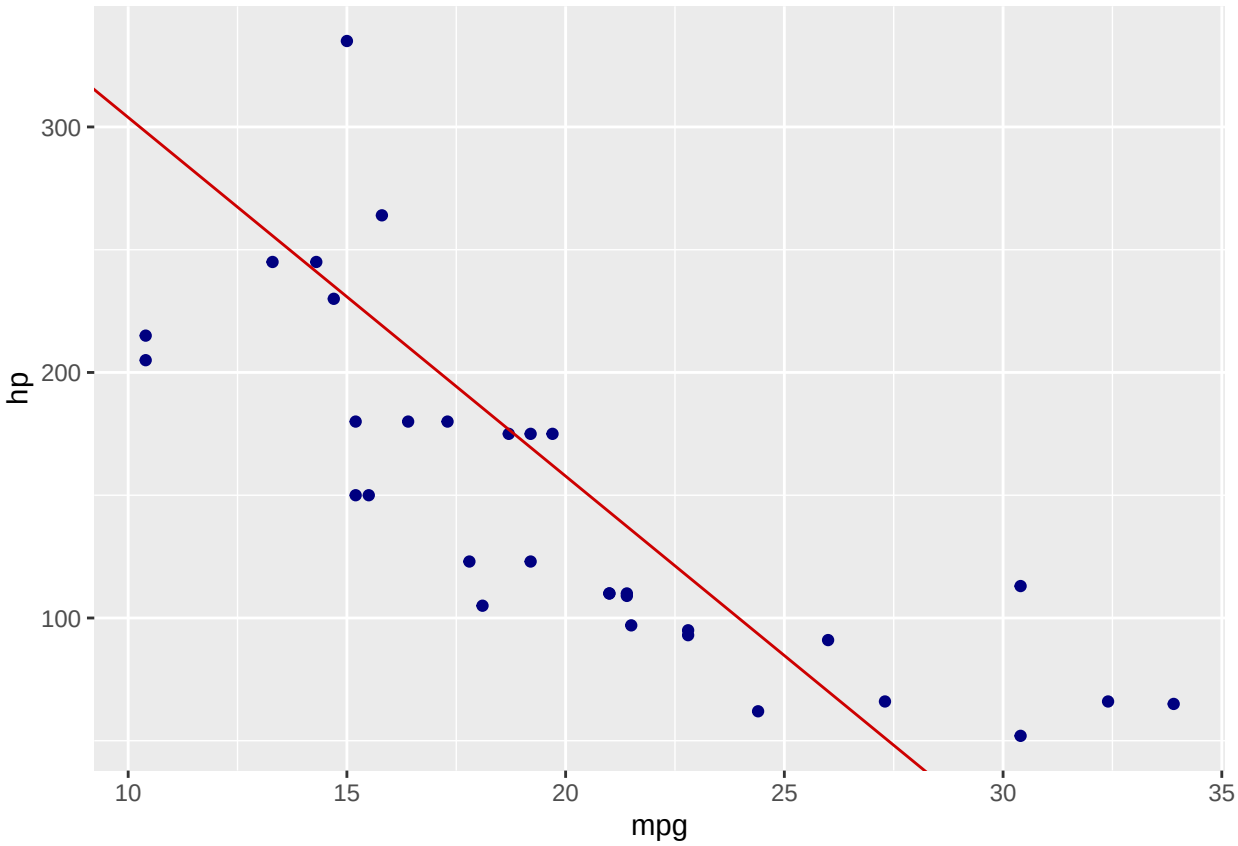
Assignment 2

```
mtcars <- mtcars

#preparing data
mtcars <- mtcars[,-c(2,3,5:11)]
```

```
cov_matrix <- cov(mtcars)
eig <- eigen(cov_matrix)
eigen_vec <- eig$vectors
eigen_val <- eig$values

cat("The components of the first principle component are", eigen_vec[,1],".")
```

```
## The components of the first principle component are -0.06827783 0.9976663 .
```

```
ggplot(data = mtcars, aes(x = mpg, y = hp)) +
  geom_point(color = "navy") +
  geom_abline( slope = eigen_vec[2,1]/eigen_vec[1,1],
               intercept = 450, color = "red3")
```



From the above plot, it is evident that the line demonstrates a negative relationship between the two values. Moreover, it is expected because, generally, more horsepower means fewer miles per gallon.

Machine Learning Exam 15 January 2020

Assignment 1.2

```
# import data
data <- read.csv("default.csv")

#preparing data
data <- data[-c(1,3,4)]
data$AGE <- data$AGE/100

#splitting data
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=data[id2,]
id3=setdiff(id1,id2)
test=data[id3,]
```

```r
#likelihood function
likelihood<-function(input_data,parameter){

  Y <- as.matrix(input_data[,3], nrow = 800)
  X <- input_data[,-3]
  X0 <- rep(1,nrow(input_data))
  X_new <- cbind(X0,X)
  n <- nrow(input_data)

  res <- (1/n) * sum(log(1 + exp(-(Y*parameter*X_new))))

  return(res)
}

parameter_a <- c(0,1,0)
likelihood_a <- likelihood(train,parameter_a)

parameter_b <- c(0,0,1)
likelihood_b <- likelihood(train,parameter_b)

parameter_c <- c(1,1,1)
likelihood_c <- likelihood(train,parameter_c)
```

The log-likelihood value of a regression model is a way to measure the goodness of fit for a model. The higher the value of the log-likelihood, the better a model fits a dataset. The parameter (0,0,1) provided the highest log-likehood value, which is equal to 2.035.

```r
optimal <- function(input_data){
  res <- optim( par = c(1,1,1), fn = likelihood, input_data = train)
  return(res)
}

best_par <- optimal(train)
best_par$par
```

```
## [1] 167.2778 189.8556 368.6556
```

The decision boundary is:

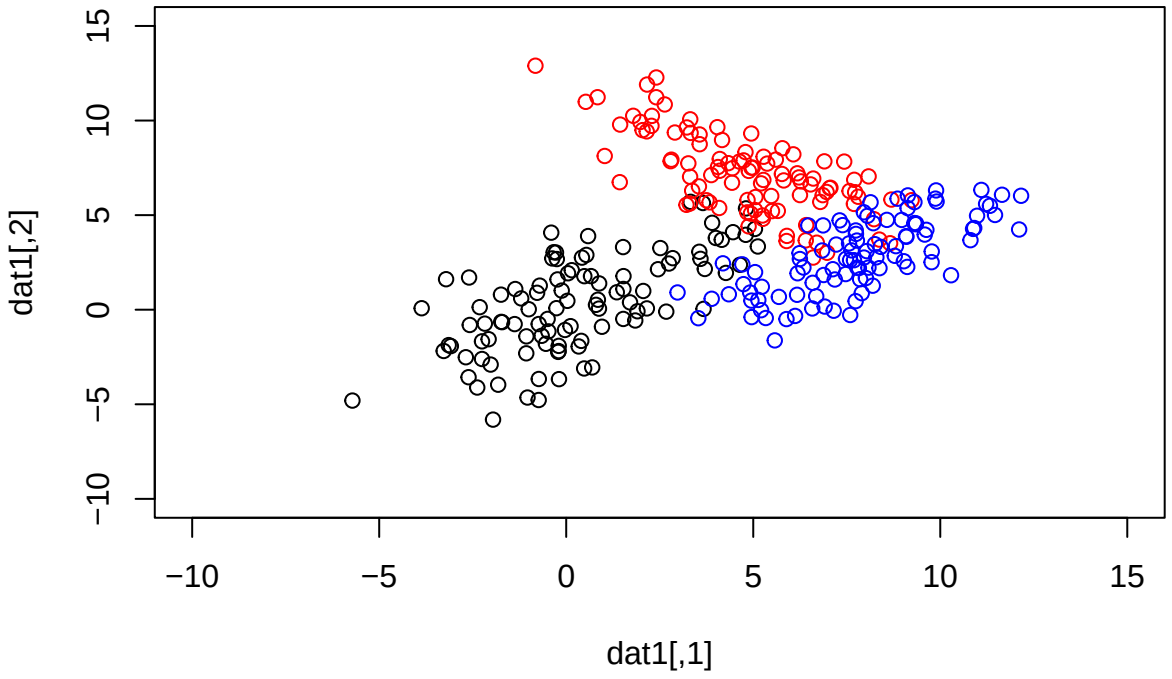
$$189.8556 - 368.6556Sex - 167.2778Age$$

```r
library(glmnet)

log_reg <- glm(default_payment ~ ., data = train, family=binomial)

pred_train <- predict(log_reg, newdata = train)
pred_test <- predict(log_reg, newdata = test)

misclass <- function(actual_val,fitted_val){
  confusion_matrix <- table(actual_val,fitted_val)
  n <- length(actual_val)
  error <- 1 - (sum(diag(confusion_matrix))/n)
  return(error)
}

train_error <- misclass(train$default_payment,pred_train)
```

```
test_error <- misclass(test$default_payment,pred_test)
```

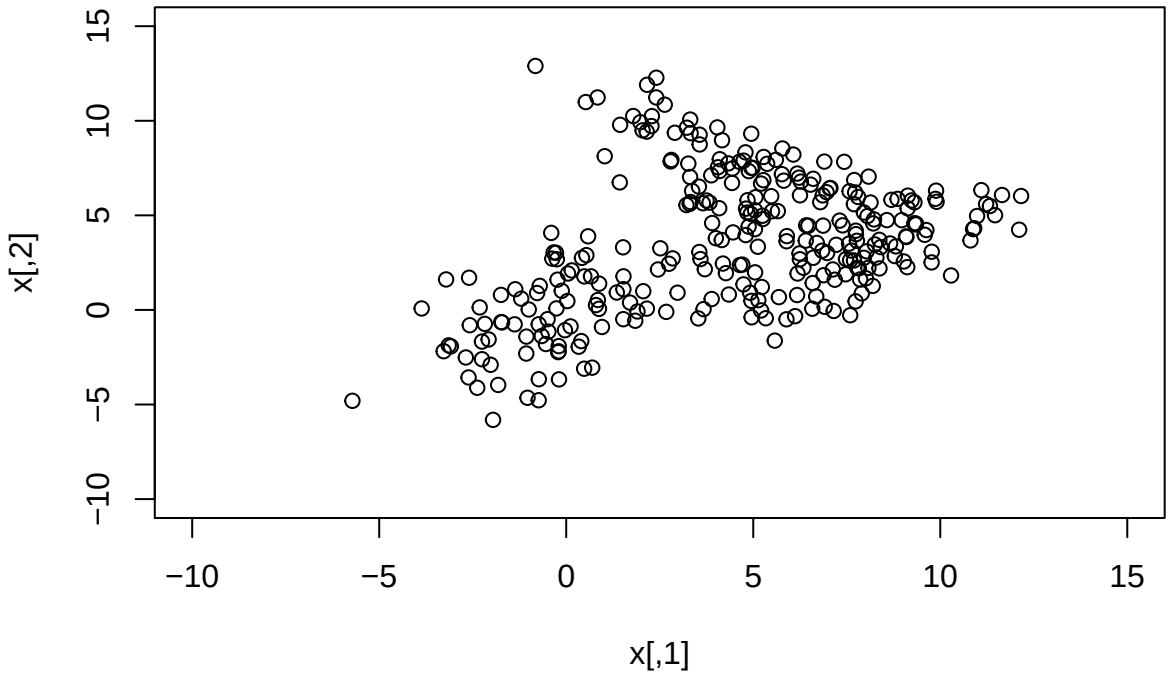
EM Algorithm

```
library(mvtnorm)

set.seed(1234567890)

max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=300 # number of training points
D=2 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data

# Producing the training data
mu1<-c(0,0)
Sigma1 <- matrix(c(5,3,3,5),D,D)
dat1<-rmvnorm(n = 100, mu1, Sigma1)
mu2<-c(5,7)
Sigma2 <- matrix(c(5,-3,-3,5),D,D)
dat2<-rmvnorm(n = 100, mu2, Sigma2)
mu3<-c(8,3)
Sigma3 <- matrix(c(3,2,2,3),D,D)
dat3<-rmvnorm(n = 100, mu3, Sigma3)
plot(dat1,xlim=c(-10,15),ylim=c(-10,15))
points(dat2,col="red")
points(dat3,col="blue")
```




```
x[1:100,]<-dat1
x[101:200,]<-dat2
x[201:300,]<-dat3
```

```
plot(x,xlim=c(-10,15),ylim=c(-10,15))
```




```
K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional means
Sigma <- array(dim=c(D,D,K)) # conditional covariances
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the parameters
pi <- runif(K,0,1)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0,5)
  Sigma[,,k]<-c(1,0,0,1)
}
pi
```

```
## [1] 0.53980633 0.37254500 0.08764867
```

```
mu
```

```
##           [,1]      [,2]
## [1,] 0.5506079 0.9707686
## [2,] 4.4511697 4.1405008
## [3,] 4.2580319 2.7822372
```

```
Sigma
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

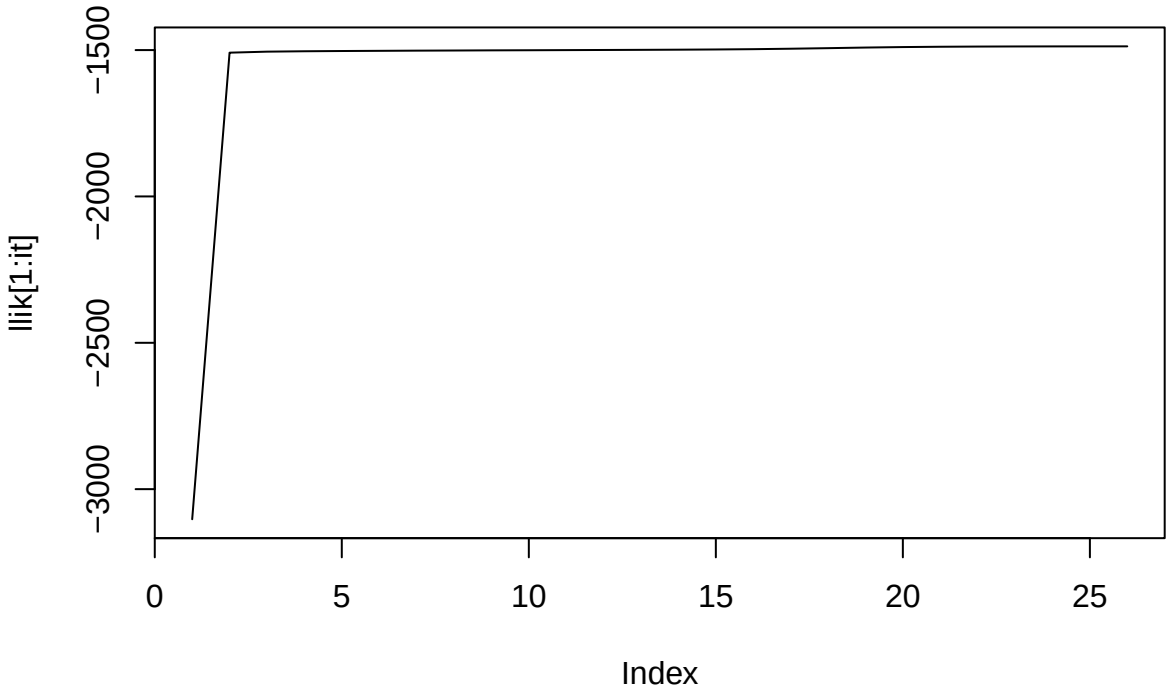
```
##
## , , 2
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## , , 3
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
for(it in 1:max_it) {
  # E-step: Computation of the fractional component assignments
  llik[it] <- 0
  for(n in 1:N) {
    for(k in 1:K) {
      z[n,k] <- pi[k]*dmvnorm(x[n,],mu[k,],Sigma[,,k])
    }

    #Log likelihood computation.
    llik[it] <- llik[it] + log(sum(z[n,]))

    z[n,] <- z[n,]/sum(z[n,])
  }

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1) {
    if(abs(llik[it] - llik[it-1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  for(k in 1:K) {
    pi[k] <- sum(z[,k]) / N
    for(d in 1:D) {
      mu[k, d] <- sum(x[, d] * z[, k]) / sum(z[,k])
    }
    for(d in 1:D) {
      for(d2 in 1:D) {
        Sigma[d,d2,k]<-sum((x[, d]-mu[k,d]) * (x[, d2]-mu[k,d2]) * z[, k]) / sum(z[,k])
      }
    }
  }
}
```

```
## iteration:  1 log likelihood:  -3102.839
## iteration:  2 log likelihood:  -1508.843
## iteration:  3 log likelihood:  -1505.373
## iteration:  4 log likelihood:  -1504.041
## iteration:  5 log likelihood:  -1503.199
```

```
## iteration:  6 log likelihood:  -1502.533
## iteration:  7 log likelihood:  -1501.954
## iteration:  8 log likelihood:  -1501.434
## iteration:  9 log likelihood:  -1500.958
## iteration:  10 log likelihood:  -1500.512
## iteration:  11 log likelihood:  -1500.082
## iteration:  12 log likelihood:  -1499.649
## iteration:  13 log likelihood:  -1499.182
## iteration:  14 log likelihood:  -1498.632
## iteration:  15 log likelihood:  -1497.914
## iteration:  16 log likelihood:  -1496.887
## iteration:  17 log likelihood:  -1495.387
## iteration:  18 log likelihood:  -1493.416
## iteration:  19 log likelihood:  -1491.349
## iteration:  20 log likelihood:  -1489.687
## iteration:  21 log likelihood:  -1488.612
## iteration:  22 log likelihood:  -1487.994
## iteration:  23 log likelihood:  -1487.651
## iteration:  24 log likelihood:  -1487.458
## iteration:  25 log likelihood:  -1487.348
## iteration:  26 log likelihood:  -1487.284
```

```
plot(llik[1:it], type="l")
```



The number of parameters in a MM model is the number of mixing coefficients minus 1 (because they all have to sum up to 1), plus the number of elements in the mean vector for each component, plus the number of entries in the covariance matrix for each component (since this matrix is symmetric, we only count the upper diagonal part of it). That is, K-1 mixing coefficients, plus KD *mean elements, plus* KD*(D+1)/2 covariance elements.

```
BIC<-llik[it] - log(N) * 0.5 * ((K-1)+K*D+K*D*(D+1)/2)
BIC
```

```
## [1] -1535.766
```

```r
# Producing the validation data
dat1<-rmvnorm(n = 1000, mu1, Sigma1)
dat2<-rmvnorm(n = 1000, mu2, Sigma2)
dat3<-rmvnorm(n = 1000, mu3, Sigma3)
v <- matrix(nrow=3000, ncol=D) # validation data
v[1:1000,]<-dat1
v[1001:2000,]<-dat2
v[2001:3000,]<-dat3
z <- matrix(nrow=3000, ncol=K)

vllik <- 0
for(n in 1:3000) {
  for(k in 1:K) {
    z[n,k] <- pi[k]*dmvnorm(v[n,],mu[k,],Sigma[,,k])
  }

  #Log likelihood computation.
  vllik <- vllik + log(sum(z[n,]))
}

pi
```

```
## [1] 0.2993551 0.3616209 0.3390240
```

```r
mu
```

```
##            [,1]       [,2]
## [1,] -0.1096711 -0.0234151
## [2,]  4.9736528  6.8964965
## [3,]  7.5481552  2.8716299
```

```r
Sigma
```

```
## , , 1
##
##          [,1]     [,2]
## [1,] 3.940797 2.706445
## [2,] 2.706445 5.796523
##
## , , 2
##
##           [,1]      [,2]
## [1,]  4.513017 -3.317680
## [2,] -3.317680  5.154163
##
## , , 3
##
##          [,1]     [,2]
## [1,] 4.638160 2.828176
## [2,] 2.828176 3.861886
```

```r
llik[it]
```

```
## [1] -1487.284
```

```r
BIC
```

```
## [1] -1535.766
```

```
vllik
```

```
## [1] -14903.88
```

```
K
```

```
## [1] 3
```

BIC for K=2: -1547.347 BIC for K=3: -1535.766 BIC for K=4: -1547.366 So, K=3 wins.

Validation log lik for K=2: -15225.83 Validation log lik for K=3: -14903.88 Validation log lik for K=4: -14982.4 So, K=3 wins.

It does NOT make sense to use the log lik on the TRAINING data to select among models because the higher the number of components the higher the log lik (since the models are nested). However, using the log lik on some VALIDATION makes sense to select among different number of components, since it measures the generalization ability of the models on some previously unseen data. This implicitly penalizes models with many components because they overfit to the training data. The previous results confirm this.