STAT6340 MiniProject 1

Lakshmipriya Narayanan

2024-09-04

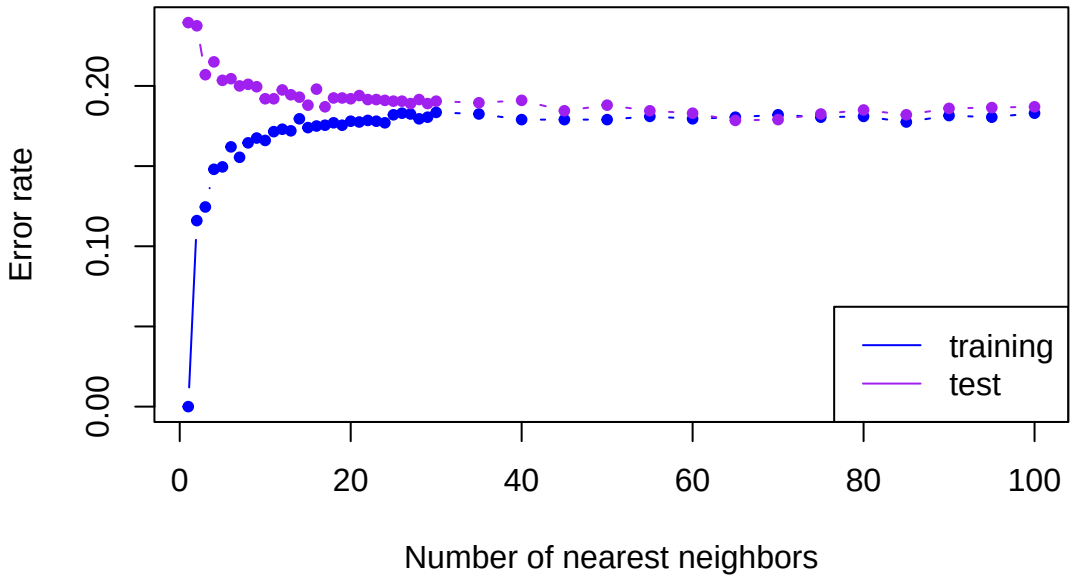
SECTION 1: Observations and Answers

1

1a: Fit KNN with $k = 1, .., 100$.

Refer to Section 2 for detailed code with comments. We have made use of a for loop.

1b: Plot of training and test error rates against K.



From the plot above, we can clearly observe that for the training data, for small values of k, we have very low error rates which indicates high accuracy. And, as k increases, the test error rate is also increasing. Whereas, for the test data, for small values of k, the error rate is very high and as k increases the test error rate decreases gradually and subsequently, for more values of k, both error rates reach the minimum at the optimal value of k which indicates a balance between the bias and the variance.

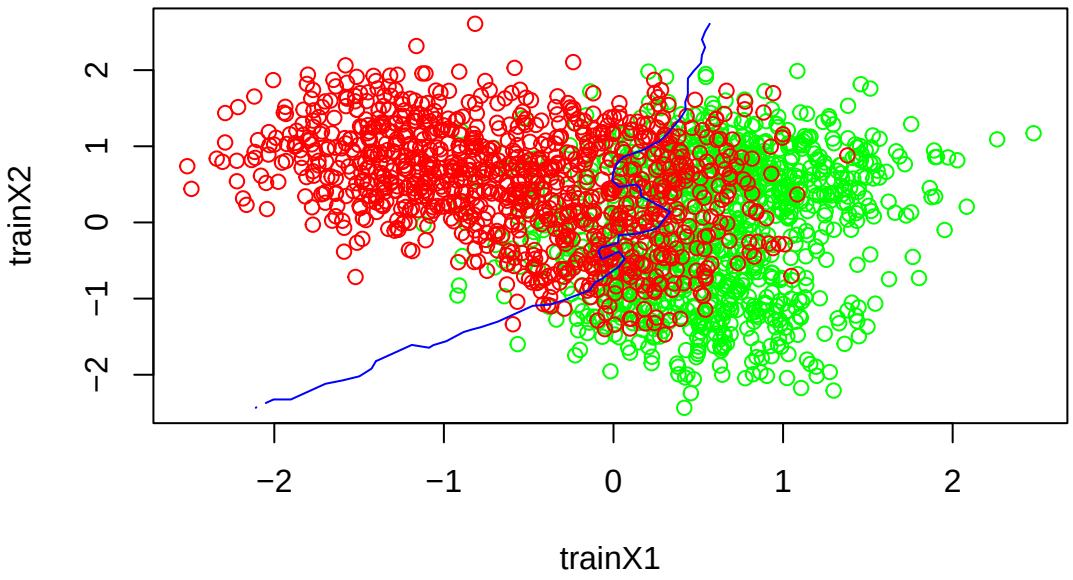
This observation is consistent with respect to the class because when k is small, training error rate is low and test error rate is high $\implies$ high variance and low bias. And for larger k values, it is vice-versa *i.e.*, data is underfit because both error

rates are high $\implies$ low variance and high bias.

1c: What is the optimal value of K? What are the training and test error rates associated with the optimal K?

The optimal value of k is 65. The corresponding training and test error rates associated with optimal $k = 65$ are 0.1805 and 0.1785 respectively.

1d: Plot of the training data that also shows the decision boundary (in blue) for the optimal K.



The decision boundary appears to classify the given data points based on the training data. It is non-linear which $\implies$ our optimal $k = 65$ makes our model less flexible.

From the contour plot we can clearly see that the decision boundary is separating the two classes with minimal overlap. Therefore, we can conclude that this decision boundary is sensible.

2

a: Proof of $MSE[\hat{f}(x_0)] = \Big[Bias[\hat{f}(x_0)]\Big]^2 + Var[\hat{f}(x_0)]$:

By definition,

$$\text{MSE}[\hat{f}(x_0)] = E\left[\Big(\hat{f}(x_0) - f(x_0)\Big)^2\right].$$

Adding and subtracting $E[\hat{f}(x_0)]$ to the RHS above gives us,

$$\text{MSE}[\hat{f}(x_0)] = E\left[\left(\hat{f}(x_0) - E[\hat{f}(x_0)] + E[\hat{f}(x_0)] - f(x_0)\right)^2\right].$$

Now, we use $(a+b)^2$ to expand the above equation where $a = \left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right]$ and $b = E[\hat{f}(x_0)] - f(x_0)$. Therefore,

$$\text{MSE}[\hat{f}(x_0)] = E\left\{\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right]^2 + 2 \cdot \left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right] \cdot \left[E[\hat{f}(x_0)] - f(x_0)\right] + \left[E[\hat{f}(x_0)] - f(x_0)\right]^2\right\}.$$

Since expectation is linear, we can distribute it to each of the terms inside as follows:

$$\text{MSE}[\hat{f}(x_0)] = E\left\{\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right]^2\right\} + 2 \cdot E\left\{\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right] \cdot \left[E[\hat{f}(x_0)] - f(x_0)\right]\right\} + E\left\{\left[E[\hat{f}(x_0)] - f(x_0)\right]^2\right\}.$$

Notice that $E\left\{\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right]^2\right\} = \text{Var}[\hat{f}(x_0)]$ by definition of variance. Similarly, $E\left\{\left[E[\hat{f}(x_0)] - f(x_0)\right]^2\right\} = \left[\text{Bias}[\hat{f}(x_0)]\right]^2$ by definition of bias.
Therefore,

$$\text{MSE}[\hat{f}(x_0)] = \text{Var}[\hat{f}(x_0)] + \left[\text{Bias}[\hat{f}(x_0)]\right]^2 + 2 \cdot E\left\{\left[\hat{f}(x_0) \cdot E[\hat{f}(x_0)]\right] - \left[E[\hat{f}(x_0)]\right]^2 + \left[E[\hat{f}(x_0)]\right]^2 - \left[\hat{f}(x_0) \cdot E[\hat{f}(x_0)]\right]\right\}.$$

But all the terms will get cancelled.
Finally,

$$\text{MSE}[\hat{f}(x_0)] = \left[\text{Bias}[\hat{f}(x_0)]\right]^2 + \text{Var}[\hat{f}(x_0)].$$

b: Proof of $E\left[\hat{Y}_0 - Y_0\right]^2 = \left[Bias[\hat{f}(x_0)]\right]^2 + Var[\hat{f}(x_0)] + \sigma^2$:

By definition, $E\left[(\hat{Y}_0 - Y_0)^2\right] = E\left[(\hat{f}(x_0) - f(x_0) - \epsilon_0)^2\right]$. Expanding the squared term, we get

$$\text{LHS} = E\left[(\hat{f}(x_0) - f(x_0) - \epsilon_0)^2\right]$$

$$= E\left[(\hat{f}(x_0) - f(x_0))^2 - 2(\hat{f}(x_0) - f(x_0))\epsilon_0 + \epsilon_0^2\right]$$

$$= E\left[(\hat{f}(x_0) - f(x_0))^2\right] - 2E\left[(\hat{f}(x_0) - f(x_0))\epsilon_0\right] + E\left[\epsilon_0^2\right].$$

Since ϵ_0 is independent of $\hat{f}(x_0) - f(x_0)$ and $E[\epsilon_0] = 0$, the second term becomes zero. Therefore:

$$E\left[(\hat{Y}_0 - Y_0)^2\right] = E\left[(\hat{f}(x_0) - f(x_0))^2\right] + E\left[\epsilon_0^2\right].$$

By definition and 2a from above and using $E[\epsilon_0^2] = \sigma^2$ given to us in the question,

$$E\left[(\hat{f}(x_0) - f(x_0))^2\right] = \text{MSE}[\hat{f}(x_0)],$$

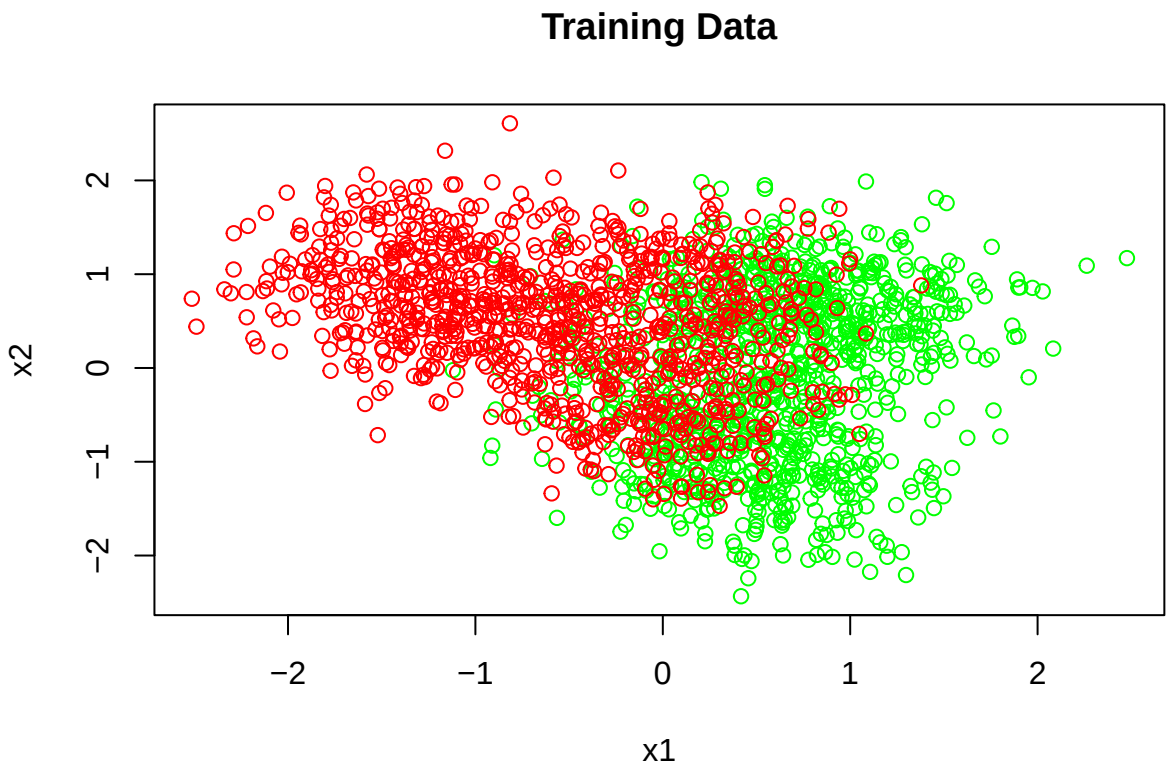
Then,

$$E\left[(\hat{Y}_0 - Y_0)^2\right] = (\text{Bias}[\hat{f}(x_0)])^2 + \text{Var}[\hat{f}(x_0)] + \sigma^2.$$

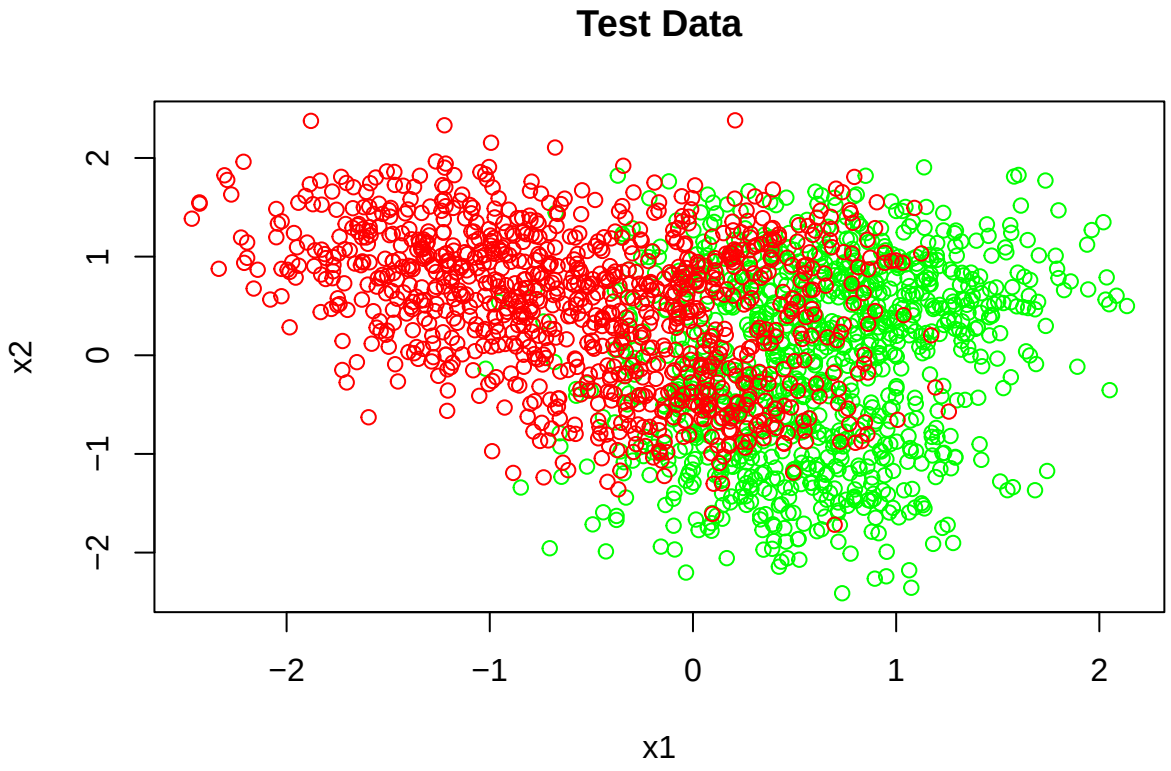
Hence, proved.

SECTION 2: Code

```r
library(class)
set.seed(1)

#Load training and test data from working directory
trainingset <- read.csv("1-training_data.csv")
testset <- read.csv("1-test_data.csv")

#Assign appropriate x1 and x2 values as predictors and y as the response for the training data.
# We convert the response from character variable "yes" or "no" to a factor variable with numeric values
#   for yes(2)  and no(1) for both training data and test data.
trainX1 <- trainingset$x.1
trainX2 <- trainingset$x.2
train_resp <- as.numeric(as.factor(trainingset$y))

testX1 <- testset$x.1
testX2 <- testset$x.2
test_resp <- as.numeric(as.factor(testset$y))

#---------DATA ANALYSIS : -------------------------
#Check for missing values in trainingset and testset
#sum(is.na(trainingset))
#sum(is.na(testset))

#Check the structure of the data we are working with to know what variables we have, etc.
#str(trainingset)
#str(testset)

#Check to see what kind of values are present in prediction values x1 and x2.
#head(trainX1)

#Now, create a vector containing predictors for both training and test data.
trainingset.X <- cbind(trainX1, trainX2)
#table(train_resp)

testset.X <- cbind(testX1, testX2)
#table(test_resp)

#Plot x1 vs x2 values to see how data is distributed for training data.
#In both plots, green dots represent response variables with class 2(yes) and red dots represent
# response variables with class 1(no).
plot(trainingset.X, xlab = "x1", ylab = "x2", col = ifelse(train_resp == "2", "green","red"),
     main = "Training Data")
```




```
#Plot x1 vs x2 values to see how data is distributed for test data.
plot(testset.X, xlab = "x1", ylab = "x2", col = ifelse(test_resp == "2", "green","red"),
     main = "Test Data")
```



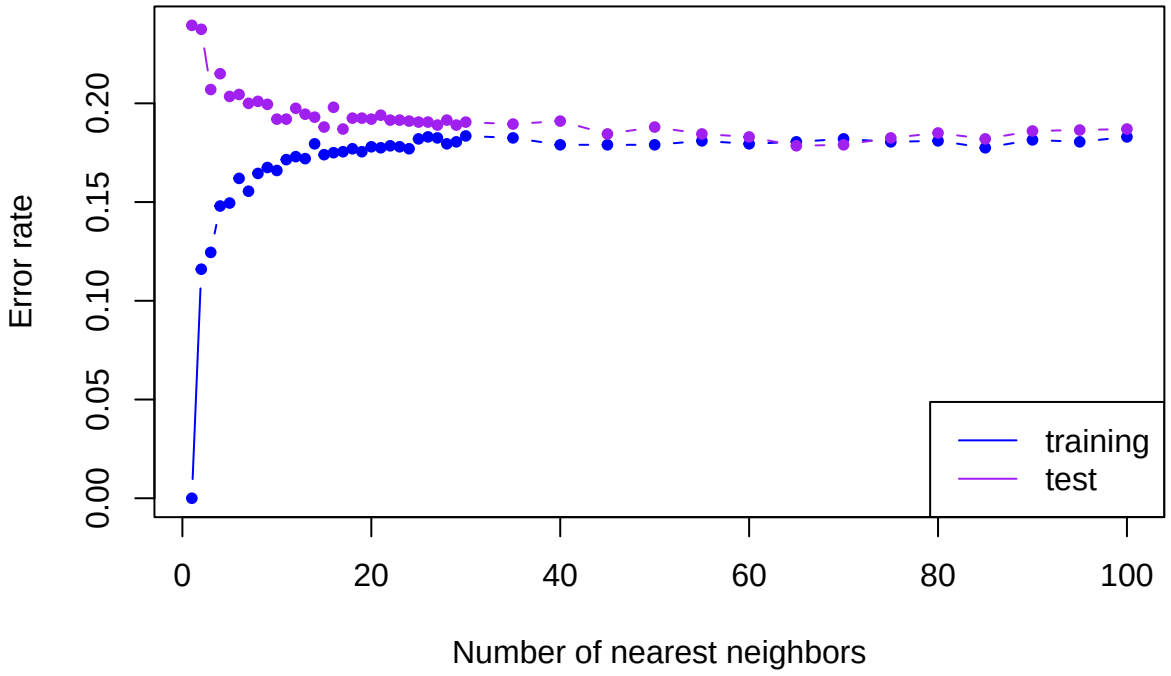
```r
# KNN with K = 1 for training data and test data with their respective error rates
mod.train <- knn(trainingset.X, trainingset.X, train_resp, k = 1)
#table(mod.train, train_resp)
mod.test <- knn(trainingset.X, testset.X, train_resp, k = 1)
#table(mod.test, test_resp)

1 - mean(mod.train == train_resp)        # Training error rate for k=1
```

```
## [1] 0
```

```r
1 - mean(mod.test == test_resp)          # Test error rate for k=1
```

```
## [1] 0.2395
```

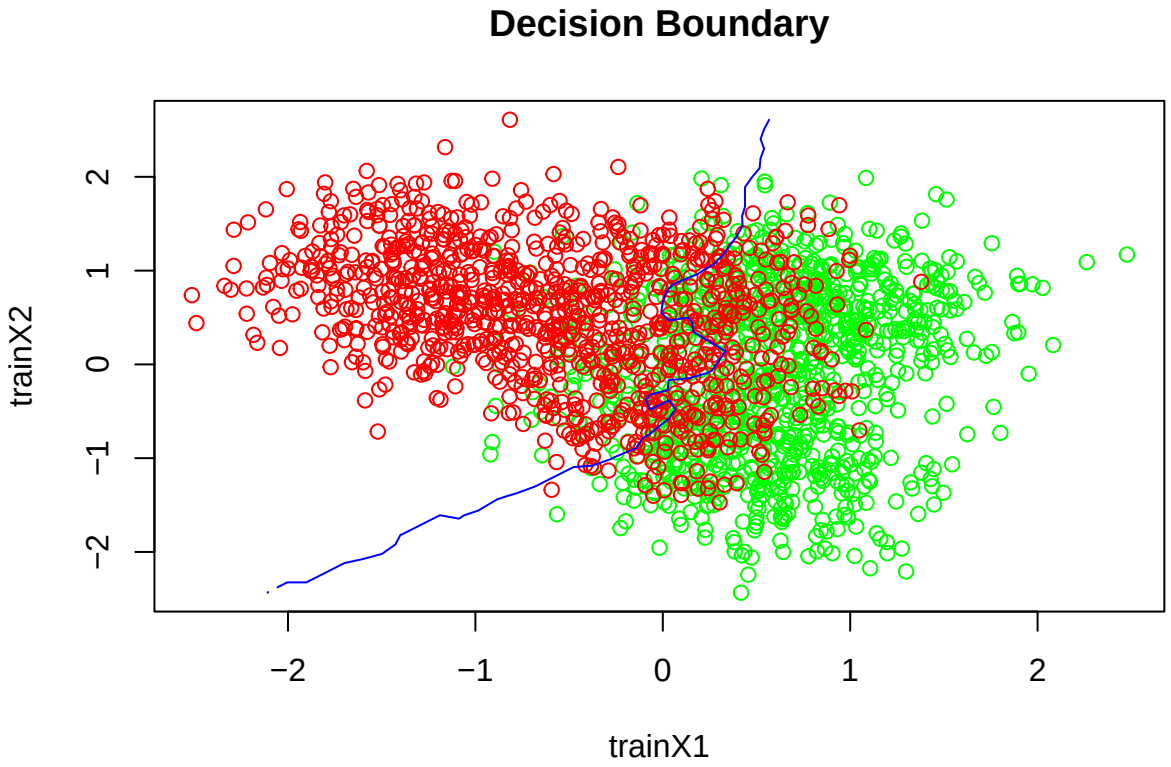
```r
# Now, we fit KNN for several values of K
ks <- c(seq(1, 30, by = 1), seq(35, 100, by = 5))

nks <- length(ks)       # Vector to store values of k from 1-100

# Store training and test error rate for each value of k ranging from 1 to 100.
err.rate.train <- numeric(length = nks)
err.rate.test <- numeric(length = nks)

#Set the names of the error rate so that they correspond to the right value of k
names(err.rate.train) <- names(err.rate.test) <- ks
for (i in seq(along = ks)) {
  mod.train <- knn(trainingset.X, trainingset.X, train_resp, k = ks[i])   # KNN for training data
  mod.test <- knn(trainingset.X, testset.X, train_resp, k = ks[i])        # KNN for test data
  err.rate.train[i] <- 1 - sum(mod.train == train_resp)/length(train_resp) # Training error rate (1 - accuracy
  err.rate.test[i] <- 1 - sum(mod.test == test_resp)/length(test_resp)     # Test error rate
}

#Plot for training error rates against values of k
plot(ks, err.rate.train, xlab = "Number of nearest neighbors", ylab = "Error rate", type = "b",
     ylim = range(c(err.rate.train, err.rate.test)), col = "blue", pch = 20)

#This line shows the plot for the test error rates against the values of k (1-100)
lines(ks, err.rate.test, type="b", col="purple", pch = 20)
legend("bottomright", lty = 1, col = c("blue", "purple"), legend = c("training", "test"))
```




```r
# Calculate optimal value of k and obtain its corresponding test and training error rates
result <- data.frame(ks, err.rate.train, err.rate.test)
result[err.rate.test == min(result$err.rate.test), ]
```

```
##    ks err.rate.train err.rate.test
## 65 65         0.1805        0.1785
```

```r
# Decision boundary for optimal K calculated above
#  (Implementation of the "general" method mentioned in the class)
n.grid <- 50
x1.grid <- seq(f = min(trainingset.X[, 1]), t = max(trainingset.X[, 1]), l = n.grid)
x2.grid <- seq(f = min(trainingset.X[, 2]), t = max(trainingset.X[, 2]), l = n.grid)
grid <- expand.grid(x1.grid, x2.grid)

k.opt <- 70
set.seed(1)
mod.opt <- knn(trainingset.X, grid, train_resp, k = k.opt, prob = T)
prob <- attr(mod.opt, "prob") # prob is fraction for response variable with class = 1(no)
prob <- ifelse(mod.opt == "2", prob, 1 - prob) # Here, it is fraction for response variable with class/factor

prob <- matrix(prob, n.grid, n.grid)

#Plot for training data representing classes (yes/2 represented by green)
plot(trainingset.X, col = ifelse(train_resp == "2", "green", "red"), main = "Decision Boundary")

#Contour plot to depict the decision boundary of the classifiers
contour(x1.grid, x2.grid, prob, levels = 0.5, labels = "", xlab = "", ylab = "", add = T, col = "blue")
```


Decision Boundary
trainX2
2
1
0
−1
−2
−2
−1
0
1
2
trainX1