Exercise 15: Introduction to Machine LEarning

Michael Hotaling

2020-10-28

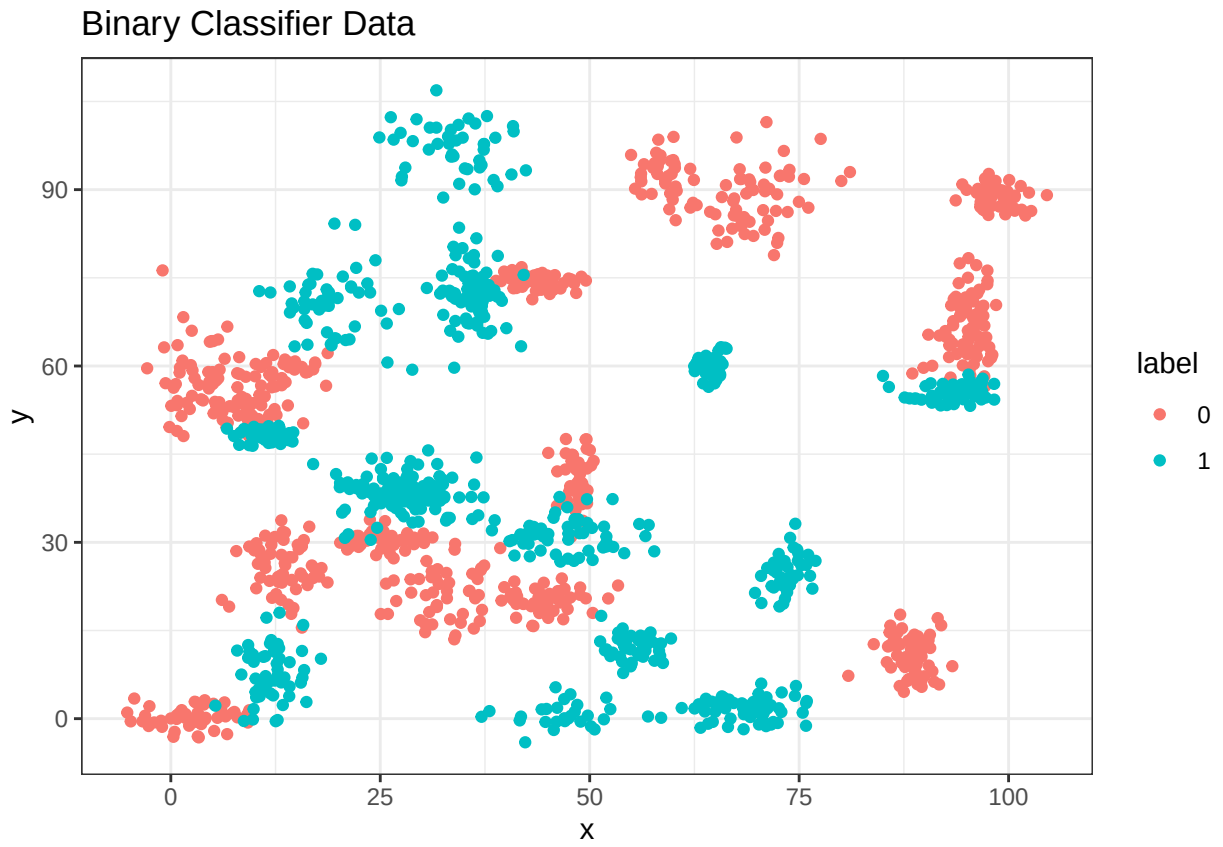
Exercise 15: Introduction to Machine Learning

In this problem, you will use the nearest neighbors algorithm to fit a model on two simplified datasets. The first dataset (found in binary-classifier-data.csv) contains three variables; label, x, and y. The label variable is either 0 or 1 and is the output we want to predict using the x and y variables. The second dataset (found in trinary-classifier-data.csv) is similar to the first dataset except that the label variable can be 0, 1, or 2.

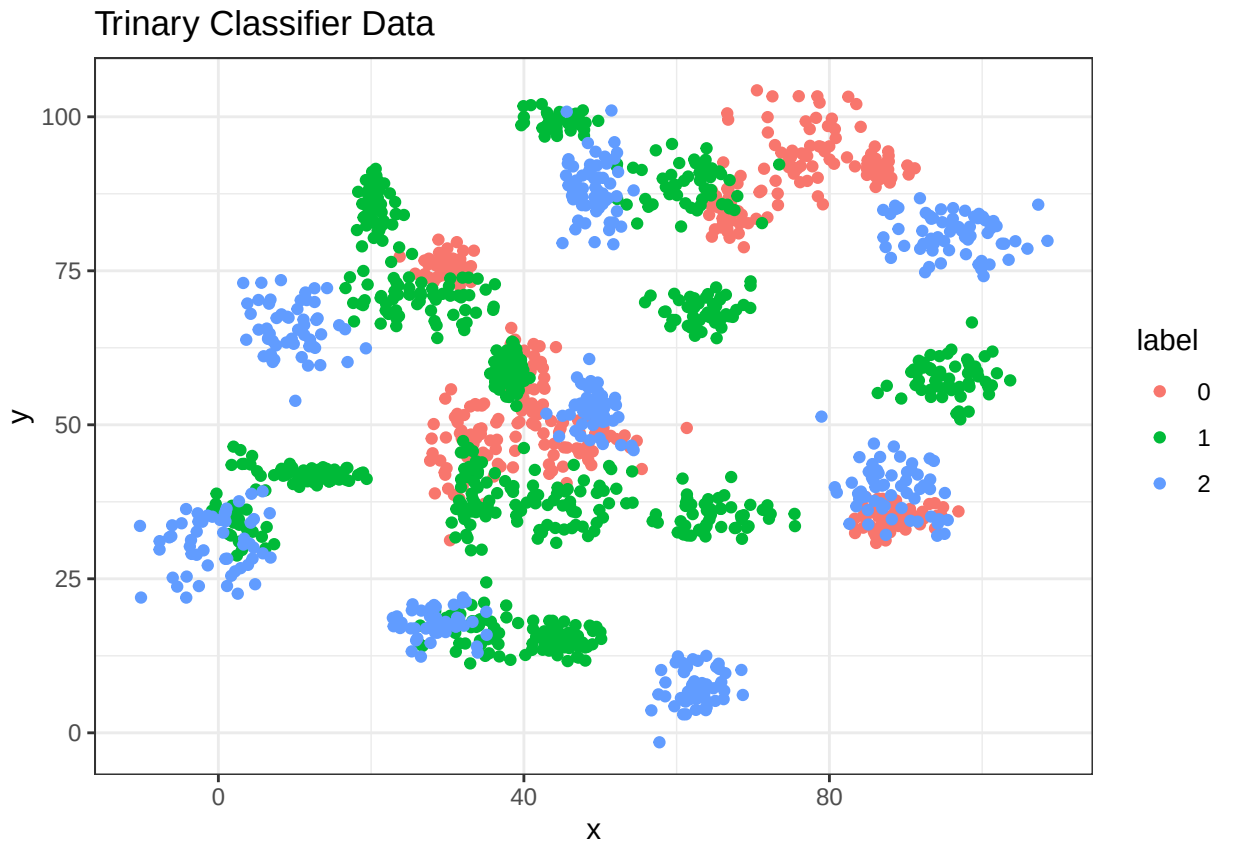
Note that in real-world datasets, your labels are usually not numbers, but text-based descriptions of the categories (e.g. spam or ham). In practice, you will encode categorical variables into numeric values.

a. Plot the data from each dataset using a scatter plot.

```
library(ggplot2)
library(caTools)
library(knitr)
library(pander)
library(class)

bi <- read.csv("binary-classifier-data.csv")
tri <- read.csv("trinary-classifier-data.csv")

bi$label <- as.factor(bi$label)
tri$label <- as.factor(tri$label)

ggplot(data = bi, aes(x = x, y = y, color = label)) + geom_point() +
ggtitle("Binary Classifier Data") +
theme_bw()
```



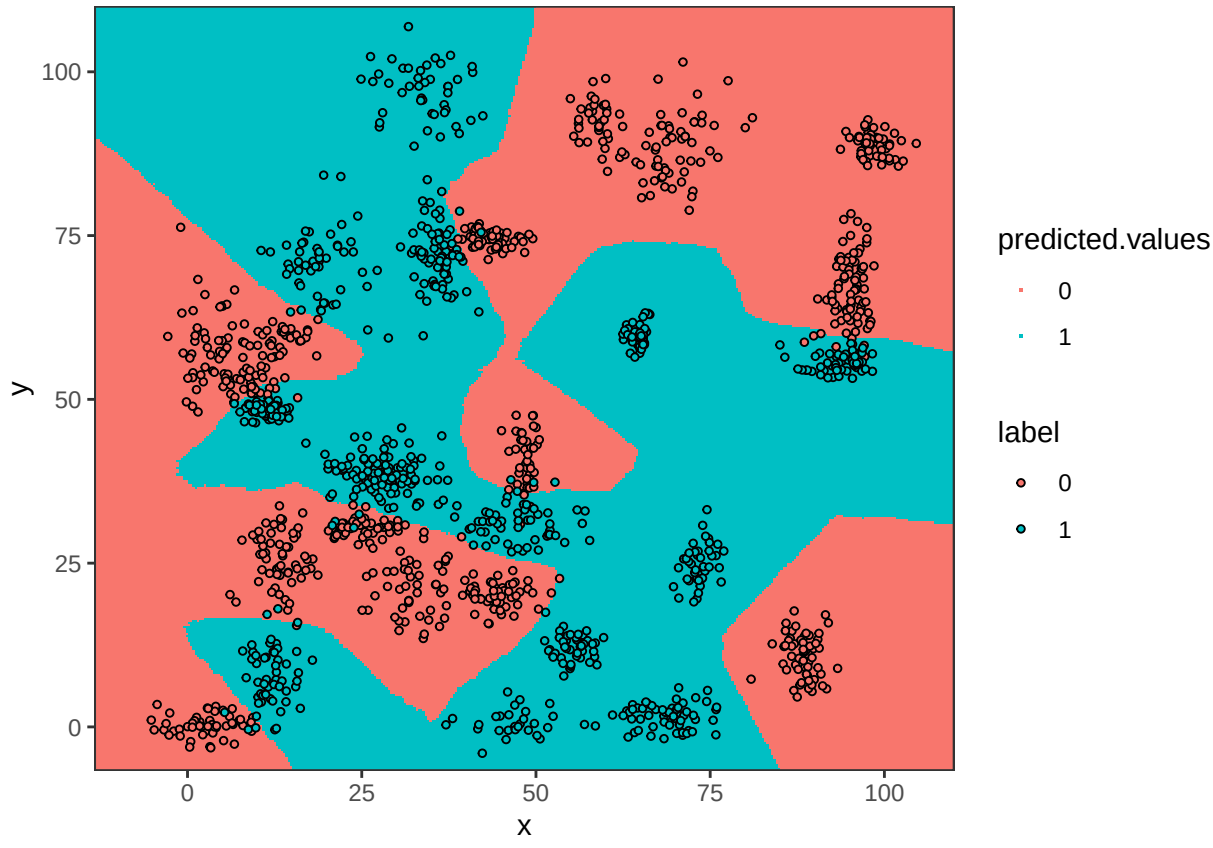

```
ggplot(data = tri, aes(x = x, y = y, color = label)) +
geom_point() +
ggtitle("Trinary Classifier Data") +
theme_bw()
```



b. The k nearest neighbors algorithm categorizes an input value by looking at the labels for the k nearest points and assigning a category based on the most common label. In this problem, you will determine which points are nearest by calculating the Euclidean distance between two points. As a refresher, the Euclidean distance between two points: Fitting a model is when you use the input data to create a predictive model. There are various metrics you can use to determine how well your model fits the data. You will learn more about these metrics in later lessons. For this problem, you will focus on a single metric; accuracy. Accuracy is simply the percentage of how often the model predicts the correct result. If the model always predicts the correct result, it is 100% accurate. If the model always predicts the incorrect result, it is 0% accurate. Fit a k nearest neighbors model for each dataset for k=3, k=5, k=10, k=15, k=20, and k=25. Compute the accuracy of the resulting models for each value of k. Plot the results in a graph where the x-axis is the different values of k and the y-axis is the accuracy of the model.

```
x = NULL
y = NULL
count = 1

for(i in -40:330){
    for(j in -20:330){
        x[count] <- i / 3
        y[count] <- j / 3
        count <- count + 1
    }
}

df <- data.frame(x, y)
```

```
df$predicted.values <- knn(bi[2:3], df, bi$label, k = 59)

ggplot() +
geom_point(data = df,
           aes(x = x,
               y = y,
               color = predicted.values),
           shape = 15,
           size = 0.3) +
geom_point(data = bi,
           aes(x = x,
               y = y,
               fill = label),
           colour = "black",
           pch = 21,
           size = 1) +
scale_x_continuous(expand = c(0, 0)) +
scale_y_continuous(expand = c(0, 0)) +
theme_bw()
```

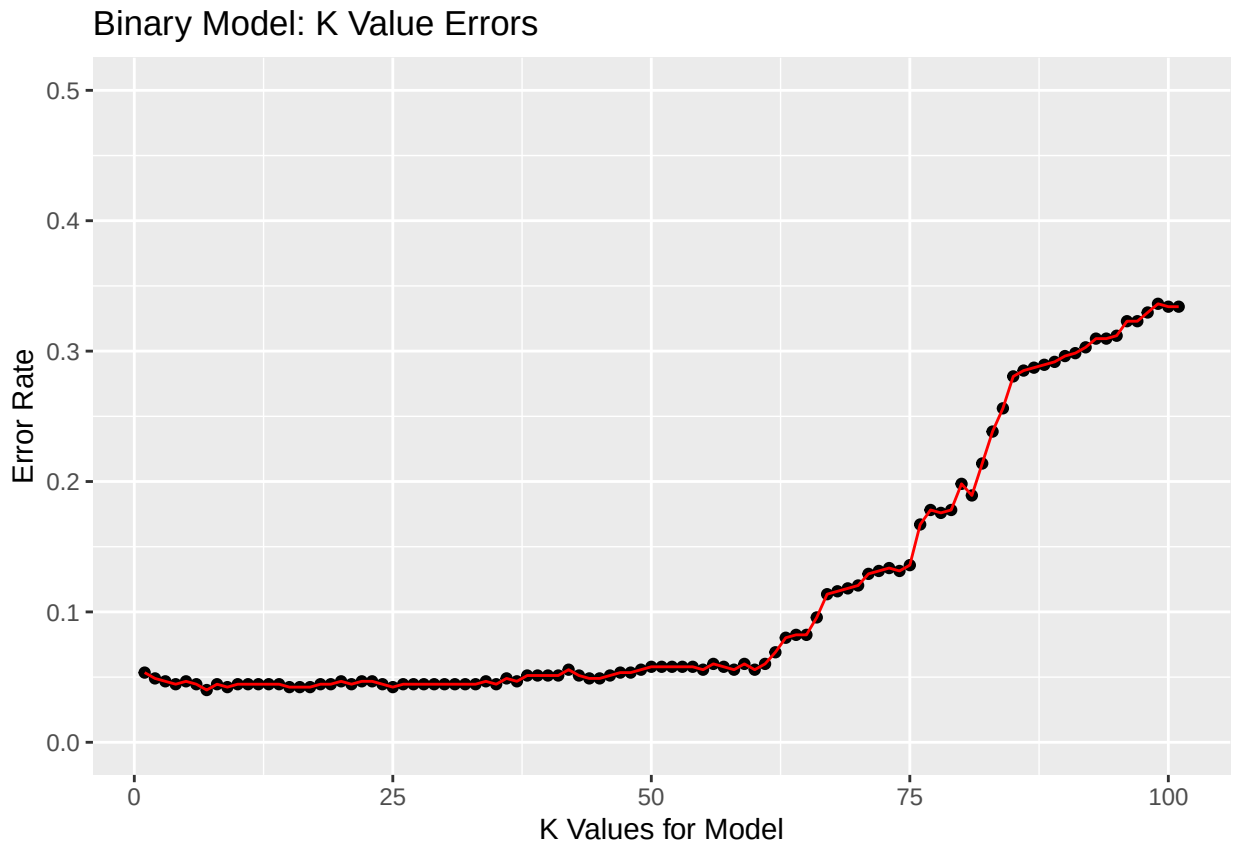



```
error.rate <- NULL
predicted.values <- NULL

set.seed(520)
```

```r
k.values <- 1:101

sample <- sample.split(bi$label, SplitRatio = 0.7 )

train <- subset(bi, sample == TRUE)
test <- subset(bi, sample == FALSE)


for(i in k.values){
    predicted.values <- knn(train[2:3], test[2:3], train$label, k = i)
    error.rate[i] <- mean(test$label != predicted.values)
}

error.df <- data.frame(k.values,error.rate)

subset.df <- subset(error.df, k.values %in% c(3,5,10,15,20,25))
rownames(subset.df) <- c()

pander(subset.df)
```

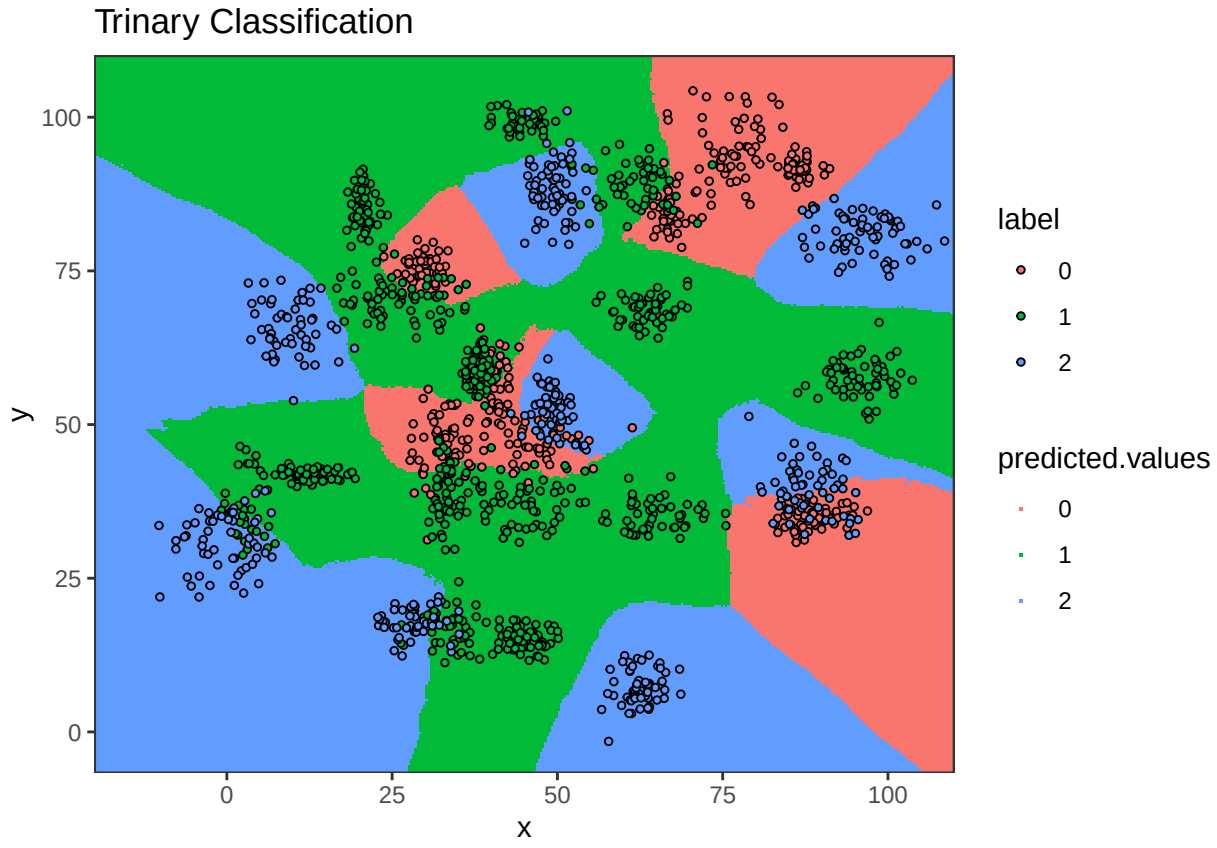
k.values	error.rate
3	0.04677
5	0.04677
10	0.04454
15	0.04232
20	0.04677
25	0.04232

```r
ggplot(data = error.df, aes(x = k.values, y = error.rate)) +
geom_point() +
geom_line(color = "red") +
xlab("K Values for Model") +
ylim(0,0.5) +
ggtitle("Binary Model: K Value Errors") +
ylab("Error Rate")
```




```
x = NULL
y = NULL
count = 1

for(i in -60:330){
    for(j in -20:330){
        x[count] <- i / 3
        y[count] <- j / 3
        count <- count + 1
    }
}

df <- data.frame(x, y)

df$predicted.values <- knn(tri[2:3], df, tri$label, k = 59)

ggplot() +
geom_point(data = df,
           aes(x = x,
               y = y,
               color = predicted.values),
           shape = 15,
           size = 0.3) +
geom_point(data = tri,
           aes(x = x,
               y = y,
```

```
              fill = label),
           colour = "black",
           pch = 21,
           size = 1) +
ggtitle("Trinary Classification") +
scale_x_continuous(expand = c(0, 0)) +
scale_y_continuous(expand = c(0, 0)) +
theme_bw()
```

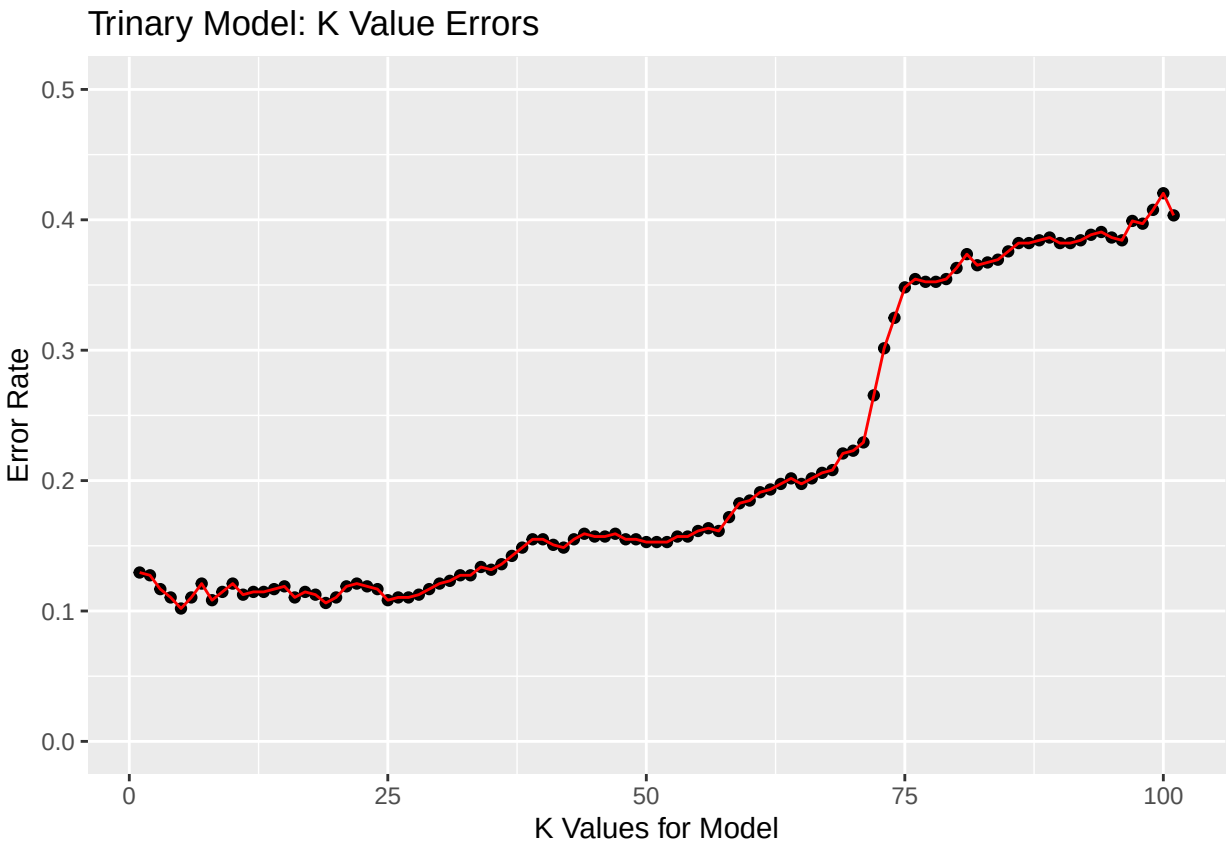



```
error.rate <- NULL
predicted.values <- NULL

set.seed(520)

k.values <- 1:101

sample <- sample.split(tri$label, SplitRatio = 0.7 )

train <- subset(tri, sample == TRUE)
test <- subset(tri, sample == FALSE)


for(i in k.values){
    predicted.values <- knn(train[2:3], test[2:3], train$label, k = i)
    error.rate[i] <- mean(test$label != predicted.values)
}
```

```
error.df <- data.frame(k.values,error.rate)

subset.df <- subset(error.df, k.values %in% c(3,5,10,15,20,25))
rownames(subset.df) <- c()

pander(subset.df)
```

k.values	error.rate
3	0.1168
5	0.1019
10	0.121
15	0.1189
20	0.1104
25	0.1083

```
ggplot(data = error.df, aes(x = k.values, y = error.rate)) +
geom_point() +
geom_line(color = "red") +
xlab("K Values for Model") +
ylim(0,0.5) +
ggtitle("Trinary Model: K Value Errors") +
ylab("Error Rate")
```



c. In later lessons, you will learn about linear classifiers. These algorithms work by defining a decision

boundary that separates the different categories.Looking back at the plots of the data, do you think a linear classifier would work well on these datasets?

No, a linear model wouldn't be a good fit for this type of data. Linear models can split data into groups based on linear algebra and this data is too complex to split with a line.