KNN Project Solution

We'll be making use of the Iris Dataset for this KNN Model

The dataset can be downloaded from the folder in git or it's by default a built-in dataset in R.

Get the Data

Iris Data Set

We'll use the famous iris data set for this project. It's a small data set with flower features that can be used to attempt to predict the species of an iris flower.

Use the ISLR libary to get the iris data set. Check the head of the iris Data Frame.

In [1]:
```
library(ISLR)
```

In [15]:
```
head(iris)
```

Out[15]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

In [16]:
```
str(iris)
```

```
'data.frame':	150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1
1 1 1 1 1 ...
```

Standardize Data

In this case, the iris data set has all its features in the same order of magnitude, but its good practice (especially with KNN) to standardize features in your data. Lets go ahead and do this even though its not necessary for this data!

Use scale() to standardize the feature columns of the iris dataset. Set this standardized version of the data as a new variable.

In [3]:
```
stand.features <- scale(iris[1:4])
```

Check that the scaling worked by checking the variance of one of the new columns.

In [4]:
```
var(stand.features[,1])
```

Out[4]: 1

Join the standardized data with the response/target/label column (the column with the species names.

In [5]:
```
final.data <- cbind(stand.features,iris[5])
```

In [6]:
```
head(final.data)
```

Out[6]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	-0.8976739	1.015602	-1.335752	-1.311052	setosa
2	-1.1392	-0.1315388	-1.335752	-1.311052	setosa
3	-1.380727	0.3273175	-1.392399	-1.311052	setosa
4	-1.50149	0.09788935	-1.279104	-1.311052	setosa
5	-1.018437	1.24503	-1.335752	-1.311052	setosa
6	-0.535384	1.933315	-1.165809	-1.048667	setosa

Train and Test Splits

Use the caTools library to split your standardized data into train and test sets. Use a 70/30 split.

In [7]:
```
set.seed(101)

library(caTools)

sample <- sample.split(final.data$Species, SplitRatio = .70)
train <- subset(final.data, sample == TRUE)
test <- subset(final.data, sample == FALSE)
```

Build a KNN model.

Call the class library.

In [8]:
```
library(class)
```

Use the knn function to predict Species of the test set. Use k=1

In [9]:
```
predicted.species <- knn(train[1:4],test[1:4],train$Species,k=1)
```

In [10]:
```
predicted.species
```

Out[10]:

setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa versicolor versicolor versicolor versicolor versicolor virginica versicolor versicolor versicolor versicolor versicolor virginica versicolor versicolor versicolor virginica virginica virginica virginica virginica virginica virginica virginica virginica virginica virginica virginica virginica virginica virginica

What was your misclassification rate?

In [11]:
```
mean(test$Species != predicted.species)
```

Out[11]: 0.0444444444444444

Choosing a K Value

Although our data is quite small for us to really get a feel for choosing a good K value, let's practice.

Create a plot of the error (misclassification) rate for k values ranging from 1 to 10.

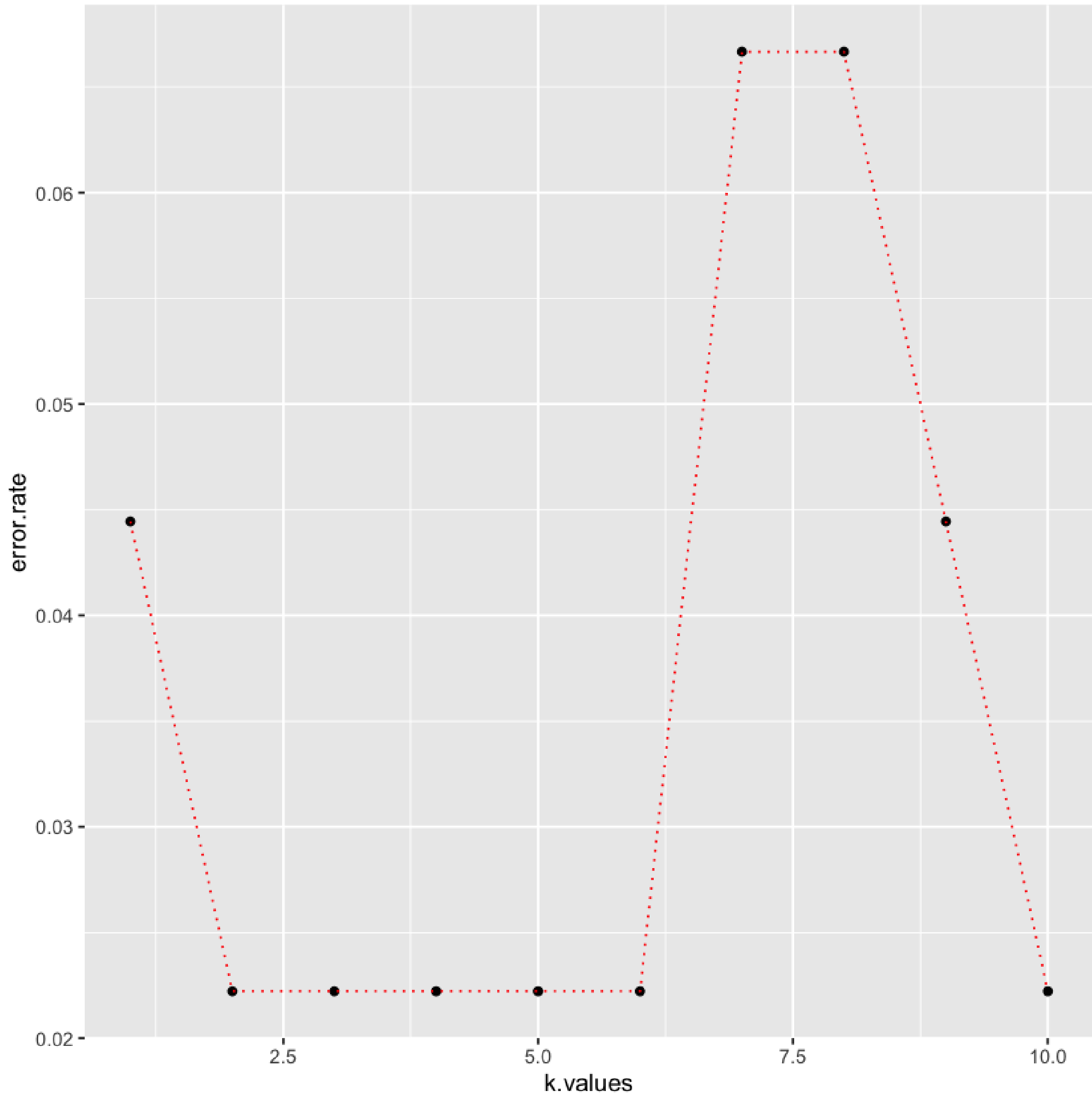
In [12]:
```
predicted.species <- NULL
error.rate <- NULL

for(i in 1:10){
    set.seed(101)
    predicted.species <- knn(train[1:4],test[1:4],train$Species,k=i)
    error.rate[i] <- mean(test$Species != predicted.species)
}
```

In [13]:
```
library(ggplot2)
```

```
k.values <- 1:10
error.df <- data.frame(error.rate,k.values)
```

In [14]:
```
pl <- ggplot(error.df,aes(x=k.values,y=error.rate)) + geom_point()
pl + geom_line(lty="dotted",color='red')
```



You should have noticed that the error drops to its lowest for k values between 2-6. Then it begins to jump back up again, this is due to how small the data set it. At k=10 you begin

to approach setting k=10% of the data, which is quite large.

Try the same on various other datasets

Feel free to check out other such data repositaries from UCI Machine Learning Repo and build different classifiers for the same

[Check out the data sets here]

Great Job!