K-NEAREST NEIGHBOR CLASSIFICATION

Preparing the data

For this type of methodology, this step is what takes the most time. We have to determine some form of standardization for each of our variables at the very least. If we were building our own unique distance functions, then lots of exploration and experimentation goes into this step. Since we're still learning, let's keep it simple for now. Looking at the PenDigits data, let's see how well we can predict the handwritten digit using the 16 variables of coordinate information. The unique thing about this dataset is that the digits are all on the exact same scale, from 0 to 100. For this reason, standardization is not of importance in this case. We can actually go ahead and start applying a model.

Making Predictions with a kNN Model

```
> #setwd("")
> #load("PenDigits.RData")
> load("/Users/shaina/Library/Mobile Documents/com~apple~CloudDocs/Data Mining 2020/Code/PenDigits.RData")
> set.seed(7515)
> library(class)
> pred = knn(train=train[,1:16], test=test[,1:16], cl=train[,17], k=5)
> conf=table(pred, test[,17])
> conf
```

```
pred   0   1   2   3   4   5   6   7   8   9
   0 354   0   0   0   0   0   0   0   1   0
   1   0 347   2   1   0   0   0  15   0   1
   2   0  15 362   0   0   0   0   1   0   0
   3   0   0   0 333   0   5   0   0   0   7
   4   0   1   0   0 354   0   0   0   0   0
   5   0   0   0   0   9 328   0   0   1   1
   6   7   0   0   0   1   0 336   0   0   0
   7   0   1   0   0   0   0   0 346   0   4
   8   1   0   0   0   0   0   0   2 334   1
   9   1   0   0   2   0   2   0   0   0 322
```

Based on initial inspection of this confusion matrix, it appears we did quite well predicting with k=5 neighbors. To compute the actual misclassification rate, we need to sum the off diagonal elements of this matrix and divide by the number of test cases. The following code does that for us:

```
> cat("Misclassification Rate = ", sum(pred!=test[,17])/length(test[,17]))
```

```
Misclassification Rate =  0.02344197
```

Wow! That's one of the lowest misclassification rates that we've seen on this dataset, with one of the simplest methods. We can take one more step to optimize the method by choosing a value of k that provides the best results on our test set:

```
> # we'll try values of k from 1 to 20 at first
> range = 1:20
> # create a vector to store the accuracy for each value of k
> accs = rep(0, length(range))
> # loop through those values and repeat the steps from above
> for (k in range) {
+   pred = knn(train=train[,1:16], test=test[,1:16], cl=train[,17], k=k)
+   accs[k] = sum(pred!=test[,17])/length(test[,17])
+ }
> # Plot the accuracies.
> plot(range, accs, xlab = "k", ylab = "Error Rate", type='b', pch=18,
+      main = 'Determining the Optimal k
+             Using Validation Data')
> abline(v=which.min(accs), col='red')
> text((which.min(accs)+0.5),0.026, paste(which.min(accs)), col='red')
```