STAT 760 - Machine/Statistical Learning Project 1

K-Nearest Neighbor Algorithm: Handwritten digit recognition

Amos Natido

February 4, 2019

```
library(tidyverse)
```

Introduction

The goal of this project is to classify handwritten digits using a K-Nearest Neighbor classifier. \ The data consists of over 7000 images that have been deslanted and resized to 16x16 pixels. The values of the pixels have also been normalized to the range $[-1, 1]$. Each observation can viewed as a point in 256-dimensional space, and any new observation will be classified according to closeness to neighboring points using the Euclidean distance.

Using linear algebra and matrices we find the L_2 distance between each observation and a target observation that we'd like to make a prediction with. Let X be a $[n \times p]$ dimensional matrix, where n is the number of observations and p is the number of predictors (in our data, there are 256), and let Y be a $[n \times k]$ matrix, where k is the number of classes in the response.

Consider a new observation, $\vec{z} = [z_1 z_2 \dots z_p]$. We want to find the closest points to $\vec{z}$, so in order to do so, we can calculate the L_2 distance in the normal way.

$$
\begin{aligned}
d^{(1)} &= \sqrt{(x_1^{(1)} - z_1)^2 + (x_2^{(1)} - z_2)^2 + \ldots + (x_p^{(1)} - z_p)^2} \\
&\vdots \\
d^{(n)} &= \sqrt{(x_1^{(n)} - z_1)^2 + (x_2^{(n)} - z_2)^2 + \ldots + (x_p^{(n)} - z_p)^2}
\end{aligned}
$$

Computing distances between each observation and new observation in a `for` loop is computationally inefficient. Instead, we can replicate $\vec{z}$ to create an $[n \times p]$ augmented matrix, Z^T. Then the L_2 distance is calculated as follows.

$$
\begin{aligned}
D &= X - Z^T \\
L_2 &= \sqrt{\sum_i d_{ij}^2},
\end{aligned}
$$

where $D = (d_{ij})$. To get the k-nearest neighbors, we take the $\min_k \{L_2\}$, which is the index of the k smallest distances. We use these indices to slice the response values in Y, and can call the responses Y_k. Finally we can take the mean of the columns of Y_j, and find the class with the highest value(posterior probability). Given a positive integer K and a test observation x_o, the KNN classifier first identifies the neighbors K points in the training data that are closest to x_o, represented by N_o based on Euclidean distance. It then

estimates the conditional probability for class j as the fraction of points in N_o whose response values equal j. The mathematical formulation is given by

$$P_r\left(Y = j | X = x_0\right) = \frac{1}{K} \sum_{i \in N_o} I(y_i = j)$$

Loading Data

```
# Read in data sets
train <- read.table("Datasets/zip.train.gz") %>%
                                              as.matrix()
test <- read.table("Datasets/zip.test.gz") %>%
                                              as.matrix()

colnames(train) <- colnames(test) <- NULL

# Split into x, y
x.train <- train[,-1]
y.train <- train[,1]

x.test <- test[,-1]
y.test <- test[,1]

rm(train, test)
```

Data Preprocessing

One-Hot encoding the target variable

The data is provided here, and a description of the data set is provided here. The target variable has the numbers 0-9. Since this is a classification problem where the response is categorical we convert the response to dummy variables using one-hot encoding.

```
# Snapshot of the response variable
y.train %>% head(5)
```

```
[1] 6 5 4 7 3
```

```
# Hot encoding
to_categorical <- function(y) {
  classes <- sort(unique(y))

  k <- length(classes)
  n <- length(y)

  y.cat <- matrix(0, nrow = n, ncol = k)

  for(i in 1:n) {
    class <- y[i] == classes
    y.cat[i, class] <- 1
  }
  colnames(y.cat) <- classes
  y.cat
}
```

```r
# Turn y into categorical response
y.train.cat <- y.train %>% to_categorical()
y.test.cat <- y.test %>% to_categorical()

class_labels <- 0:9

named_class_labels <- c("zero", "one", "two", "three", "four",
                        "five", "six", "seven", "eight", "nine")
```

Writing a KNN Function

Here, we determine the k nearest neighbor points using the Euclidean distance

```
# Returns the label index for each prediction
knn <- function(x_train, y_train, x_test, y_test, k = 1) {

  # observation matrix dimensions
  n <- nrow(x_train)
  p <- ncol(x_train)

  # basic checks to ensure dimensional conformability
  stopifnot(n == nrow(y_train))
  stopifnot(p == ncol(x_test))
  stopifnot(nrow(x_test) == nrow(y_test))
  stopifnot(ncol(y_train) == ncol(y_test))
  stopifnot(k > 0)


  pred_func <- function(z) {
    z_star <- replicate(n, z) %>% t()

    # Calculated L2 distance
    D <- x_train - z_star
    L2 <- rowSums(D^2)

    # Get indices of k nearest neighbors
    k_nearest <- order(L2) %>% head(k)

    # Make prediction
    if (k == 1) {
      knn_response <- y_train[k_nearest, ] %>% as.matrix %>% t()
      knn_mean <- knn_response
    } else {
      knn_response <- y_train[k_nearest, ]

      knn_mean <- colMeans(knn_response)
    }
    return(which.max(knn_mean))
  }

  apply(x_test, 1, pred_func)
}
```

Generating Predictions

```
results <- knn(x_train = x.train, y_train = y.train.cat,
                x_test = x.test, y_test = y.test.cat,
                k = 5)
```

Evaluation

```
# Calculate error rate
correct <- class_labels[results] == y.test
incorrect <- !correct

mean(incorrect)
```

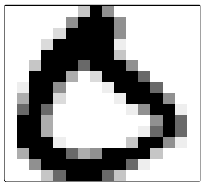
```
[1] 0.05530643
```

With $k = 5$, the error rate is just around 5.5%. Let's take a look at some of the correctly classified and misclassified digits. We'll see that for many of them, it's no surprise that the knn classifier failed. Most of the errors come from numbers being generally similar to each other: nines look like sevens, twos and zeros look like eights, and ones look like sevens.

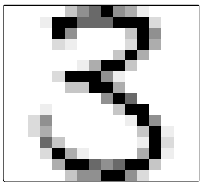
```
print_image <- function(index, x, label) {
                                        rotate <- function(x) t(apply(x, 2, rev))
                                        matrix(x[index, ], nrow = 16, byrow = T) %>%
                                        rotate %>%
                                        image(col = gray.colors(n = 8, 1, 0), xlab = label,
                                        xaxt = 'n', yaxt = 'n')
}
```

Sample of correctly classified Digits

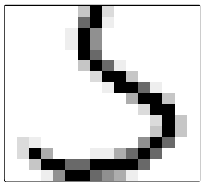
```
for(i in sample(which(correct), 12, F)) {
  title <- paste0("Pred: ", named_class_labels[results[i]],
                  " | Obs: ", named_class_labels[which.max(y.test.cat[i,])])
  print_image(i, x.test, title)
}
```



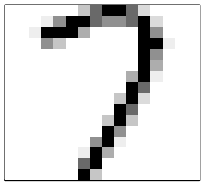
Pred: zero | Obs: zero



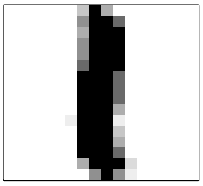
Pred: three | Obs: three



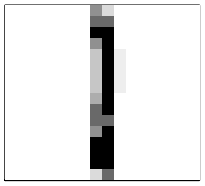
Pred: five | Obs: five



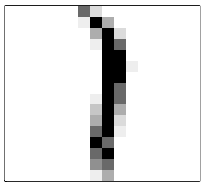
Pred: seven | Obs: seven



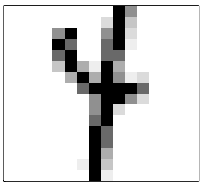
Pred: one | Obs: one



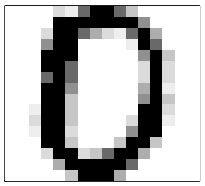
Pred: one | Obs: one



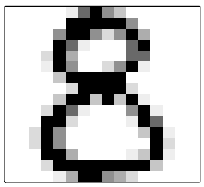
Pred: one | Obs: one



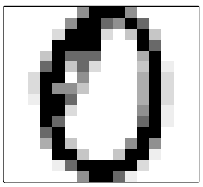
Pred: four | Obs: four



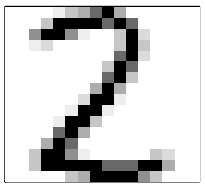
Pred: zero | Obs: zero



Pred: eight | Obs: eight



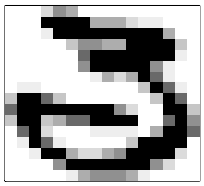
Pred: zero | Obs: zero



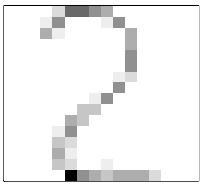
Pred: two | Obs: two

Sample of misclassified Digits

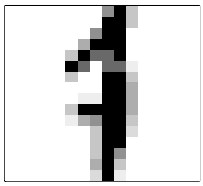
```
for(i in sample(which(incorrect), 12, F)) {
  title <- paste0("Pred: ", named_class_labels[results[i]],
                  " | Obs: ", named_class_labels[which.max(y.test.cat[i,])])
  print_image(i, x.test, title)
}
```



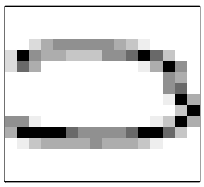
Pred: two | Obs: three



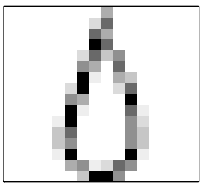
Pred: seven | Obs: two



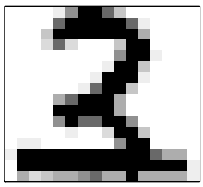
Pred: one | Obs: four



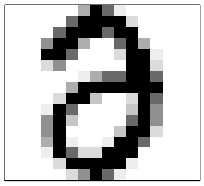
Pred: zero | Obs: three



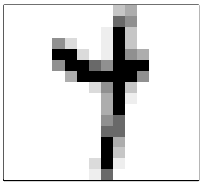
Pred: six | Obs: zero



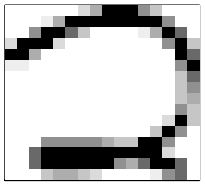
Pred: two | Obs: three



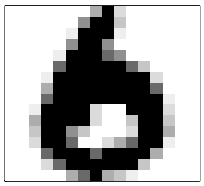
Pred: zero | Obs: two



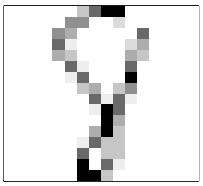
Pred: nine | Obs: four



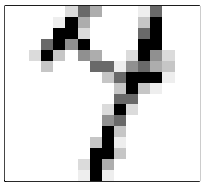
Pred: zero | Obs: two



Pred: zero | Obs: six



Pred: seven | Obs: eight



Pred: seven | Obs: four