Section 1.7

k-Nearest Neighbors

This chapter covers

- The k-Nearest Neighbors classification algorithm
- The concept behind k-Nearest Neighbors
- The Handwriting Recognition Example
- A method for choosing the best k

k-Nearest Neighbors is one of the simplest machine learning algorithms and yet it can be used solve difficult problems such as the automatic recognition of handwritten characters. The United States Post Office has machines to automatically sort letters using the zip codes that are written on the envelop. The first step in this process is for the machine to read the handwritten numbers of the zip code. The concept behind k-Nearest Neighbors has been described as the following: if something looks like a duck, walks like a duck, and quacks like a duck then it must be a duck.

In this chapter, the concept behind the machine learning algorithm, k-Nearest Neighbors, is first illustrated through pictures. Next the data set used in the Handwriting Recognition Example is described. The R code which classifies this data set is presented along with reasons for choosing a specific k. Finally, data set types that k-Nearest Neighbors can be applied to are discussed along with the pros and cons of using the k-nearest neighbor algorithm.

1.7.1 Concept behind k-Nearest Neighbors

k-Nearest Neighbors

Pros: High accuracy, insensitive to outliers

Cons: Computationally expensive, requires a lot of memory

Works with: Numeric values and nominal values

The concept behind the k-Nearest Neighbors (kNN) algorithm is best illustrated with a few simple pictures. The solid dots in Figure 1.7.1 have been labeled with the color they have been filled with. The 0 represents a new data point that should be filled with the appropriate color. In this case as the nearest neighbor to the 0 is a red dot, this new data point should also be classified as red under the kNN algorithm when k is set to one. However, if k is set to three as seen in Figure 1.7.2, the neighborhood of 0 now contains 3 dots. Two of these dots are blue and only one is red. When k = 3, the kNN algorithm would classify the 0 as blue.

The kNN algorithm starts with a training data set in which each observation is labeled and a guess for the number k. For this example assume k = 3. When a new data point which needs to be classified arrives the kNN algorithm looks at the 3 observations from the training data set which are most like the new data point. These 3 observations are called the 3 nearest neighbors of this new data point. The label which occurs the most frequently in this particular neighborhood is assigned as the label for the new data point. In the colored dot example, the distance between the points is used to determine which training observations are close to the new data point.

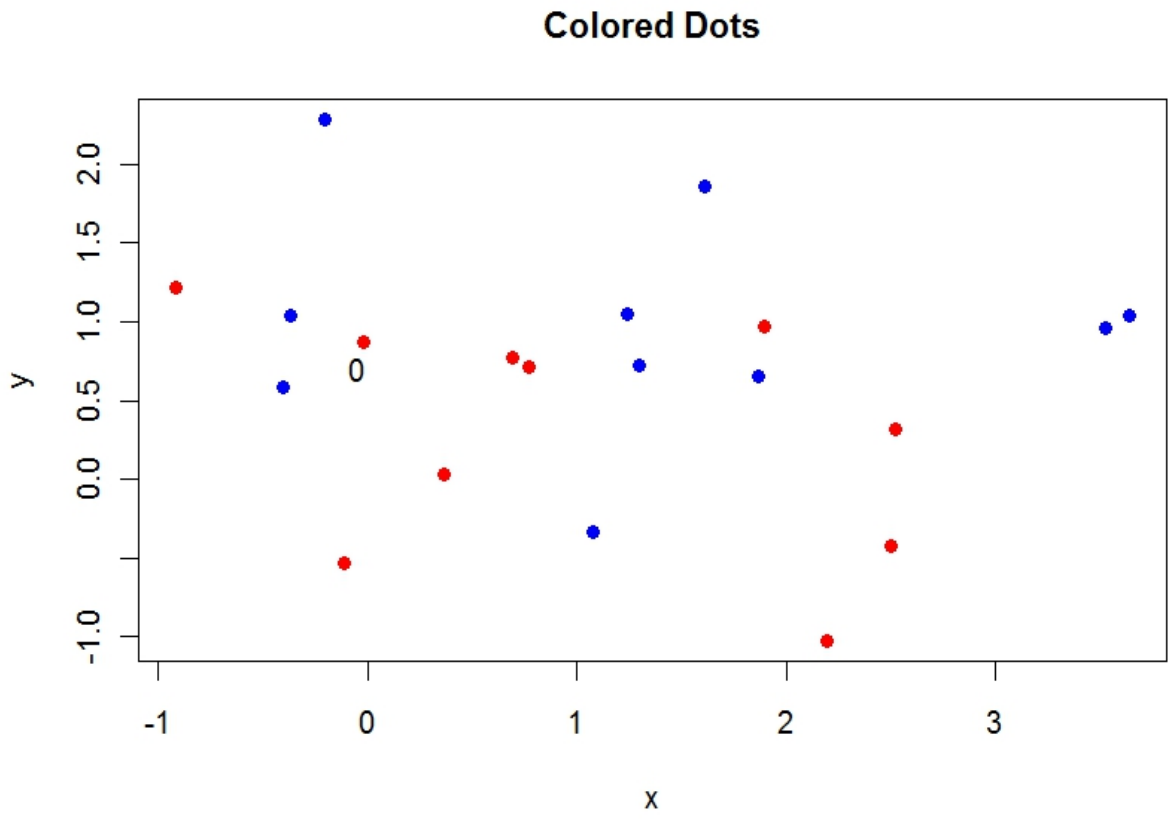

Figure 1.7.1 Colored Dots: classify 0 as red when k = 1

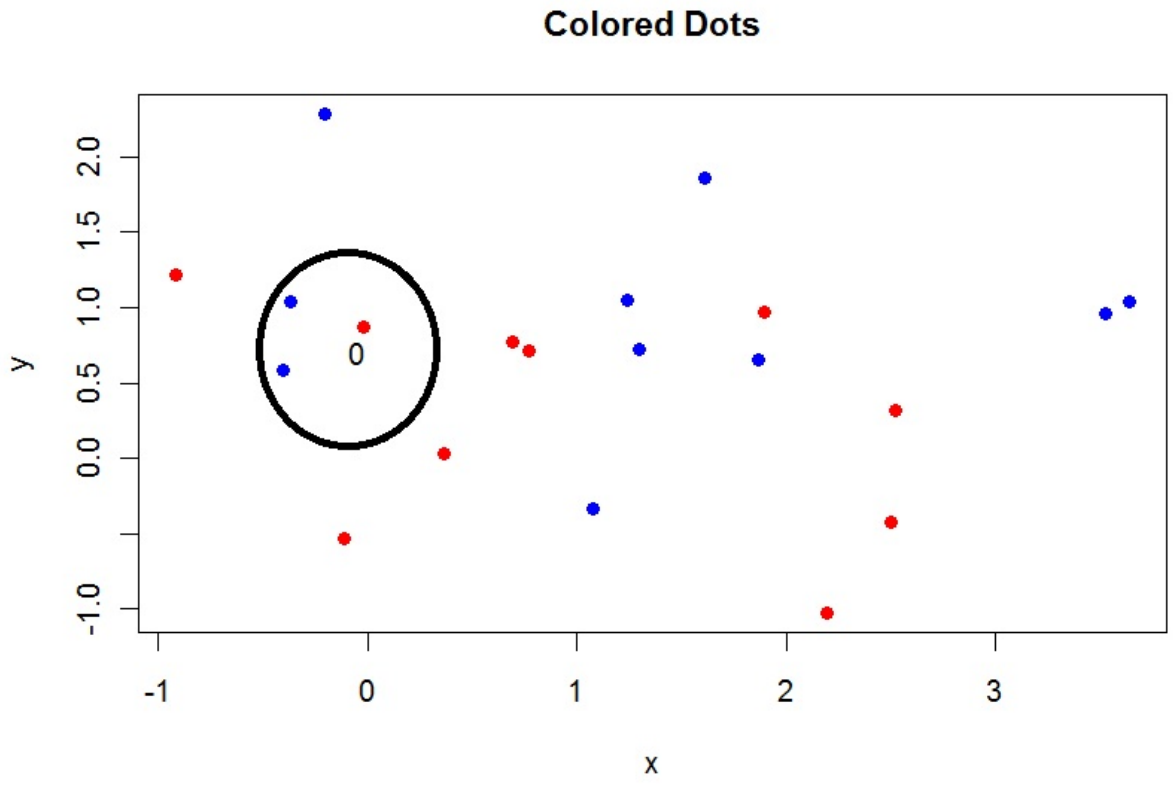

Figure 1.7.2 Colored Dots: classify 0 as blue when k = 3

1.7.2 The Handwriting Data Example

Optical Recognition of Handwritten Digits - Digits 0 through 9

- Each handwritten digit is represented by an 8x8 grid of squares
  (i.e. 64 numbers representing gray scale)
- Each cell in the grid represents one of seventeen different shades of gray
  (i.e. a number between 0 and 16)
- There are 5620 observations of digits (zero through nine)

The data for the Handwriting Data Example was obtained by having 43 different people write numeric digits. A total of 5620 numeric digits were written for this data set. Each individual digit was then divided into an 8x8 grid. Each cell in the grid was then coded with a number from zero to sixteen representing the shade of gray for that cell. Sixteen represents a totally white cell, while zero represents a cell that is completely black. An example of several digits is shown in Figure 1.7.3. The Handwriting Data Set is a table of numbers with size 5620 rows by 65 columns. For this data set each of the 5620 numeric digit observations becomes a row in the data set. Each handwritten digit is represented by 64 cells. The first 64 columns contain the code for the gray scale of that cell. The number in the last column is the label for that row. For example if the person had written a zero, the last element in the row representing that handwritten digit is the number zero. Similarly, if the person had written a one, the last element in the row representing that handwritten digit is the number one.

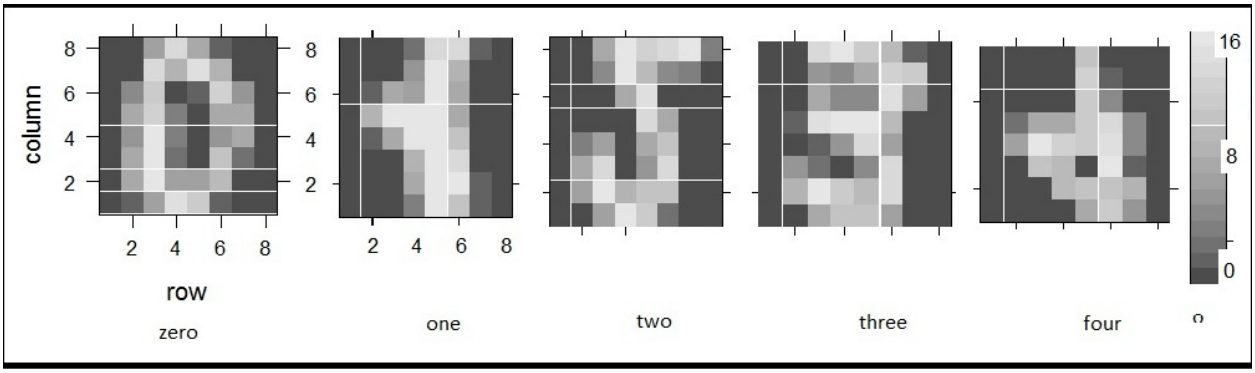

Figure 1.7.3 Example from the Optical Recognition of Handwritten Digits Data Set

The five data rows that are used to create each of the five digits in Figure 1.7.3 are listed here:

0,1,6,15,12,1,0,0,0,7,16,6,6,10,0,0,0,8,16,2,0,11,2,0,0,5,16,3,0,5,7,0,0,7,...14,9,15,9,0,0,0,0,6,14,7,1,0,0,0

0,0,0,3,16,11,1,0,0,0,0,8,16,16,1,0,0,0,0,9,16,14,0,0,0,1,7,16,16,11,0,...0,0,0,5,16,9,0,0,0,0,0,2,14,14,1,0,1

0,0,6,16,12,2,0,0,0,6,16,9,11,11,0,0,0,7,14,0,5,14,0,0,0,3,6,0,7,11,0,0,...4,16,10,4,3,0,0,0,7,16,13,14,16,3,2

0,0,7,11,11,6,0,0,0,9,16,12,10,14,0,0,0,5,2,0,4,14,0,0,0,0,1,5,14,6,...0,0,5,4,8,13,12,0,0,0,14,16,12,10,1,0,3

0,0,0,2,13,1,0,0,0,0,1,15,11,0,0,0,0,0,8,15,2,2,0,0,0,1,16,7,3,16,3,...,0,0,0,1,14,10,0,0,0,0,0,0,15,11,0,0,4

Notice that each row ends with the label for the digit that it represents. Also notice that the first digit of each row happens to be a zero which represents a black cell. The top left hand corner of each of the digits in the figure are black. Each row of the data set contains 65 numbers. A few of the numbers in each row were replaced by ... to save space on this page.

A machine that automatically reads text from a sheet of paper is called an optical character reader. The Optical Recognition of Handwritten Digits Data Set is part of the University of California at Irvine Machine Learning Repository: http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits This data set has been divided into two parts. The training portion has 3823 observations (rows) while the test set has 1797 observations making a total of 5620 observations.

The R code for creating Figure 1.7.3 is in the following listing:

Listing 1.7.1 Illustrating Handwritten Digits - Creating Figure 1.7.3

```
# Download the Training Data Set to your computer from
# http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/
# name the file: optdigits-tra.txt
# setwd("location of Data Set on your computer")

#1 input training digits
odtra<-read.table("optdigits-tra.txt",sep=",", header = FALSE)

#2 Function to diplay greyscale digits
dispDig <- function(k, digData) {
        onerow <- digData[k,1:64] #Digit on row k
        onerow_nmatrix <- matrix(as.numeric(onerow),nrow=8,ncol=8)
        levelplot(onerow_nmatrix, col=gray)
}

#3 Display digits
dispDig(1,odtra)   # Digit 0
dispDig(12,odtra)  # Digit 1
dispDig(42,odtra)  # Digit 2
dispDig(15,odtra)  # Digit 3
dispDig(101,odtra) # Digit 4
```

#1 Statement which reads in the training data set : odtra is the variable representing the data frame. odtra has 65 columns. (remember the table in R is called a data frame)
#2 Function used to plot one data point: Turns 64 numbers from a row of data into a 8x8 matrix and plots this matrix
#3 Five calls to the function to plot the example digits : Rows 1, 12, 42 , 15, and 101 of the odtra data frame are plotted

1.7.3 Applying k-Nearest Neighbors to the Handwriting Data Set

k-Nearest Neighbors

1: Read in the Data

2: Use the training data to determine the best value for k

3: Use the testing data to calculate the error

The function in R used to execute the kNN algorithm is simply knn. The R function, knn, requires four arguments. The first argument is the training data. The second argument is the test data. Next are the correct labels for the training data, followed by k, the size of the neighborhood. Euclidian distance is used by the knn function to determine the closeness of two data points. Start with the first test observation. This test observation

is compared to each of the training data points. The labels for the k training points which are the closest to the test observation are used to determine the label for that test observation. The label which occurs most frequently in the k nearest neighbors is assigned to this first test observation. The knn functions assigns a label to each of the test observations and returns these labels as a vector. The knn function is illustrated in the example Listing 1.7.2: "Using kNN to classify Handwritten Digits".

Besides Euclidian distance, there are other ways to determine how close data points are to each other. There is an R Package, knnflex, which contains the R function knn.dist which allows user defined distance functions.

Recall that cross validation has been previously discussed. It is a technique for assessing how the results of a prediction method will generalize to an independent data set. It is use to estimate how accurately a predictive model will perform in practice. Leave on out cross validation involves using a single observation from the original training set for validation and using the remaining observations as the training data. This is repeated until each observation in the sample is used once as the validation data.

The R function knn.cv is used to determine the best k to use in the kNN algorithm for a particular data set. The best k is the value which results in the smallest error. The leave one out cross validation function for k-Nearest Neighbors is described here. The function knn.cv only requires the training data set along with the labels for the training data set and a value for k. Each row of the training data set represents one observation. To determine the best k, knn.cv is called with a selection of k values. In the example code, k runs from one to twenty. The first call of knn.cv is with k set to one. The first step of this call is to predict the label for the first observation using all the training data but this first observation. In other words, the first row is "left out" when making the prediction for the first row. knn.cv looks through the rest of the training set to find the observation that is closest to the first row. The label predicted for the first row is then set to the label from that closest observation. The second step of knn.cv is to predict the label for the second row. This time the second row is "left out" when making the prediction for the second row. In this second step of knn.cv, only the second row is removed from the training set. The label predicted for the second row is then set to the label from the closest observation in the remaining training set. These steps are continued until a prediction has been made for each row. In the last step of this call to knn.cv, only the last row is "left out" when making the prediction of the label for the last row. The prediction for each row is compared with the actual label read from the data set for that row. If there is a match the prediction is correct, however if there is not a match, the prediction is wrong. In Listing 1.7.2: "Using kNN to classify Handwritten Digits", knn.cv is called with 20 different values for k. The value of k that results in the largest number of correct predictions is the best k. This best k is used for deploying this algorithm. The test data set is then used to validate the use of this algorithm. An error is calculated using the kNN algorithm with this best k on the test data set.

The R code for classifying the Handwritten Digits using kNN is in the following listing:

Listing 1.7.2 Using kNN to classify Handwritten Digits

```
# Read the Training Data
odtra<-read.table("optdigits-tra.txt",sep=",", header = FALSE)
train <- odtra[,1:64]
labels <- odtra[,65]
Err <- rep(0,20) #variable used to store the error

#1 Find the best k to use
for(kk in seq(from=1,to=20)){
       out <- knn.cv(train,labels,k=kk)
       Error <- 1-sum(abs(labels == out))/length(out)
       Err[kk] <- Error
}
Err
plot(Err)

bestk = which.min(Err) #bestk is the index of the smallest number in the Err vector
bestk # k = 3 results in the lowest error

# Classify the test data and calculate the error

# Read in the Test data
```

```
odtes<-read.table("optdigits-tes.txt",sep=",", header = FALSE)
test <- odtes[,1:64]
testLabels <- odtes[,65]

#2 Classify the Test Data using the best k
out <- knn(train, test, labels, k = bestk)

#3 Calculate the Error
Error <- 1-sum(abs(testLabels == out))/length(out)
Error # 0.02
```

#1 Find the best k: knn.cv is the built in R function which executes the kNN algorithm using only the training data. It uses leave one out cross validation. This loops through various values of k and computes the error.
#2 Classify the Test Data: After the call to the built in R function, knn, the vector variable, out, contains the predicted labels for the test data
#3 Calculate the Error: The predicted labels are compared with the actual labels which are stored in the vector testLabels

1.7.4 k-Nearest Summarized

The kNN algorithm is very easy to implement. As long as the training set is large enough, it is a very accurate algorithm. It is robust to outliers. It can be used with both numeric and nominal values as long as there is a way to compute the "closeness" of data points. There are methods to determine the closeness of nominal values. It can be run online. That is as new data arrives, the new data can easily be added for making future predictions.

There are a few draw backs for kNN. There is no "model". All of the training data is required to make predictions. If the training data set is too small or in high dimensional cases, kNN results in poor accuracy. In these cases the distance between observations in the training data set and the point requiring a prediction may be too far. When making a prediction, it must be compared to each observation in the training data set. Although this is time consuming, there are cleaver ways to overcome this obstacle. As all of the training data set is continually used it must be kept in storage.