Lab 4-1 KNN and Cancer Diagnosis

Joseph Yang

A KNN analysis on a cancer diagnostic data set.

I will be creating a KNN classifier for data on breast cancer biopsies. Data: Wisconsin Diagnostic Breast Cancer (WDBC) data set. From the UC Irvine Machine Learning Repository, http://archive.ics.uci.edu/ml/index.html

First, establish a seed value for the entire lab. This value is used whenever a call to a random process is required.

```
seed.val <- 1234
```

Read in the data from the file "wdbc_data.csv" and assign it to the variable "df.wdbc".

```
df.wdbc <- read.csv("data/week6/wdbc_data.csv")
```

Data exploration and transformation. The column "diagnosis" is character data, with levels: "B" for benign, or non-cancerous, and "M" for malignant, or cancerous. Let's see how many of each are in the data set.

```
summary(as.factor(df.wdbc[, 2]))
```

```
##   B   M
## 357 212
```

We can see there are 357 benign and 212 malignant cases are in this data set.

I will apply a transformation to the data set to scale all values to the same range 0-1. This transformation is a type of "normalization". The formula for normalization is: normalized = (value-min)/(max-min) When normalize or otherwise apply a transformation to your data, be careful that it will not change the proportions of the values. Normalization, standardization, and log are examples of safe transformations to use. The following function will do the normalization.

```
normalize.it <- function(vec) {
    y <- (vec - min(vec))/(max(vec) - min(vec))
    y
}
```

Now I will create the normalized data set to use for the KNN classifier. Apply the "normalize.it" function to the dataset- but only to the columns 3 to 32 as we do not want to normalize the id and the diagnosis columns. There are three statements to write here.

1- Normalize a subset of the data. A call to lapply and pass it a data set you create by selecting all rows and only the columns 3 through 32 (i.e. all columns starting at index 3 and ending at index 32) from df.wdbc, and also pass in the function normalize.it, which was defined above.

Assign the call to lapply to a variable "wbcd.lst".

2- Convert list to dataframe. The lapply() function returns a list (that is the "l" in "lapply"), so we have to convert it to a data frame. Assign that result to a variable called df.wbcd.normed. That is now our normalized data set that I will use to create a KNN classifier.

3- Note that df.wbcd.normed should not have the id or the diagnosis columns. The "head" function allows you to check that before moving on.

```
wbcd.lst <- lapply(df.wdbc[, 3:32], normalize.it)
df.wbcd.normed <- data.frame(wbcd.lst)
head(df.wbcd.normed)
```

```
##   radius_mean texture_mean perimeter_mean area_mean smoothness_mean
## 1   0.5210374    0.0226581      0.5459885 0.3637328       0.5937528
## 2   0.6431445    0.2725736      0.6157833 0.5015907       0.2898799
## 3   0.6014956    0.3902604      0.5957432 0.4494168       0.5143089
## 4   0.2100904    0.3608387      0.2335015 0.1029056       0.8113208
## 5   0.6298926    0.1565776      0.6309861 0.4892895       0.4303512
## 6   0.2588386    0.2025702      0.2679842 0.1415058       0.6786133
##   compactness_mean concavity_mean points_mean symmetry_mean dimension_mean
## 1        0.7920373      0.7031396   0.7311133     0.6863636      0.6055181
## 2        0.1817680      0.2036082   0.3487575     0.3797980      0.1413227
## 3        0.4310165      0.4625117   0.6356859     0.5095960      0.2112468
## 4        0.8113613      0.5656045   0.5228628     0.7762626      1.0000000
## 5        0.3478928      0.4639175   0.5183897     0.3782828      0.1868155
## 6        0.4619962      0.3697282   0.4020378     0.5186869      0.5511794
##    radius_se texture_se perimeter_se    area_se smoothness_se compactness_se
## 1 0.35614702 0.12046941   0.36903360 0.27381126     0.1592956     0.35139844
## 2 0.15643672 0.08258929   0.12444047 0.12565979     0.1193867     0.08132304
## 3 0.22962158 0.09430251   0.18037035 0.16292179     0.1508312     0.28395470
## 4 0.13909107 0.17587518   0.12665504 0.03815479     0.2514532     0.54321507
## 5 0.23382220 0.09306489   0.22056260 0.16368757     0.3323588     0.16791841
## 6 0.08075321 0.11713225   0.06879329 0.03808008     0.1970629     0.23431069
##   concavity_se points_se symmetry_se dimension_se radius_worst texture_worst
## 1   0.13568182 0.3006251  0.31164518    0.1830424    0.6207755     0.1415245
## 2   0.04696970 0.2538360  0.08453875    0.0911101    0.6069015     0.3035714
## 3   0.09676768 0.3898466  0.20569032    0.1270055    0.5563856     0.3600746
## 4   0.14295455 0.3536655  0.72814769    0.2872048    0.2483102     0.3859275
## 5   0.14363636 0.3570752  0.13617943    0.1457996    0.5197439     0.1239339
## 6   0.09272727 0.2153817  0.19372995    0.1446596    0.2682319     0.3126333
##   perimeter_worst area_worst smoothness_worst compactness_worst concavity_worst
## 1       0.6683102 0.45069799        0.6011358         0.6192916       0.5686102
## 2       0.5398177 0.43521431        0.3475533         0.1545634       0.1929712
## 3       0.5084417 0.37450845        0.4835898         0.3853751       0.3597444
## 4       0.2413467 0.09400806        0.9154725         0.8140117       0.5486422
## 5       0.5069476 0.34157491        0.4373638         0.1724151       0.3194888
## 6       0.2639076 0.13674794        0.7127386         0.4827837       0.4277157
##   points_worst symmetry_worst dimension_worst
## 1    0.9120275      0.5984624       0.4188640
## 2    0.6391753      0.2335896       0.2228781
## 3    0.8350515      0.4037059       0.2134330
## 4    0.8848797      1.0000000       0.7737111
## 5    0.5584192      0.1575005       0.1425948
## 6    0.5982818      0.4770353       0.4549390
```

Create train and test sets. Let's use 2/3 for train and 1/3 for test. In a series of three statements, I will create variables that hold the data size, training set size, and the number of labels in the test set, which is essentially the test set size.

1- First, calculate the data size, i.e. the number of rows, in the normed data set we created in above and assign it to a variable called "data.size". 2- Then create a variable called training.size and assign it to 0.66, which is about 2/3. 3- Finally, create a variable "num.test.labels" and assign to it the number of labels in the test data set (this is the number of rows that will be in the test set, or about 1/3 of the data). Use the variables we just created in the first two steps to calculate num.test.labels. This is the size of the entire data set * 1- size of the training set.

```
data.size <- nrow(df.wbcd.normed)
training.size <- 0.66
num.test.labels <- (1 - training.size) * data.size
```

Create the training set. This will be done in three steps: 1- Call the set.seed function using the seed value.

2- Assign to a variable "train.row.nums" the row numbers to be used in the training set, selected by random.

3- Obtain the training set. Assign to the variable train.data the rows sepcified by train.row.nums from the normed dataframe, df.wbcd.normed.

```
RNGversion("3.4.3")
```

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-uniform
## 'Rounding' sampler used
```

```
set.seed(seed.val)
train.row.nums <- sample(1:data.size, data.size*training.size, replace = FALSE)
train.data <- subset(df.wbcd.normed[train.row.nums, ])
```

Create the test set in a similar way. First create a vector that specifies the rows we want to be in the test set. This set should contain all of the rows that are not in the training set. This can be seen as the set difference (like subtraction for sets) between the data and the training data: test set rows = data set rows - train set rows

Use the setdiff function to create this vector and assign it to a variable called test.row.nums.

```
test.row.nums <- setdiff(1:data.size, train.row.nums)
test.data <- subset(df.wbcd.normed[test.row.nums, ])
```

Let's get the training and test set labels. Create a variable called class.labels and assign to it the values for diagnosis for the row numbers specified by train.row.nums. Create a variable called true.labels and assign to it the values for diagnosis for the row numbers specified by test.row.nums. Remember that the diagnosis cilumn was not included in the normalized data set, so use the df.wdbc data frame here.

```
class.labels <- df.wdbc[train.row.nums, 2]
true.labels <- df.wdbc[test.row.nums, 2]
```

Choose k, do modeling with knn, predict(classify) and evaluate. Next choose a number for k, the number of nearest neighbors. One "rule of thumb" is to set k to be the nearest integer to the square root of the size of the training set. We'll be using the "floor" function on the call to sqrt so the value of k is an integer.

```
k <- floor(sqrt(training.size * data.size))
k
```

```
## [1] 19
```

Load the library, "class"", for knn, then call the knn function and assign the result to the variable knn.pred.

```
library(class)
knn.pred <- knn(train.data, test.data, class.labels, k)
```

Calculate the misclassification rate. The misclassification rate is the number of rows where the knn prediction does not match the true labels.

```
num.incorrect.labels <- sum(knn.pred != true.labels)
misc.rate <- num.incorrect.labels / num.test.labels
misc.rate
```

```
## [1] 0.04135222
```

Create a confusion matrix using the table function and assign it to a variable called "conf.matrix".

```
conf.matrix <- table(knn.pred, true.labels)
conf.matrix
```

```
##          true.labels
## knn.pred   B   M
##        B 116   5
##        M   3  70
```

Let's explain the results in this table using the following terminology. (we assume Malignant is a "positive" result): A true positive: the model and true label = Malignant. A false positive: the model = Malignant and true label = Benign A false negative: the model = Benign and true label = Malignant A true negative: the model and true label = Benign

> true positive: (M,M): 70. false positive: (M,B): 3. false negative: (B,M): 5. true negative: (B,B): 116. I think the model did good with a misc rate of 0.041 and true positive rate of a 36% and true negative 60%. It did good, not too bad but a lower false negative rate would be preferable.

Explore other values of k. We calculated the misclassification rate for k=19. Now let's find out is there a value for k that would result in a lower misclassification rate? Using a "for" loop, run knn on values of k from 1 to 20 (use k as the loop index). Declare an empty vector, call it "results", before the loop, then update it in the loop with the misclassification rate for each value of k. In the loop.

After the loop finishes, we have the vector "results" which contains 20 misclassification rates for the values of k from 1 to 20.

```
results <- c()
for (k in 1:20) {
  knn.pred.labels <- knn(train.data, test.data, class.labels, k)
  num.incorrect.labels <- sum(knn.pred.labels != true.labels)
  misc.rate <- num.incorrect.labels / num.test.labels
  results[k] <- misc.rate
}
results
```

```
##  [1] 0.03101416 0.05169027 0.02584514 0.02584514 0.01550708 0.01550708
##  [7] 0.02067611 0.01550708 0.01550708 0.02584514 0.02067611 0.02584514
## [13] 0.03101416 0.02584514 0.02584514 0.03618319 0.03618319 0.02584514
## [19] 0.04135222 0.03618319
```

As we can see when k=5 we have the lowest misclassification rate at 0.01550708.