Sean Toon

IT-460-18500-M01 Machine Learning 2024 C-5 (Sep - Oct)

October 12, 2024

Module Five Labs: k-NN and k -Means

The k-NN Algorithm:

The data set that I am using for the k-NN lab is the Breast Cancer Wisconsin (Diagnostic) dataset donated from the University of Wisconsin and it includes 569 examples of breast cancer biopsies, each with 32 features. I used the command below to show the data structure and some of the features included in the data set after importing the CSV file:

```
> str(wbcd)
'data.frame':   569 obs. of  32 variables:
 $ id                     : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844981 84501001 ...
 $ diagnosis              : chr  "M" "M" "M" "M" ...
 $ radius_mean            : num  18 20.6 19.7 11.4 20.3 ...
 $ texture_mean           : num  10.4 17.8 21.2 20.4 14.3 ...
 $ perimeter_mean         : num  122.8 132.9 130 77.6 135.1 ...
 $ area_mean              : num  1001 1326 1203 386 1297 ...
 $ smoothness_mean        : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
 $ compactness_mean       : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
 $ concavity_mean         : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
 $ concave.points_mean    : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
 $ symmetry_mean          : num  0.242 0.181 0.207 0.26 0.181 ...
```

Next, I used the following command to drop the first column of the data set “id” to ensure that it doesn’t overfit or cause poor generalizations on future data:

```
> wbcd <- wbcd[-1]
```

One important variable that will need to be considered for the outcome is the “diagnosis”, but the feature is not considered a factor yet. So, I used the code below to recode the variable and renamed the labels so that they were more informative:

```
> wbcd$diagnosis <- factor(wbcd$diagnosis, levels = c("B", "M"),labels = c("Benign", "Malignant"))
```

I then looked at the prop.table() output with the following command:

```
> round(prop.table(table(wbcd$diagnosis)) * 100, digits = 1)

   Benign Malignant
     62.7      37.3
```

Another issue with the data is that the values of the features vary greatly in ranges. This could cause problems for the future outcome, so I used the following normalization process to rescale the features to a standard range of values, with the following code: (I also include the code I used to test if the normalization function was working correctly on a couple of vectors)

```
> normalize <- function(x) {
+       return ((x - min(x)) / (max(x) - min(x)))
+     }
> normalize(c(1, 2, 3, 4, 5))
[1] 0.00 0.25 0.50 0.75 1.00
> normalize(c(10, 20, 30, 40, 50))
[1] 0.00 0.25 0.50 0.75 1.00
>
```

Now time to apply the normalization function to all the numeric variables in the data set using the lapply() function, and then apply the list returned to a data frame using the as.data.frame() function:

```
wbcd_n <- as.data.frame(lapply(wbcd[2:31], normalize))
```

Then I used the following command to check if it was applied correctly:

```
> summary(wbcd_n$area_mean)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.1174  0.1729  0.2169  0.2711  1.0000
```

The next step is to prepare the data by creating training and test datasets. The training and test datasets that I created represent a simulation of a scenario of 100 masses of unknown cancer status, compared to diagnoses obtained using conventional methods. I split the data by using 469

records for the training set and the remaining 100 to simulate new patients using the following code:

```
> wbcd_train <- wbcd_n[1:469, ]
> wbcd_test <- wbcd_n[470:569, ]
```

Because I excluded the target variable "diagnosis" in the normalizing training and test datasets, I had to store those class labels in factor vectors, split between the training and test datasets using the following code:

Now it is time to train a model on the data using the k-NN algorithm. The process simply involves storing the input data in a structured format, and then using Euclidean distance to classify our factor vectors by using a certain number of nearest neighbors that have also been classified. First, I needed to install the "class" package to use a basic set of R functions for classification:

```
install.packages("class")
```

At this point, I gathered everything needed for the k-NN algorithm, except for the value of k. The value of k specifies the number of neighbors that will be included in the final vote. I used the value of 21, to make sure that there is not a tie vote. I used the following code and the parameters are the test and training sets, the factor vectors, and the number of nearest neighbors.

```
> wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
+                       cl = wbcd_train_labels, k = 21)
```

Next, to evaluate the model performance, I compared the predicted classes stored in "wbcd_test_pred" with the acutal values in the "wbcd_test_labels" vector, using the CrossTable() function in the "gmodels" package below:

```
> CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,
+              prop.chisq = FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  100


                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        77 |         0 |        77 |
                 |     1.000 |     0.000 |     0.770 |
                 |     0.975 |     0.000 |           |
                 |     0.770 |     0.000 |           |
-----------------|-----------|-----------|-----------|
       Malignant |         2 |        21 |        23 |
                 |     0.087 |     0.913 |     0.230 |
                 |     0.025 |     1.000 |           |
                 |     0.020 |     0.210 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        79 |        21 |       100 |
                 |     0.790 |     0.210 |           |
-----------------|-----------|-----------|-----------|
```

The results show that the k-NN model produced results of 98 percent accurate!

The results were great, but there could be ways to optimize the model. I then used the z-z- score standardization method for rescaling the numeric features and tried different values of k. For the z-score standardization, I used the scale() function below and then created all of the sets as well as the cross table to check the new accuracy:

```
wbcd_z <- as.data.frame(scale(wbcd[-1]))
summary(wbcd_z$area_mean)
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
.4532 -0.6666 -0.2949  0.0000  0.3632  5.2459
 wbcd_train <- wbcd_z[1:469, ]
wbcd_test <- wbcd_z[470:569, ]
wbcd_train_labels <- wbcd[1:469, 1]
wbcd_test_labels <- wbcd[470:569, 1]
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
                      cl = wbcd_train_labels, k = 21)
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,
           prop.chisq = FALSE)
```

It turns out that the method decreased the accuracy of the model, so I tried to use the caret package to try and find a k parameter to increase the accuracy. I used the following commands to find that the best model accuracy used a value of k=9:

```
> train_control <- trainControl(method = "cv", number = 10)
```

```
set.seed(123)
```

```
> knn_fit <- train(wbcd_train, wbcd_train_labels,
+                  method = "knn",
+                  tuneGrid = expand.grid(k = seq(1, 30, by = 2)),
+                  trControl = train_control)
> print(knn_fit)
k-Nearest Neighbors

469 samples
 30 predictor
  2 classes: 'Benign', 'Malignant'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 422, 422, 423, 422, 422, 422, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.9553191  0.9068889
   3  0.9702128  0.9373217
   5  0.9659574  0.9284884
   7  0.9659574  0.9284871
   9  0.9702128  0.9373983
  11  0.9680851  0.9332481
  13  0.9637835  0.9237258
  15  0.9659574  0.9283351
  17  0.9637835  0.9240286
  19  0.9594820  0.9141050
  21  0.9594820  0.9141050
  23  0.9573543  0.9094947
  25  0.9530990  0.9005054
  27  0.9488437  0.8911999
  29  0.9530990  0.9001865

Accuracy was used to select the optimal model using the largest valu
The final value used for the model was k = 9.
```

```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|

Total Observations in Table:  100

```

wbcd_test_labels	wbcd_test_pred: Benign	Malignant	Row Total
Benign	75	2	77
	0.974	0.026	0.770
	0.987	0.083	
	0.750	0.020	
Malignant	1	22	23
	0.043	0.957	0.230
	0.013	0.917	
	0.010	0.220	
Column Total	76	24	100
	0.760	0.240	

The results showed that the k value of 9 decreased the accuracy of the previous value of 21!

The k-Means Algorithm:

The k-Means algorithm is very similar to the k-NN algorithm, except that it uses clusters instead of classifications to model data. The way that the k-Means algorithm works is, first, examples are assigned to an initial set of k clusters, then assignments are updated by adjusting the cluster boundaries according to the examples that currently fall into the cluster.

The data that I used for an example of the k-Means Algorithm is a random sample of SNS (Social Networking Services) profiles of 30,000 high school students. The data sampled over four high school graduation years representing senior, junior, sophomore, and freshman classes. The data recorded includes text from SNS profiles, gender, age and number of SNS friends. From the downloaded texts of the SNS profiles, out of the top 500 words used, 36 words were chosen to represent the categories: extracurricular, activities, fashion, religion, romance and antisocial behavior. The final dataset shows each person and how many times the words chosen to represent the categories were used. I used the following command to look at the data:

```
> teens <- read.csv("C:\\Users\\toons\\Downloads\\snsdata.csv")
> str(teens)
'data.frame':   30000 obs. of  40 variables:
 $ gradyear    : int  2006 2006 2006 2006 2006 2006 2006 2006 2006 2006 ...
 $ gender      : chr  "M" "F" "M" "F" ...
 $ age         : num  19 18.8 18.3 18.9 19 ...
 $ friends     : int  7 0 69 0 10 142 72 17 52 39 ...
 $ basketball  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ football    : int  0 1 1 0 0 0 0 0 0 0 ...
 $ soccer      : int  0 0 0 0 0 0 0 0 0 0 ...
 $ softball    : int  0 0 0 0 0 0 0 1 0 0 ...
 $ volleyball  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ swimming    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ cheerleading: int  0 0 0 0 0 0 0 0 0 0 ...
 $ baseball    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ tennis      : int  0 0 0 0 0 0 0 0 0 0 ...
 $ sports      : int  0 0 0 0 0 0 0 0 0 0 ...
 $ cute        : int  0 1 0 1 0 0 0 0 0 1 ...
 $ sex         : int  0 0 0 0 1 1 0 2 0 0 ...
 $ sexy        : int  0 0 0 0 0 0 0 1 0 0 ...
 $ hot         : int  0 0 0 0 0 0 0 0 0 1 ...
 $ kissed      : int  0 0 0 0 5 0 0 0 0 0 ...
 $ dance       : int  1 0 0 0 1 0 0 0 0 0 ...
```

Next, upon analyzing the data, I can see that there are missing values in gender column and the data in the age column could cause problems for the final analysis because the range does not represent the correct ages of high school students. So, I used the following code first, include gender data that is recorded as NA and then to recode the age variable with an ifelse() function that assigns a value of NA to age if it is not in the range of 13 to 20 years old:

```
>  table(teens$gender, useNA = "ifany")

    F     M  <NA>
22054  5222  2724
> teens$age <- ifelse(teens$age >= 13 & teens$age < 20,
+                         teens$age, NA)
> summary(teens$age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  13.03   16.30   17.27   17.25   18.22   20.00    5523
```

The methods caused a serious missing data issue, so I used dummy coding in the following code for the gender data to add a category for unknown gender to get back some of the NA values.

```
> teens$female <- ifelse(teens$gender == "F" &
+                           !is.na(teens$gender), 1, 0)
> teens$no_gender <- ifelse(is.na(teens$gender), 1, 0)
> table(teens$gender, useNA = "ifany")

    F     M  <NA>
22054  5222  2724
> table(teens$female, useNA = "ifany")

    0     1
 7946 22054
> table(teens$no_gender, useNA = "ifany")

    0     1
27276  2724
```

Now time to eliminate the missing age values using the strategy of imputation by guessing the age of a student based off graduation year. I used the following to calculate the mean value of the ages by removing missing values first and then performing the calculation:

```
> mean(teens$age, na.rm = TRUE)
[1] 17.25243
```

Next, I used the following code to calculate the mean age based on each graduation year with the aggregate() function:

```
> aggregate(data = teens, age ~ gradyear, mean, na.rm = TRUE)
  gradyear      age
1     2006 18.65586
2     2007 17.70617
3     2008 16.76770
4     2009 15.81957
```

Because the mean age of each graduation year is about a year apart, I know that the data is reasonable and it can be used to merge back into the original data containing the missing values. I did so using the ave() function and an ifelse() function:

```
> ave_age <- ave(teens$age, teens$gradyear, FUN =
+                    function(x) mean(x, na.rm = TRUE))
>  teens$age <- ifelse(is.na(teens$age), ave_age, teens$age)
> summary(teens$age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  13.03   16.28   17.24   17.24   18.21   20.00
```

After cleaning up the data, I was able to begin my cluster analysis. I used the stats package for the implementation of the algorithm. First I started by making a data frame that only considered the 36 features that represented the number of times various interests appeared on SNS profiles with the following code:

```
interests <- teens[5:40]
```

Then, I applied a z-score standardization to the interests data frame, with the scale() and lapply() functions. Because the lappy() function returns a matrix, I then had to use as.data.frame() function to coerce the matrix back to data frame form. By doing so I avoided the problem of which some features could come to dominate because of a larger range of values:

```
> interests_z <- as.data.frame(lapply(interests, scale))
> summary(interests$basketball)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.2673  0.0000 24.0000
> summary(interests_z$basketball)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-0.3322 -0.3322 -0.3322  0.0000 -0.3322 29.4923
```

The value of k in for the kmeans() function determines how many clusters are used. For my first trial, I used 5 clusters and identified the clusters as a brain, an athlete, a basket case, a princess, and a criminal. Also, to use random starting points I used the set.seed() function:

```
> RNGversion("3.5.2")
Warning message:
In RNGkind("Mersenne-Twister", "Inversion", "Rounding") :
  non-uniform 'Rounding' sampler used
> set.seed(2345)
>
> teen_clusters <- kmeans(interests_z, 5)
```

I used the following command to check the size of the clusters to determine if they are too large or too small for use:

```
> teen_clusters$size
[1]   871   600  5981  1034 21514
```

I then used the following command to examine the clusters further by determining the coordinates of the cluster centroids of each interest, and gather more information about how accurate the clusters were based on the interests of teenagers:

```
> teen_clusters$centers
   basketball   football      soccer    softball
1  0.16001227  0.2364174  0.10385512  0.07232021
2 -0.09195886  0.0652625 -0.09932124 -0.01739428
3  0.52755083  0.4873480  0.29778605  0.37178877
4  0.34081039  0.3593965  0.12722250  0.16384661
5 -0.16695523 -0.1641499 -0.09033520 -0.11367669
```

Next, I applied the cluster back on to the full dataset. I created a teen_clusters object with the kmeans() function with a component named cluster that contained the cluster assignments of all 30,000 individuals. I added the data as a teens column on the data frame:

```
 teens$cluster <- teen_clusters$cluster
teens[1:5, c("cluster", "gender", "age", "friends")]
cluster gender    age friends
      5      M 18.982       7
      3      F 18.801       0
      5      M 18.335      69
      5      F 18.875       0
      4   <NA> 18.995      10
```

I then used the aggregate function to look at the demographic characteristics, the gender, and the number of friends relative to the clusters:

```
>  aggregate(data = teens, age ~ cluster, mean)
  cluster      age
1       1 16.86497
2       2 17.39037
3       3 17.07656
4       4 17.11957
5       5 17.29849
```

```
> aggregate(data = teens, female ~ cluster, mean)
  cluster    female
1       1 0.8381171
2       2 0.7250000
3       3 0.8378198
4       4 0.8027079
5       5 0.6994515
```

```
> aggregate(data = teens, friends ~ cluster, mean)
  cluster  friends
1       1 41.43054
2       2 32.57333
3       3 37.16185
4       4 30.50290
5       5 27.70052
```

Overall, the associations found prove that the clusters could be useful predictors of behavior.

Lastly, I changed trained the k-means model with the values k =3 and then k=10. My results showed that for k = 3 the mean age and proportion of females is more balanced across clusters compared to k = 5 and there is less variability of data in the clusters. When I trained the model with a value of k = 10, the results showed that the age distribution was similar, but female proportions were more various with more clusters. There were significant changes of proportions in certain clusters.

k=3:

```
>  aggregate(data = teens, age ~ cluster, mean)
  cluster      age
1       1 17.29839
2       2 17.02952
3       3 17.09781
> aggregate(data = teens, female ~ cluster, mean)
  cluster    female
1       1 0.7007816
2       2 0.8533690
3       3 0.8064516
```

k=10:

```
>  aggregate(data = teens, age ~ cluster, mean)
   cluster      age
1        1 17.10684
2        2 17.38079
3        3 17.21900
4        4 17.34051
5        5 17.10586
6        6 17.07997
7        7 17.07847
8        8 17.02195
9        9 16.86958
10      10 17.30867
> aggregate(data = teens, female ~ cluster, mean)
   cluster    female
1        1 0.8731707
2        2 0.7198642
3        3 0.7856561
4        4 0.7467018
5        5 0.8968553
6        6 0.4023669
7        7 0.6939163
8        8 0.9561753
9        9 0.8406139
10      10 0.7010816
```

References

Lantz, B. (2019). *Machine Learning with R* (3rd ed.). Packt

Publishing. https://mbsdirect.vitalsource.com/books/9781788291552