BIOS 512 HW 8

Homework 08

Student Name: Jama Brookes

This homework is based on the clustering lectures. Check the lecture notes and TA notes - they should help!

Question 1

This question will walk you through creating your own `kmeans` function.

a) What are the steps of `kmeans`? **Hint**: There are 4 steps/builder functions that you'll need.

1. Assign points to clusters at random:
2. Compute cluster means: Calculate average position of points inside cluster
3. Iterate and reassign labels
4. Recompute means Treat the calculated centers as the official centers

```
label_randomly <- function(n_points, n_clusters){
  sample((1:n_points %% n_clusters) + 1, n_points)
}
```

b) Create the builder function for step 1.

```
get_cluster_means <- function(data, labels){
  data %>%
    mutate(label__ = labels) %>%        # create the label column
    group_by(label__) %>%               # group by the same name
    summarize(across(everything(), mean), .groups = "drop") %>%
    arrange(label__)
}
```

c) Create the builder function for step 2.

d) Create the builder function for step 3. *Hint*: There are two ways to do this part - one is significantly more efficient than the other. You can do either.

```r
assign_cluster <- function(data, means){
  data_matrix <- as.matrix(data)
  means_matrix <- as.matrix(means %>% dplyr::select(-label__))
  dii <- sort(rep(1:nrow(data), nrow(means)))
  mii <- rep(1:nrow(means), nrow(data))
  data_repped <- data_matrix[dii, ]
  means_repped <- means_matrix[mii, ]
  diff_squared <- (data_repped - means_repped)^2
  all_distances <- rowSums(diff_squared)
  tibble(dii=dii, mii=mii, distance=all_distances) %>%
    group_by(dii) %>%
    arrange(distance) %>%
    filter(row_number()==1) %>%
    ungroup() %>%
    arrange(dii) %>%
    pull(mii)
}
```

```r
kmeans_done <- function(old_means, new_means, eps = 1e-6){
  om <- as.matrix(select(old_means, -label__))
  nm <- as.matrix(select(new_means, -label__))
  m <- mean(sqrt(rowSums((om - nm)^2)))
  m < eps
}
```

e) Create the builder function for step 4.

```r
mykmeans <- function(data, n_clusters, eps = 1e-6){
  labels <- label_randomly(nrow(data), n_clusters)
  old_means <- get_cluster_means(data, labels)
  done <- FALSE
  while(!done){
    labels <- assign_cluster(data, old_means)
    new_means <- get_cluster_means(data, labels)
    if(kmeans_done(old_means, new_means)){
      done <- TRUE
    } else {
      old_means <- new_means
    }
  }
  list(labels = labels, means = new_means)
}
```

f) Combine them all into your own `kmeans` function.

Question 2

This is when we'll test your `kmeans` function. #### a) Read in the `voltages_df.csv` data set.

```r
#load in data
voltage_df <- read.csv("./voltages_df.csv")
```

```r
results_mykmeans <- mykmeans(voltage_df, n_clusters = 3)

print(results_mykmeans$labels)
```

b) Call your `kmeans` function with 3 clusters. Print the results with `results$labels` and `results$means`.

```
##   [1] 3 1 1 1 1 1 2 2 1 3 3 3 2 3 1 3 3 3 3 3 1 1 1 1 1 3 3 3 2 2 2 2 1 2 2 1 1
##  [38] 3 2 1 3 3 3 2 1 2 2 1 2 2 1 2 3 3 2 3 1 3 2 1 3 3 1 3 2 2 1 2 3 2 3 1 2 2
##  [75] 1 1 3 2 2 2 2 2 1 2 2 1 1 1 3 2 2 2 1 2 1 1 2 1 1 2 2 2 1 2 3 3 2 3 3 1 3
## [112] 3 1 2 3 2 3 3 1 1 1 1 3 3 2 1 1 2 3 2 1 3 1 2 3 1 1 1 3 2 3 1 1 2 3 3 1 3
## [149] 3 2 3 2 1 1 2 3 2 2 1 3 2 2 3 1 3 2 1 3 1 3 3 2 1 3 2 2 1 2 3 1 2 1 2 3 2
## [186] 1 1 3 1 3 3 1 3 3 2 2 3 1 3 3 2 3 2 2 1 3 1 2 1 3 1 2 3 2 2 1 2 2 1 1 1 1
## [223] 1 1 1 2 2 2 2 3 1 3 2 1 1 2 2 3 3 2 2 3 2 2 1 2 3 2 2 1 2 1 3 2 1 2 2 3 3
## [260] 3 1 1 2 3 1 1 1 2 2 2 3 1 1 1 2 3 2 2 1 1 2 3 2 3 3 1 2 1 1 2 2 3 2 2 3 1
## [297] 3 2 2 1 1 2 3 3 3 2 3 1 1 1 3 3 3 3 3 3 1 3 1 3 3 2 1 2 3 3 3 3 1 3 1 3 2
## [334] 3 1 1 2 2 3 1 2 1 2 2 3 2 1 2 3 3 2 1 1 1 3 2 1 3 2 2 1 1 1 1 3 2 3 1 3 3
## [371] 3 3 3 1 1 2 2 3 3 3 3 2 2 3 3 2 2 1 1 2 3 1 1 1 1 2 1 1 2 3 3 3 3 1 1 3 3
## [408] 2 2 2 1 2 2 3 1 1 1 2 1 1 1 3 3 1 3 2 1 1 1 1 2 2 1 2 3 1 3 3 3 2 2 2 2 3
## [445] 1 3 3 3 2 1 3 3 3 1 1 3 1 1 3 2 1 1 3 2 1 1 1 3 2 2 3 2 3 1 1 2 3 2 3 1 1
## [482] 2 3 2 2 2 1 2 2 2 2 2 1 3 3 2 2 3 1 1 1 3 3 3 1 1 2 2 3 2 3 1 1 2 2 3 2 2
## [519] 1 3 3 3 1 2 3 1 2 2 3 1 2 2 2 2 3 3 1 3 3 1 1 1 1 2 3 3 2 3 3 3 1 1 3 1 2
## [556] 2 1 2 2 2 1 2 2 1 2 2 1 2 2 3 2 1 2 1 2 1 1 1 3 2 2 2 2 3 1 2 2 2 1 2 2 1
## [593] 1 1 2 3 1 1 3 1 2 1 1 2 3 3 2 2 2 1 2 3 1 3 2 1 3 1 1 3 2 1 3 2 2 3 1 3 2
## [630] 1 3 2 3 1 1 3 1 3 3 3 3 3 1 1 3 2 3 2 2 3 2 1 2 3 2 3 3 3 1 3 3 1 1 3 3 1
## [667] 3 1 2 2 1 3 1 3 3 2 1 1 3 2 3 1 2 2 3 3 1 3 1 1 2 2 2 2 1 1 1 3 2 1 3 2 2
## [704] 3 3 2 1 2 2 3 1 3 2 1 3 1 1 3 1 1 3 3 1 1 3 1 2 3 3 3 3 3 1 1 3 3 1 1 3 1
## [741] 3 1 2 3 3 3 1 3 2 1 1 3 3 2 3 3 2 3 2 2 3 3 1 2 3 1 2 1 1 3 2 1 2 3 1 3 3
## [778] 2 3 2 1 3 3 1 2 1 3 1 3 2 3 3 1 1 1 3 1 3 2 1 1 1 1 3 3 2 1 3 3 3 3 2 2 3
## [815] 2 3 1 3 1 2 1 1 1 1 1 2 2 1 2 2 2 2 2 2 1 3 3 1 2 2 3 1 2 2 2 2 3 3 1 2 1
## [852] 3 2 2 2 2 3 3 2 2 2 2 2 3 1 2 1 3 1 1 2 3 2 2 3 3 1 3 3 3 2 1 2 1 1 3 3 2
## [889] 2 1 3 2 2 1 1 1 1 3 2 2
```

```r
print(results_mykmeans$means)
```

```
## # A tibble: 3 x 251
##   label__    X0 X1.00401606425703 X2.00803212851406 X3.01204819277108
##     <int> <dbl>             <dbl>             <dbl>             <dbl>
## 1       1 -1.03             1.24              1.09              0.900
## 2       2 -1.03             1.31              1.16              0.979
## 3       3 -1.03             0.938             0.762             0.363
## # i 246 more variables: X4.01606425702811 <dbl>, X5.02008032128514 <dbl>,
## #   X6.02409638554217 <dbl>, X7.0281124497992 <dbl>, X8.03212851405623 <dbl>,
## #   X9.03614457831325 <dbl>, X10.0401606425703 <dbl>, X11.0441767068273 <dbl>,
## #   X12.0481927710843 <dbl>, X13.0522088353414 <dbl>, X14.0562248995984 <dbl>,
## #   X15.0602409638554 <dbl>, X16.0642570281125 <dbl>, X17.0682730923695 <dbl>,
## #   X18.0722891566265 <dbl>, X19.0763052208835 <dbl>, X20.0803212851406 <dbl>,
## #   X21.0843373493976 <dbl>, X22.0883534136546 <dbl>, ...
```

c) Call R's `kmeans` function with 3 clusters. Print the results with `results$centers` and `results$clusterr`. *Hint*: Use the `as.matrix()` function to make the `voltages_df` data frame a matrix before calling `kmeans()`.

```
results_Rkmeans <- kmeans(as.matrix(voltage_df), centers = 3)

print(results_Rkmeans$cluster)
```

```
##   [1] 2 1 1 1 1 3 2 2 1 2 2 2 2 2 1 2 2 2 2 2 3 1 1 1 1 2 2 2 2 2 2 2 1 2 2 1 1
##  [38] 2 2 3 2 2 2 2 3 2 2 1 2 2 3 2 2 2 2 2 3 2 2 1 2 2 1 2 2 2 3 2 2 2 2 1 2 2
##  [75] 1 3 2 2 2 2 2 2 1 2 2 1 1 1 2 2 2 2 1 2 1 3 2 1 1 2 2 2 1 2 2 2 2 2 2 3 2
## [112] 2 3 2 2 2 2 2 2 1 1 3 1 2 2 2 3 1 2 2 2 2 1 2 1 2 2 3 1 3 2 2 2 1 1 2 2 2 1 2
## [149] 2 2 2 2 1 1 2 2 2 2 3 2 2 2 2 2 1 2 2 1 2 1 2 2 2 1 2 2 2 1 2 2 3 2 1 2 2 2
## [186] 3 3 2 3 2 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 1 2 1 2 3 2 3 2 2 2 2 2 1 2 2 1 1 1 3
## [223] 1 1 3 2 2 2 2 2 1 2 2 1 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 3 2 1 2 2 3 2 2 2 2
## [260] 2 1 1 2 2 1 1 1 2 2 2 2 3 3 3 2 2 2 2 2 3 1 2 2 2 2 2 2 3 2 1 1 2 2 2 2 2 2 3
## [297] 2 2 2 1 1 2 2 2 2 2 2 2 3 1 3 2 2 2 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 2 1 2 3 2 2
## [334] 2 3 1 2 2 2 1 2 1 2 2 2 2 1 2 2 2 2 1 3 1 2 2 1 2 2 2 3 1 1 1 2 2 2 3 2 2
## [371] 2 2 2 1 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 1 2 2 1 1 3 1 2 3 1 2 2 2 2 2 1 1 2 2
## [408] 2 2 2 1 2 2 2 1 3 3 2 1 1 1 2 2 3 2 2 1 3 1 3 2 2 3 2 2 3 2 2 2 2 2 2 2 2 2
## [445] 1 2 2 2 2 1 2 2 2 3 1 2 1 1 2 2 1 1 2 2 1 1 1 2 2 2 2 2 2 1 1 2 2 2 2 1 1
## [482] 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 1 1 2 2 2 1 1 2 2 2 2 2 1 3 2 2 2 2 2
## [519] 1 2 2 2 2 3 2 2 3 2 2 2 1 2 2 2 2 2 2 2 1 2 2 1 1 1 3 2 2 2 2 2 2 2 2 3 1 2 1 2
## [556] 2 1 2 2 2 1 2 2 1 2 2 1 2 2 2 2 3 2 1 2 3 3 1 2 2 2 2 2 2 2 1 2 2 2 1 2 2 1
## [593] 1 3 2 2 1 1 2 1 2 1 3 2 2 2 2 2 2 3 2 2 3 2 2 1 2 1 1 2 2 1 2 2 2 2 1 2 2
## [630] 3 2 2 2 2 3 1 2 1 2 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 1 2 2 1 1 2 2 1
## [667] 2 3 2 2 1 2 1 2 2 2 3 1 2 2 2 1 2 2 2 2 2 1 2 3 1 2 2 2 2 2 1 3 3 2 2 1 2 2 2
## [704] 2 2 2 1 2 2 2 3 2 2 1 2 3 1 2 1 1 2 2 1 3 2 3 2 2 2 2 2 2 2 1 1 2 2 1 1 2 3
## [741] 2 3 2 2 2 2 2 1 2 2 1 3 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 3 2 3 1 2 2 1 2 2 3 2 2
## [778] 2 2 2 1 2 2 1 2 3 2 3 2 2 2 2 2 1 1 1 2 1 2 2 2 1 1 1 1 2 2 2 3 2 2 2 2 2 2 2
## [815] 2 2 3 2 1 2 3 1 1 1 1 2 2 1 2 2 2 2 2 2 2 1 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 1 2 1
## [852] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 1 2 3 1 2 2 2 2 2 2 2 1 2 2 2 2 2 3 2 1 1 2 2 2
## [889] 2 1 2 2 2 1 1 1 1 2 2 2
```

```
as_tibble(results_Rkmeans$centers)
```

```
## # A tibble: 3 x 250
##      X0 X1.00401606425703 X2.00803212851406 X3.01204819277108 X4.01606425702811
##   <dbl>             <dbl>             <dbl>             <dbl>             <dbl>
## 1 -1.03              1.24              1.09             0.898             0.279
## 2 -1.03              1.12             0.962             0.671            -0.235
## 3 -1.03              1.25              1.10             0.905             0.351
## # i 245 more variables: X5.02008032128514 <dbl>, X6.02409638554217 <dbl>,
## #   X7.0281124497992 <dbl>, X8.03212851405623 <dbl>, X9.03614457831325 <dbl>,
## #   X10.0401606425703 <dbl>, X11.0441767068273 <dbl>, X12.0481927710843 <dbl>,
## #   X13.0522088353414 <dbl>, X14.0562248995984 <dbl>, X15.0602409638554 <dbl>,
## #   X16.0642570281125 <dbl>, X17.0682730923695 <dbl>, X18.0722891566265 <dbl>,
## #   X19.0763052208835 <dbl>, X20.0803212851406 <dbl>, X21.0843373493976 <dbl>,
## #   X22.0883534136546 <dbl>, X23.0923694779116 <dbl>, ...
```

d) Are your labels/clusters the same? If not, why? Are your means the same? The labels are not the same, but this is expected because the labels are randomly generated in the beginning. Thus,

the labels are arbitrary. The means that were calculated are the same across the created kmeans and R's built-in kmeans, but may be under different labels.

Question 3

a) Explain the process of using a for loop to assign clusters for kmeans. It loops over all the points one at a time. It calculates the distance from all the cluster means and determines which is closest. It is essentially a double loop.

b) Explain the process of vectorizing the code to assign clusters for kmeans. It creates two matrices of data the same size, and it is able to compare the data points to the cluster means at once by subtracting them.

c) State which (for loops or vectorizing) is more efficient and why. Vectorizing! It repeats entries to create an index vector that then can be easily compared in on vectorized operation.

Question 4

When does `kmeans` fail? What assumption does `kmeans` use that causes it to fail in this situation? Kmeans fails when we do not have spherical Gaussian data with similar standard deviations. For example, if we have two layered rings of groups, kmeans would not be able to classify the "inner" and "outer" rings as groups. It would also fail if we have elongated groups that were not circular in nature. This is because this data is not spherical Gaussian data and they would not be roughly the same size, which is a kmeans assumption.

Question 5

What assumption do Gaussian mixture models make? Gaussian mixture models do not require that each cluster to be the same size because it assumes that the N Gaussian distributions have individual parameters calculated from the data. This allows Gaussian mixture models to identify clusters that are elongated or unequal sizes.

Question 6

What assumption does spectral clustering make? Why does this help us? Special clustering makes very minimal assumptions on the data. It assumes that two points close to one another are likely to be in the same cluster. Because it does not rely on a vector space, it groups two things together by any critera. This helps us because it can identify points in similar clusters based on a criteria (like distance). It then creates a Graph Laplacian, which maps the connections. This then can be runned through kmeans to create clusters.

Question 7

Define the gap statistic method. What do we use it for? The gap statistic method compares the clustering results for different numbers of clusters (k) to randomized data in the same data range. The tightness or density of the observed data is compared to the tightness and density of the randomzied data with the same number of k clusters. This is used to identify the best number of clusters to use to prevent overfitting or underfitting the data.