Kmeans

Chase Enzweiler

11/27/2017

K-means Clustering

Implement a function my_kmeans() that performs the k-means algorithm.

```
#input: X: an nxp matrix, k: the desired number of clusters
# output: list(cluster_sizes: length k vector of cluster sizes, cluster means: kxp matrix with centroid


my_kmeans <- function(X, k){

  # 1. select k random rows from the data and make them initial centroids

  select <- sample.int(dim(X)[1], size = k, replace = FALSE)

  # matrix of initial centroids

  old_centroid <- matrix(0, ncol = dim(X)[2], nrow = k)

  new_centroid <- X[select,]

  while(all(old_centroid != new_centroid)){

  # 2. calculate the distances between observations and centroids

  # vectors of calculated distances stored in a list
  dist_vectors <- list()

  for (i in 1:k){

    # distances for each observation to centroid stored in vector
    dist_store <- c(rep(0,dim(X)[1]))

    for (j in 1:dim(X)[1]){

      dist_store[j] <- sqrt(sum((X[j,] - new_centroid[i,])^2))

    }

    dist_vectors[[i]] <- dist_store

  }
  # now we can create an nxk matrix that has all the distances for each point in the rows with respect

  dist_matrix <- do.call(cbind, dist_vectors)

  # create a list storing vectors of indices assigned to each cluster
  assign_vect <- c(rep(0,dim(dist_matrix)[1]))
```

```
  for (i in 1:dim(dist_matrix)[1]){

    # vector of assigned cluster for each observation
    assign_vect[i] <- which.min(dist_matrix[i,])


  }

  # assign the old centroid to new and create new centroid from clusters

  old_centroid <- new_centroid

  cluster_sizes <- c(rep(0,k))

  for (i in 1:k){

     cluster <- X[which(assign_vect == i),]

     new_centroid[i,] <- colMeans(cluster)

     cluster_sizes[i] <- dim(cluster)[1]

  }

  }

  cluster_means <- new_centroid

  clustering_vector <- assign_vect

  # calculate the wss

  wss <- c(rep(0, k))

  for (i in 1:k){

    wss[i] <-sum((t(t(X[which(assign_vect == i),]) - cluster_means[i,]))^2)

  }

  tss <- sum((t(t(X) - colMeans(X)))^2)

  bss_over_tss <- (tss - sum(wss))/tss

return(list(cluster_sizes = cluster_sizes,cluster_means = cluster_means,clustering_vector = clustering_

}
```

run my_kmeans with iris data and k = 3 and compare it to the function kmeans

```
# my kmeans function
set.seed(90)
my_kmeans(iris[,1:4], 3)
```

```
## $cluster_sizes
## [1] 50 37 63
##
## $cluster_means
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 80      5.006000    3.428000     1.462000    0.246000
## 132     6.870270    3.086486     5.745946    2.089189
## 66      5.904762    2.746032     4.412698    1.433333
##
## $clustering_vector
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 2 2 2
## [106] 2 3 2 2 2 2 2 2 3 3 2 2 2 2 3 2 3 2 3 2 2 3 3 2 2 2 2 2 3 3 2 2 2 3 2
## [141] 2 2 3 2 2 2 3 2 2 3
##
## $wss_cluster
## [1] 0.019980 1.063835 1.489287
##
## $bss_over_tss
## [1] 0.9962236
```

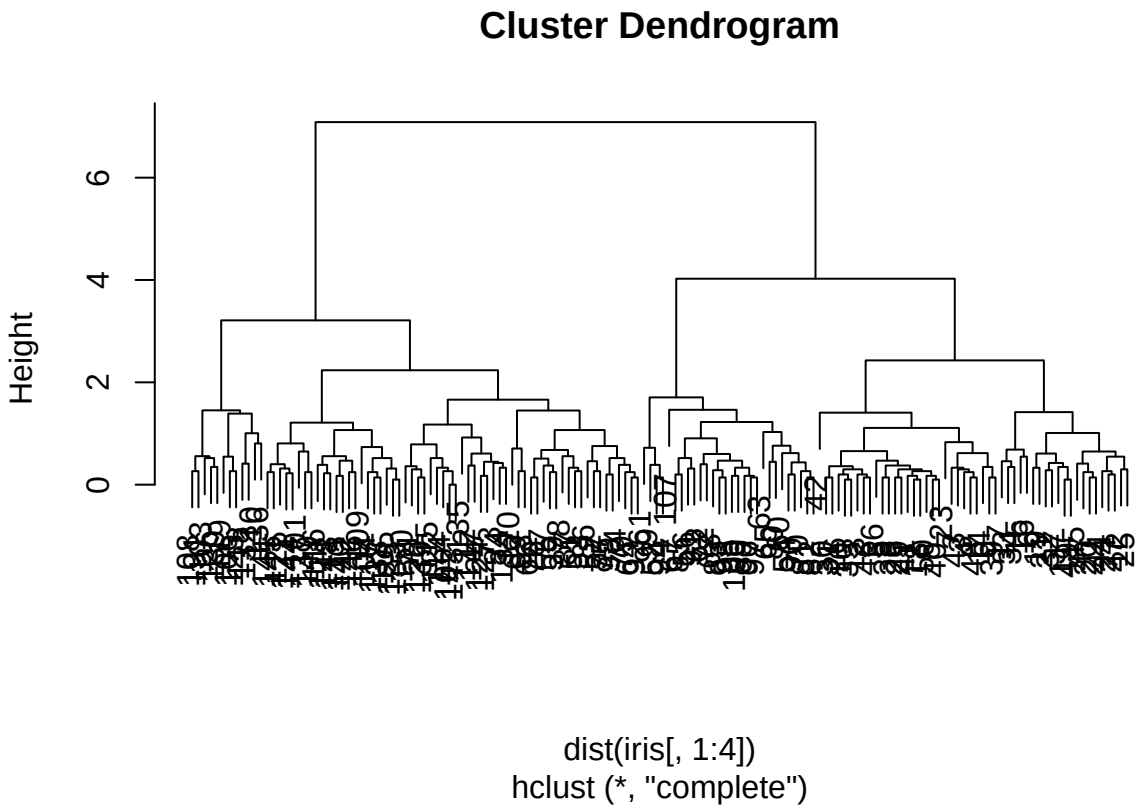
R's k means function

```
# R's kmean function
set.seed(1)
kmeans(iris[,1:4], 3)
```

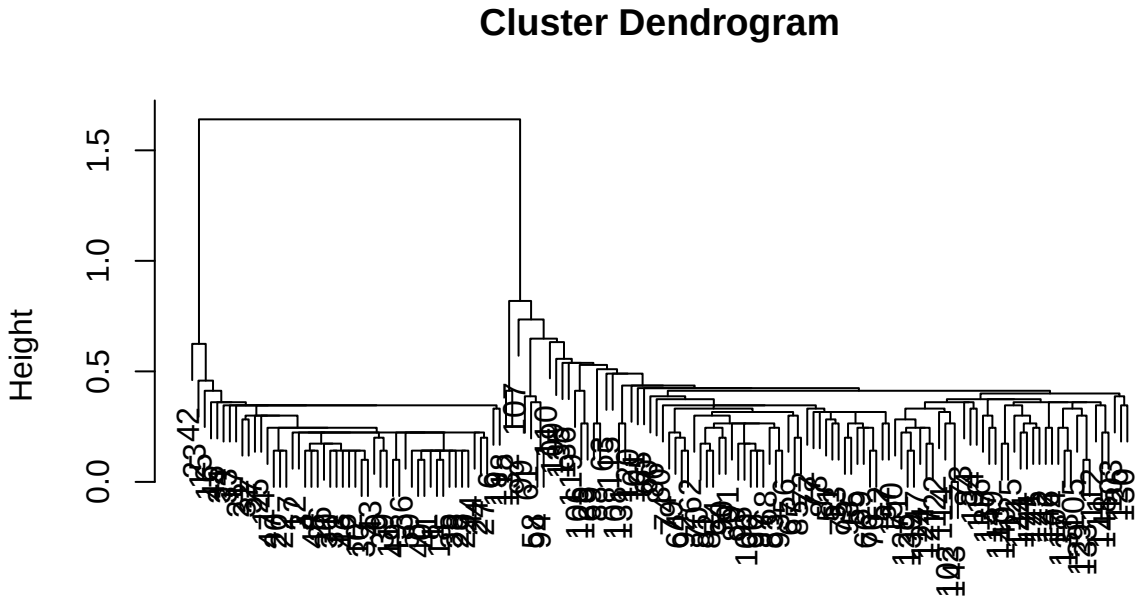
```
## K-means clustering with 3 clusters of sizes 50, 38, 62
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     5.006000    3.428000     1.462000    0.246000
## 2     6.850000    3.073684     5.742105    2.071053
## 3     5.901613    2.748387     4.393548    1.433871
##
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 2 2 2
## [106] 2 3 2 2 2 2 2 2 3 3 2 2 2 2 3 2 3 2 3 2 2 3 3 2 2 2 2 2 3 2 2 2 2 3 2
## [141] 2 2 3 2 2 2 3 2 2 3
##
## Within cluster sum of squares by cluster:
## [1] 15.15100 23.87947 39.82097
##  (between_SS / total_SS =  88.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

Hierarchical Clustering

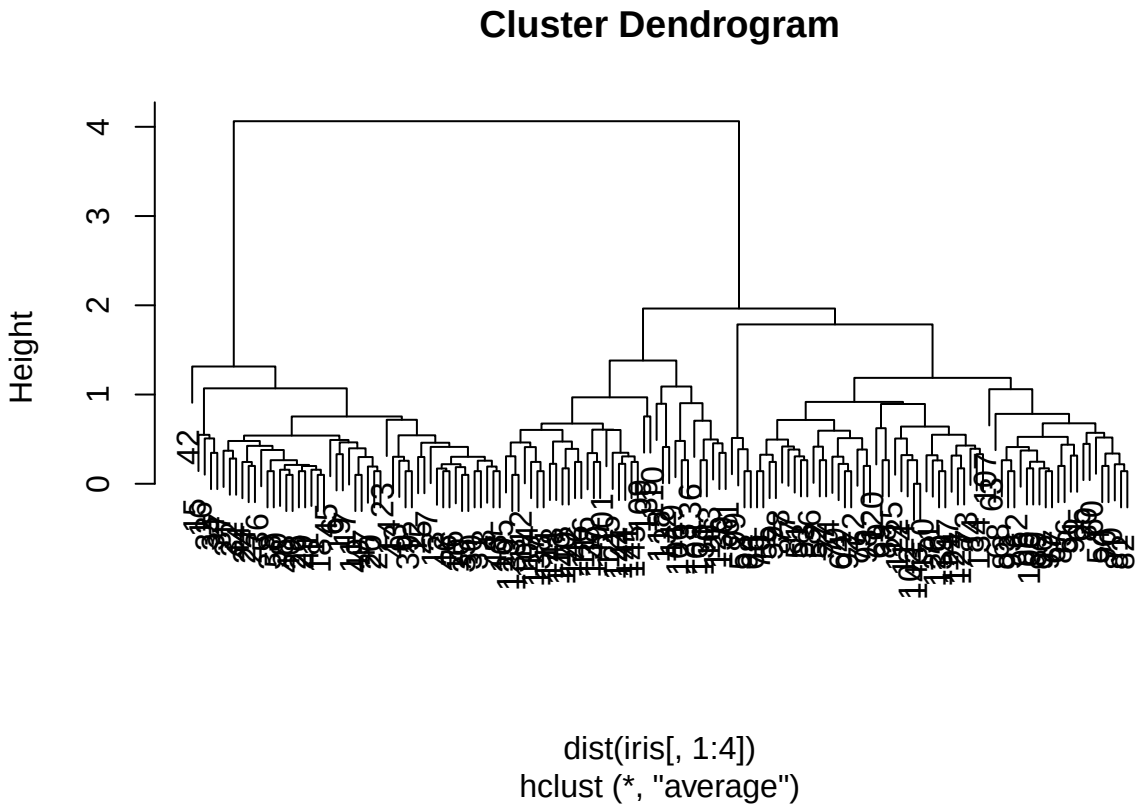
```
# complete
hc.complete <- hclust(dist(iris[,1:4]), method = "complete")

# single
hc.single <- hclust(dist(iris[,1:4]), method = "single")

# average
hc.average <- hclust(dist(iris[,1:4]), method = "average")

plot(hc.complete)
```




```
plot(hc.single)
```




```
plot(hc.average)
```



```r
# use cutree with k = 3
cut_complete <- cutree(hc.complete, k = 3)

cut_single <- cutree(hc.single, k = 3)

cut_average <- cutree(hc.average, k = 3)

cut_complete
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 2 3 2 3 2 3 3 3 3 2 3 2 3 3 2 3
##  [71] 2 3 2 2 2 2 2 2 2 3 3 3 3 2 3 2 2 2 3 3 3 2 3 3 3 3 3 2 3 3 2 2 2 2 2
## [106] 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [141] 2 2 2 2 2 2 2 2 2 2
```

```r
cut_single
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [106] 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2
## [141] 2 2 2 2 2 2 2 2 2 2
```

```r
cut_average
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3
## [106] 3 2 3 3 3 3 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3
## [141] 3 3 2 3 3 3 2 3 3 2
```

Based on the results from cutree it seems that complete and average perform well for 3 clusters. Single does not perform as well because it has singletons for the third cluster where the third cluster only has 2 observations