Stat 432 Homework 10

Assigned: Oct 28, 2024; Due: 11:59 PM CT, Nov 7, 2024

- Instruction
- Question 1: K-means Clustering [65 pts]
- Question 2: Hierarchical Clustering
- Question 3: Spectral Clustering [15 pts]

Instruction

Please remove this section when submitting your homework.

Students are encouraged to work together on homework and/or utilize advanced AI tools. However, **sharing, copying, or providing any part of a homework solution or code to others** is an infraction of the University's rules on Academic Integrity (https://studentcode.illinois.edu/article1/part4/1-401/). Any violation will be punished as severely as possible. Final submissions must be uploaded to Gradescope (https://www.gradescope.com/courses/570816). No email or hard copy will be accepted. For **late submission policy and grading rubrics** (https://teazrq.github.io/stat432/syllabus.html), please refer to the course website.

- You are required to submit the rendered file `HWx_yourNetID.pdf`. For example, `HW01_rqzhu.pdf`. Please note that this must be a `.pdf` file. `.html` format **cannot** be accepted. Make all of your `R` code chunks visible for grading.
- Include your Name and NetID in the report.
- If you use this file or the example homework `.Rmd` file as a template, be sure to **remove this instruction** section.
- Make sure that you **set seed** properly so that the results can be replicated if needed.
- For some questions, there will be restrictions on what packages/functions you can use. Please read the requirements carefully. As long as the question does not specify s uch restrictions, you can use anything.
- **When using AI tools**, you are encouraged to document your comment on your experience with AI tools especially when it's difficult for them to grasp the idea of the question.
- **On random seed and reproducibility**: Make sure the version of your `R` is ≥ $4.0.0$. This will ensure your random seed generation is the same as everyone else. Please note that updating the `R` version may require you to reinstall all of your packages.

Question 1: K-means Clustering [65 pts]

In this question, we will code our own k-means clustering algorithm. The **key requirement** is that you **cannot write your code directly**. You **must write a proper prompt** to describe your intention for each of the function so that GPT (or whatever AI tools you are using) can understand your way of thinking clearly, and provide you with the correct code. We will use the handwritten digits dataset from HW9 (2600 observations). Recall that the k-means algorithm iterates between two steps:

- Assign each observation to the cluster with the closest centroid.
- Update the centroids to be the mean of the observations assigned to each cluster.

You do not need to split the data into train and test. We will use the whole dataset as the training data. Restrict the data to just the digits 2, 4 and 8. And then perform marginal variance screening to **reduce to the top 50** features. After this, complete the following tasks. Please read all sub-questions a, b, and c before you start, and think about how different pieces of the code should be structured and what the inputs and outputs should be so that they can be integrated. For each question, you need to document your prompt to GPT (or whatever AI tools you are using) to generate the code. **You cannot wirte your own code or modify the code generated by the AI tool in any of the function definitions.**

a. [20 pts] In this question, we want to ask GPT to write a function called `cluster_mean_update()` that takes in three arguments, the data X, the number of clusters K, and the cluster assignments. And it outputs the updated centroids. Think about how you should describe the task to GPT (your specific requirements of how these arguments and the output should structured) so that it can understand your intention. You need to request the AI tool to provide sufficient comments for each step of the function. After this, test your function with the training data, $K = 3$ and a random cluster assignment.

Solution:

```
load(file = "mnist_first2600.RData")
dim(mnist2600)
```

```
## [1] 2600  785
```

```
train <- mnist2600
train <- train[train$Digit == 2 | train$Digit == 4 | train$Digit == 8, ]

# perform marginal screening
var = apply(train[, -1], 2, var)
varuse = order(var, decreasing = TRUE)[1:30]
train = train[, c(1, varuse + 1)]
```

Here is my prompt to GPT:

I want to write a function to implement the kmeans clustering. I want to break that down into three steps. In this first step, I want to write a function called `cluster_mean_update()` that takes in three arguments,

- the data matrix X, which is a n by p matrix
- the number of clusters K,
- the cluster (call this argument C), which is a vector with its i-th element being the cluster label of the i-th subject.

This function `cluster_mean_update()` should output a K by p matrix of cluster means, with the jth row representing the center of the jth cluster.

Here is the output I obtained from GPT:

```
cluster_mean_update <- function(X, K, C) {
  # Initialize an empty K x p matrix for cluster means
  p <- ncol(X)
  cluster_means <- matrix(0, nrow = K, ncol = p)

  # Loop through each cluster and compute the mean for each
  for (j in 1:K) {
    # Select rows where the cluster assignment equals the current cluster j
    cluster_data <- X[C == j, ]
    # Calculate mean for each column (feature) in the selected cluster data
    if (nrow(cluster_data) > 0) {
      cluster_means[j, ] <- colMeans(cluster_data)
    } else {
      cluster_means[j, ] <- NA  # If no points in the cluster, assign NA
    }
  }

  return(cluster_means)
}
```

To test the function, I will use the training data, $K = 3$ and a random cluster assignment. This part is completed by GitHub Copilot automatically in R, with my slight modification and comments.

```
set.seed(432)

# generate random cluster assignments
cluster_assignments <- sample(1:3, nrow(train), replace = TRUE)

# use our AI generated function
cluster_means <- cluster_mean_update(train[, -1], 3, cluster_assignments)

# check dimension
dim(cluster_means)
```

```
## [1]  3 30
```

b. [20 pts] Next, we want to ask GPT to write a function called `cluster_assignments()` that takes in two arguments, the data X and the centroids. And it outputs the cluster assignments. Think about how you should describe the task to GPT so that this function would be compatible with the previous function to achieve the k-means clustering. You need to request the AI tool to provide sufficient comments for each step of the function. After this, test your function with the training data and the centroids from the previous step.

Solution:

Here is my prompt to GPT:

In this second step, I want to write a function called `cluster_assignments()` that takes in two arguments,

- the data matrix X, which is a n by p matrix
- the centroids, which is a K by p matrix of cluster means.

This function `cluster_assignments()` should output a vector of cluster assignments, with its i-th element being the cluster label of the i-th subject. In the calculation of the cluster assignments, I want to use the Euclidean distance.

Here is the output I obtained from GPT:

```
cluster_assignments <- function(X, centroids) {
  # Get dimensions
  n <- nrow(X)
  K <- nrow(centroids)

  # Initialize a vector to store cluster assignments for each subject
  assignments <- integer(n)

  # Loop through each subject (row in X)
  for (i in 1:n) {
    # Calculate the Euclidean distance between X[i, ] and each centroid
    distances <- apply(centroids, 1, function(centroid) {
      sum((X[i, ] - centroid)^2)
    })
    # Assign the subject to the nearest centroid (cluster with minimum distance)
    assignments[i] <- which.min(distances)
  }

  return(assignments)
}
```

To test the function, I will use the training data and the centroids from the previous step. This part is completed by GitHub Copilot automatically in R, with my slight modification and comments.

```r
# use our AI generated function
clusters <- cluster_assignments(train[, -1], cluster_means)

# check the first 10 cluster assignments
head(clusters, 10)
```

```
##  [1] 3 1 3 1 2 3 2 3 1 2
```

c. [20 pts] Finally, we want to ask GPT to write a function called `kmeans()`. What arguments should you supply? And what outputs should be requested? Again, think about how you should describe the task to GPT. Test your function with the training data, $K = 3$, and the maximum number of iterations set to 20. For this code, you can skip the multiple starting points strategy. However, keep in mind that your solution maybe suboptimal.

Solution:

Here is my prompt to GPT:

In this third step, I want to write a function called `kmeans()` that takes in three arguments,

- the data matrix X, which is a n by p matrix
- the number of clusters K,
- the maximum number of iterations.

The algorithm should iterate between the two steps:

1. Update the centroids to be the mean of the observations assigned to each cluster, using the `cluster_mean_update()` defined earlier.
2. Assign each observation to the cluster with the closest centroid, using the `cluster_assignments()` function.

At the first iteration, I want to randomly assign each observation to a cluster. After each iteration, I want to check if the cluster assignments have changed. If the cluster assignments have not changed, I want to stop the algorithm. If the maximum number of iterations is reached, I also want to stop the algorithm.

Here is the output I obtained from GPT:

```
kmeans <- function(X, K, max_iter = 100) {
  # Step 1: Initial random cluster assignments
  n <- nrow(X)
  assignments <- sample(1:K, n, replace = TRUE)

  # Initialize a variable to keep track of the previous assignments
  prev_assignments <- NULL

  # Initialize iteration counter
  iter <- 0

  # Main loop
  repeat {
    # Step 2: Update centroids based on current cluster assignments
    centroids <- cluster_mean_update(X, K, assignments)

    # Step 3: Assign each observation to the closest centroid
    new_assignments <- cluster_assignments(X, centroids)

    # Check if the assignments have changed
    if (identical(assignments, new_assignments) || iter >= max_iter) {
      break
    }

    # Update assignments and increment iteration counter
    prev_assignments <- assignments
    assignments <- new_assignments
    iter <- iter + 1
  }

  # Return final assignments and centroids
  list(assignments = assignments, centroids = centroids, iterations = iter)
}
```

To test the function, I will use the training data, $K = 3$, and the maximum number of iterations set to 20. This part is completed by GitHub Copilot automatically in R, with my slight modification and comments.

```
set.seed(432)

# use our AI generated function
kmeans_result <- kmeans(train[, -1], 3, 20)

# check the first 10 cluster assignments
head(kmeans_result$assignments, 10)
```

```
##  [1] 3 1 3 2 2 3 2 3 2 2
```

d. [5 pts] After completing the above tasks, check your clustering results with the true labels in the training dataset. Is your code working as expected? What is the accuracy of the clustering? You are not restricted to use the AI tool from now on. Comment on whether you think the code generated by GPT can be improved (in any ways).

Solution:

```
# check the clustering results
table(kmeans_result$assignments, train$Digit)
```

```
##
##       2   4   8
##   1 114   7  29
##   2  93   5 189
##   3  49 277  10
```

```
# calculate the accuracy
(114 + 277 + 189) / nrow(train)
```

```
## [1] 0.7503234
```

Since the label is not ordered, we select the cluster with the highest frequency to be the label for that cluster. The accuracy of the clustering is `0.7503234`. The current code for the cluster membership update is very inefficient as it loops through each subject. A more efficient implementation would be to use matrix operations to substract the centroids from the data matrix and calculate the sum of squares. Moreover, our code does not implement the multiple starting points strategy to avoid local minima. This could be added to improve the performance of the algorithm.

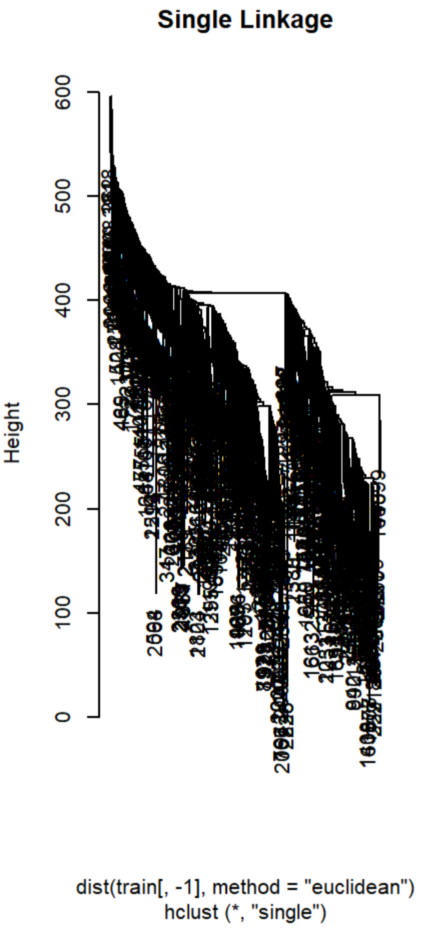
Question 2: Hierarchical Clustering

In this question, we will use the hierarchical clustering algorithm to cluster the training data. We will use the same training data as in Question 1. Directly use the `hclust()` function in R to perform hierarchical clustering, but test different linkage methods (single, complete, and average) and euclidean distance.

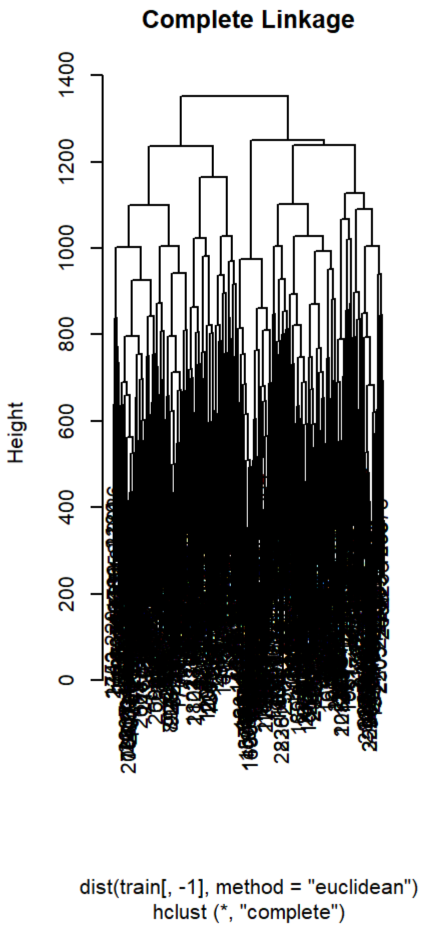
a. [10 pts] Plot the three dendrograms and compare them. What do you observe? Which linkage method do you think is the most appropriate for this dataset?

Solution:

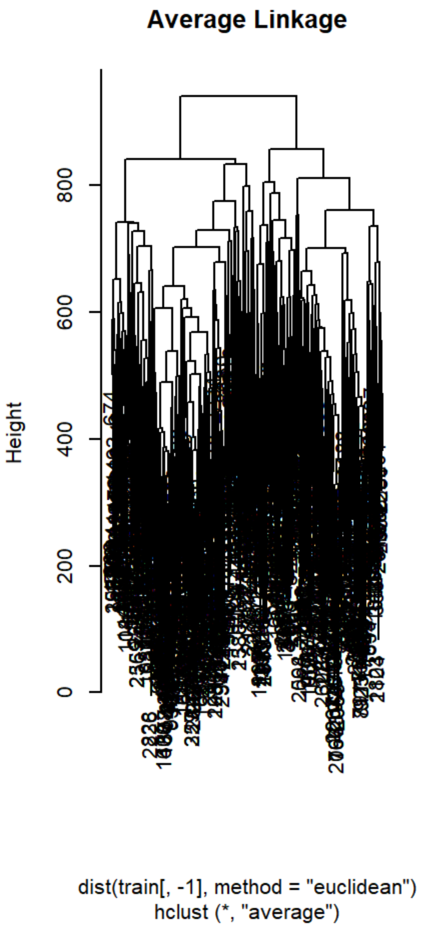
```
# Perform hierarchical clustering
hclust_single <- hclust(dist(train[, -1], method = "euclidean"), method = "single")
hclust_complete <- hclust(dist(train[, -1], method = "euclidean"), method = "complete")
hclust_average <- hclust(dist(train[, -1], method = "euclidean"), method = "average")

# Plot the dendrograms
par(mfrow = c(1, 3))
plot(hclust_single, main = "Single Linkage")
plot(hclust_complete, main = "Complete Linkage")
plot(hclust_average, main = "Average Linkage")
```


Single Linkage

dist(train[, -1], method = "euclidean")
hclust (*, "single")


Complete Linkage

dist(train[, -1], method = "euclidean")
hclust (*, "complete")


Average Linkage

dist(train[, -1], method = "euclidean")
hclust (*, "average")

b. [10 pts] Choose your linkage method, cut the dendrogram to obtain 3 clusters and compare the clustering results with the true labels in the training dataset. What is the accuracy of the clustering? Comment on its performance.

Solution:

```
# I choose to use the average linkage
clusters_average <- cutree(hclust_average, k = 3)

# Check the clustering results
table(clusters_average, train$Digit)
```

```
##
## clusters_average   2   4   8
##                1  80 284  50
##                2  91   5  14
##                3  85   0 164
```

```
# Calculate the accuracy
(284 + 164 + 91) / nrow(train)
```

```
## [1] 0.6972833
```

The accuracy for digit 2 is very poor. It often get mixed with the other two digits. Both 4 and 8 are reasonably well clustered. The overall accuracy is `0.6972833`.

Question 3: Spectral Clustering [15 pts]

For this question, let's use the spectral clustering function `specc()` from the `kernlab` package. Let's also consider all pixels, instead of just the top 50 features. Specify your own choice of the kernel and the number of clusters. Report your results and compare them with the previous clustering methods.

Solution:

```
library(kernlab)
train <- mnist2600
train <- train[train$Digit == 2 | train$Digit == 4 | train$Digit == 8, ]

# Perform spectral clustering
set.seed(432)
specc_result <- specc(as.matrix(train[, -1]), centers = 3, kernel = "rbfdot")

# Check the clustering results
table(specc_result, train$Digit)
```

```
##             
## specc_result   2   4   8
##            1 181   4   6
##            2  16 279   9
##            3  59   6 213
```

```
# Calculate the accuracy
(181 + 279 + 213) / nrow(train)
```

```
## [1] 0.8706339
```

I was able to improve the accuracy slightly with this approach, using radial basis kernel function. The overall accuracy is `0.8706339`.