Homework 3
Introduction to Data Analysis and Mining
Spring 2018
CSCI-B 365

Yining Wang

March 1, 2018

Directions

Please follow the syllabus guidelines in turning in your homework. I am providing the LaTeX of this document too. This homework is due Monday, Feb 26, 2018 10:00p.m. **OBSERVE THE TIME**. Absolutely no homework will be accepted after that time. All the work should be your own. Within a week, AIs can contact students to examine code; students must meet within three days. The session will last no longer than 5 minutes. If the code does not work, the grade for the program may be reduced. Lastly, source code cannot be modified post due date.

All the work herein is solely mine.

k-means Algorithm in Theory

This part is provided to help you implement k-means clustering algorithm.

ALGORITHM `k-means`
INPUT (data Δ, distance $d : \Delta^2 \to \mathbb{R}_{\geq 0}$, centoid number k, threshold τ)
OUTPUT (Set of centoids $\{c_1, c_2, \ldots, c_k\}$)

$\star\star\star$ $Dom(\Delta)$ denotes domain of data.

$\star\star\star$ Assume centroid is structure $c = (v \in DOM(\Delta), B \subseteq \Delta)$
$\star\star\star$ $c.v$ is the centroid value and $c.B$ is the set of nearest points.
$\star\star\star$ c^i means centroid at i^{th} iteration.

$i = 0$
$\star\star\star$ Initialize Centroids
for $j = 1, k$ **do**
    $c_j^i.v \leftarrow random(Dom(\Delta))$
    $c_j^i.B \leftarrow \emptyset$
end for

repeat
    $i \leftarrow i + 1$
    $\star\star\star$ Assign data point to *nearest* centroid
    for $\delta \in \Delta$ **do**
        $c_j^i.B \leftarrow c.B \cup \{\delta\}$, where $\min_{c_j^i}\{d(\delta, c_j^i.v)\}$
    end for

24: **for** $j = 1, k$ **do**
25: $\star\star\star$ Get size of centroid
26: $n \leftarrow |c_j^i.B|$
27: $\star\star\star$ Update centroid with average
28: $c_j^i.v \leftarrow (1/n) \sum_{\delta \in c_j^i.B} \delta$
29: $\star\star\star$ Remove data from centroid
30: $c_j^i.B \leftarrow \emptyset$
31: **end for**
32: $\star\star\star$ Calculate scalar product (abuse notation and structure slightly)
33: $\star\star\star$ See notes
34: **until** $((1/k) \sum_{j=1}^{k} \ ||c_j^{i-1} - c_j^i||) < \tau$
35: **return** $(\{c_1^i, c_2^i, \ldots, c_k^i\})$

k-means on a tiny data set.

Here are the inputs:

$$\Delta = \{(2,5),(1,5),(22,55),(42,12),(15,16)\} \quad (1)$$

$$d((x_1,y_1),(x_2,y_2)) = [(x_1-x_2)^2 + (y_1-y_2)^2)]^{1/2} \quad (2)$$

$$k = 2 \quad (3)$$

$$\tau = 10 \quad (4)$$

Observe that $Dom(\Delta) = \mathbb{R}^2$. We now work through k-means. We ignore the uninformative assignments. We remind the reader that T means transpose.

1: $i \leftarrow 0$
2: $\star\star\star$ Randomly assign value to first centroid.
3: $c_1^0.v \leftarrow random(Dom(\Delta)) = (16, 19)$
4: $\star\star\star$ Randomly assign value to second centroid.
5: $c_2^0.v \leftarrow random(Dom(\Delta)) = (2, 5)$
6: $i \leftarrow i + 1$
7: $\star\star\star$ Associate each datum with nearest centroid
8: $c_1^1.B = \{(22,55),(42,12),(15,16)\}$
9: $c_2^1.B = \{(2,5),(1,5)\}$
10: $\star\star\star$ Update centroids
11: $c_1^1.v \leftarrow (26.3, 27.7) = (1/3)((22,55) + (42,12) + (15,16))$
12: $c_2^1.v \leftarrow (1.5, 5) = (1/2)((2,5) + (1,5))$
13: $\star\star\star$ The convergence condition is split over the next few lines to explicitly show the calculations
14: $(1/k) \sum_{j=1}^{k} \ ||c_j^{i-1} - c_j^i|| = (1/2)(||c_1^0 - c_1^1|| + ||c_2^0 - c_2^1||) = (1/2)(||\binom{2}{5} - \binom{1.5}{5}|| + ||\binom{16}{19} - \binom{26.3}{27.7}||)$
15: $= (1/2)[(\binom{.5}{0}^{\mathsf{T}}\binom{.5}{0})^{(1/2)} + (\binom{-9.7}{-8.7}^{\mathsf{T}}\binom{-9.7}{-8.7})^{(1/2)})] = (1/2)(\sqrt{.5} + \sqrt{169.7}) \sim (1/2)(13.7) = 6.9$
16: Since the threshold is met $(6.9 < 10)$, k-means stops, returning $\{(26.3, 27.7), (1.5, 5)\}$

Problem 1 [10 points]

Answer the following questions for Ionosphere Data Set:

1.1 Briefly describe this data set–what is its purpose? How should it be used? What are the kinds of data it's using?
The purpose of this data set is to use the attributes (17 pulse numbers and each pulse number contains two attributes) to predicate whether the radar return is good, meaning showing evidence of some type of structure in the ionosphere, or bad, meaning not showing.
They can be used as the training set to build model(s) to predict whether the radar return is good or bad based off of necessary attributes.
The data that is used in the data sets are attributes that predict whether radar return shows evidence of some type of structure in the ionosphere or not. There are 34 attributes (coming from 17 pulse number) plus a 35th attribute is either "good" or "bad". The 34 attributes are continuous numbers. The 35th attribute is a binary classification task.

1.2 Using R, show code that answers the following questions:

1.2.1 How many entries are in the data set?

Listing 1: Sample R Script With Highlighting

```
library(data.table)
mydata <- fread("https://archive.ics.uci.edu/ml/machine-learning-databases
                /ionosphere/ionosphere.data")
nrow(mydata)
# There are 351 entries in the data set.
```

1.2.2 How many unknown or missing data are in the data set?

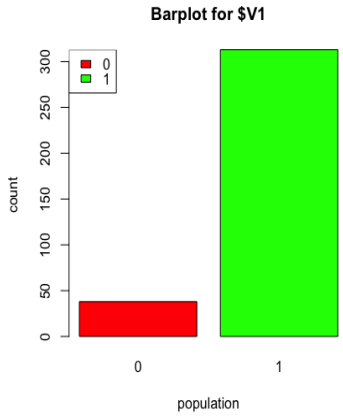
Listing 2: Sample R Script With Highlighting

```
library(data.table)
mydata <- fread("https://archive.ics.uci.edu/ml/machine-learning-databases
                /ionosphere/ionosphere.data")
idx <- mydata == "?"
is.na(mydata) <- idx
table(is.na(mydata))
# There's no missing data. (0 missing data)
```

1.2.3 Create a bar plot of 1st, 2nd, and 35th variables. Label the plots properly. Discuss the distribution of values *e.g.*, are uniform, skewed, normal. Place images of these bar plots into the document. Show the R code that you used below and discussion below that.

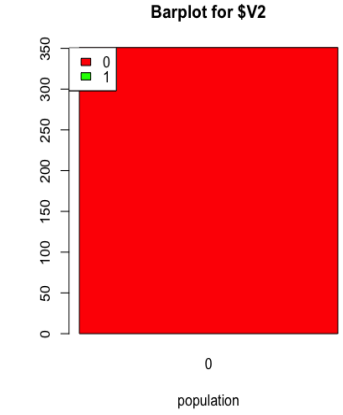
Listing 3: V1

```
install.packages("data.table")
library(data.table)
install.packages("curl")
mydata <- fread("https://archive.ics.uci.edu/ml/machine-learning-databases
                /ionosphere/ionosphere.data")
barplot(table(mydata$V1), main = "Barplot for $V1",xlab = "population",
        col = c("red","green"))
legend("topleft", c("0","1"), fill = c("red","green"))
#The barplot is skewed left. This graph indicate that the population for 0
#is less than 50 and the majority of the population is in 1 which is over 300.
```



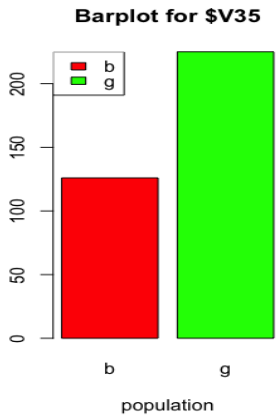
Listing 4: V2

```
install.packages("data.table")
library(data.table)
install.packages("curl")
mydata <- fread("https://archive.ics.uci.edu/ml/machine-learning-databases
                /ionosphere/ionosphere.data")
barplot(table(mydata$V2), main = "Barplot for $V2",
        xlab = "population", col = c("red","green"))
legend("topleft", c("0","1"), fill = c("red","green"))
#The barplot is uniform. This graph indicate that there's no
#data for population 1 and all 350 data are for population 0.
```



Listing 5: V35

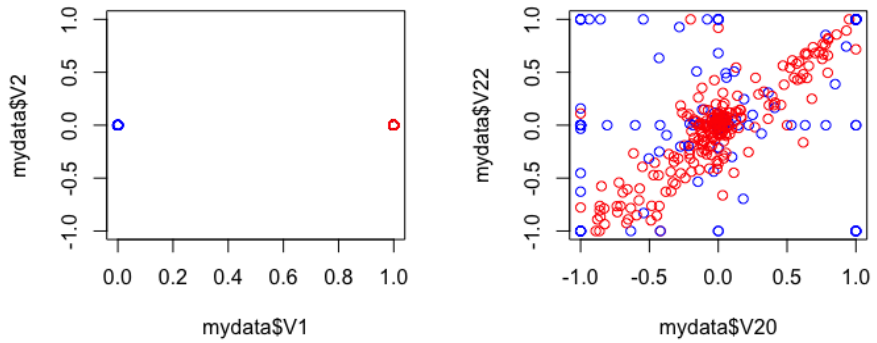
```
install.packages("data.table")
library(data.table)
install.packages("curl")
mydata <- fread("https://archive.ics.uci.edu/ml/machine-learning-databases
                /ionosphere/ionosphere.data")
barplot(table(mydata$V35), main = "Barplot for $V35",
        xlab = "population", col = c("red","green"))
legend("topleft", c("b","g"), fill = c("red","green"))
#The barplot is skewed to the left as well. This graph indicate that there's
#more data (over 200) that fall into the "good" category and less data
#(about 130)falls into the "bad" category.
```



1.2.4 Make a scatter plots of $[V22, V20]$ and $[V1, V2]$ variables and color the data points with the class variable $[V35]$. Discuss the plots, i.e., do you observe any relationships between variables?

Listing 6: Sample R Script With Highlighting

```
install.packages("data.table")
library(data.table)
install.packages("curl")
mydata <- fread("https://archive.ics.uci.edu/ml/machine-learning-databases/
                 ionosphere/ionosphere.data")
plot(mydata$V1,mydata$V2,
     col=ifelse(mydata$V35=='g','red','blue'))
plot(mydata$V20,mydata$V22,
     col=ifelse(mydata$V35=='g','red','blue'))
```



There's only 2 points shown in plot1 and the value of V2 is always 0. Red points indicate good and blue points indicate bad.
Red points indicate good and blue points indicate bad. In plot2, the red points are somewhat linear with some outliers. However, the blue (bad) points are very scattered and there's no obvious relationship.

Problem 2 [10 points]

The pseudo-code for k-means and a running example of k-means on a small data set are provided above. Answer the following questions:

2.1 Does k-means always converge? Given your answer, a bound on the iterate must be included. How is its value determined?
Yes, kmeans always converge. In every iteration, the value of $((1/k)\sum_{j=1}^{k} ||c_j^{i-1} - c_j^i||)$ gets smaller and smaller every time there's a new centroid created, until it the value becomes less than τ. Therefore, eventually, it will converge in a finite number of iteration.

2.2 What is the run-time of this algorithm?
The run-time of this algorithm is $O(tnk)$. We assume that it takes that t times to get the the value of $((1/k)\sum_{j=1}^{k} ||c_j^{i-1} - c_j^i||)$ to become less than τ.

Problem 3 [20 points]

Implement Lloyd's algorithm for k-means (see algorithm k-means above) in R and call this program C_k. As you present your code explain your protocol for

Listing 7: Kmeans

```
#the distance function
distance <- function(cent, dp){
  d = 0
  for(i in 1:length(cent)){
    d = d + (cent[i] - dp[i])^2
  }
  return(sqrt(d))
}

scalar_product <- function(cents_old, cents_new){
  sp = 0
  k = nrow(cents_new)
  for (i in 1: k){
    sp = sp + distance(cents_old[i,], cents_new[i,])
  }
  return(sp / k)
}

kmeans <- function(dataset, d=distance, k, tau){
  dimension = ncol(dataset)
  dataset_min = min(dataset)
  dataset_max = max(dataset)
  #initialization of centroids
  centroids = matrix(runif(k*dimension, min = dataset_min,
                           max = dataset_max),
                     nrow = k, ncol = dimension)
  iteration = 0
  repeat{
    iteration = iteration+1
    print(iteration)
    #First row in every matrix is not used
    B <- lapply(1:k, function(x) matrix(rep(0,dimension), nrow=1,
                                        ncol=dimension))

    #Assign data point to nearest centroid
    for (j in 1: nrow(dataset)){
      dp = dataset[j,]
      min = d(centroids[1,], dp)
      index = 1
```

```
      for (t in 2 : k){
        dist = d(centroids[t,], dp)
        if(dist < min) {
          min = dist
          index = t
        }
      }
      B[[index]] = rbind(B[[index]], dp)
    }
    #Get size of centroid & Update centroid with average
    new_centroids = matrix(rep(0, k*dimension), nrow = k, ncol = dimension)
    for (j in 1:k){
      if (nrow(B[[j]]) == 1) {
        new_centroids[j,] = runif(dimension, min=dataset_min, max=dataset_max)
      } else if (nrow(B[[j]]) == 2){
        new_centroids[j,] = B[[j]][2,]
      } else {
        new_centroids[j,] = colMeans(B[[j]][-1,])
      }
    }
    #if any two centroids collapse, randomize one of them
    for(i in 1:(k-1)){
      for(j in (i+1):k){
        print(d(new_centroids[i,], new_centroids[j,]))
        if(d(new_centroids[i,], new_centroids[j,]) <= 0.00001){
          new_centroids[j,] = runif(dimension, min=dataset_min, max=dataset_max)
        }
      }
    }

    # converge condition
    old_centroids = centroids;
    centroids = new_centroids;

    if(scalar_product(old_centroids, centroids) < tau) {
      break;
    }
  }
  print(centroids)
}

# a test
fakedata=matrix(runif(30, 0, 9999), nrow=2, ncol=3)
kmeans(fakedata, distance, 2, 1)
```

3.1 initializing centroids

To initialize centroids, we need to find k centroids by random selecting the centroids. And creating a B which stores the matrices of data-points that to there nearest centroid. But when initializing it, we set it to empty.

3.2 maintaining k centroids

Calculate the distance between each datapoint and a centroid. Assign the data point to the centroid that has the least distance between them. Update the centroids with the average of the distances. Remove the old data points from the centroids. And repeat the process until all data points are assigned to the nearest centroid with the average distance less than τ.

3.3 deciding ties

If any two centroids collapse, randomize one of the using $runif()$

3.4 stopping criteria

There are 2 sets of iterations. First iteration provides one set of centroids and second iteration provides another set of centroids. Each centroid in every set contains a pair of old and new centroids. Compare and calculate the distance between the old centroid and the new centroids for every centroid. Compare the average of the sum of the distances to τ. Stop if the distance is less than τ.

Problem 4 [40 points]

In this question, you are asked to run your program, C_k, against Ionosphere Data Set. Upon stopping, you will calculate the quality of the centroids and of the partition. For each centroid c_i, form two counts:

$$g_i \leftarrow \sum_{\delta \in c_i.B} [\delta.C = \text{“g”}], \quad \text{good}$$

$$b_i \leftarrow \sum_{\delta \in c_i.B} [\delta.C == \text{“b”}], \quad \text{bad}$$

where $[x = y]$ returns 1 if True, 0 otherwise. For example, $[2 = 3] + [0 = 0] + [34 = 34] = 2$

The centroid c_i is classified as good if $g_i > b_i$ and bad otherwise. We can now calculate a simple error rate. Assume c_i is good. Then the error is:

$$error(c_i) = \frac{b_i}{b_i + g_i}$$

We can find the total error rate easily:

$$Error(\{c_1, c_2, \ldots, c_k\}) = \sum_{i=1}^{k} error(c_i)$$

Report the total error rates for $k = 2, \ldots 5$ for 20 runs each, presenting the results that are easily understandable. Plots are generally a good way to convey complex ideas quickly, i.e., box plot. Discuss your results.

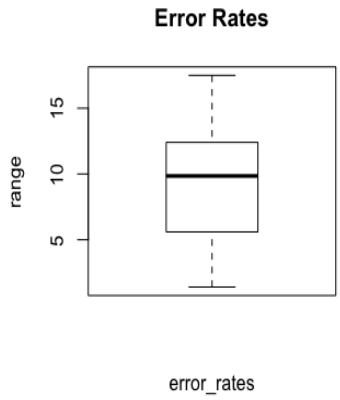
Listing 8: Sample R Script With Highlighting

```
#Problem 4

#the distance function
distance <- function(cent, dp){
  d = 0
  for(i in 1:length(cent)){
    d = d + (cent[i] - dp[i])^2
  }
  return(sqrt(d))
}

scalar_product <- function(cents_old, cents_new){
  sp = 0
  k = nrow(cents_new)
  for (i in 1: k){
    sp = sp + distance(cents_old[i,], cents_new[i,])
  }
  return(sp / k)
}
# the last column of input dataset must be class
kmeans <- function(input_dataset, d=distance, k, tau){
  dataset = input_dataset[, -ncol(input_dataset)]
```

```
classes = input_dataset[, ncol(input_dataset)]
dimension = ncol(dataset)
dataset_min = min(dataset)
dataset_max = max(dataset)
#initialization of centroids
centroids = matrix(runif(k*dimension, min = dataset_min,
                         max = dataset_max),
                   nrow = k, ncol = dimension)
iteration = 0
repeat{
  iteration = iteration+1
  print(iteration)
  #First row in every matrix is not used
  B <- lapply(1:k, function(x) matrix(rep(0,dimension), nrow=1,
                                      ncol=dimension))
  goods_and_bads=matrix(rep(0,2*k), nrow=k, ncol=2)

  #Assign data point to nearest centroid
  for (j in 1: nrow(dataset)){
    dp = dataset[j,]
    min = d(centroids[1,], dp)
    nearest_centroid = 1
    for (t in 2 : k){
      dist = d(centroids[t,], dp)
      if(dist < min) {
        min = dist
        nearest_centroid = t
      }
    }
    B[[nearest_centroid]] = rbind(B[[nearest_centroid]], dp)
    class = classes[j]
    #print(class)
    if(class == "g"){
      goods_and_bads[nearest_centroid,1] =
        goods_and_bads[nearest_centroid,1] + 1

    }else if(class == "b"){
      goods_and_bads[nearest_centroid,2] =
        goods_and_bads[nearest_centroid, 2] +1
    }
  }
  #error rate
  #goods_and_bads; k*2 matrix
  error_rate = 0
  for(i in 1:k){
    total = goods_and_bads[i,1]+goods_and_bads[i,2]
    if (goods_and_bads[i,1] > goods_and_bads[i,2]){
      error_rate = error_rate + (goods_and_bads[i,1]/total)
    }else{
      error_rate = error_rate + (goods_and_bads[i,2]/total)
    }
  }


  #Get size of centroid & Update centroid with average
  new_centroids = matrix(rep(0, k*dimension), nrow = k, ncol = dimension)
  for (j in 1:k){
    if (nrow(B[[j]]) == 1) {
```

```
        new_centroids[j,] = runif(dimension, min=dataset_min, max=dataset_max)
      } else if (nrow(B[[j]]) == 2){
        new_centroids[j,] = B[[j]][2,]
      } else {
        new_centroids[j,] = colMeans(B[[j]][-1,])
      }
    }
    #if any two centroids collapse, randomize one of them
    for(i in 1:(k-1)){
      for(j in (i+1):k){
        #print(d(new_centroids[i,], new_centroids[j,]))
        if(d(new_centroids[i,], new_centroids[j,]) <= 0.00001){
          new_centroids[j,] = runif(dimension, min=dataset_min, max=dataset_max)
        }
      }
    }

    # converge condition
    old_centroids = centroids;
    centroids = new_centroids;

    if(scalar_product(old_centroids, centroids) < tau) {
      break;
    }
  }
  print("centroids:")
  print(centroids)
  print(paste("error" , error_rate))
  return(error_rate)
}



mydata <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases
                    /ionosphere/ionosphere.data", header=FALSE)
error_rates=rep(0,19)
for (k in 2:20){
  print(paste("#",k))
error_rates[k] = kmeans(mydata, distance, k, 0.001)
}
print(error_rates)
boxplot(error_rates, main = "Error Rates",
        xlab = "rates")


#error rates:
#1.40395554102648
#2.47724482392991
#2.82668067226891
#4.1728585602737
#5.00529339281043
#6.18542598347609
#7.07628339140534
#7.73451959917477
#8.97970072239422
#9.84774581558355
#10.5117448812816
#10.6331483619288
```

```
#11.5573929250240
#11.7044553624000
#13.0979561139829
#14.1036852248944
#14.5068002210464
#16.4620827650651
#17.4989232306834
```



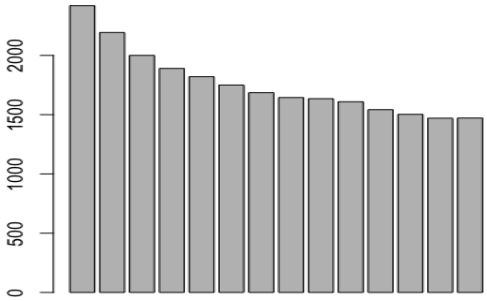
The boxplot indicates that the mean of the 19 error rates are around 10. The smallest number is around 1 and the largest number is around 17. The data is skewed left which means that the error rates is increasing.

Problem 5 [10 points]

In this question, you are asked to make use of the R package for k-means. Elbow method is one of the techniques to decide the optimal cluster number. Find the optimal cluster number for Ionosphere data set using elbow method (for $2 \leq k \leq 15$). Provide a plot that shows the total SSE for each k. Discuss your results.

Listing 9: Sample R Script With Highlighting

```
#library(data.table)
mydata <- fread("https://archive.ics.uci.edu/ml/machine-learning-databases
                /ionosphere/ionosphere.data", drop = "V35")

#SSE
k = c()
for (n in 2:15) {
  k[n-1] = kmeans(mydata, n, iter.max = 36, nstart = n,
                  algorithm = c("Lloyd"))$ sum(withinss)
  }
barplot(k)

#The graph is skewed to the right. Although the graphs change every time you
#run the program (because kmeans is choosing random centroids every time
#in the first place), the result differs just a little. Overall it is still
#skewed to the right.
```



Problem 6 [10 points]

Let $X \subset \mathbb{R}^n$ ($\mathbb{R}$ is the set of reals) for positive integer $n > 0$. Define a distance $d : X \times X \to \mathbb{R}_{\geq 0}$ as

$$d(x, y) = max\{|x_i - y_i|\}, \forall i \ 1 \leq i \leq n$$

Prove/disprove d is a metric?
Yes, it is a metric. For it to be metric, it has to meet three conditions:

1. d(x,y) $\geq$ 0
   With $|x_i - y_i|$, the result will always be a positive value that is greater than or equal to 0 given the nature of absolute value.
2. d(x,y) = d(y,x)
   $d(x, y) = max\{|x_i - y_i|\}$
   $|x_i - y_i| = |y_i - x_i|$
   With the absolute value sign, the difference between x_i and y_i is the same. Therefore, $d(x, y) = d(y, x)$.
3. d(x,z) $\leq$ d(x,y) + d(y,z)
   $|x, z| = |x - y + y - z| \leq |x - y| + |y - z|$

Extra credit [30 points]

This part is optional.

1 The k-means algorithm provided above stops when centroids become stable (Line 34). In theory, k-means converges once SSE is minimized

$$SSE = \sum_{j}^{k} \sum_{x \in c_j.B} ||\mathbf{x} - c_j.v||_2^2$$

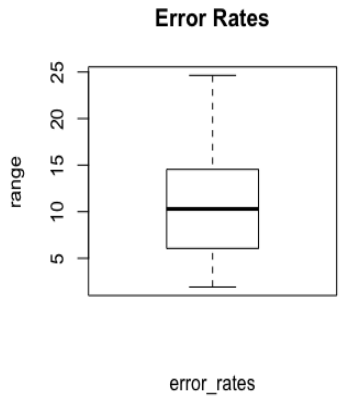
In this question, you are asked to use SSE as stopping criterion. Run k-means over Breast Cancer Wisconsin Data Set and report the total SSE in a plot for $k = 2, \ldots, 5$ for 20 runs each. Discuss your results. [15 points].

Listing 10: Sample R Script With Highlighting

```
#extra credit #1

#the distance function
distance <- function(cent, dp){
  d = 0
  for(i in 1:length(cent)){
    d = d + (cent[i] - dp[i])^2
  }
  return(sqrt(d))
}

#SSE
sse <- function(B, centroids){
  err = 0
  for(j in 1:k){
    if (nrow(B[[j]] >= 2)) {
      for(x in 2:nrow(B[[j]])){
        err = err + distance(B[[j]][x,], centroids[j,])^2
      }
    }
  }
  return(err)
}
# the last column of input dataset must be class
kmeans <- function(input_dataset, d=distance, k, max_iteration){
  dataset = input_dataset[, -ncol(input_dataset)]
  classes = input_dataset[, ncol(input_dataset)]
  rownames(dataset)<-c()
  dimension = ncol(dataset)
  dataset_min = min(dataset)
  dataset_max = max(dataset)
  #initialization of centroids
  centroids = matrix(runif(k*dimension, min = dataset_min,
                           max = dataset_max),
                     nrow = k, ncol = dimension)
  iteration = 0
  SSEs=c()
  ERROR_RATEs=c()
  repeat{
    iteration = iteration+1
    print(iteration)
    #First row in every matrix is not used
    B <- lapply(1:k, function(x) {
      m=matrix(rep(0,dimension), nrow=1, ncol=dimension)
      rownames(m)<-1 })
    goods_and_bads=matrix(rep(0,2*k), nrow=k, ncol=2)

    #Assign data point to nearest centroid
    for (j in 1: nrow(dataset)){
      dp = dataset[j,]
      min = d(centroids[1,], dp)
      nearest_centroid = 1
      for (t in 2 : k){
        dist = d(centroids[t,], dp)
        if(dist < min) {
          min = dist
          nearest_centroid = t
```

```
      }
    }

    B[[nearest_centroid]] = rbind(B[[nearest_centroid]], dp)
    class = classes[j]
    #print(class)
    if(class == "2"){
      goods_and_bads[nearest_centroid,1] =
        goods_and_bads[nearest_centroid,1] + 1

    }else if(class == "4"){
      goods_and_bads[nearest_centroid,2] =
        goods_and_bads[nearest_centroid, 2] +1
    }
  }
  #error rate
  #goods_and_bads; k*2 matrix
  error_rate = 0
  for(i in 1:k){
    total = goods_and_bads[i,1]+goods_and_bads[i,2]
    if (total == 0) {
      next
    }
    if (goods_and_bads[i,1] > goods_and_bads[i,2]){
      error_rate = error_rate + (goods_and_bads[i,1]/total)
    }else{
      error_rate = error_rate + (goods_and_bads[i,2]/total)
    }
  }

  #Get size of centroid & Update centroid with average
  new_centroids = matrix(rep(0, k*dimension), nrow = k, ncol = dimension)
  for (j in 1:k){
    if (nrow(B[[j]]) == 1) {
      new_centroids[j,] = runif(dimension, min=dataset_min, max=dataset_max)
    } else if (nrow(B[[j]]) == 2){
      new_centroids[j,] = B[[j]][2,]
    } else {
      new_centroids[j,] = colMeans(B[[j]][-1,])
    }
  }
  #if any two centroids collapse, randomize one of them
  for(i in 1:(k-1)){
    for(j in (i+1):k){
      #print(d(new_centroids[i,], new_centroids[j,]))
      if(d(new_centroids[i,], new_centroids[j,]) <= 0.00001){
        new_centroids[j,] = runif(dimension, min=dataset_min, max=dataset_max)
      }
    }
  }

  # converge condition
  old_centroids = centroids;
  centroids = new_centroids;

  SSEs = c(SSEs,sse(B, centroids))
  print(SSEs)
  ERROR_RATEs = c(ERROR_RATEs, error_rate)
```

```
    print(ERROR_RATEs)
    if (iteration >= max_iteration){
      break
    }

  }

  # find minimum SSE
  res_err = ERROR_RATEs[which.min(SSEs)]

  print(paste("error" , res_err))
  return(res_err)
}



mydata <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases
                   /breast-cancer-wisconsin/breast-cancer-wisconsin.data",
                   header=FALSE, na.strings="?")
for(i in 1:ncol(mydata)){
  mydata[is.na(mydata[,i]), i] <- mean(mydata[,i], na.rm = TRUE)
}
error_rates=rep(0,19)
for (k in 2:20){
  error_rates[k] = kmeans(mydata[,-1], distance, k, 10)
}
print(error_rates)
boxplot(error_rates, main = "Error Rates",xlab = "rates")



#error rates:
#1.91201716738197,
#2.84301006011532,
#3.80491393387978,
#4.65413462110455,
#5.72880728466935,
#6.39091359432829,
#6.54658768741247,
#7.55780061631261,
#8.33263440860215,
#10.2926470588235,
#11.1006291113664,
#12.0163677877314,
#12.9200509914472,
#13.7124120658264,
#15.374637089204,
#16.7459158238489,
#19.8201864706352,
#23.0603588584345,
#24.639750580769
```



The boxplot indicates that the mean of the 19 error rates are around 10. The smallest number is around 1 and the largest number is around 25. The data is skewed left which means that the error rates is increasing.

2 Traditional k-means initialization is based on choosing values from a uniform distribution. In this question, you are asked to improve k-means through initialization. k-means ++ is an extended k-means clustering algorithm and induces non-uniform distributions over the data that serve as the initial centroids. Read the paper and discuss the idea in a paragraph. Implement this idea to improve your k-means program. [15 points]
The idea of $k-$means++ is to improve the running time of the program when randomly finding the first set of centroids. k-means++ algorithm only randomly select the first centroid instead of randomly selecting the first k centroids. After finding the first centroid, select the second centroid that has the largest distance to the first center. Then use the first and second centers to calculate the third centroid. This algorithm assigns each data point to its nearest center and the make the farthest point as another centroid.

What to Turn-in

Submit a .zip file that includes the files below. Name the .zip file as "usename-section number", i.e., hakurban-B365.

- The *tex and *pdf of the written answers to this document.
- *`R`files for:
  - implementation of k-means [Problem 4]
  - `R` package usage [Problem 5]
  - extra credit questions–optional
- A README file that explains how to run your code and other files in the folder