Math 156: Hw3

Sijia Hua
Assisted by Haoxian Chen

June 5th, 2019

1 Program the Batch K-Means and test synthetic dataset

I use a permutation of k(number of clusters)to find out the true label instead of purely clustering, so that the error and loss is reasonable. The trend of misclassification is shown as in Fig 1. It shows a decreasing trend as s grows at first, then performs some fluctuations. The lowest misclassification rate appears at s = 5.5 and s = 6.5 with around 0.1 misclassficiation.

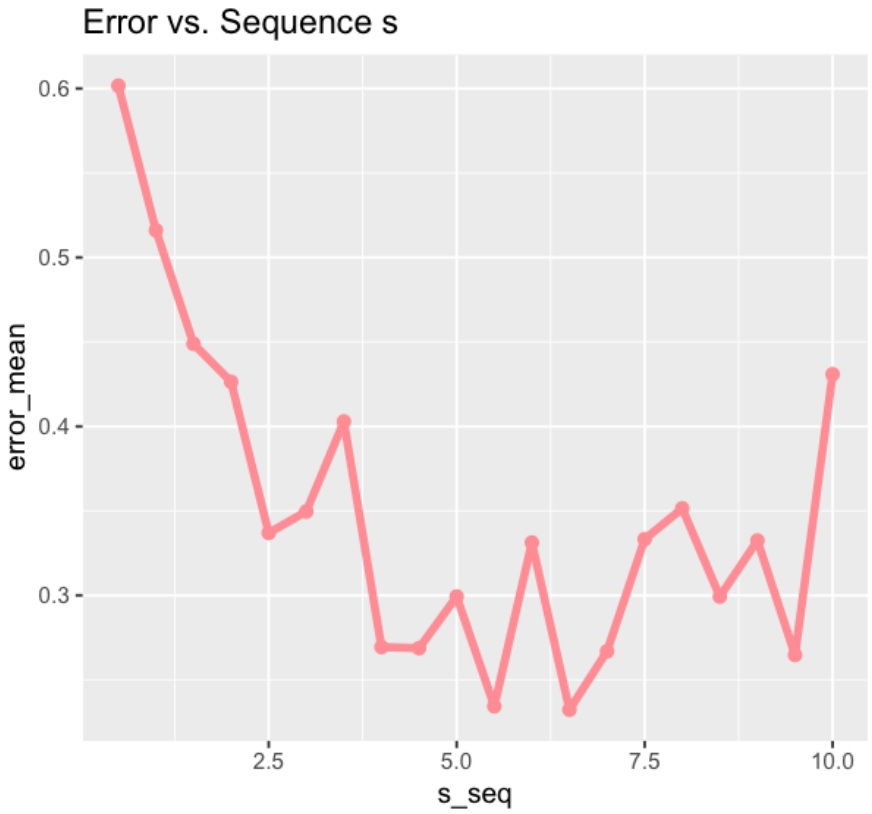

Figure 1: Misclassification Rate

2 Compute Loss Lower Bound

The lower bound of loss function is the average sum square of singular values σ obtained from SVD. The plot of both loss function and its lower bound is in Fig 2, results in Fig 3. The lower bound of the loss is always around 6, while the loss function is first increasing and then fluctuated.

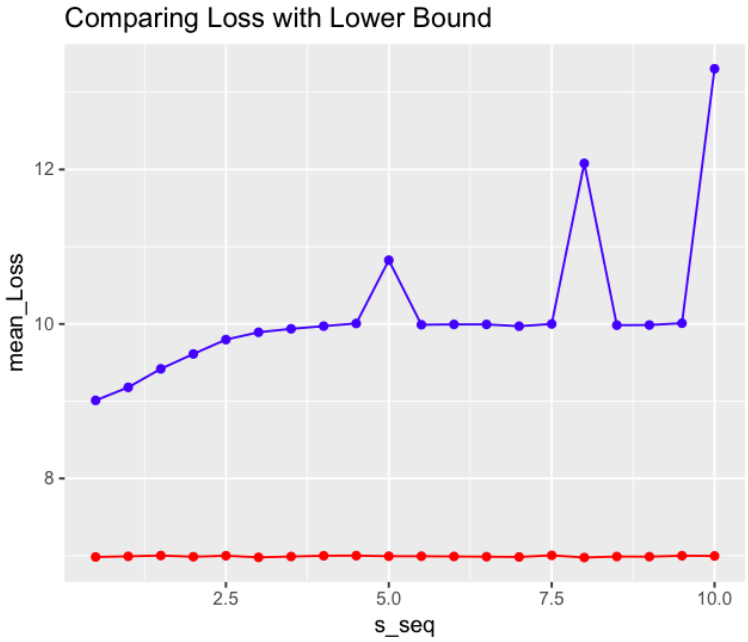

Figure 2: Loss Function

```
> print('s values:')
[1] "s values:"
> print(s_seq)
 [1]  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5 10.0
> print('Mean error in each s value:')
[1] "Mean error in each s value:"
> print(error_mean)
 [1] 0.60164 0.51608 0.44886 0.42634 0.33693 0.34960 0.40289 0.26938 0.26871 0.29928 0.23433 0.33122 0.23229 0.26686
[15] 0.33322 0.35145 0.29922 0.33240 0.26468 0.43081
> parta_result <- data.frame(s_seq, error_mean)
> print('mean loss:')
[1] "mean loss:"
> print(mean_Loss)
 [1]  9.010842  9.179744  9.418993  9.611949  9.797843  9.892644  9.937372  9.972298 10.007109 10.825967  9.990072
[12]  9.994685  9.994671  9.971335 10.000360 12.079454  9.984549  9.986777 10.010316 13.304054
> print('mean loss lower bound:')
[1] "mean loss lower bound:"
> print(mean_Loss_lower)
 [1] 6.984139 6.992215 7.002633 6.987050 7.000088 6.979374 6.989807 6.999963 7.001372 6.993847 6.993898 6.990578
[13] 6.987495 6.984321 7.005167 6.976796 6.989599 6.988208 7.000317 6.997029
```

Figure 3: Error and Loss Result

The derivation of Loss function's lower bound:

3 Use k-means to classify seeds data

Before data processing, loss function gives a relatively larger value than the data after processing. The error rate is also smaller in normalized data. Because each feature in the seed data have different units, the feature with a much smaller magnitude may result in a incredibly huge significance in the k-mean algorithm. Hence, the predictions of raw data would be not reasonable. The sample normalized data is in Fig 5.

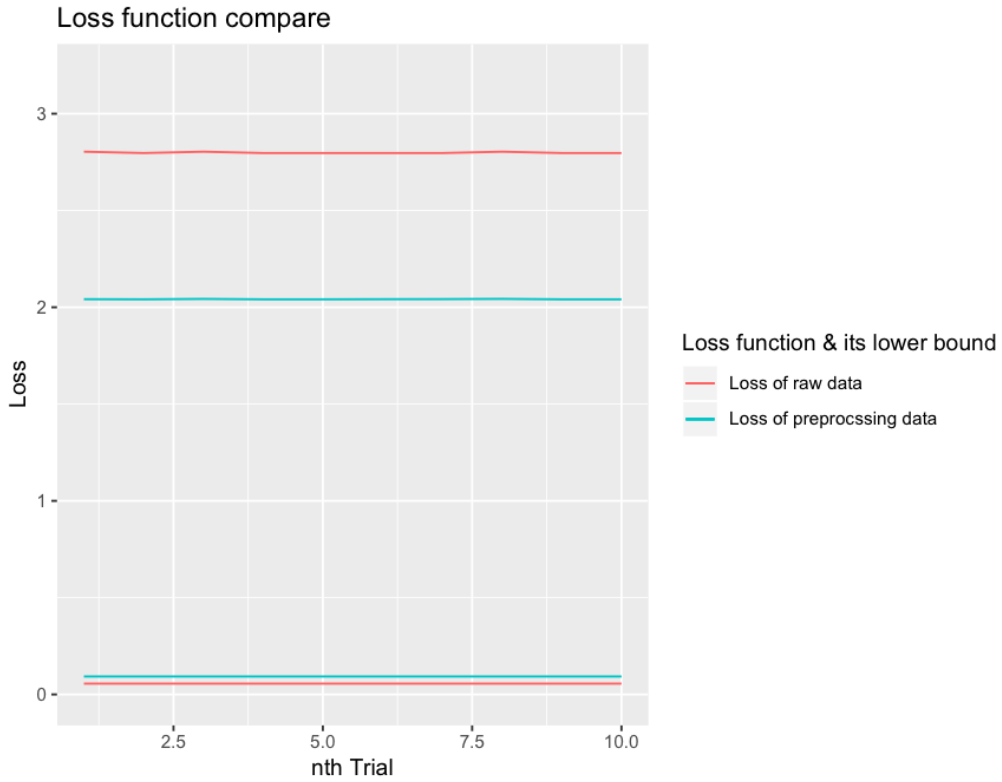

Figure 4: Loss Function Compare between raw data and pre-processing data

	V1	V2	V3	V4	V5	V6	V7
1	0.141759037	0.2149488188	6.045733e-05	0.303493006	0.141364035	-0.983800962	-0.38266305
2	0.011161356	0.0082041534	4.274938e-01	-0.168222697	0.196961591	-1.783903583	-0.91981560
3	-0.191608729	-0.3593419185	1.438945e+00	-0.761817099	0.207551602	-0.665888201	-1.18635720
4	-0.346263878	-0.4742000660	1.036904e+00	-0.687335672	0.318746714	-0.958527563	-1.22705057
5	0.444195774	0.3298069663	1.371233e+00	0.066506648	0.803239702	-1.559768435	-0.47422315
6	-0.160677699	-0.2674554005	1.019976e+00	-0.547400870	0.141364035	-0.823514403	-0.91981560
7	-0.054137485	-0.0530535253	3.767096e-01	-0.147909581	0.001046394	-0.075953850	-0.38469772
8	-0.253470789	-0.3516847087	8.506951e-01	-0.470662431	0.114889009	-0.665223111	-0.83029017
9	0.612598048	0.6896958284	1.566449e-01	0.958026757	0.546431943	-1.104182155	0.95411430
10	0.547299207	0.5288944219	7.195027e-01	0.576591571	0.652332050	-1.151403506	0.25418826
11	0.141759037	0.2226060287	-5.918773e-02	0.192899372	-0.043961151	0.560536763	-0.19140419
12	-0.280965037	-0.3057414497	3.640136e-01	-0.430036198	-0.152508761	-1.319006050	-0.82825550
13	-0.329079973	-0.4129423873	7.195027e-01	-0.427779185	-0.157803766	0.190081934	-1.36337338

Figure 5: Sample normalized data

4 R Source code

```r
## Math 156 Hw3
## Sijia Hua
library(combinat) # to use permutation
library(base) # to use svd
library(dplyr)
library(ggplot2)


# given parameters
n <- 10^4 # number of observations
d <- 10  # number of features
k <- 3 # number of clusters
s_seq <- seq(0.5, 10, by = 0.5) # different s we want
e <- 10^-6  # expected minimum convergence error
j <- 10 # number of trials

# norm function
norm2 <- function(x) sum(x^2)
# compute cost function (L) for k-means algorithm
cost<- function(label, x, new_center){   # input label that we want to test
  sum <- 0
  n <- dim(x)[1]
  for(i in 1:n){
    sum <- sum + norm2((x[i, ] - new_center[label[i], ]))
  }
  sum / n
}

# lower bound function
lb <- function(x, k){
  n <- dim(x)[1]
  svd_r <- svd(x)
  u <- svd_r$u
  s <- svd_r$d
  vh <- svd_r$v
  k_2 <- k+1
  L_lower <- sum((s[k_2:length(s)]^2))/n
  return(L_lower)
}

### Batch K-Means
bk <- function(x, k, label_0, e){
  ## x here is the data set and k is the number of clusters
  ## e is expected minimum convergence error
  # construct random initial classification(very random)
  label_new <- label_0
  continue <- TRUE #decide whether continue while loop or not
  n <- dim(x)[1]
  d <- dim(x)[2]
  while(continue){
    continue <- FALSE
    label_reference <- label_new # make a copy of label_new
    new_center <- matrix(0, 3, d) # each time update new_center
    # compute the new centroids for j in range(k) :
    # I set the output new_center as a 3*10 matrix
    for(j in 1:k){
      labj <- which(label_new == j) #the set of index that has label j
      x_inj <- x[labj,] #subset of x with label j
      new_center[j,] <- apply(x_inj, 2, mean) # 3 center as three rows of new_center matrix
    }
    # for loop for computing new labels
    for(i in 1:n){
      length <- vector()
      # calculate lost functions of different centers and find the smallest one
      for(j in 1:k){
        length[j] <- norm2(x[i,] - new_center[j,])
      }
      label_new[i] <- which(length == min(length))
    }
    # set up condition for stopping while loop (enough accuracy)
    L_new <- cost(label_new, x, new_center)
    L_ref <- cost(label_reference, x, new_center)
    if(L_new <= L_ref - e) {continue <- TRUE}
  }
  return(list(label_new, L_new, new_center))
}

### Batch K-Means Error

bkerror <- function(k, label_true, label)
{
  # input k is the number clusters
  # label_true is the true label while label is the output from bk algorithm
  permn_list <- permn(1:k)  # possible permutations
  error <- mean(label_true != label) # original error
  # use for loop to find the most possible permutation
  for (i in (1: length(permn_list))){
    permn_try <- permn_list[[i]] # try ith permutation
    label_new <- label
    label_new[label_new == 1] <- permn_try[1]
    label_new[label_new == 2] <- permn_try[2]
    label_new[label_new == 3] <- permn_try[3]
    new_error <- mean(label_true != label_new) # calculate new error
    if(new_error < error){
      error <- new_error
    }
  }
  return(error)
}


### part a
# construct required matrix
```

```
loss_m <- matrix(0, length(s_seq), j) # loss matrix
error_m <- matrix(0, length(s_seq), j) # error matrix
loss_lb <- matrix(0, length(s_seq), j)  # loss lower bound matrix
# run for different s size
for(i in (1:length(s_seq))){
  s <- s_seq[i]
  # run for j different trials
  for(j in (1:j)){
    x <- matrix(rnorm(n*d), n, d)
    label_true <- sample(1:3, n, replace = TRUE) # true label
    # make distinction in data to distinguish from different clusters
    for(p in (1:n)){
      x[p, label_true[p]] <- x[p, label_true[p]] - s
    }
    # run k means algorithm
    label_0 <- sample(1:3, n, replace = TRUE) # randomly run out initial labels
    list_result<- bk(x, k, label_0, e)# return result from algorithm
    label <- list_result[[1]]
    L <- list_result[[2]]
    center <- list_result[[3]]
    loss_m[i,j] <- L # loss matrix
    error_m[i,j] <- bkerror(k, label_true, label)
    loss_lb[i,j] <- lb(x, k)
  }
  print(s)
  print("s")
}
error_mean <- apply(error_m, 1, mean)

## plot error analysis graph
print('s values:')
print(s_seq)
print('Mean error in each s value:')
print(error_mean)
parta_result <- data.frame(s_seq, error_mean)
#plot(s_seq, error_mean, type = "l", xlab = "s value", ylab = "Error", main = "Error vs. s value")
# part a plot
quartz('part a')
ggplot(parta_result, aes(s_seq, error_mean)) +
  geom_line(size = 1.5, colour = "#FF9999") +
  geom_point(size = 2, colour = "#FF9999") +
  ggtitle("Error vs. Sequence s")

# part b plot
mean_Loss <- apply(loss_m, 1, mean)
mean_Loss_lower <- apply(loss_lb, 1, mean)
print('mean loss:')
print(mean_Loss)
print('mean loss lower bound:')
print(mean_Loss_lower)
partb_result <- data.frame(mean_Loss, mean_Loss_lower,s_seq)

quartz('partb')
ggplot(partb_result, aes(s_seq)) +
  geom_line(aes(y=mean_Loss), color = "blue") +
  geom_line(aes(y=mean_Loss_lower), color = "red") +
  geom_point(aes(y=mean_Loss), color = "blue") +
  geom_point(aes(y=mean_Loss_lower), color = "red") +
  ggtitle("Comparing Loss with Lower Bound")


#### part c
setwd("/Users/Renaissance/Desktop")
seed <- read.table("seeds_dataset.txt")

j <- 10 # number of trials
k <- 3
e <- 1e-6

x_seed <- seed[c(-8)] # x_bar
truelabel_seed <-seed[c(8)]
truelabel_seed <- as.vector(unlist(truelabel_seed), mode = "numeric")
n2 <- dim(x_seed)[1]
d2 <- dim(x_seed)[2]
# without standardize

Loss <- vector(mode = "numeric", length = j)
Error <- vector(mode = "numeric", length = j)

for (j in 1: j) {
  label_0_seed <- sample(1:k, n2, replace = TRUE)
  result_seed <- bk(x_seed, k, label_0_seed, e)
  label_seed <- result_seed[[1]]
  L_seed <- result_seed[[2]]
  center_seed <- result_seed[[3]]

  Loss[j] <- L_seed
  Error[j] <- bkerror(k,truelabel_seed,label_seed)
  print(j)
}
Loss_lower_seeds <- lb(x_seed,k)
print('loss:')
print(Loss)
print('error frac:')
print(Error)
print('num wrong:')
print(Error*n)
print('lower bound error')
print(Loss_lower_seeds)

# pre-process the data
x_seed_proc <- matrix(0, 210, 7)

for (i in 1:d2){
  x_seed_proc[,i] <- (x_seed[,i] - mean(x_seed[,i])) / sd(x_seed[,i])
}
```

```
Loss_p <- vector(mode = "numeric", length = j)
Error_p <- vector(mode = "numeric", length = j)

for (j in 1: j) {
  label_0_seed <- sample(1:k, n2, replace = TRUE)
  result_seed <- bk(x_seed_proc, k, label_0_seed, e)
  label_seed <- result_seed[[1]]
  L_seed <- result_seed[[2]]
  center_seed <- result_seed[[3]]

  Loss_p[j] <- L_seed
  Error_p[j] <- bkerror(k,truelabel_seed,label_seed)
  print(j)
}
Loss_lower_seeds_p <- lb(x_seed_proc,k)
print('loss:')
print(Loss_p)
print('error frac:')
print(Error_p)
print('num wrong:')
print(Error_p*n)
print('lower bound error')
print(Loss_lower_seeds_p)

j_seq <- seq(1,10, by = 1)
part3result <- data.frame(j_seq, Loss, Loss_p) #construct a dataframe to store result

quartz()
ggplot(part3result, aes(j_seq)) +
  ylim(0,3.2) +
  geom_line(aes(y = Loss, color = "blue")) +
  geom_line(aes(y = Loss_p, color = "red")) +
  geom_line(aes(y = Loss_lower_seeds, color = "blue")) +
  geom_line(aes(y = Loss_lower_seeds_p, color = "red")) +
  labs(title = "Loss function compare", x = "nth Trial", y = "Loss") +
  scale_color_discrete(name = "Loss function & its lower bound", labels = c("Loss of raw data", "Loss of preprocssing data"))
```