Homework 1

CS 498, Spring 2018, Xiaoming Ji

Problem 1

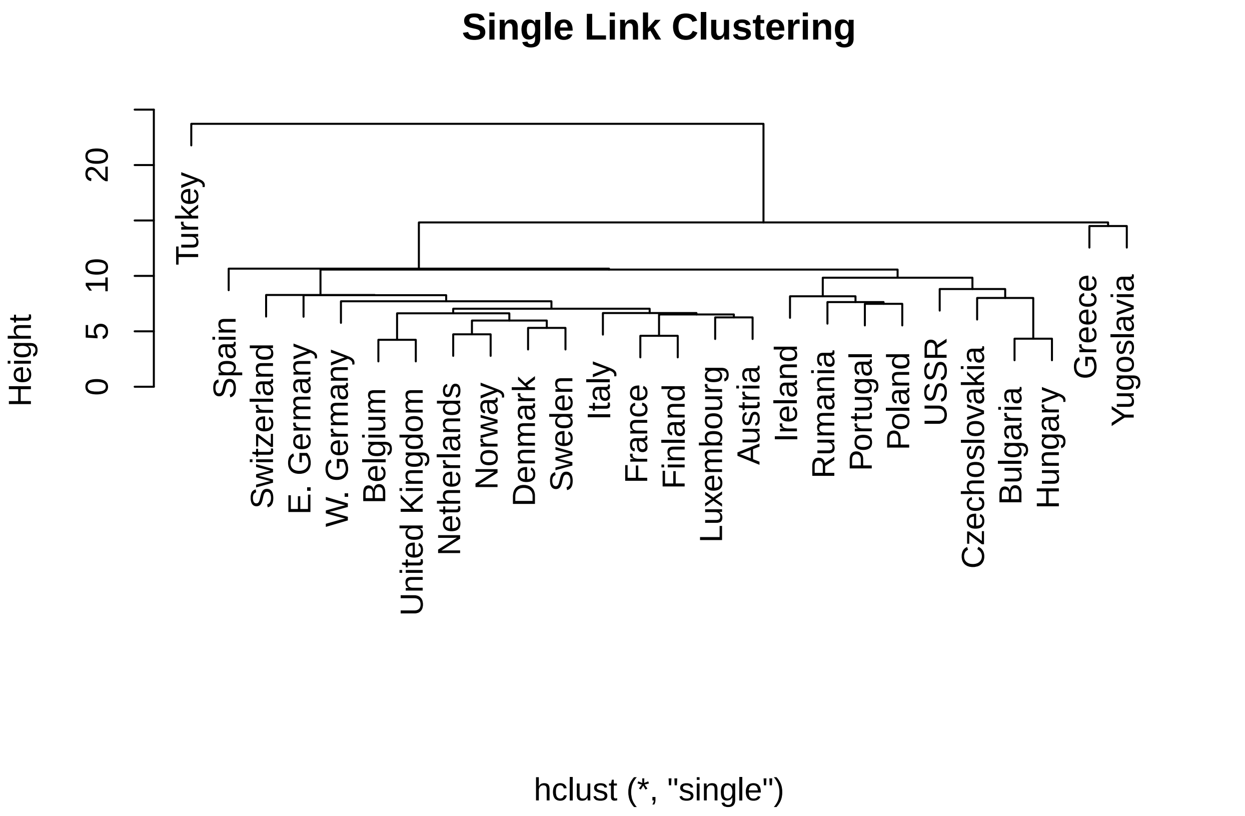
Find a dataset dealing with European employment in 1979 at http://lib.stat.cmu.edu/DASL/Stories/EuropeanJobs.html. This dataset gives the percentage of people employed in each of a set of areas in 1979 for each of a set of European countries. Notice this dataset contains only 26 data points.

Part (1)

Use an agglomerative clusterer to cluster this data. Produce a dendrogram of this data for each of single link, complete link, and group average clustering.

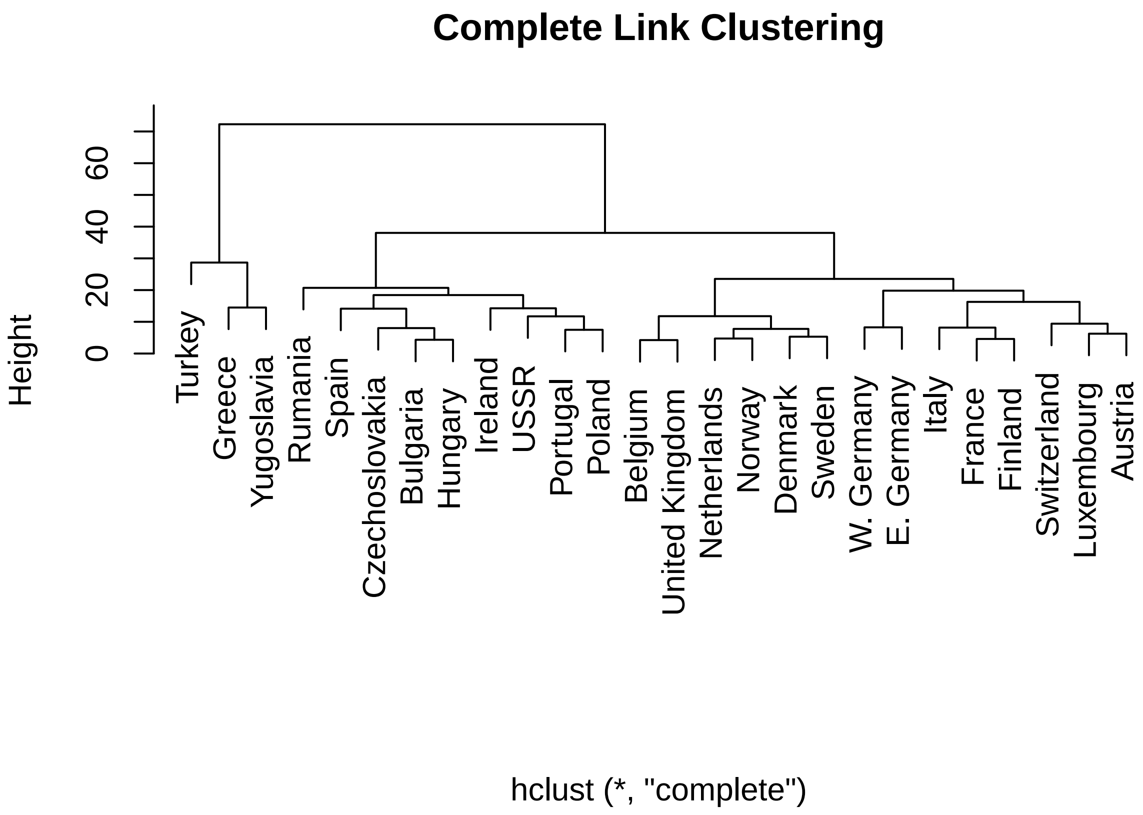
```r
library(readr)
EuropeanJobs = read_csv("EuropeanJobs.csv")
Countries = EuropeanJobs$Country
JobsStat = EuropeanJobs[, -1]
d = dist(JobsStat)
```

```r
plot(hclust(d, method = "single"), labels = Countries, main = "Single Link Clustering",
     xlab = "")
```



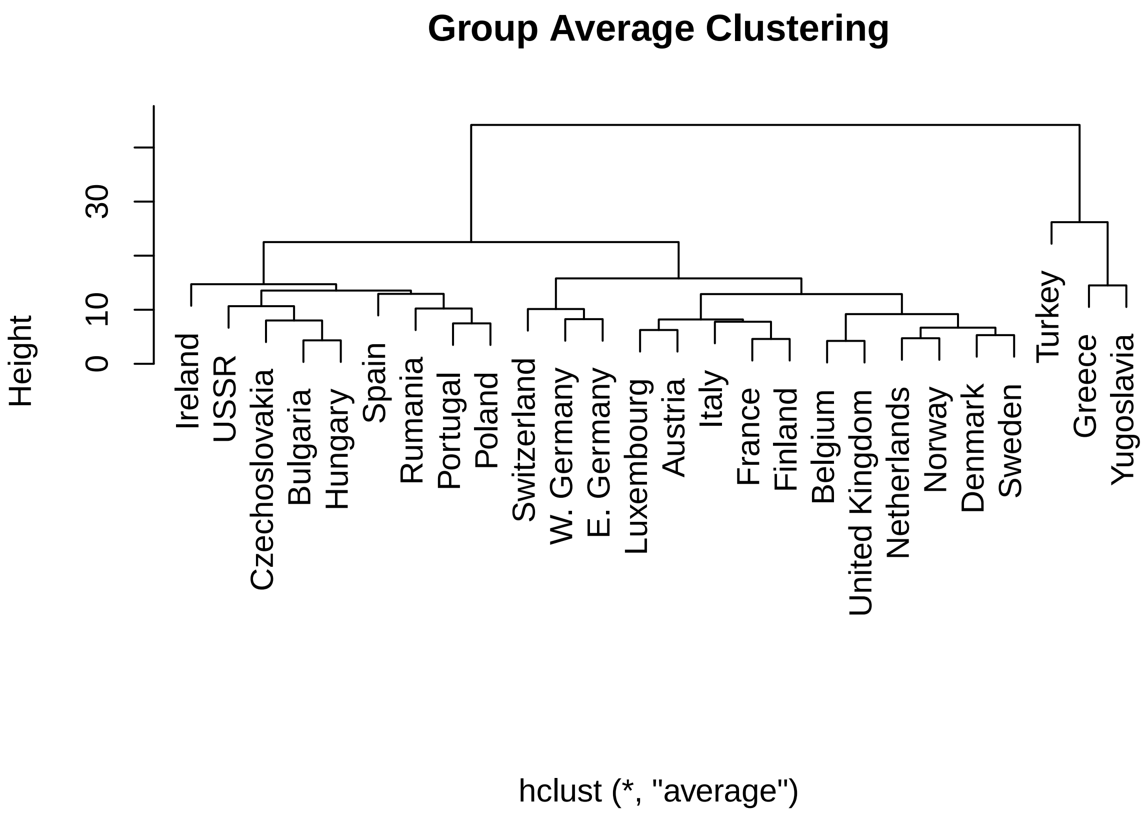
- This clustering expose Turkey as a very special single node cluster against every other countries. It makes some sense as Turkey is a country span both Asia and Europe.

```r
plot(hclust(d, method = "complete"), labels = Countries,
     main = "Complete Link Clustering", xlab = "")
```



- From some height level, this clustering makes Turkey, Greece and Yugoslavia one cluster. Romania, Czechoslovakia, USSR etc a second cluster and the rest counties a 3rd cluster. If we check the history, countries in the second cluster are most socialism countries at that time (except Spain, Portugal and Ireland) and also close to each other. Countries in the 3rd cluster are capitalism countries except East Germany which is closely grouped with West Germany (since they were one country).
- Countries close to each other will more likely to be clustered together. For example: East Germany and West Germany. Netherlands, Norway, Denmark and Sweden. Belgium and United Kingdom.

```r
plot(hclust(d, method = "average"), labels = Countries,
     main = "Group Average Clustering", xlab = "")
```



- The results of this clustering is very similar to the Complete Link Clustering.

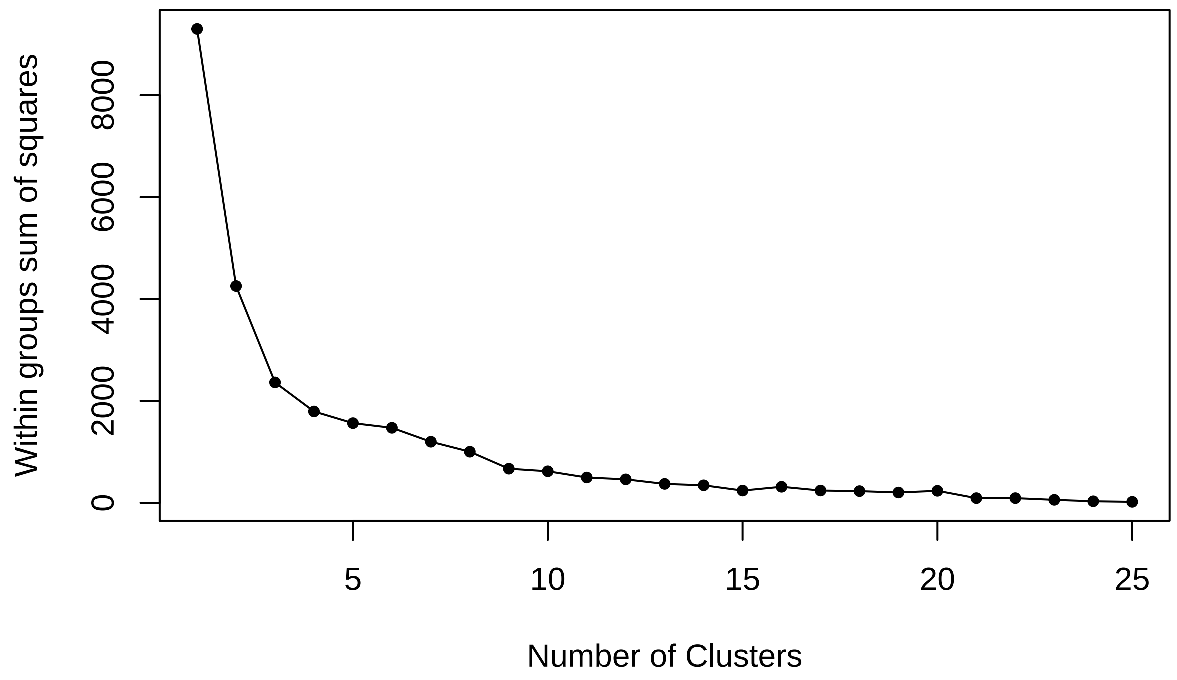
Part (2)

Using k-means, cluster this dataset. determine what is a good choice of k for this data.

We make plot of the cost (total within-cluster sum of squares) for different values of k.

```r
set.seed(1972)
ws = lapply(1:(nrow(JobsStat) - 1), function(x){kmeans(JobsStat, centers = x)$tot.withinss})

plot(1:(nrow(JobsStat) - 1), ws, type="o", xlab = "Number of Clusters",
     ylab = "Within groups sum of squares", pch = 20)
```



- Notice there is a sharp drop in cost from k = 1 to k = 4. After that, the cost falls off slowly. This suggests using k = 4 or k = 5 are the good choices for k. The plots from part (1) support this conclusion.

Problem 2

Obtain the activities of daily life dataset from the UC Irvine machine learning website (https://archive.ics.uci.edu/ml/datasets/Dataset+for+ADL+Recognition+with+Wrist-worn+Accelerometer data provided by Barbara Bruno, Fulvio Mastrogiovanni and Antonio Sgorbissa).

We will load all data and split them among training and test dataset.

```
ActivityLabels = c("Brush_teeth", "Climb_stairs", "Comb_hair", "Descend_stairs",
                   "Drink_glass", "Eat_meat", "Eat_soup", "Getup_bed",
                   "Liedown_bed", "Pour_water", "Sitdown_chair",
                   "Standup_chair", "Use_telephone", "Walk")
DataFolderName = "HMP_Dataset"
AllRawData = list()
RawTrainData = list()
RawTestData = list()

for (i in 1:length(ActivityLabels)){
  activity_data = list()
  label = ActivityLabels[i]
  folder_name = paste("./HMP_Dataset/", label, sep = "")
  file_names = dir(folder_name)
  for (j in 1:length(file_names)){
    full_file_name = paste(folder_name, "/", file_names[j], sep = "")
    data =  read_delim(full_file_name, " ", col_names = c("X", "Y", "Z"),
                       col_types = cols(X="i", Y="i", Z="i"), trim_ws = TRUE)
    activity_data[[j]] = data
  }
  AllRawData[[i]] = activity_data
}

set.seed(19720816)
# Sample 80% as training data and the rest 20% as test data. Save them to RawTrainData
# and RawTestData
for (i in 1:length(ActivityLabels)){
  all_indexes = 1:(length(AllRawData[[i]]))
  train_indexes = sample(all_indexes, floor((length(AllRawData[[i]])) * 0.8))
  test_indexes = all_indexes[-train_indexes]
  train_data = list()
  test_data = list()
  for(j in 1:length(train_indexes)){
    train_data[[j]] = AllRawData[[i]][[train_indexes[j]]]
  }
  for(j in 1:length(test_indexes)){
    test_data[[j]] = AllRawData[[i]][[test_indexes[j]]]
  }
  RawTrainData[[i]] = train_data
  RawTestData[[i]] = test_data
}
```

```
#Cut n*3 dataframe as samples vectors
cut.as.vector = function(df3, sample_length){
  row_count = floor(dim(df3)[1] / sample_length)
  m_data = matrix(0, row_count, sample_length * 3)

  index = 1
  for(i in 1:row_count){
    d = df3[index:(index + sample_length -1),]
    d = as.matrix(d)
    dim(d) = c(sample_length * 3, 1)
    d = as.vector(d)
    m_data[i,] = d

    index = index + sample_length
  }

  return (m_data)
}

# Cut activity data as fixed size data
cut.listdf = function(ll_df, sample_length){
  new_ll_df = list()

  for (i in 1:length(ll_df)){
    activity_data = list()
    for (j in 1:length(ll_df[[i]])){
      activity_data[[j]] = cut.as.vector(ll_df[[i]][[j]], sample_length)
    }
    new_ll_df[[i]] = activity_data
  }

  return (new_ll_df)
}
```

Cut data to fixed size vector (32*3)

```r
TrainData = cut.listdf(RawTrainData, 32)
TestData = cut.listdf(RawTestData, 32)
```

Part (a) - Build a classifier that classifies sequences into one of the 14 activities provided

```r
#Merge all training data
merge_list_df = function (ll_df) {
  all_df = data.frame()
  for (i in 1:length(ll_df)){
    for (j in 1:length(ll_df[[i]])){
      all_df = rbind(all_df, ll_df[[i]][[j]])
    }
  }
  return (all_df)
}

vector_quantize = function(df, centers){
  df = as.matrix(df)
  centers = as.matrix(centers)
  cv = rep(0, dim(df)[1])

  for (i in 1:(dim(df)[1])){
    c = t(centers) - df[i,]
    d = colSums(c^2)
    cv[i] = which.min(d)
  }
  return (cv)
}

# Compute histgram of the data and assign label number. This will make
# feature vectors for classification.
featurize_data = function(ll_df, centers){
  all_features = data.frame()
  sample_interval = seq(0, dim(centers)[1], 1)

  for (i in 1:length(ll_df)){
    df = data.frame()

    mv = matrix(0, length(ll_df[[i]]), length(sample_interval) - 1)
    for (j in 1:length(ll_df[[i]])){
      mv[j,] = hist(vector_quantize(ll_df[[i]][[j]], centers), breaks = sample_interval,
                    plot = FALSE)$density
    }
    all_features = rbind(all_features, cbind(as.data.frame(mv),label=i))
  }

  return (all_features)
}
```

We build a dictionary by clustering the training data with $k = 480$.

```r
AllTrainDF = merge_list_df(TrainData)

library(factoextra)
km = hkmeans(AllTrainDF, 480)
```

We vector quantize training and test dataset and compute the histgram of the centers for each dataset. This is to build features for later classification.

```r
HistTrainData = featurize_data(TrainData, km$centers)
HistTestData = featurize_data(TestData, km$centers)
```

We then use random forest to train and calculate the accuracy of the test dataset. In the classifier, we try different depth of tree and number of trees to get the best model.

```r
library(h2o)

get_best_classifier = function(train_frame, test_frame, ntrees, max_depths) {
  accuracies = rep(0, length(ntrees) * length(max_depths))
  models = list()

  index = 1
  for (ntree in 1:length(ntrees)) {
    for (depth in 1:length(max_depths)) {
      rf = h2o.randomForest(y = "label", training_frame = train_frame,
                            ntrees = ntrees[ntree], max_depth = max_depths[depth])
      prediction = h2o.predict(rf, newdata = test_frame[-dim(test_frame)[2]])

      models[[index]] = rf
      accuracies[index] = sum(prediction[,1] == test_frame["label"]) / dim(test_frame)[1]
      index = index + 1
    }
  }

  accuracy = max(accuracies)
  model = models[[which.max(accuracies)]]

  return (list(accuracy = accuracy, model = model))
}

h2o.init(nthreads=-1, max_mem_size = "2G")
h2o.removeAll()

HistTrainData$label = as.factor(ActivityLabels[HistTrainData$label])
HistTestData$label = as.factor(ActivityLabels[HistTestData$label])

h2o_train_data = as.h2o(HistTrainData)
h2o_test_data = as.h2o(HistTestData)

result = get_best_classifier(h2o_train_data, h2o_test_data, c(10, 20, 30), c(4, 8, 16))
result[["CM"]] = h2o.confusionMatrix(result$model)

save(result, file="result.data")
```

We got the result as,

- Error rate: 0.2254335

- Confusion Matrix:

```
result[["CM"]]
```

```
##                 Brush_teeth Climb_stairs Comb_hair Descend_stairs
## Brush_teeth               8            0         0              0
## Climb_stairs              0           62         0              0
## Comb_hair                 0            0        22              0
## Descend_stairs            0           10         0             19
## Drink_glass               0            0         0              0
## Eat_meat                  0            0         0              0
## Eat_soup                  0            0         0              0
## Getup_bed                 0            2         0              0
## Liedown_bed               0            0         0              1
## Pour_water                0            0         0              0
## Sitdown_chair             0            1         0              0
## Standup_chair             0            2         0              2
## Use_telephone             0            0         0              0
## Walk                      0            9         0              1
## Totals                    8           86        22             23
##                 Drink_glass Eat_meat Eat_soup Getup_bed Liedown_bed
## Brush_teeth               0        0        0         0           0
## Climb_stairs              1        0        0         1           0
## Comb_hair                 0        0        0         1           1
## Descend_stairs            0        0        0         1           0
## Drink_glass              74        0        0         1           0
## Eat_meat                  0        3        0         0           0
## Eat_soup                  0        0        1         0           0
## Getup_bed                 3        0        0        58           1
## Liedown_bed               1        0        0        10           1
## Pour_water                2        0        0         1           0
## Sitdown_chair             2        0        0         5           0
## Standup_chair             4        0        0         6           0
## Use_telephone             3        0        0         0           0
## Walk                      0        0        0         2           0
## Totals                   90        3        1        86           3
##                 Pour_water Sitdown_chair Standup_chair Use_telephone Walk
## Brush_teeth              0             0             1             0    0
## Climb_stairs             2             2             2             0   11
## Comb_hair                0             0             0             0    0
## Descend_stairs           0             1             0             0    2
## Drink_glass              4             0             0             1    0
## Eat_meat                 1             0             0             0    0
## Eat_soup                 1             0             0             0    0
## Getup_bed                8             1             6             0    1
## Liedown_bed              2             6             1             0    0
## Pour_water              77             0             0             0    0
## Sitdown_chair            7            52            13             0    0
## Standup_chair            5             9            52             0    1
## Use_telephone            2             0             0             5    0
## Walk                     1             1             2             0   64
## Totals                 110            72            77             6   79
##                      Error      Rate
## Brush_teeth     0.11111111     1 / 9
## Climb_stairs    0.23456790   19 / 81
## Comb_hair      0.08333333    2 / 24
## Descend_stairs 0.42424242   14 / 33
## Drink_glass    0.07500000    6 / 80
## Eat_meat       0.25000000     1 / 4
## Eat_soup       0.50000000     1 / 2
## Getup_bed      0.27500000   22 / 80
## Liedown_bed    0.95454545   21 / 22
## Pour_water     0.03750000    3 / 80
## Sitdown_chair  0.35000000   28 / 80
## Standup_chair  0.35802469   29 / 81
## Use_telephone  0.50000000    5 / 10
## Walk           0.20000000   16 / 80
## Totals         0.25225225 168 / 666
```

Part (b) - Improve classifier

1.Modifying the number of cluster centers in k-means

```
cluster_number = c(100, 200, 300, 480, 500, 700, 800, 1000)
accuracies = rep(0, length(cluster_number))

for (i in 1:length(cluster_number)){
  km = hkmeans(AllTrainDF, cluster_number[i])
  hist_train_data = featurize_data(TrainData, km$centers)
  hist_test_data = featurize_data(TestData, km$centers)

  hist_train_data$label = as.factor(ActivityLabels[hist_train_data$label])
  hist_test_data$label = as.factor(ActivityLabels[hist_test_data$label])

  h2o_train_data = as.h2o(hist_train_data)
  h2o_test_data = as.h2o(hist_test_data)

  accuracies[i] = get_best_classifier(h2o_train_data, h2o_test_data, c(20, 30),
                                      c(8, 16))$accuracy

  cat(accuracies[i])
}

improve_result_1 = list(cluster_number = cluster_number, accuracies = accuracies)
save(improve_result_1, file = "improve_result_1.data")
```

This experiment shows cluster of 100 improved the accuracy to:0.7919075.

2.Modifying the size of the fixed length samples

```
sample_size = c(8, 16, 24, 32, 40, 48,  64)
accuracies = rep(0, length(sample_size))

for (i in 1:length(sample_size)){
  train_data = cut.listdf(RawTrainData, sample_size[i])
  test_data = cut.listdf(RawTestData, sample_size[i])
  all_train_df = merge_list_df(train_data)

  km = hkmeans(all_train_df, 480)
  hist_train_data = featurize_data(train_data, km$centers)
  hist_test_data = featurize_data(test_data, km$centers)

  hist_train_data$label = as.factor(ActivityLabels[hist_train_data$label])
  hist_test_data$label = as.factor(ActivityLabels[hist_test_data$label])

  h2o_train_data = as.h2o(hist_train_data)
  h2o_test_data = as.h2o(hist_test_data)

  accuracies[i] = get_best_classifier(h2o_train_data, h2o_test_data, c(20, 30),
                                      c(8, 16))$accuracy

  cat(accuracies[i])
}

improve_result_2 = list(sample_size = sample_size, accuracies = accuracies)
save(improve_result_2, file = "improve_result_2.data")
```

This experiment shows sample size of 16 improved the accuracy to:0.8034682.

Note: both experiments don't give us consistent results. But as rule of thumb, smaller size of sample (8, 16, 24) usually give us better results than 32.