Stat 6910, Section 003
Statistical Learning and Data Mining II
Fall 2018

Homework 6

Jared Hansen

Due: 5:00 PM, Friday 12/07/18

A-number: `A01439768`

e-mail: `jrdhansen@gmail.com`

1. **PHATE (35 pts).** Download code for the PHATE visualization tool at `github.com/KrishnaswamyLab/PHATE`. Download the `EBdata.mat` file from Canvas. This data file contains a sparse matrix labeled "data" and a vector labeled "cells". The data matrix contains cleaned single-cell RNA-sequencing data from embryoid bodies with 16,825 cells and 17,580 genes. The cells vector indicates which time point the cell was extracted from. You will apply PHATE to visualize this data.

   (a) (5 pts) Load the data. Indicate what packages or libraries you used. In particular, the data matrix is saved as a sparse Matlab matrix. You should be able to find R packages that can load sparse matlab matrices into R.

   Python libraries used:

   - os
   - zipfile
   - urlopen
   - pandas
   - numpy
   - scprep

   Completed this step, see code at bottom of problem 1.

   (b) (10 pts) Run PHATE on the data using default parameters. Plot the PHATE coordinates colored by time point (the "cells" vector) and include the plot.

   Default parameters for the PHATE function (used to create this plot):

   - k = 5 (k is the number of nearest neighbors)
   - a = 40 (a is alpha decay)
   - t = "auto", in this case, t = 21 (t is the number of times to power the operator)
   - gamma = 1 (gamma is the informational distance constant)

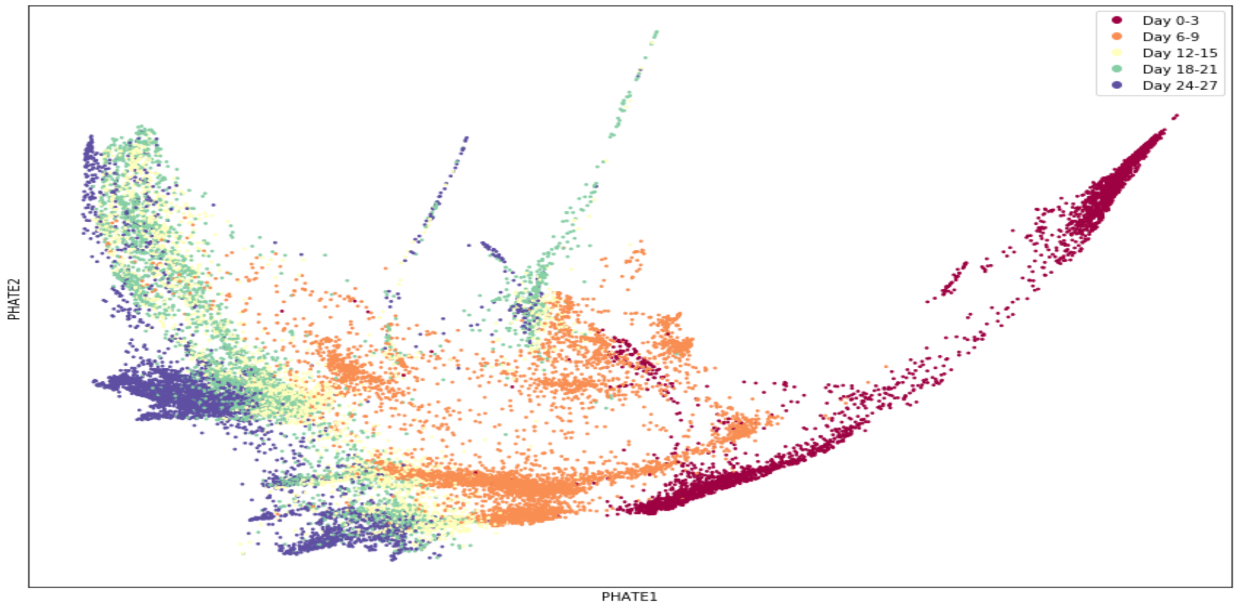


Figure 1: PHATE plot using default parameters

(c) (10 pts) Run PHATE on the data using a different value of t. Plot the PHATE coordinates colored by time point and include the plot. Based on the results, do you think your chosen value of t is better than the parameter chosen using the "knee point" of the VNE plot (the value chosen using default parameters)? Will the VNE be higher or lower for your chosen value of t than that selected in part (b)?

I chose a value of $t = 30$ to see if we could see any additional de-noising or clearer structure in the plot. Based on the results, I don't think that my chosen value of t is better than the default parameter chosen using the "knee point" of the VNE plot. This is unsurprising, since this dataset is used as the illustrative example for PHATE. Typically a dataset that works well with an algorithm is used to illustrate it; in other words, we'd expect that the default parameters work pretty well for the canonical example.
I'd say that my value of t isn't any better than the default since it yields a plot that is nearly identical to the plot obtained by using the default value of t ($t = 21$ in this case). Since the plots are nearly the same and a larger t requires greater computational time, we can fairly definitively say that $t = 30$ is a worse choice than $t = 21$ in this case.
We know that as t increases the Von Neumann Entropy decreases. Thus the VNE will be lower for my chosen t of 30 than it will be for the default t of 21 chosen in part (b). However, based on the fact that the plot is almost the same for these values, I'd bet that the decrease in VNE is fairly negligible going from $t = 21$ to $t = 30$.

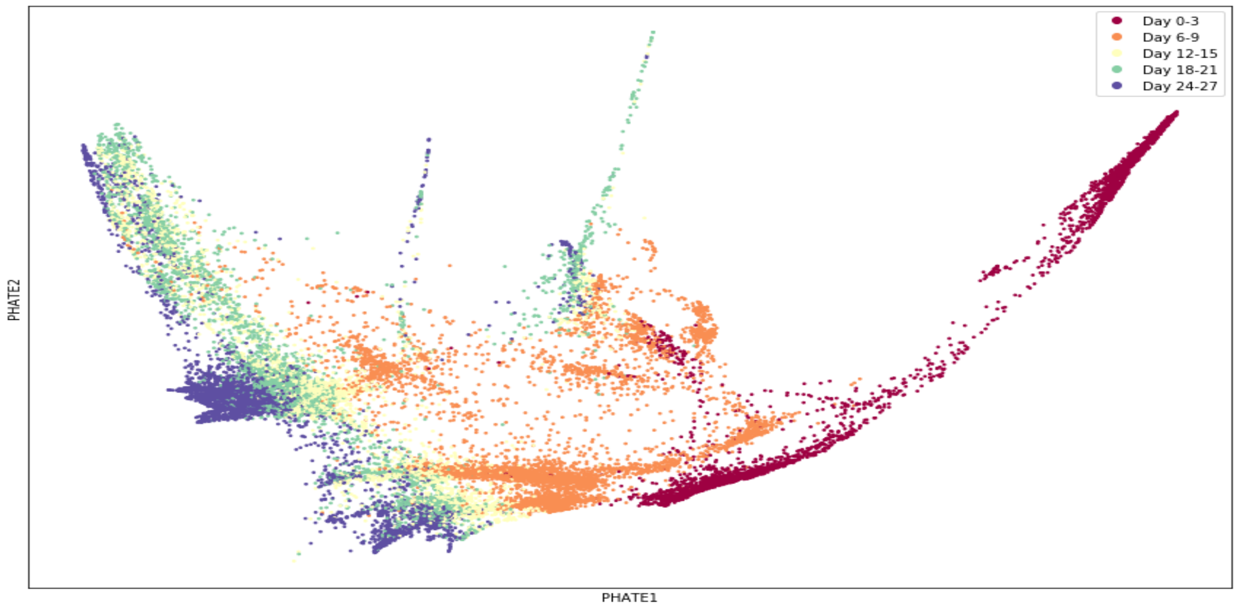


Figure 2: PHATE plot using $t = 30$ (nearly identical to Figure 1, but I promise they're slightly different. Look at noteable features of the plot and you can spot small differences.)

(d) (10 pts) Run PHATE on the data using default parameters to obtain 3d coordinates. Plot the 3d coordinates. You may need to rotate the plot to get a good view.

I had a hard time explicitly rotating the plot a given amount. So I just generated a GIF, and then took a screenshot at a point that was rotated different from the tutorial example. I tried to pick a rotation that gave some insight into the 3D structure.

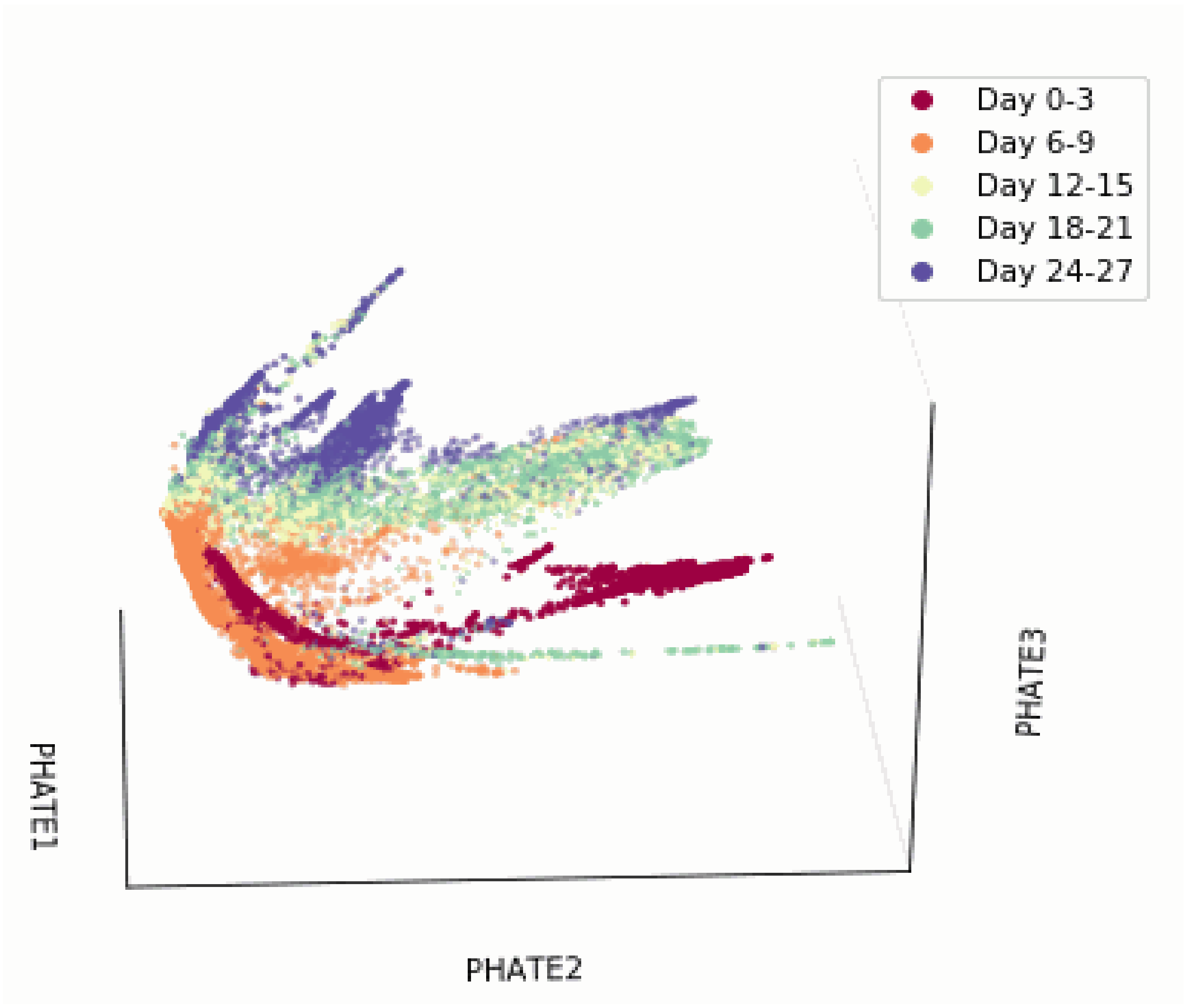


Figure 3: 3D PHATE plot using default parameters

BELOW IS MY CODE FOR THIS PROBLEM

```
#==============================================================================
#====== SLDM hw6, PROBLEM 1 ===================================================
#==============================================================================



# Need to pip install: "phate", "scprep" on the command line before running
# the rest of this code in an IDE.



#==============================================================================
#==== Loading the 10X data ====================================================
#==============================================================================
# PART A from the homework

# Import the data (code given on the tutorial website at:
# https://github.com/KrishnaswamyLab/PHATE/blob/master/Python
#         /tutorial/EmbryoidBody.ipynb)
import os
import zipfile
from urllib.request import urlopen
download_path = os.path.expanduser("~")
print(download_path)

if not os.path.isdir(os.path.join(download_path, "scRNAseq", "T0_1A")):
    if not os.path.isdir(download_path):
        os.mkdir(download_path)
    zip_data = os.path.join(download_path, "scRNAseq.zip")
    if not os.path.isfile(zip_data):
        with urlopen("https://data.mendeley.com/datasets/v6n743h5ng"
                     "/1/files/7489a88f-9ef6-4dff-a8f8-1381d046afe3"
                     "/scRNAseq.zip?dl=1") as url:
            print("Downloading data file...")
            # Open our local file for writing
            with open(zip_data, "wb") as handle:
                handle.write(url.read())
    print("Unzipping...")
    with zipfile.ZipFile(zip_data, 'r') as handle:
        handle.extractall(download_path)
    print("Done.")

# Need these libraries
import pandas as pd
import numpy as np
import phate
import scprep

# Now use scprep to import data into a Pandas dataframe, using the
# scprep.io.load_10x function.
sparse=True
T1 = scprep.io.load_10X(os.path.join(download_path,"scRNAseq", "T0_1A"),
                        sparse=sparse,
                        gene_labels='both')
T2 = scprep.io.load_10X(os.path.join(download_path, "scRNAseq", "T2_3B"),
                        sparse=sparse,
                        gene_labels='both')
T3 = scprep.io.load_10X(os.path.join(download_path, "scRNAseq", "T4_5C"),
                        sparse=sparse,
                        gene_labels='both')
T4 = scprep.io.load_10X(os.path.join(download_path, "scRNAseq", "T6_7D"),
                            sparse=sparse,
                            gene_labels='both')
T5 = scprep.io.load_10X(os.path.join(download_path, "scRNAseq", "T8_9E"),
                            sparse=sparse,
                            gene_labels='both')
T1.head()

# Now, merge all datasets and create a vector representing the time point of
# each sample
EBT_counts, sample_labels = scprep.utils.combine_batches(
    [T1, T2, T3, T4, T5],
    ["Day 0-3", "Day 6-9", "Day 12-15", "Day 18-21", "Day 24-27"],
    append_to_cell_names=True
)
del T1, T2, T3, T4, T5 # removes objects from memory
EBT_counts.head()


#===============================================================================
#===== Preprocessing: filtering, normalizing, and transforming ================
#===============================================================================
# This is PART B from the homework (technically most of this isn’t asked for in
# the homework, but I figured it made more sense to follow the tutorial exactly

# Remove (suspected) dead cells
mito_genes = scprep.utils.get_gene_set(EBT_counts, starts_with="MT-")
# Get all mitochondrial genes. There are 14, FYI.
scprep.plot.plot_gene_set_expression(EBT_counts, mito_genes, percentile=90)
# Plot number of cells that have a certain amount of mitochondrial RNA,
# remove cells that are above the 90th percentile. (Line below)
EBT_counts, sample_labels = scprep.filter.filter_gene_set_expression(
    EBT_counts, mito_genes,
    percentile=90,
    keep_cells='below',
    sample_labels=sample_labels)

# Now filter out cells that have either very large or very small library sizes.
# Library size is somewhat analogous to sample size. We'll eliminate the
# bottom 20% of cells for each sample.
scprep.plot.plot_library_size(EBT_counts, percentile=20)
EBT_counts, sample_labels = scprep.filter.filter_library_size(
    EBT_counts, percentile=20,
    keep_cells='above',
    sample_labels=sample_labels,
    filter_per_sample=True)

# Now remove rare genes (genes expressed in 10 or fewer cells)
EBT_counts = scprep.filter.remove_rare_genes(EBT_counts, min_cells=10)

# Normalization: accounting for differences in library sizes, divide each cell
# by its library size and then rescale by the median library size.
EBT_counts = scprep.normalize.library_size_normalize(EBT_counts)

# Transformation: use square root transform (similar to using log transform
# but has the added benefit of dealing with 0's automatically).
EBT_counts = scprep.transform.sqrt(EBT_counts)


#===============================================================================
#===== Applying PHATE to data ==================================================
#===============================================================================
```

```python
# (In the tutorial, this section is "Embedding Data Using PHATE")

# Default parameters for the PHATE function are:
# k: number of nearest neighbors, default is 5
# a: alpha decay, default is 40
# t: number of times to power the operator, default "auto", 21 for these data
# gamma: informational distance constant, default is 1.
phate.PHATE()
phate_operator = phate.PHATE(n_jobs=-2)
Y_phate = phate_operator.fit_transform(EBT_counts)

# Now plot using phate.plot.scatter2d
phate.plot.scatter2d(Y_phate,
                     c=sample_labels,
                     s=3,
                     figsize=(12,8),
                     cmap="Spectral")

# PART D (from homework): run PHATE on the data using a different value of t.
# Plot the PHATE coordinates colored by time point and include the plot.
# Based on the results, do you think your chosen value of t is better than the
# parameter chosen using the "knee point" of the VNE plot (the default value)?
# Will the VNE be higher or lower for your chosen value of t than that
# selected (by default) in part(b)?
phate_op_2 = phate.PHATE(n_jobs = -2,
                         t = 30)
Y_phate_2 = phate_op_2.fit_transform(EBT_counts)

# Now plot using phate.plot.scatter2d having used t = 30
phate.plot.scatter2d(Y_phate_2,
                     c=sample_labels,
                     s=3,
                     figsize=(12,8),
                     cmap="Spectral")

# PART E (from homework): run PHATE on the data using default parameters to
# obtain 3d coordinates. Plot the 3d coordinates. Rotate the plot such that
# it's different from what the tutorial has.
phate.plot.scatter3d(phate_operator, c=sample_labels, s=3, figsize=(8,6),
                     cmap="Spectral")
# This saves the 3D plot as a gif
phate.plot.rotate_scatter3d(phate_operator, c=sample_labels,
                            s=3, figsize=(8,6), cmap="Spectral",
                            filename="phate.gif")
# This saves the 3D plot as an MP4 (which is also a cool gun in my opinion)
phate.plot.rotate_scatter3d(phate_operator, c=sample_labels,
                            s=3, figsize=(8,6), cmap="Spectral",
                            filename="phate.mp4")
```

2. **Spectral clustering and GMM (55 pts).** Download all of the data for the MNIST dataset from `yann.lecun.com/exdb/mnist/`. This should give you 60,000 images in the training data and 10,000 images in the test data. You will apply a couple of clustering algorithms to this dataset. You may use any existing packages or libraries for this problem. For all parts, you may assume the same number of clusters as classes (10 in this case).

   (a) (7 pts) Apply k-means clustering to just the features of the training data (do not include labels). Compute the adjusted Rand index (ARI) between your cluster outputs and the true labels of the data points and report the value.
   Note: You may need to do subsampling to make this computationally feasible. If you do, repeat the clustering for multiple (say 10-20) subsamples and report the average ARI.

   For computational feasibility, I calculated the ARI for 20 randomly selected subsamples of size 2000. The average ARI of these 20 subsamples was **0.3603834**.

   (b) (5 pts) Apply k-means clustering to just the features of the test data (do not include labels). Compute the ARI between your cluster outputs and the true labels of the data points and report the value.

   There was no need to do subsampling for this part, so I computed an ARI of **0.3814961** on all of the test data.

   (c) (10 pts) Apply spectral clustering to just the features of the training data (do not include labels) using a radial or Gaussian kernel. Compute the ARI between your cluster outputs and the true labels of the data points. Use the ARI to tune the kernel bandwidth parameter. Report the ARI using your selected bandwidth.
   Note: You will likely need to do subsampling to make this computationally feasible. If you do, repeat the final clustering for multiple subsamples and report the average ARI.

   For computational feasibility, I calculated the ARI for 20 randomly selected subsamples of size 2000. The average ARI of these 20 subsamples was **0.4731655**. I did this using a radial kernel with a value of $\sigma = 5.011872 \times 10^{-6}$ in R. (It's my understanding that σ is just a rescaled version of bandwidth. If not that, it is a proxy of bandwidth in some sense.)

   (d) (7 pts) Apply spectral clustering to the features of the test data using the same kernel and bandwidth you selected in part (c). Report the ARI.

   Again, I used a radial kernel with $\sigma = 5.011872 \times 10^{-6}$ as the bandwidth parameter (or σ is related to bandwidth as discussed in part (c).) This gave an ARI of **0.5388696**.

(e) (10 pts) Learn a Gaussian mixture model from the training data using the EM algorithm to cluster the data. Calculate and report the ARI.
Note: You may need to do subsampling to make this computationally feasible. If you do, repeat the clustering for multiple (say 10-20) subsamples and report the average ARI.

For computational feasibility, I calculated the ARI for 20 randomly selected subsamples of size 500. (I tried samples of size 1000, but the computation time was far greater. I guess computation time doesn't scale linearly for GMM's). The average ARI was **0.3224209**.

(f) (7 pts) Learn a Gaussian mixture model from the test data to cluster the data. Calculate and report the ARI.
Again, for computational feasibility I calculated the ARI for 20 randomly selected subsamples of size 500. (I tried samples of size 1000, but far greater computation time was needed.) The average ARI was **0.3275036**.

(g) (5 pts) Based on the reported ARI numbers, which clustering algorithm seems to work the best on this data?

Based on our results, we can see that the **spectral clustering** algorithm is the clear winner based on the metric of (average) ARI (the closer to an ARI of 1, the more concordant a clustering algorithm's predictions with the true labels). Spectral clutering outperforms both k-means and the GMM for both the training data and the test data.

(h) (4 pts) Generally when we're clustering data, we don't have access to the true labels which makes it difficult to tune parameters like the kernel bandwidth. What is another way you could tune the bandwidth without using cluster or class labels?

To be honest, I don't have a particularly good idea of how to do this. A much less quantitative approach might be to consult with a subject matter expert for the data we're analyzing. The expert might have an idea of how many clusters we should expect to see. Let's say they expect to see M clusters in a given data set. We could then manually label a subset of observations (assigning values of $1, 2, \ldots, M$), use the manually labeled data as training data, and assign predicted labels to the rest of the data. (We'd treat this labeled data as if the labels are actually true.) Then we could tune parameters by calculating ARI as we've done throughout this problem. However, this isn't a very mathematically defensible approach. Also, there's a good chance it wouldn't be feasible to label data as I've suggested.
However, if we're flying blind in this regard, I'm not really sure what we'd do. I can't think of a good way to mathematically define a measure of accuracy if there's no standard to compare things to (true labels). If nothing else, this has given me an appreciation of how difficult the problems are in unsupervised learning; when we have labels we're trying to predict, the objective of success is straightforward, but here it's a much more ambiguous thing to try and define.

NOTE: my code for PROBLEM 2 is given below

```r
#================================================================================
#============= STATISTICAL LEARNING, HW6 problem 2 ==============================
#================================================================================



# Necessary libraries
library(kknn)
library(fcd)
library(mclust)
library(speccalt)
library(dplyr)
library(kernlab)

# Set working directory
setwd("C:/Users/jrdha/OneDrive/Desktop/USU_Fa2018/Moon__SLDM2/hw6/problem2")

#======================== READ IN, FORMAT DATA ==================================
#================================================================================
# NOTE: used the helpful code that Matt shared on Piazza to do this part.
# need R.utils package installed and files downloaded in working directory.

# gunzip the files
R.utils::gunzip("train-images-idx3-ubyte.gz")
R.utils::gunzip("train-labels-idx1-ubyte.gz")
R.utils::gunzip("t10k-images-idx3-ubyte.gz")
R.utils::gunzip("t10k-labels-idx1-ubyte.gz")

# helper function for visualization
show_digit = function(arr784, col = gray(12:1 / 12), ...) {
  image(matrix(as.matrix(arr784[-785]), nrow = 28)[, 28:1], col = col, ...)
}

# load image files
load_image_file = function(filename) {
  ret = list()
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n    = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  nrow = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  ncol = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  x = readBin(f, 'integer', n = n * nrow * ncol, size = 1, signed = FALSE)
  close(f)
  data.frame(matrix(x, ncol = nrow * ncol, byrow = TRUE))
}
```

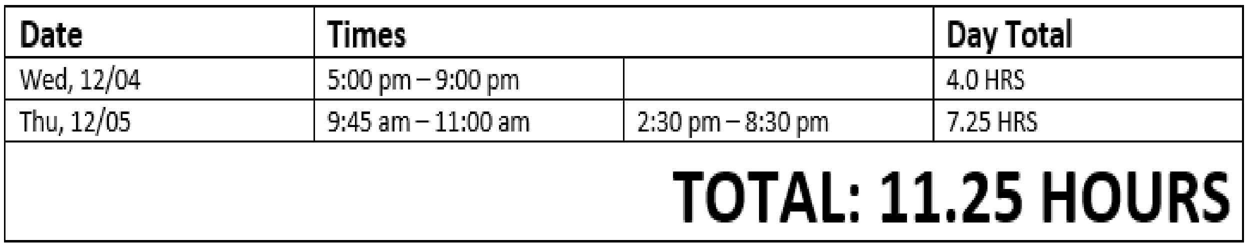
```r
# load label files
load_label_file = function(filename) {
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  y = readBin(f, 'integer', n = n, size = 1, signed = FALSE)
  close(f)
  y
}

# load images
train = load_image_file("train-images-idx3-ubyte")
test  = load_image_file("t10k-images-idx3-ubyte")

# load labels
train$y = as.factor(load_label_file("train-labels-idx1-ubyte"))
test$y  = as.factor(load_label_file("t10k-labels-idx1-ubyte"))

# view test image
# show_digit(train[10000, ])


#=================== SubSamp FUNCTION =========================================
#==============================================================================
# SubSamp takes as arguments: givenData and sampSize. Generates (and returns) a
# random sub-sample of sampSize from givenData.
SubSamp <- function(givenData, sampSize){
  samp <- dplyr::sample_n(tbl = givenData, size = sampSize, replace = FALSE)
  return(samp)
}


#==== GENERATE SUBSAMPLES TO BE USED FOR K-MEANS AND SPECTRAL CLUSTERING =======
#==============================================================================

# Set seed for reproducibility
set.seed(2345)
# Specify the number of sub-samples to generate
numSubSamp <- 20
# Specify the number of observations in each sub-sample
subSampSize <- 2000

# Generate a list ofsub-samples (will be used for k-means and spectral
# clustering)
subSampList <- list()
for (i in 1:numSubSamp) {
  subSampList[[i]] <- SubSamp(givenData = train, sampSize = subSampSize)
}
```

```r
#========= TRAINING K-MEANS CLUSTERING ON SUB-SAMPLED DATA ====================
#==============================================================================

# Store the ARI value for each sub-sample in this object
ARI.km.train <- double(numSubSamp)

# Loop over subsamples
counter <- 1 # Initialize counter var
for(ss in subSampList){
  km.train <- kmeans(x = select(ss, -y), centers = 10, nstart = 20)
  ARI.km.train[counter] <- adjustedRandIndex(km.train$cluster, ss$y)
  counter <- counter + 1
}
ARI.km.train.val <- mean(ARI.km.train)
ARI.km.train.val


#=============== APPLYING K-MEANS CLUSTERING TO TEST DATA =====================
#==============================================================================
ARI.km.test <- kmeans(x = select(test, -y), centers = 10, nstart = 20)
ARI.km.test.val <- adjustedRandIndex(ARI.km.test$cluster, test$y)
ARI.km.test.val


#========= TRAINING SPECTRAL CLUSTERING ON SUB-SAMPLED DATA ===================
#==============================================================================
# Selecting one sub-sample to be used for tuning of simga parameter
# (The sigma parameter is a proxy for bandwidth here)
set.seed(2345)
subSampTune <- SubSamp(train, 400)
# sigmaGrid <- 1*10^(-10:-5) # Initial rough grid: ARI max at 1e-06
sigmaGrid <- 1*10^(seq(-7, -5, by = 0.1)) # Narrowed down to this grid

ARI.sc.tune <- double(length(sigmaGrid))
counter <- 1
for(sg in sigmaGrid){
  sc.train <- specc(as.matrix(dplyr::select(subSampTune, -y)),
                    centers = 10,
                    kernel = "rbfdot",
                    kpar = list("sigma" = sg))
  ARI.sc.tune[counter] <- adjustedRandIndex(sc.train@.Data, subSampTune$y)
  counter <- counter + 1
}
max(ARI.sc.tune)

# selected sigma = 5.011872e-06
sigma = 5.011872 * (10^(-6))
```

```r
ARI.sc.train <- double(numSubSamp)
counter <- 1
for(ss in subSampList){
  sc.train <- specc(as.matrix(dplyr::select(ss, -y)),
                    centers = 10,
                    kernel = "rbfdot",
                    kpar = list("sigma" = sigma))
  ARI.sc.train[counter] <- adjustedRandIndex(sc.train@.Data, ss$y)
  counter <- counter + 1
}
ARI.sc.train.val <- mean(ARI.sc.train)
ARI.sc.train.val


#=============== APPLYING SPECTRAL CLUSTERING TO TEST DATA =====================
#===============================================================================
set.seed(2345)
ARI.sc.test <- specc(as.matrix(dplyr::select(test, -y)),
                     centers = 10,
                     kernel = "rbfdot",
                     kpar = list("sigma" = sigma))
ARI.sc.test.val <- adjustedRandIndex(ARI.sc.test@.Data, test$y)
ARI.sc.test.val

#========= TRAINING GMM ON SUB-SAMPLED DATA ====================================
#===============================================================================
# these seem to take longer to train, so at this point I regenerated the sub-sample
s,
# this time with 500 observations per sample.
set.seed(2345)

numSubSamp <- 20 # number of sub samples to generate
subSampSize <- 1000 # number of observations in each sub sample

# generate list of subsamples to be used for GMM
subSampList <- list()
for (i in 1:numSubSamp) {
  subSampList[[i]] <- SubSamp(givenData = train, sampSize = subSampSize)
}

ARI.gmm.train <- double(numSubSamp)
counter <- 1
for(ss in subSampList){
  gmm.train <- Mclust(dplyr::select(ss, -y), centers = 10)
  ARI.gmm.train[counter] <- adjustedRandIndex(gmm.train$classification, ss$y)
  counter <- counter + 1
}
ARI.gmm.train.val <- mean(ARI.gmm.train)
ARI.gmm.train.val
```

```r
#=============== APPLYING GMM TO TEST DATA =====================================
#==============================================================================

# Generate list of subsample from the test data
subSampList <- list()
for (i in 1:numSubSamp) {
  subSampList[[i]] <- SubSamp(givenData = test, sampSize = subSampSize)
}

ARI.gmm.test <- double(numSubSamp)
counter <- 1
for(ss in subSampList){
  gmm.test <- Mclust(dplyr::select(ss, -y), centers = 10)
  ARI.gmm.test[counter] <- adjustedRandIndex(gmm.test$classification, ss$y)
  counter <- counter + 1
}
ARI.gmm.test.val <- mean(ARI.gmm.test)
ARI.gmm.test.val
```

3. How long did this assignment take you? (5 pts)



Date	Times		Day Total
Wed, 12/04	5:00 pm – 9:00 pm		4.0 HRS
Thu, 12/05	9:45 am – 11:00 am	2:30 pm – 8:30 pm	7.25 HRS
			TOTAL: 11.25 HOURS

4. Type up homework solutions (5 pts)