CS 6375 - ASSIGNMENT 4

Please read the instructions below before starting the assignment.

- This assignment has three parts. The first part requires you to answer theoretical questions, while the second and third parts ask involve working with supervised learning problems.

- Please create three different folders, named partI, partII, and partIII, and keep your files in the respective folders.

- For the second part, you are free to use any pre-processing and loading library, but not a library that computes K-means.

- In the third part, you are free to use any image processing or machine learning library.

- For each of the code folders, please include a README file indicating how to compile and run your code. Also, mention clearly which packages/libraries you have used.

- You should use a cover sheet, which can be downloaded at: http://www.utdallas.edu/~axn112530/cs6375/CS6375_CoverPage.docx

- You are allowed to work in pairs i.e. a group of two students is allowed. Please write the names of the group members on the cover page.

- You have a total of 4 free late days for the entire semester. You can use at most 2 days for any one assignment. After that, there will be a penalty of 10% for each late day. **The submission for this assignment will be closed 2 days after the due date.**

- Please ask all questions through Piazza, and not through email.

PART I

(10 points)
1. Consider a regression problem of trying to estimate the function f: X -> y where X is a vector of feature attributes and y is a continuous real valued output variable. You would like to use a bagging model where you first create M bootstrap samples and then use them to create M different models – h_1, h_2, …, h_M. You can assume that all models are of the same type.

The error for each of the models would be described as:

$$\epsilon_i(x) = f(x) - h_i(x)$$

where x is the input data and h_i is the model created using i^{th} bootstrap sample

The expected value of the squared error for any of the models will be defined as:

$$E(\varepsilon_i(x)^2) = E[\big(f(x) - h_i(x)\big)^2]$$

The average value of the expected squared error for each of the models acting individually is defined as:

$$E_{avg} = \frac{1}{M}\sum_{i=1}^{M} E(\varepsilon_i(x)^2)$$

Now, you decide to aggregate the models using a committee approach as follows:

$$h_{agg}(x) = \frac{1}{M}\sum_{i=1}^{M} h_i\,(x)$$

The error using the aggregated model is defined as:

$$E_{agg}(x) = E[\left\{\frac{1}{M}\sum_{i=1}^{M} h_i\,(x) - f(x)\right\}^2]$$

which can be simplified as:

$$E_{agg}(x) = E[\left\{\frac{1}{M}\sum_{i=1}^{M} \epsilon_i\,(x)\right\}^2]$$

where we used the value of ε_i is defined above.

Prove that

$$E_{agg} = \frac{1}{M} E_{avg}$$

provided you make the following assumptions:

1. Each of the errors have a 0 mean

$$E(\epsilon_i(x)) = 0 \text{ for all i}$$

2. Errors are uncorrelated

$$E\left(\epsilon_i(x)\epsilon_j(x)\right) = 0 \text{ for all i} \neq \text{j}$$

(10 points)

2. Jensen's inequality states that for any *convex* function *f*:

$$f\left(\sum_{i=1}^{M} \lambda_i x_i\right) \leq \sum_{i=1}^{M} \lambda_i f(x_i)$$

In question 1, we had assumed that each of the errors are uncorrelated i.e.

$$E\left(\epsilon_i(x)\epsilon_j(x)\right) = 0 \text{ for all i} \neq \text{j}$$

This is not really true, as the models are created using bootstrap samples and have correlation with each other. Now, let's remove that assumption. Show that using Jensen's inequality, it is still possible to prove that:

$$E_{agg} \leq E_{avg}$$

(10 points)
3. Deriving the training error for AdaBoost:
In class, we discussed the steps of Adaboost algorithm. Recall that the final hypothesis for a Boolean classification problem at the end of T iterations is given by:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

The above equation says that the final hypothesis is the weighted hypothesis generated at the end of each individual step.
Also recall that the weight for the point i at step t+1 is given by:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times e^{-\alpha_t h_t(i) y(i)}$$

where:
$D_t(i)$ is the normalized weight of point i in step t
$h_t(i)$ is the hypothesis (prediction) at step t for point i
α_t is the final "voting power" of hypothesis h_t
$y(i)$ is the true label for point i
Z_t is the normalization factor at step t (it ensures that the weights sum up to 1.0)

Note that at step 1, the points have equal weight

$$D_1 = \frac{1}{N}$$

where N is the total number of data points.
At each of the steps, the total error of h_t will be defined as $\varepsilon_t = \frac{1}{2} - \gamma_t$, which is a way of saying that the error will be better than 50% by a value γ_t.

Prove that at the end of T steps, the overall training error will be bounded by:

$$\exp(-2\sum_{i=1}^{T} \gamma_t^2)$$

That is, the overall training error of the hypothesis H will be less than or equal to the amount indicated above.

Part II (40 points)

Tweets Clustering using k-means

Twitter provides a service for posting short messages. In practice, many of the tweets are very similar to each other and can be clustered together. By clustering similar tweets together, we can generate a more concise and organized representation of the raw tweets, which will be very useful for many Twitter-based applications (e.g., truth discovery, trend analysis, search ranking, etc.)

In this assignment, you will learn how to cluster tweets by utilizing Jaccard Distance metric and K-means clustering algorithm.

Objectives:

- Compute the similarity between tweets using the Jaccard Distance metric.
- Cluster tweets using the K-means clustering algorithm.

Introduction to Jaccard Distance:
The Jaccard distance, which measures dissimilarity between two sample sets (A and B). It is defined as the difference of the sizes of the union and the intersection of two sets divided by the size of the union of the sets.

$$Dist(A,B) = 1 - \frac{|A \bigcap B|}{|A \bigcup B|} = \frac{|A \bigcup B| - |A \bigcap B|}{|A \bigcup B|}$$

For example, consider the following tweets:

Tweet A: the long march

Tweet B: ides of march

|A ∩ B | = 1 and |A U B | = 5, therefore the distance is 1 – (1/5)

In this assignment, a tweet can be considered as an unordered set of words such as {a,b,c}. By "unordered", we mean that {a,b,c}={b,a,c}={a,c,b}=...
A Jaccard Distance Dist(A, B) between tweet A and B has the following properties:

- It is small if tweet A and B are similar.
- It is large if they are not similar.
- It is 0 if they are the same.
- It is 1 if they are completely different (i.e., no overlapping words).

Here is the reference for more details about Jaccard Distance:
http://en.wikipedia.org/wiki/Jaccard_index

Hint: Note that the tweets do not have the numerical coordinates in Euclidean space, you might want to think of a sensible way to compute the "centroid" of a tweet cluster. *This could be the tweet having minimum distance to all of the other tweets in a cluster.*

Exercise:
Implement the tweet clustering function using the Jaccard Distance metric and K-means clustering algorithm to cluster redundant/repeated tweets into the same cluster. You are free to use any language or package, but it should be clearly mentioned in your report.

Note that while the K-means algorithm is proved to converge, the algorithm is sensitive to the k initial selected cluster centroids (i.e., seeds) and the clustering result is not necessarily optimal on a random selection of seeds. In this assignment, we provide you with a list of K initial centroids that have been tested to generate good results.

Inputs to your K-means Algorithm:

(1) The number of clusters K (default to K=25).

(2) A real world dataset sampled from Twitter during the Boston Marathon Bombing event in April 2013 that contains 251 tweets. The tweet dataset is in JSON format and can be downloaded from http://www.utdallas.edu/~axn112530/cs6375/unsupervised/Tweets.json

(3) The list of initial centroids can be downloaded from:
http://www.utdallas.edu/~axn112530/cs6375/unsupervised/InitialSeeds.txt

Note that each element in this list is the tweet ID (i.e., the id field in JSON format) of the tweet in the dataset.

How to run your code:

The TA should be able to run your code as follows:

tweets-k-means <numberOfClusters> <initialSeedsFile> <TweetsDataFile> <outputFile>

<numberOfClusters> is the number of Clusters and
<initialSeedsFile> is a text file containing value of the initial seed data.
<TweetsDataFile> is the data file containing Tweets in JSON format
<outputFile> is the output file explained below.

The default value for <numberOfClusters> should be 25 and for the <initialSeeds> to be the file provided to you containing number of seeds.

Output:

Your code should also output to a file called tweets-k-means-output.txt. It should contain following:

<cluster-id> <List of tweet ids separated by comma>

For example,
1 323906397735641088, 900906397735641088, ..

It should also contain the SSE value.

Validation:

The same method as described in part I will be used. The distance metric will be Jaccard.

What to Turn In for Part II :
(1) A result file that contains the clustering results. Each line represents a cluster. It is in the form of *cluster_id: a list of tweet IDs that belongs to this cluster*
(2) The source code in any of the following languages - Java, Python, C#, Ruby. Be sure to include a README file indicating how to build and compile your code.
(3) A report file that contains details on how to compile your code, and any other relevant details.
* Idea and data courtesy of Dong Wang, University of Notre Dame *

Part III (30 points)

1. Color Quantization

In this part, you will use unsupervised learning to reduce the number of color points in an image. This application is also referred to as *color quantization*. You are free to use either k-means or EM algorithm for the unsupervised learning. You are also free to use any suitable library or package, but you have to clearly specify the name and installation instructions in the README file.

Below are some resources for learning more about color quantization:

- https://en.wikipedia.org/wiki/Color_quantization
- https://lmcaraig.com/color-quantization-using-k-means/
- https://appliedmachinelearning.blog/2017/03/08/image-compression-using-k-means-clustering/

Input:

You are given a set of 5 images that you can download from:

http://www.utdallas.edu/~axn112530/cs6375/unsupervised/images

You can select any 3 out of these and run the color quantization algorithm on them.

Parameters:

It is up to you to select the best value of the number of clusters and any other parameters for the algorithm. You should try at least 3 of k values for each image.

Output:

You should include the output of the 3 images i.e. quantized images for any **one value of k**. Create a folder called "quantizedImages" and place the images there.

How to run:

Please include instructions on how to run your code in the report file. Do not use hard-coded paths or parameters.

Report File:

Please include a report file indicating which language/tools/packages you used, and results of your experiments in the following format:

Image	Original File Size	Value of k used	Image Quality	Image Size
Image 1	500K	3	Appears Blurred	200K
Image 2	600K	6	Looks almost like original	300K
….				

What to Turn In for Part III-1:

(1) Source code in any language of your choice. Be sure to include a README file indicating how to build and compile your code.
(2) 3 examples of quantized images as output (in a separate folder)
(3) Report containing details mentioned above.

2. PCA for Images

In the second part of this section, you will use Principal Component Analysis to compress the same images as part one above. As before, you are free to use any language, library, and package. Just be sure to specify it in the README file.

Some resources for learning about PCA are below:
- http://people.ciirc.cvut.cz/~hlavac/TeachPresEn/11ImageProc/15PCA.pdf
- http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1679-45082012000200004
- https://www.kaggle.com/marnixk/image-compression-with-pca-from-scratch-math

Input:

You will use the same 3 images that you used in part 1 of this section.

Parameters:

It is up to you to select the best value of the number of clusters and any other parameters for the algorithm. You should try at least 3 different values of the number of principal components for each image.

Output:

You should include the output of the 3 images i.e. compressed images for any **one value of number of principal components**. Create a folder called "compressedImages" and place the images there.

How to run:

Please include instructions on how to run your code in the report file. Do not use hard-coded paths or parameters.

Report File:

Please include a report file indicating which language/tools/packages you used, and results of your experiments in the following format:

Image	Original File Size	Value of Principal Components	Image Quality	Image Size
Image 1	500K	2	Hazy	200K
Image 2	600K	4	Looks almost like original	300K
….				

What to Turn In for Part III-2:

(1) Source code in any language of your choice. Be sure to include a README file indicating how to build and compile your code.
(2) 3 examples of compressed images as output (in a separate folder)
(3) Report containing details mentioned above.