CS550 **Problem Set 2**
Spring 2018 **Due:** Mar 18, 2018, 11:59 p.m.

This homework has a total of 80 points + 20 bonus points, it will be rescaled to 10 points + 2.5 bonus points eventually.

Submission instructions: These questions require thought but do not require long answers. Please be as concise as possible. You should submit your answers as a writeup in PDF format, for those questions that require coding, write your code for a question in a single source code file, and name the file as the question number (e.g., question_1.java or question_1.py). Finally, put your PDF answer file and all the code files in a folder named as your NetID (e.g., ab123), compress the folder as a zip file (e.g., ab123.zip), and submit the zip file via Sakai. For the answer writeup PDF file, we have provided both a word template and a latex template for you, after you finished the writing, save the file as a PDF file, and submit both the original file (word or latex) and the PDF file.

1. **Singular Value Decomposition [40pt]**

   In this problem we will explore the relationship between two of the most popular dimensionality reduction techniques, SVD and PCA at a basic conceptual level. Before we proceed with the question itself, let us briefly recap the SVD and PCA techniques and a few important observations:

   • First, recall that the eigenvalue decomposition of a *real, symmetric*, and *square* matrix B (of size $d \times d$) can be written as the following product:

   $$B = Q\Lambda Q^T$$

   where $\Lambda = diag(\lambda_1, \cdots, \lambda_d)$ contains the eigenvalues of B (which are always real) along its main diagonal and Q is an orthogonal matrix containing the eigenvectors of B as its columns.

   • Principal Component Analysis (PCA): Given a data matrix M (of size $p \times q$), PCA involves the computation of the eigenvectors of MM^T or M^TM. The matrix of these eigenvectors can be thought of as a rigid rotation in a high dimensional space. When you apply this transformation to the original data, the axis corresponding to the principal eigenvector is the one along which the points are most spread out. More precisely, this axis is the one along which the variance of the data is maximized. Put another way, the points can best be viewed as lying along this axis, with small deviations from this axis. Likewise, the axis corresponding to the second eigenvector (the eigenvector corresponding to the second-largest eigenvalue) is the axis along which the variance of distances from the first axis is greatest, and so on.

   • Singular Value Decomposition (SVD): SVD involves the decomposition of a data matrix M (of size $p \times q$) into a product: $U\Sigma V^T$ where U (of size $p \times r$) and V (of size $q \times r$) are column-orthonormal matrices[1] and Σ (of size $r \times r$) is a diagonal matrix. The entries

[1] A matrix $U \in \mathbb{R}^{p \times q}$ is column-orthonormal if and only if $U^TU = I$ where I denotes the identity matrix

along the diagonal of Σ are referred to as singular values of M. The key to understanding what SVD offers is in viewing the r columns of U, Σ, and V as representing concepts that are hidden in the original matrix M.

For answering the questions below, let us define a matrix M (of size $p \times q$) and let us assume this matrix corresponds to a dataset with p data points and q dimensions.

(a) [5pt] Are the matrices MM^T and M^TM symmetric, square and real? Explain.

(b) [5pt] Prove that the eigenvalues of MM^T are the same as that of M^TM. Are their eigenvectors the same?

(c) [5pt] Given that we now understand certain properties of M^TM, write an expression for M^TM in terms of Q, Q^T and Λ where $\Lambda = diag(\lambda_1, ..., \lambda_d)$ contains the eigenvalues of M^TM along its main diagonal and Q is an orthogonal matrix containing the eigenvectors of M^TM as its columns. (*Hint: Check the definition of eigenvalue decomposition provided in the beginning of the question to see if it is applicable.*)

(d) [5pt] SVD decomposes the matrix M into the product $U\Sigma V^T$ where U and V are column-orthonormal and Σ is a diagonal matrix. Given that $M = U\Sigma V^T$, write a simplified expression for M^TM in terms of V, V^T and Σ.

(e) In this question, let us experimentally test if SVD decomposition of M actually provides us the eigenvectors (PCA dimensions) of M^TM. We strongly recommend students to use Python and suggested functions for this exercise[2].

Initialize matrix M as follows:

$$M = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 4 \\ 4 & 3 \end{bmatrix}$$

• (e)(a) [5pt] Compute the SVD of M (*Use scipy.linalg.svd function in Python and set the argument* ***full_matrices*** *to False*). The function returns values corresponding to U, Σ and V^T. What are the values returned for U, Σ and V^T? Note: Make sure that the first element of the returned array Σ has a greater value than the second element.

• (e)(b) [5pt] Compute the eigenvalue decomposition of M^TM (*Use scipy.linalg.eigh function in Python*). The function returns two parameters: a list of eigenvalues (let us call this list *Evals*) and a matrix whose columns correspond to the eigenvectors of the respective eigenvalues (let us call this matrix *Evecs*). Sort the list *Evals* in descending order such that the largest eigenvalue appears first in the list. Also, re-arrange the columns in *Evecs* such that the eigenvector corresponding to the largest eigenvalue appears in the first column of *Evecs*. What are the values of *Evals* and *Evecs* (after the sorting and re-arranging process)?

• (e)(c) [5pt] Based on the experiment and your derivations in part (c) and (d), do you see any correspondence between V produced by SVD and the matrix of eigenvectors *Evecs*

[2]Other implementations of SVD and PCA might give slightly different results. Besides, you will just need fewer than five python commands to answer this entire question

(after the sorting and re-arranging process) produced by eigenvalue decomposition? If so, what is it? (*Note: The function scipy.linalg.svd returns V^T, not V.*)

• (e)(d) [5pt] Based on the experiment and the expressions obtained in part (c) and part (d) for M^TM, what is the relationship (if any) between the eigenvalues of M^TM and the singular values of M? Explain.

Note: The entries along the diagonal of Σ (part (d)) are referred to as singular values of M. The eigenvalues of M^TM are captured by the diagonal elements in Λ (part (c)).

What to submit:

(i) Written solutions to questions 1(a) to 1(e) with explanations wherever required.

(ii) Include the code as a single Python file as question_1.py.

2. **Dead ends in PageRank computations [20pt]**

Let the *matrix of the Web* M be an n-by-n matrix, where n is the number of Web pages. The entry m_{ij} in row i and column j is 0, unless there is an arc from node (page) j to node i. In that case, the value of m_{ij} is $1/k$, where k is the number of arcs (links) out of node j. Notice that if node j has $k > 0$ arcs out, then column j has k values of $1/k$ and the rest 0's. If node j is a dead end (i.e., it has zero arcs out), then column j is all 0's.

Let $\mathbf{r} = [r_1, r_2, ..., r_n]^T$ be (an estimate of) the PageRank vector; that is, r_i is the estimate of the PageRank of node i. Define $w(\mathbf{r})$ to be the sum of the components of $\mathbf{r}$; that is $w(\mathbf{r}) = \sum_{j=1}^{n} r_i$.

In one iteration of the PageRank algorithm, we compute the next estimate $\mathbf{r'}$ of the PageRank as: $\mathbf{r'} = M\mathbf{r}$. Specifically, for each i we compute $r'_i = \sum_{j=1}^{n} M_{ij} r_j$.

(a) [5pt] Suppose the Web has no dead ends. Prove that $w(\mathbf{r'}) = w(\mathbf{r})$.

(b) [5pt] Suppose there are still no dead ends, but we use a teleportation probability of $1 - \beta$, where $0 < \beta < 1$. The expression for the next estimate of r_i becomes $r'_i = \beta \sum_{j=1}^{n} M_{ij} r_j + (1 - \beta)/n$. Under what circumstances will $w(\mathbf{r'}) = w(\mathbf{r})$? Prove your conclusion.

(c) Now, let us assume a teleportation probability of $1 - \beta$ in addition to the fact that there are one or more dead ends. Call a node "dead" if it is a dead end and "live" if not. Assume $w(\mathbf{r}) = 1$. At each iteration, we will distribute equally to each node the sum of:

- 1. $(1 - \beta)r_j$ if node j is live.
- 2. r_j if node j is dead.

• (c)(a) [5pt] Write the equation for r'_i in terms of β, M, and $\mathbf{r}$.

• (c)(b) [5pt] Then, prove that $w(\mathbf{r'})$ is also 1.

What to submit:

(i) Proof of 2(a);

(ii) Condition for $w(\mathbf{r'}) = w(\mathbf{r})$ and Proof of 2(b);

(iii) Equation for r'_i and Proof of 2(c).

3. **Implementing PageRank [20pt]**

In this problem, you will learn how to implement the PageRank algorithm. You will be experimenting with the provided graph (assume graph has no dead-ends), which is stores in the `graph.txt` file.

It has $n = 100$ nodes (numbered $1, 2, ..., 100$), and $m = 1024$ edges, 100 of which form a directed cycle (through all the nodes) which ensures that the graph is connected. It is easy to see that the existence of such a cycle ensures that there are no dead ends in the graph. There may be multiple edges between a pair of nodes, your program should handle these instead of ignoring them. The first column in `graph.txt` refers to the source node, and the second column refers to the destination node.

Assume the directed graph $G = (V, E)$ has n nodes (numbered $1, 2, ..., n$) and m edges, all nodes have positive out-degree, and $M = [M_{ji}]_{n \times n}$ is an $n \times n$ matrix as defined in class such that for any $i, j \in [1, n]$:

$$M_{ji} = \begin{cases} \frac{1}{deg(i)}, \ if \ (i \to j) \in E \\ 0 \ otherwise \end{cases}$$

Here, $deg(i)$ is the number of outgoing edges of node i in G. By the definition of PageRank, assuming $1 - \beta$ to be the teleport probability, and denoting the PageRank vector by the column vector $\mathbf{r}$, we have the following equation:

$$\mathbf{r} = \frac{1 - \beta}{n}\mathbf{1} + \beta M\mathbf{r}$$

where $\mathbf{1}$ is the $n \times 1$ vector with all entries equal to 1.

Based on this equation, the iterative procedure to compute PageRank works as follows:

1. Initialize: $\mathbf{r}^{(0)} = \frac{1}{n}\mathbf{1}$;
2. For i from 1 to k, iterate: $\mathbf{r}^{(i)} = \frac{1-\beta}{n}\mathbf{1} + \beta M\mathbf{r}^{(i-1)}$.

Run the aforementioned iterative process for 40 iterations (assuming $\beta = 0.8$) and obtain the PageRank vector $\mathbf{r}$. Compute the following:

(a) [10pt] List the top 5 node IDs with the highest PageRank scores.

(b) [10pt] List the bottom 5 node IDs with the lowest PageRank scores.

What to submit:

(i) List 5 node IDs with the highest and least PageRank scores in your writeup.

(ii) Include your code as a single source code file such as question_2.py.

4. **k-Means on MapReduce [20pt bonus]**

Note: This problem requires substantial computing time. Don't start it at the last minute.

This problem will help you understand the nitty gritty details of implementing clustering algorithms on Hadoop. In addition, this problem will also help you understand the impact

of using various initialization strategies in practice. Let us say we have a set $\mathcal{X}$ of n data points in the d-dimensional space $\mathbb{R}^d$. Given the number of clusters k and the set of k centroids $\mathcal{C}$, we now proceed to define the distance metric and the corresponding cost function that we minimize.

Euclidean distance: Given two points A and B in d dimensional space such that $A = [a_1, a_2...a_d]$ and $B = [b_1, b_2...b_d]$, the Euclidean distance between A and B is defined as:

$$\|a - b\| = \sqrt{\sum_{i=1}^{d}(a_i - b_i)^2}$$

The corresponding cost function ϕ that is minimized when we assign points to clusters using the Euclidean distance metric is given by:

$$\phi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} \|x - c\|^2$$

Iterative k-Means Algorithm: We learned the basic k-Means algorithm in class which is as follows: k centroids are initialized, each point is assigned to the nearest centroid and the centroids are recomputed based on the assignments of points to clusters. In practice, the above steps are run for several iterations. We present the resulting iterative version of k-Means in Algorithm 1.

Algorithm 1 Iterative k-Means Algorithm

```
1: procedure ITERATIVE k-MEANS
2:     Select k points as initial centroids of the k clusters.
3:     for iterations := 1 to MAX_ITER do
4:         for each point p in the dataset do
5:             Assign point p to the cluster with the closest centroid
6:         end for
7:         for each cluster c do
8:             Recompute the centroid of c as the mean of all the data points assigned to c
9:         end for
10:    end for
11: end procedure
```

Iterative k-Means clustering on Hadoop: Implement iterative k-means using MapReduce where a single step of MapReduce completes one iteration of the k-means algorithm. So, to run k-means for i iterations, you will have to run a sequence of i MapReduce jobs.

Please use our provided dataset `hw2-q4-kmeans.zip` for this problem. The zip has 4 files:

- 1. `data.txt` contains the dataset which has 4601 rows and 58 columns. Each row is a document represented as a 58 dimensional vector of features. Each component in the vector represents the importance of a word in the document.
- 2. `c1.txt` contains k initial cluster centroids. These centroids were chosen by selecting $k = 10$ random points from the input data.

- 3. `c2.txt` contains initial cluster centroids which are as far apart as possible. (You can do this by choosing the 1st centroid c1 randomly, and then finding the point c2 that is farthest from c1, then selecting c3 which is farthest from c1 and c2, and so on).
- 4. `vocab.txt` is the vocabulary document that contains the words, this file is only for your reference and you do not need it to do the experiment.

Set number of iterations (`MAX_ITER`) to 20 and number of clusters k to 10 for all the experiments carried out in this question.

*Hint about **job chaining**:*

We need to run a sequence of Hadoop jobs where the output of one job will be the input for the next one. There are multiple ways to do this and you are free to use any method you are comfortable with. One simple way to handle such a multistage job is to configure the output path of the first job to be the input path of the second and so on.

The following pseudo code demonstrates job chaining.

```
var inputDir
var outputDir
var centroidDir


for i in number-of-iterations (
  Configure job here with all params
  Set job input directory = inputDir
  Set job output directory = outputDir + i
  Run job
  centroidDir = outputDir + i
)
```

You will also need to share the location of the centroid file with the mapper. There are many ways to do this and you can use any method you find suitable. One way is to use the Hadoop Configuration object. You can set it as a property in the Configuration object and retrieve the property value in the Mapper setup function. For more details see :

- http://hadoop.apache.org/docs/r1.0.4/api/org/apache/hadoop/conf/Configuration.html#set(java.lang.String,java.lang.String)
- http://hadoop.apache.org/docs/r1.0.4/api/org/apache/hadoop/conf/Configuration.html#get(java.lang.String)
- http://hadoop.apache.org/docs/r1.0.4/api/org/apache/hadoop/mapreduce/Mapper.html#setup(org.apache.hadoop.mapreduce.Mapper.Context)

(a) [10pt bonus] Using the Euclidean distance as the distance measure, compute the cost function $\phi(i)$ for every iteration i. This means that, for your first MapReduce job iteration, you'll be computing the cost function using the initial centroids located in one of the two text files. Run the k-means on `data.txt` using `c1.txt` and `c2.txt`. Generate

a graph where you plot the cost function $\phi(i)$ as a function of the number of iterations $i = 1, 2...20$ for `c1.txt` and also for `c2.txt`.

(Hint: Note that you do not need to write a separate MapReduce job to compute $\phi(i)$. You can just incorporate the computation of $\phi(i)$ into the Mapper/Reducer.)

(b) [10pt bonus] What is the percentage change in cost after 10 iterations of the k-Means algorithm when the cluster centroids are initialized using `c1.txt` vs. `c2.txt`? Is random initialization of k-means using `c1.txt` better than initialization using `c2.txt` in terms of cost $\phi(i)$? Explain your reasoning.

What to submit:

(i) A plot of cost vs. iteration for two initialization strategies for 4(a)

(ii) Percentage improvement values and your explanation for 4(b)

(iii) The code named as question_4.py or question_4.java.