CSE 547: Machine Learning for Big Data **Spring 2019**

Problem Set 2

Please read the homework submission policies.

1 Principal Component Analysis and Reconstruction (25 points)

Let's do PCA and reconstruct pictures of faces in the PCA basis. As we will be making many visualizations of images, plot these images in reasonable way (e.g. make the images smaller).

(a) Matrix Algebra Review [5 pts]

The trace of a square matrix M, denoted, by $Tr(M)$ is defined as the sum of the diagonal entries of M

1. **[2 pts]** Show that $Tr(AB^{\mathrm{T}}) = Tr(B^{\mathrm{T}}A)$ for two matrices A and B of size $n \times d$.
2. **[3 pts]** Now we prove a few claims that will be helpful in the next problem. Define $\mathbf{\Sigma} := \frac{1}{n}\mathbf{X}^{\mathrm{T}}\mathbf{X}$, where $\mathbf{X}$ is the $n \times d$ data matrix. Let $\mathbf{X}_i$ be the i-th row of $\mathbf{X}$ (so $\mathbf{X}_i$ is a d-dimensional vector). Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ be the eigenvalues of $\mathbf{\Sigma}$. Show the following two properties:

$$Tr(\Sigma) = \sum_{i=1}^{d} \lambda_i \tag{1}$$

$$Tr(\Sigma) = \frac{1}{n}\sum_{i=1}^{n} \|X_i\|_2^2 \tag{2}$$

(b) PCA [6 pts]

For this question we will use the dataset `faces.csv` from `q1/data` (adapted from the Extended Yale Face Database B), which is composed of facial pictures of 38 individuals under 64 different lighting conditions such as lit from the top, front, and side (some lighting conditions are missing for some people). In total, there are 2414 images, where each image is 96 pixels high and 84 pixels wide (for a total of 8064 pixels per image). Each row in the dataset is a single image flattened into a vector in a column-major order. The images are in grayscale, so each pixel is represented by a single real number between 0 (black) and 1 (white).

Define Σ, a 8064×8064 matrix, as follows:

$$\Sigma = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^{\mathrm{T}}$$

where the x_i's are points in our dataset as **column** vectors and $n = 2414$ is the number of points in the dataset. Now compute the top 50 PCA dimensions; these are the 50 dimensions which best reconstruct the data.

We will be implementing PCA using eigendecomposition. Thus, you may use library functions for obtaining the eigenvalues and eigenvectors of a matrix (e.g. `numpy.linalg.eig` in Python and `eig` or `eigs` in MATLAB). You are should **not** use functions which directly compute the principal components of a matrix. Please ask on Piazza regarding other (non-basic) linear algebra functions you may be interested in using.

1. **[2 pts]** What are the eigenvalues λ_1, λ_2, λ_{10}, λ_{30}, and λ_{50}? Also, what is the sum of eigenvalues $\sum_{i=1}^{d} \lambda_i$? *(Hint: use the answer from the previous question).*
2. **[3 pts]** It is straight forward to see that the fractional reconstruction error of using the top k out of d directions is $1 - \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$. Plot this fractional reconstruction error for the each of the first 50 values of k (i.e. k from 1 to 50). So the X-axis is k and the Y-axis is the fractional reconstruction error. Make sure your plots are legible at reasonable zoom in your write-up.
3. **[1 pt]** What does the first eigenvalue capture, and why do you think λ_1 is much larger than the other eigenvalues?

(c) Visualization of the Eigen-Directions [6 pts]

Now let us get a sense of the what the top PCA directions are capturing (recall these are the directions which capture the most variance).

1. **[3 pts]** Display the first 10 eigenvectors as images.

   Hint 1: If the images appear like random lines, try reshaping differently and then transposing.

   Hint 2: The eigenvectors you obtain are normalized to have length 1 in the L_2 norm, thus their entries are extremely small. To avoid getting all-black images, make sure to re-normalize the image. in Python, `matplotlib.pyplot.imshow` does this by default. If you are using `imshow` in MATLAB, you should pass an extra empty vector as an argument, e.g. "`imshow(im, []);`".
2. **[3 pt]** Provide a brief interpretation of what you think each eigenvalue captures.

(d) Visualization and Reconstruction [8 pts]

1. **[6 pt]** We will now observe the reconstruction using PCA on a sample of images composed of the following images:

   (a) image 1 (row 0).

   (b) image 24 (row 23).

   (c) image 65 (row 64).

   (d) image 68 (row 67).

   (e) image 257 (row 256).

   For each of these images, plot the original image and plot the projection of the image onto the top k eigenvectors of Σ for $k = 1, 2, 5, 10, 50$. In particular, if U is the $d \times k$ matrix of the top k eigenvectors, the reconstruction matrix will be UU^{T}.

   Specifically, you should obtain a 5×6 table where in each row corresponds to a single image, the first cell in every row is the original image and the following cells are the reconstructions of the image with the 5 different values of k.

   Hint: In this problem, we are observing (partial) combinations of images that already have a meaningful scale. Therefore, we want to keep the scale of the results and not re-normalize the images (in contrast to part (c)). If you are using `matplotlib.pyplot.imshow` in Python, you should pass the additional arguments `vmin=0, vmax=1`, e.g. `imshow(im, vmin=0, vmax=1)`. If you are using `imshow` in MATLAB, the default is not re-normalizing, so you should **not** pass additional arguments, e.g. simply use "`imshow(im);`".

2. **[2 pt]** Provide a brief interpretation, in terms of your perceptions of the quality of these reconstructions. Explain the results in light of your answers for part (c).

What to submit:

(i) Proofs for the claims in 1(a).

(ii) The values of λ_1, λ_2, λ_{10}, $\lambda_{30}, \lambda_{50}$ and $\sum_i \lambda_i$, a plot of the reconstruction error vs. k and an explanation for 1(b).

(iii) Plots of the first 10 eigenvectors as images and their interpretation [1(c)].

(iv) Table of original and reconstructed images and their interpretation [1(d)].

(v) Upload the code to Gradescope.

2 k-means on Spark (20 points)

Note: This problem requires substantial computing time. Don't start it at the last minute. Also, you should **not** use the Spark MLlib clustering library for this problem.

This problem will help you understand the nitty gritty details of implementing clustering algorithms on Spark. In addition, this problem will also help you understand the impact of using various distance metrics and 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 various distance metrics and the corresponding cost functions that they minimize.

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

$$\|a - b\|_2 = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2} \tag{3}$$

The corresponding cost ϕ that is associated with a given set of centroids $\mathcal{C}$ when we assign points to clusters using the Euclidean distance metric is given by:

$$\phi(\mathcal{C}) = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} \|x - c\|_2 \tag{4}$$

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

$$\|a - b\|_1 = \sum_{i=1}^{d} |a_i - b_i| \tag{5}$$

The corresponding cost ψ that is associated with a given set of centroids $\mathcal{C}$ when we assign points to clusters using the Manhattan distance metric is given by:

$$\psi(\mathcal{C}) = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} \|x - c\|_1 \tag{6}$$

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.

Note: What we are implementing in this problem is, in fact, not the correct adaptation of k-means to the L_1 norm. The correct way to implement k-means for L_1 would be to define

Algorithm 1 Iterative k-Means Algorithm

```
1: procedure ITERATIVE k-MEANS
2:     Select k points as initial centroids of the k clusters.
3:     for iteration := 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:         Calculate the cost for this iteration.
8:         for each cluster c do
9:             Recompute the centroid of c as the mean of all the data points assigned to c
10:        end for
11:    end for
12: end procedure
```

the centroids of the clusters as their **medians**, rather than their means. Analogously, this algorithm is called "k-medians".

Iterative k-Means clustering on Spark: Implement iterative k-means using Spark. Please use the dataset from `q2/data` within the bundle for this problem.

The folder has 3 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 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).

Set the number of iterations (`MAX_ITER`) to 20 and the number of clusters k to 10 for all the experiments carried out in this question. Your driver program should ensure that the correct amount of iterations are run.

(a) Exploring initialization strategies with Euclidean distance [10 pts]

1. **[5 pts]** Using the Euclidean distance (refer to Equation 3) as the distance measure, compute the cost function $\phi(i)$ (refer to Equation 4) for every iteration i [1]. This means that, for your first 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`

[1] We are overloading the notation here. The more precise way to express $\phi(i)$ would be $\phi(\mathcal{C}_i)$, where $\mathcal{C}_i$ is the set of centroids at iteration i

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,. . . ,20 for `c1.txt` and also for `c2.txt`.

Hint: Note that you do not need to write a separate Spark job to compute $\phi(i)$. You should be able to calculate costs while partitioning points into clusters.

2. **[5 pts]** By how many percent does the cost change after 10 iterations of k-means with the Euclidean distance when the cluster centroids are initialized using `c1.txt` and when using `c2.txt`? Is the random initialization of k-means using `c1.txt` better than the initialization using `c2.txt` in terms of cost $\phi(i)$? Explain your reasoning.

(b) Exploring initialization strategies with Manhattan distance [10 pts]

1. **[5 pts]** Using the Manhattan distance metric (refer to Equation 5) as the distance measure, compute the cost function $\psi(i)$ (refer to Equation 6) for every iteration i [2]. This means that, for your first 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 $\psi(i)$ as a function of the number of iterations i=1,. . . ,20 for `c1.txt` and also for `c2.txt`.

   Hint: This problem can be solved in a similar manner to that of part (a).

2. **[5 pts]** By how many percent does the cost change after 10 iterations of k-means with the Manhattan distance when the cluster centroids are initialized using `c1.txt` and when using `c2.txt`? Is the random initialization of k-means using `c1.txt` better than the initialization using `c2.txt` in terms of cost $\psi(i)$? Explain your reasoning.

What to submit:

(i) Upload the code for 2(a) and 2(b) to Gradescope.

(ii) A plot of cost vs. iteration for two initialization strategies for 2(a).

(iii) Percentage of improvement values and your explanation for 2(a).

(iv) A plot of cost vs. iteration for two initialization strategies for 2(b).

(v) Percentage of improvement values and your explanation for 2(b).

3 Latent Features for Recommendations (30 points)

Warning: This problem requires substantial computing time (it can be a few hours on some systems). Don't start it at the last minute.

[2] Same as footnote 1

* * *

The goal of this problem is to implement the *Stochastic Gradient Descent* algorithm to build a Latent Factor Recommendation system. We can use it to recommend movies to users. We encourage you to read the slides of the lecture "Recommender Systems 2" again before attempting the problem.

Suppose we are given a matrix R of recommendations. The element R_{iu} of this matrix corresponds to the rating given by user u to item i. The size of R is $m \times n$, where m is the number of movies, and n the number of users.

Most of the elements of the matrix are unknown because each user can only rate a few movies.

Our goal is to find two matrices P and Q, such that $R \simeq QP^T$. The dimensions of Q are $m \times k$, and the dimensions of P are $n \times k$. k is a parameter of the algorithm.

We define the error as

$$E(R, P, Q) = \sum_{(i,u)\in \text{ratings}} (R_{iu} - q_i \cdot p_u)^2 + \lambda \left(\sum_{u=1}^{n} \|p_u\|_2^2 + \sum_{i=1}^{m} \|q_i\|_2^2 \right). \tag{7}$$

The $\sum_{(i,u)\in \text{ratings}}$ means that we sum only on the pairs (user, item) for which the user has rated the item, *i.e.* the (i, u) entry of the matrix R is known. q_i denotes the i^{th} row of the matrix Q (corresponding to an item), and p_u the u^{th} row of the matrix P (corresponding to a user u). λ is the regularization parameter. $\|\cdot\|_2$ is the L_2 norm and $\|p_u\|_2^2$ is square of the L_2 norm, *i.e.,* it is the sum of squares of elements of p_u.

(a) [10 points]

Let ε_{iu} denote the derivative of the error E with respect to R_{iu}. What is the expression for ε_{iu}? What are the update equations for q_i and p_u in the Stochastic Gradient Descent algorithm with learning rate η when processing the observation R_{iu}?

(b) [20 points]

Implement the algorithm. Read each entry of the matrix R from disk and update ε_{iu}, q_i and p_u for each entry.

To emphasize, you are not allowed to store the matrix R in memory. You have to read each element R_{iu} one at a time from disk and apply your update equations (to each element). Then, iterate until both q_i and p_u stop changing. Each iteration of the algorithm will read the whole file.

Choose $k = 20$, $\lambda = 0.1$ and number of iterations $= 40$. Find a good value for the learning rate η. Start with $\eta = 0.1$ (you should search for the best value you can). The error E

on the training set `ratings.train.txt` discussed below should be less than 65000 after 40 iterations.

Based on values of η, you may encounter the following cases:

- If η is too big, the error function can converge to a high value or may not monotonically decrease. It can even diverge and make the components of vectors p and q equal to ∞.
- If η is too small, the error function doesn't have time to significantly decrease and reach convergence. So, it can monotonically decrease but not converge *i.e.* it could have a high value after 40 iterations because it has not converged yet.

Use the dataset at `q3/data` within the bundle for this problem. It contains the following files:

- `ratings.train.txt`: This is the matrix R. Each entry is made of a user id, a movie id, and a rating.

Plot the value of the objective function E (defined in equation 7) on the training set as a function of the number of iterations. What value of η did you find?

You can use any programming language to implement this part, but Java, C/C++, and Python are recommended for speed. (In particular, Matlab can be rather slow reading from disk.) It should be possible to get a solution that takes on the order of minutes to run with these languages.

Hint: These hints will help you if you are not sure about how to proceed for certain steps of the algorithm, although you don't have to follow them if you have another method.

- *Initialization of P and Q: We would like q_i and p_u for all users u and items i such that $q_i \cdot p_u \in [0, 5]$. A good way to achieve that is to initialize all elements of P and Q to random values in $[0, \sqrt{5/k}]$.*
- *Update the equations: In each update, we update q_i using p_u and p_u using q_i. Compute the new values for q_i and p_u using the old values, and then update the vectors q_i and p_u.*
- *You should compute E at the end of a full iteration of training. Computing E in pieces during the iteration is incorrect since P and Q are still being updated.*

What to submit

(i) Equation for ε_{iu}. Update equations in the Stochastic Gradient Descent algorithm. [3(a)]

(ii) Value of η. Plot of E vs. number of iterations. Make sure your graph has a y-axis so that we can read the value of E. [3(b)]

(iii) Please upload all the code to Gradescope. [3(b)]

4 Recommendation Systems (25 points)

Consider a user-item bipartite graph where each edge in the graph between user U to item I, indicates that user U likes item I. We also represent the ratings matrix for this set of users and items as R, where each row in R corresponds to a user and each column corresponds to an item. If user i likes item j, then $R_{i,j} = 1$, otherwise $R_{i,j} = 0$. Also assume we have m users and n items, so matrix R is $m \times n$.

Let's define a matrix P, $m \times m$, as a diagonal matrix whose i-th diagonal element is the degree of user node i, *i.e.* the number of items that user i likes. Similarly, a matrix Q, $n \times n$, is a diagonal matrix whose i-th diagonal element is the degree of item node i or the number of users that liked item i. See figure below for an example.

(a) [4 points]

Define the non-normalized user similarity matrix $T = RR^T$. Explain the meaning of T_{ii} and T_{ij} $(i \neq j)$, in terms of bipartite graph structures (See Figure 1) (e.g. node degrees, path between nodes, etc.).

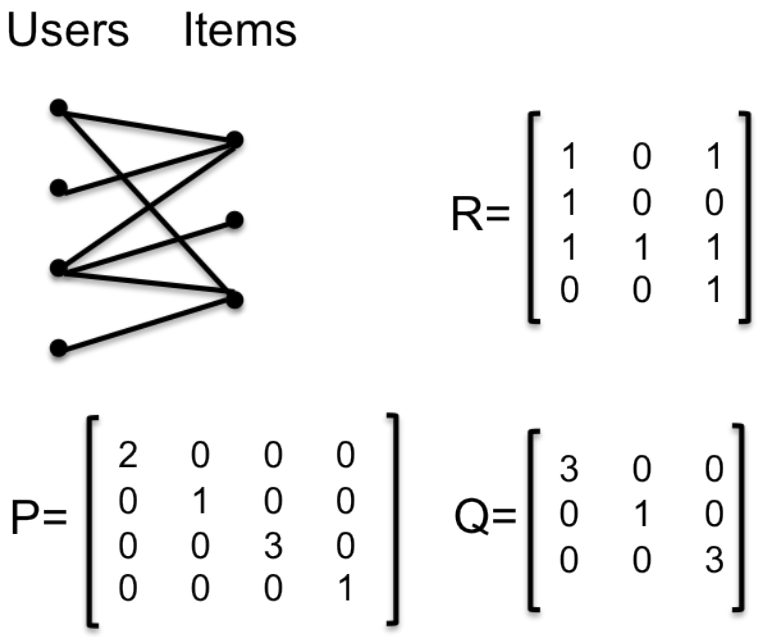

Figure 1: User-Item bipartite graph.

Cosine Similarity: Recall that the cosine similarity of two vectors u and v is defined as:

$$\textit{cos-sim}(u,v) = \frac{u \cdot v}{\|u\|\|v\|}$$

(b) [6 points]

Let's define the *item similarity matrix*, S_I, $n \times n$, such that the element in row i and column j is the cosine similarity of *item* i and *item* j which correspond to column i and column j of the matrix R. Show that $S_I = Q^{-1/2}R^TRQ^{-1/2}$, where $Q^{-1/2}$ is defined by $Q_{rc}^{-1/2} = 1/\sqrt{Q_{rc}}$ for all nonzero entries of the matrix, and 0 at all other positions.

Repeat the same question for *user similarity matrix*, S_U where the element in row i and column j is the cosine similarity of *user* i and *user* j which correspond to row i and row j of the matrix R. That is, your expression for S_U should also be in terms of some combination of R, P, and Q. Your answer should be an operation on the matrices, in particular you should not define each coefficient of S_U individually.

Your answer should show how you derived the expressions.

Note: To make the element-wise square root of a diagonal matrix, you may write it as matrix to the power of $\frac{1}{2}$.

(c) [5 points]

The recommendation method using user-user collaborative filtering for user u, can be described as follows: for all items s, compute $r_{u,s} = \Sigma_{x \in users}\,\text{cos-sim}(x,u) \cdot R_{xs}$ and recommend the k items for which $r_{u,s}$ is the largest.

Similarly, the recommendation method using item-item collaborative filtering for user u can be described as follows: for all items s, compute $r_{u,s} = \Sigma_{x \in items} R_{ux} \cdot \text{cos-sim}(x,s)$ and recommend the k items for which $r_{u,s}$ is the largest.

Let's define the recommendation matrix, Γ, $m \times n$, such that $\Gamma(i,j) = r_{i,j}$. Find Γ for both item-item and user-user collaborative filtering approaches, in terms of R, P and Q.

Hint: For the item-item case, $\Gamma = RQ^{-1/2}R^TRQ^{-1/2}$.

Your answer should show how you derived the expressions (even for the item-item case, where we give you the final expression).

(d) [10 points]

In this question you will apply these methods to a real dataset. The data contains information about TV shows. More precisely, for 9985 users and 563 popular TV shows, we know if a given user watched a given show over a 3 month period.

Use the dataset from `q4/data` within the bundle for this problem.

The folder contains:

- `user-shows.txt` This is the ratings matrix R, where each row corresponds to a user and each column corresponds to a TV show. $R_{ij} = 1$ if user i watched the show j over a period of three months. The columns are separated by a space.
- `shows.txt` This is a file containing the titles of the TV shows, in the same order as the columns of R.

We will compare the user-user and item-item collaborative filtering recommendations for the 500^{th} user of the dataset. Let's call him Alex.

In order to do so, we have erased the first 100 entries of Alex's row in the matrix, and replaced them by 0s. This means that we don't know which of the first 100 shows Alex has watched. Based on Alex's behaviour on the other shows, we will give Alex recommendations on the first 100 shows. We will then see if our recommendations match what Alex had in fact watched.

- Compute the matrices P and Q.
- Using the formulas found in part (c), compute Γ for the user-user collaborative filtering. Let S denote the set of the first 100 shows (the first 100 columns of the matrix). From all the TV shows in S, which are the five that have the highest similarity scores for Alex? What are their similarity scores? In case of ties between two shows, choose the one with smaller index. Do not write the index of the TV shows, write their names using the file `shows.txt`.
- Compute the matrix Γ for the movie-movie collaborative filtering. From all the TV shows in S, which are the five that have the highest similarity scores for Alex? In case of ties between two shows, choose the one with smaller index. Again, hand in the names of the shows and their similarity score.

What to submit:

(i) Interpretation of T_{ii} and T_{ij} [for 4(a)]

(ii) Expression of S_I and S_U in terms of R, P and Q and accompanying explanation [for 4(b)]

(iii) Expression of Γ in terms of R, P and Q and accompanying explanation [for 4(c)]

(iv) The answer to this question should include the followings: [for 4(d)]

- The five TV shows that have the highest similarity scores for Alex for the user-user collaborative filtering

- The five TV shows that have the highest similarity scores for Alex for the item-item collaborative filtering item-item collaborative filtering
- Upload the source code to Gradescope.