CS246: Mining Massive Data Sets **Winter 2018**

Problem Set 2

Due 11:59pm February 8, 2018

Only one late period is allowed for this homework (11:59pm 2/13).

General Instructions

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 via GradeScope and code via the Snap submission site.

Submitting writeup: Prepare answers to the homework questions into a single PDF file and submit it via http://gradescope.com. Make sure that the answer to each question is on a *separate page*. It is also important to tag your answers correctly on Gradescope. We will deduct 1 point for each incorrectly tagged subproblem. This means you can lose up to 12 points for incorrect tagging.

Submitting code: Upload your code at http://snap.stanford.edu/submit. Put all the code for a single question into a single file and upload it.

Questions

1 Singular Value Decomposition and Principal Component Analysis (20 points) [Praty, Heather, Dylan]

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:

$$B = Q\Lambda Q^{\mathrm{T}}$$

where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$ 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 $p \times k$) and V (of size $q \times k$) are column-orthonormal matrices[1] and Σ (of size $k \times k$) is a diagonal matrix. The entries 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) [3 points]

Are the matrices MM^T and M^TM symmetric, square and real? Explain.

★ **SOLUTION:** Yes, both the matrices are symmetric, square and real.

- Symmetric: $(MM^T)^T = MM^T$ and $(M^TM)^T = M^TM$
- Square: When you multiply a matrix of size $p \times q$ by its transpose of size $q \times p$, you end up with a square matrix of size $p \times p$.
- Real: Given that M is real, the product of M and its transpose will also be real.

(b) [5 points]

Prove that the eigenvalues of MM^T are the same as that of M^TM. Are their eigenvectors the same?

[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

★ **SOLUTION:** Let e denote the eigenvector of MM^T and let the corresponding eigen value be λ i.e $MM^T(e) = \lambda(e)$. Multiply both sides of equation by M^T and the result is $M^TMM^Te = M^T\lambda e$ which can be reduced to $M^TM(M^Te) = \lambda(M^Te)$. Therefore, the eigenvalue of $M^TM = \lambda$ too but the eigenvector is M^Te which is different from the eigenvector of MM^T which is e.

(c) [2 points]

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 $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_d)$ 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.

★ **SOLUTION:** $M^TM = Q\Lambda Q^T$

(d) [5 points]

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?

★ **SOLUTION:** $M^TM = (U\Sigma V^T)^TU\Sigma V^T$
$= V\Sigma^TU^TU\Sigma V^T$
$= V\Sigma^T\Sigma V^T$
$= V\Sigma^2V^T$

(e) [5 points]

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:

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

[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

- 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.*

- Compute the eigenvalue decomposition of $M^T M$ (*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)?

★ **SOLUTION:** One acceptable answer is

```
>>> U
array([[ 0.27854301,  0.5       ],
       [ 0.27854301, -0.5       ],
       [ 0.64993368,  0.5       ],
       [ 0.64993368, -0.5       ]])
>>> S
array([ 7.61577311,  1.41421356])
>>> Vh
array([[ 0.70710678,  0.70710678],
       [-0.70710678,  0.70710678]])
>>> P
array([  2.,  58.])
>>> Q
array([[-0.70710678,  0.70710678],
       [ 0.70710678,  0.70710678]])
```

Another acceptable answer to part e is

```
>>> import numpy
>>> from scipy import linalg
>>> M = numpy.array([[1, 2], [2, 1], [3, 4], [4, 3]])
>>> linalg.svd(M, full_matrices=False)
(array([[-0.27854301, 0.5 ],
[-0.27854301, -0.5 ],
[-0.64993368, 0.5 ],
[-0.64993368, -0.5 ]]),
array([ 7.61577311, 1.41421356]),
array([[-0.70710678, -0.70710678],
[-0.70710678, 0.70710678]]))
>>> linalg.eigh(numpy.dot(numpy.transpose(M), M))
(array([ 2., 58.]), array([[-0.70710678, 0.70710678],
```

```
[ 0.70710678, 0.70710678]]))
```

- 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* (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).

  ★ **SOLUTION:** V is equivalent to the matrix of eigenvectors if we reorder the columns as per the ordering of the singular values.

- 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 (e)) are referred to as singular values of M. The eigenvalues of M^TM are captured by the diagonal elements in Λ (part (d))

  ★ **SOLUTION:** The singular values of M are square roots of the eigenvalues of M^TM

What to submit:

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

(ii) Upload the code via Snap submission site

2 k-means on Spark (20 points) [Qijia, Sanyam, Hiroto]

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 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:

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

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

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

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:

$$|a - b| = \sum_{i=1}^{d} |a_i - b_i| \tag{3}$$

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

$$\psi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} |x - c| \tag{4}$$

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 Spark: Implement iterative k-means using Spark. Please use the dataset at http://snap.stanford.edu/class/cs246-data/hw2-q2-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 1^{st} 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).

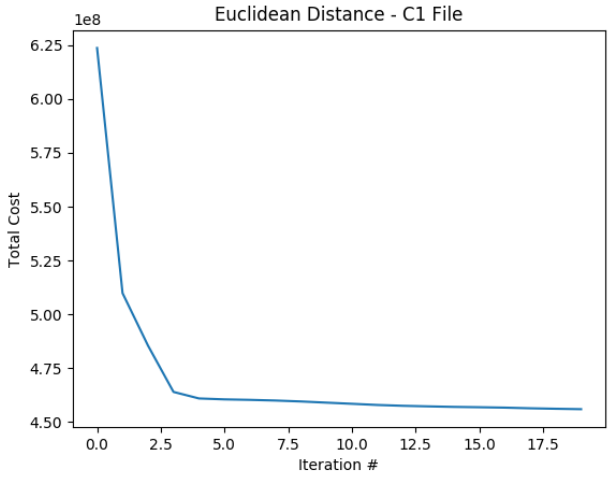
Set number of iterations (`MAX_ITER`) to 20 and number of clusters k to 10 for all the experiments carried out in this question. Your driver program should ensure that the correct amount of iterations are run.

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

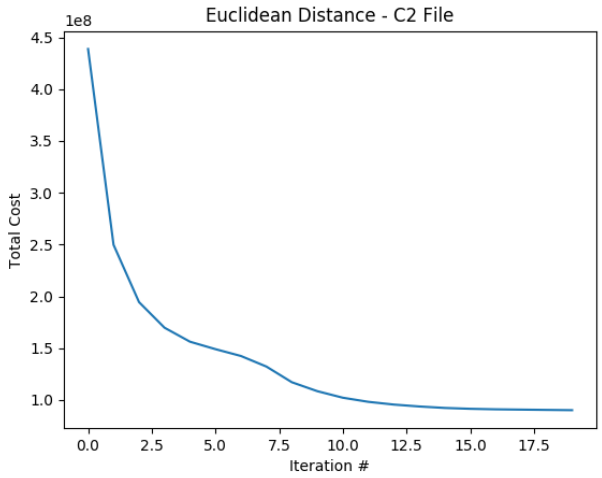
1. **[5 pts]** Using the Euclidean distance (refer to Equation 1) as the distance measure, compute the cost function $\phi(i)$ (refer to Equation 2) for every iteration i. This means that, for your first iteration, you'll be computing the cost function using the initial centroids located in one of the 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 $\phi(i)$ 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.)

   ★ **SOLUTION:** Cost vs Iteration: For c1.txt:



For c2.txt:



2. **[5 pts]** 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` and the distance metric being used is Euclidean distance? Is random initialization of k-means using `c1.txt` better than initialization using `c2.txt` in terms of cost $\phi(i)$? Explain your reasoning.

★ **SOLUTION:** c1 improves by 26% after 10 iterations.
c2 improves by 75% after 10 iterations.

c2 is better than c1 because it distributes the initial clusters far apart. Because there is less overlap, true clusters are split less often, leading to a better final set of clusters.

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

1. **[5 pts]** Using the Manhattan distance metric (refer to Equation 3) as the distance measure, compute the cost function $\psi(i)$ (refer to Equation 4) for every iteration i. 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]** 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` and the distance metric being used is Manhattan distance? Is random initialization of k-means

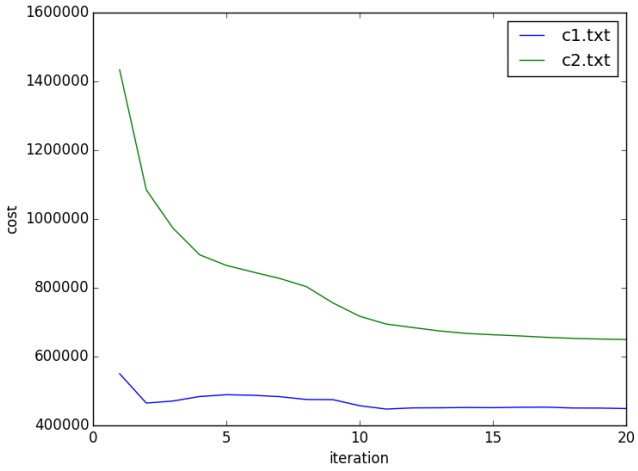


Figure 1: Cost vs Iteration

using `c1.txt` better than initialization using `c2.txt` in terms of cost $\psi(i)$? Explain your reasoning.

★ **SOLUTION:** c1 improved by 19%.
c2 improved by 52%.
c2 is better than c1 because it distributes the initial clusters far apart. Because there is less overlap, true clusters are split less often, leading to a better final set of clusters. Note that c2 is not better than c1 for Manhattan distance because the points in c2.txt were as far apart from each other as possible using the Euclidean distance metric, and weren't necessarily far apart in Manhattan distance.

What to submit:

(i) Upload the code for 2(a) and 2(b) to Snap submission site

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

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

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

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

3 Latent Features for Recommendations (35 points) [Yutian, Kush, Chang]

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

* * *

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

$$E = \sum_{(i,u)\in \text{ratings}} (R_{iu} - q_i \cdot p_u^T)^2 + \lambda \left[\sum_u \|p_u\|_2^2 + \sum_i \|q_i\|_2^2 \right]. \tag{5}$$

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?

★ SOLUTION:

$$\varepsilon_{iu} = 2 * (R_{iu} - q_i \cdot p_u^T)$$

$$q_i := q_i + \eta * (\varepsilon_{iu} * p_u - 2 * \lambda * q_i)$$

$$p_u := p_u + \eta * (\varepsilon_{iu} * q_i - 2 * \lambda * p_u)$$

(b) [25 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 $\eta = 0.1$. 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 http://snap.stanford.edu/class/cs246-data/hw2-recommendations.zip. 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.
- `ratings.val.txt`: This is the test set. We will not be using it for this problem this year.

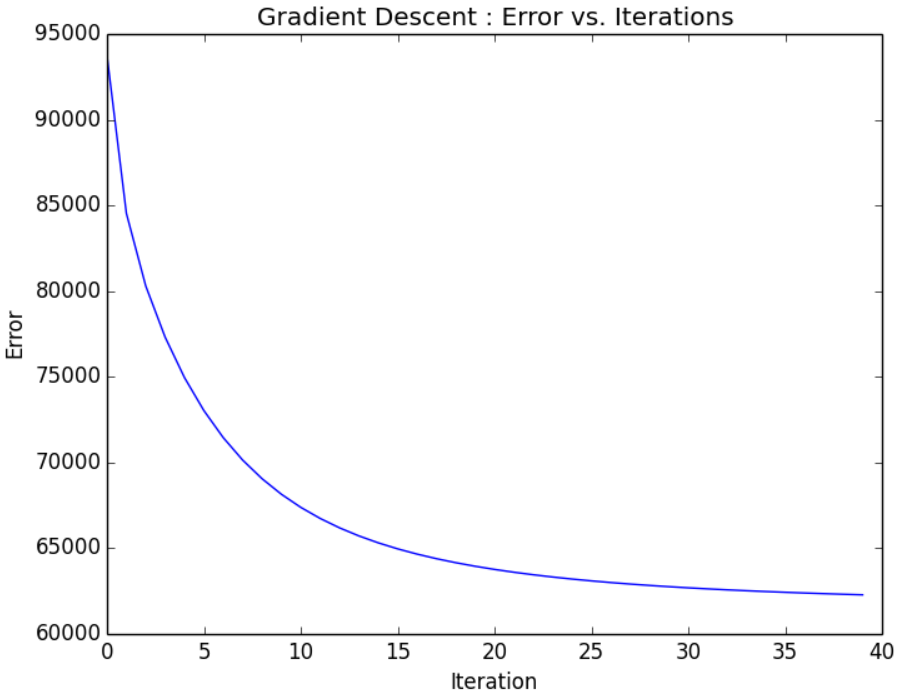
Plot the value of the objective function E (defined in equation 5) on the training set as a function of the number of iterations. What value of η did you find?

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 $q_i \cdot p_u^T \in [0, 5]$. 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.*

★ **SOLUTION:** $\eta = 0.03$



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 snap submission site. [3(b)]

4 Recommendation Systems (25 points) [Wanzi, Sen, Jessica]

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.

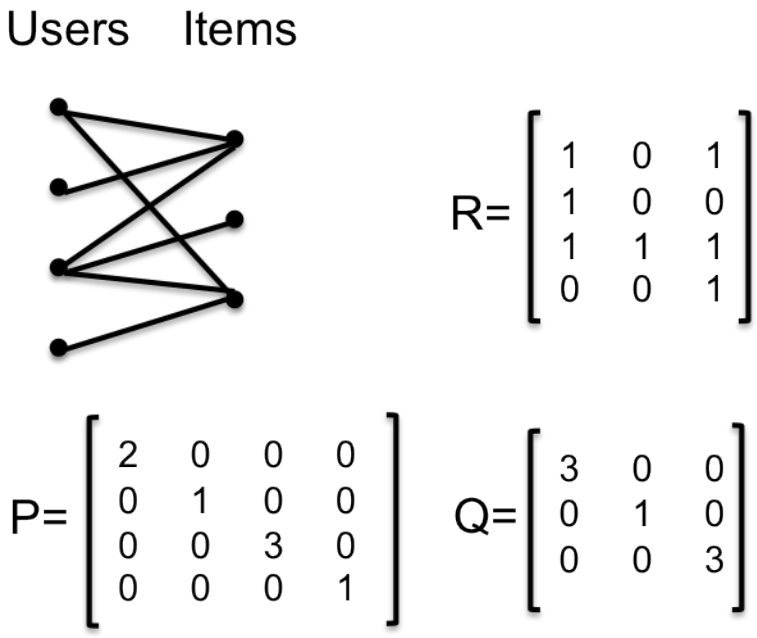


Figure 2: User-Item bipartite graph.

(a) [4 points]

Define the non-normalized user similarity matrix $T = R * R^T$. Explain the meaning of T_{ii} and T_{ij} $(i \neq j)$, in terms of bipartite graph structures (See Figure 2) (e.g. node degrees, path between nodes, etc.).

★ **SOLUTION:** In the user-item bipartite graph, T_{ii} equals the degree of user i. Since $T_{ii} = \sum_{j=1}^{n} R_{ij} * (R^T)_{ji} = \sum_{j=1}^{n} R_{ij}^2 = \sum_{j=1}^{n} R_{ij}$. Since R_{ij} is 0 or 1, so $T_{ii} = \text{degree(user}i\text{)}$.

$T_{ij}(i \neq j)$ is the number of paths between user i and user j with the length of 2, it also represents the number of items that they both like. $T_{ij} = \sum_{k=1}^{n} R_{ik} * R_{jk}$, $R_{ik} * R_{jk}(\neq 0)$ means there exists a 2-step path starts from user i to user j via item_k.

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 $S_I = Q^{-1/2} R^T R Q^{-1/2}$, 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 matrix, you may write it as matrix to the power of $\frac{1}{2}$.)

★ **SOLUTION:** We denote by $R_{i\cdot}$ the i-th row of R, and by $R_{\cdot j}$ its j-th column.

We know that the vector of an item is defined by one column is R. Furthermore, the norm of, say, item_i is defined by $\sqrt{\sum_{k=1}^{m} R_{ki}^2}$. The sum is in fact the number of users that like this item (because R_{ki} can be either 0 or 1, we have $R_{ki} = R_{ki}^2$). So the norm of item_i is in fact $\sqrt{Q_{ii}}$. We thus have:

$$\begin{aligned}\text{cos-sim}(\text{item}_i, \text{item}_j) &= \frac{R_{\cdot i} \cdot R_{\cdot j}}{\sqrt{Q_{ii}}\sqrt{Q_{jj}}} \\ &= \frac{\sum_{k=1}^{m} R_{ki} R_{kj}}{\sqrt{Q_{ii}}\sqrt{Q_{jj}}}\end{aligned}$$

Now, the matrix Q is diagonal, and its diagonal coefficients are all non-zero (otherwise, some items are liked by nobody, and we might as well remove them, because they are useless and

because the angle they form with other items would be ill-defined), so we can denote by $Q^\star$ the diagonal matrix whose diagonal coefficients are defined by $Q^\star_{ii} = Q_{ii}^{-1/2}$. We then have that:

$$\begin{aligned}
\left(Q^\star R^T R Q^\star\right)_{ij} &= \sum_{k,l,m} Q^\star_{ik} R_{kl} R_{lm} Q^\star_{mj} \\
&= \sum_{l} Q^\star_{ii} R_{li} R_{lj} Q^\star_{jj} \text{ (because } Q^\star \text{ is diagonal)} \\
&= \sum_{l} \frac{R_{li} R_{lj}}{\sqrt{Q_{ii}}\sqrt{Q_{jj}}} \\
&= \frac{R_{\cdot i} \cdot R_{\cdot j}}{\sqrt{Q_{ii}}\sqrt{Q_{jj}}}
\end{aligned}$$

So, the matrix S_I can be expressed in terms of Q and R:

$$S_I = Q^\star R^T R Q^\star$$

To compute a similar expression for S_u, we notice that (R, Q, S_I) and (R^T, P, S_u) play similar roles. Indeed, the relation "user_u likes item_i" can be put backward into "item_i is liked by user_u", which is equivalent to switching users and items, ie to transpose the matrix R. Thus, S_u is given by:

$$S_u = P^\star R R^T P^\star$$

with $P^\star$ being a diagonal matrix whose coefficients are defined by $P^\star_{ii} = P_{ii}^{-1/2}$.

(c) [5 points]

The recommendation method using user-user collaborative filtering for user u, can be described as follows: for all items s, compute $r_{u,s} = \Sigma_{x \in users} \text{cos-sim}(x, u) * R_{xs}$ 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 $r_{u,s} = \Sigma_{x \in items} R_{ux} * \text{cos-sim}(x, s)$ 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, $\Gamma = RQ^{-1/2}R^T RQ^{-1/2}$.

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

★ **SOLUTION:** For the user-user collaborative filtering recommendation, we have that:

$$\begin{aligned}\Gamma_{ij} &= r_{ij} \\ &= \sum_{x \in \text{users}} \text{cos-sim}(x, i) \times R_{x,j} \\ &= \sum_{x \in \text{users}} \text{cos-sim}(i, x) \times R_{x,j} \\ &= \sum_{x=1}^{m} (S_u)_{i,x} \times R(x, j) \\ &= \left(S_u R\right)_{ij} \\ &= \left(P^\star R R^T P^\star R\right)_{ij}\end{aligned}$$

Thus,

$$\Gamma = P^\star R R^T P^\star R$$

Similarly, for the item-item collaborative filtering recommendation, we have that:

$$\begin{aligned}\Gamma_{ij} &= r_{ij} \\ &= \sum_{x \in \text{items}} R_{i,x} \times \text{cos-sim}(x, j) \\ &= \sum_{x=1}^{n} R_{i,x} \times (S_I)_{x,j} \\ &= \left(R S_I\right)_{ij} \\ &= \left(R Q^\star R^T R Q^\star\right)_{ij}\end{aligned}$$

Thus,

$$\Gamma = R Q^\star R^T R Q^\star$$

(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.

Download the dataset[3] on http://snap.stanford.edu/class/cs246-data/hw2-q4-dataset.zip.

The ZIP file contains:

[3]The original data is from Chris Volinksy's website at http://www2.research.att.com/~volinsky/DataMining/Columbia2011/HW/HW6.html.

- `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 via the SNAP electronic submission website

★ **SOLUTION:** user_user shows:

- FOX 28 News at 10pm
- Family Guy
- 2009 NCAA Basketball Tournament
- NBC 4 at Eleven
- Two and a Half Men

item_item shows:

- FOX 28 News at 10pm
- Family Guy
- NBC 4 at Eleven
- 2009 NCAA Basketball Tournament
- Access Hollywood

★ **SOLUTION:** Comments: open question. e.g. There is no significant advantage to any of the methods. Or Precision decreases both for user-user and item-item as k increases.