CS246: Mining Massive Data Sets **Winter 2013**

Problem Set 2

Due 9:30am February 7, 2013

Only one late period is allowed for this homework (2/12).

General Instructions

These questions require thought, but do not require long answers. Please be as concise as possible. Please fill the cover sheet (http://cs246.stanford.edu/cover.pdf) and submit it as a front page with your answers. Include the names of your collaborators (see the course website for the collaboration or honor code policy).

Regular (non-SCPD) students should submit hard copies of the answers and upload the source code (see course website FAQ for further instructions).

SCPD students should submit their answers through SCPD and upload the code via the submission website. The submission must include the usual SCPD routing form available at http://scpd.stanford.edu/generalInformation/pdf/SCPD_HomeworkRouteForm.pdf.

Questions

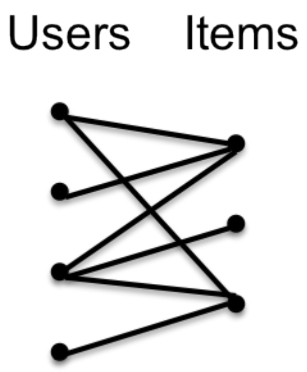
1 Recommendation Systems (25 points) [Le, Tongda]

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 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 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) [5 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. Recall that the cosine similarity of two vectors u and v is defined as $\frac{u \cdot v}{\|u\|\|v\|}$. Express S_I in terms of R, P and Q. Your answer should be a product of matrices, in particular you should not define each coefficient of S_I individually.



$$R=\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P=\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Q=\begin{bmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

Figure 1: User-Item bipartite graph.

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.

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

(b) [5 points]

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

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

(c) [5 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 1) (e.g. node degrees, path between nodes, etc.).

(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[1] on `http://snap.stanford.edu/class/cs246-data/hw2-q1-dataset.zip`.

The ZIP file 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 500th 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 or not. 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 (b), 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 are the most similar to Alex? What are their similarity score? 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 are the most similar to 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.

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

Alex's original row is given in the file `alex.txt`. For a given number k, **the precision at top-k** is defined as follows: using the matrix Γ computed previously, compute the top-k TV shows in S that are most similar to Alex (break ties as before). The precision is the fraction of the top-k TV shows that were watched by Alex in reality.

- Plot the **precision at top-k** (defined above) as a function of k, for $k \in [1, 19]$, with predictions obtained by the user-user collaborative filtering.
- On the same figure, plot the precision at top-k as a function of k, for $k \in [1, 19]$, with predictions obtained by the item-item collaborative filtering.
- Briefly comment your results (one sentence is enough).

What to submit:

(a) Expression of S_I and S_U in terms of R, P and Q

(b) Expression of Γ in terms of R, P and Q

(c) Interpretation of T_{ii} and T_{ij}

(d) The answer to this question should include the followings:

- The five TV shows that are most similar to Alex for the user-user collaborative filtering.
- The five TV shows that are most similar to Alex for the item-item collaborative filtering.
- The graph of the precision at top-k for both the user-user and the item-item collaborative filtering.
- A brief comment on this graph.
- Submit the source code via the electronic submission website.

2 Singular Value Decomposition (30 points) [Rose, Dingyi]

In this problem we study some of the ways the SVD can be computed. Assume throughout that all matrices have real entries.

First, let us recall a simpler version of the "eigen decomposition theorem" (or the eigen-vector/eigen-value decomposition that we all know and love):

Theorem 1 *If B is a real symmetric square matrix, then it can be decomposed as follows:*

$$B = PDP^{\mathrm{T}},$$

where D is a square diagonal matrix with the eigenvalues of B on the diagonal, and P is an orthogonal matrix with eigenvectors of B as the columns.

The way we find eigenvalues and eigenvectors is to simply solve the equation $Bx = \lambda x$ by transforming this into the determinant equation $|B - \lambda I| = 0$.

Recall that the SVD decomposition can be written as follows:

$$A = USV^{\mathrm{T}},$$

where A is of size $m \times n$ (m rows and n columns), U and V are orthogonal matrices of size $m \times m$ and $n \times n$, and S is a diagonal matrix with the singular values of A on the diagonal.

(Note: In class, it was defined as U is $m \times r$, S is of size $r \times r$ and V is $n \times r$, where $r < m, n$. In this exercise, U is simply the U defined in class, padded with $m - r$ orthonormal columns, S is the $r \times r$ matrix S defined in class, padded with 0's to make it $m \times n$, and V is the $n \times r$ matrix defined in class padded with $r - n$ additional orthonormal columns.)

In this problem, let A be a matrix with size 100×10000. We wish to compute the SVD decomposition of A. We will make use of the eigen decomposition theorem to find the SVD of A (even though A is not square).

(a) [10 points]

Finding the SVD using AA^T or A^TA:

It can be shown that both these matrices AA^{T} and $A^{\mathrm{T}}A$ are square symmetric matrices with values UDU^{T} and VEV^{T}. (D is a square matrix with size $m \times m$, E is a square matrix with size $n \times n$. Both these matrices are diagonal matrices with the squares of the singular values of A along the diagonal.) Note that the eigenvalues of AA^{T} or $A^{\mathrm{T}}A$ are simply the values along the diagonal of D or E.

1. Lets say we simply want the magnitudes of the singular values (*i.e.*, the values ignoring +/- signs). We can use either AA^{T} or $A^{\mathrm{T}}A$. Which one should we use and why?

2. Here, we allow ourselves to use both AA^{T} and $A^{\mathrm{T}}A$. How would you compute U and V?

3. Prove that $A^{\mathrm{T}}u_i = \lambda_i v_i$ for columns u_i and v_i in U and V and λ_i is the i^{th} entry along the diagonal in S.

4. Prove that we can compute U and V using only the eigenvalue decomposition of AA^{T} (without using the decomposition of $A^{\mathrm{T}}A$).

   (Hint: prove that once you have U, you can compute V.)

5. Similarly, prove that we can compute U and V using only the eigenvalue decomposition of $A^{\mathrm{T}}A$.

(b) [10 points]

Finding the SVD using $M = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$:

1. What is the relationship between M and M^{T}?
2. Prove that if u_i, v_i are corresponding columns in U and V, then $\begin{bmatrix} v_i \\ -u_i \end{bmatrix}$ and $\begin{bmatrix} v_i \\ u_i \end{bmatrix}$ are eigenvectors of matrix M.
3. Prove that for these eigenvectors, the eigenvalues are $-\lambda_i$ and λ_i. (Thus, we can directly get the SVD from the eigen-decomposition of M.)

(c)[10 points]

Document similarity using SVD:

In this exercise, you will play around with SVD on some real data. To complete the problem it will be the easiest to use Matlab. In Matlab SVD of a matrix A can be retrieved as: `[U, S, V] = svd(A, 'econ');` If you do not have access to Matlab you can also use the open source "Matlab" implementation called Octave. You can also use any other software package/language.

First, download the input file http://snap.stanford.edu/class/cs246-data/hw2-q2-Matrix.txt.zip [2] which is a document-term incidence matrix I (which includes idf to give a higher weight to rarer terms) of various news articles. Here, each of the rows i can be considered to be a document d_i while the columns represent the occurrence of words within the corresponding document. In this example, the first 100 rows correspond to baseball stories while the remaining documents belong to various other categories such as politics and technology.

We want to figure out how well decomposition performs when it comes to identifying the average cosine similarity between just baseball documents and all documents in the data set. Let's define $r = ((\sum_{i=1}^{99}\sum_{j=i+1}^{100} \text{cossim}(d_i, d_j))/\binom{100}{2})/((\sum_{i=1}^{496}\sum_{j=i+1}^{497} \text{cossim}(d_i, d_j))/\binom{497}{2})$. Ideally, we would want r to be as large as possible in order to maximize the similarity between baseball documents while minimizing the similarity between other types of documents.

We can use SVD to reduce the dimensionality of the matrix. This can be accomplished by only keeping the k highest significant values and setting the other values to zero. Recompute I from the original U and V and the new S matrix containing only the k highest significant singular values and compute the r of Matrix I when $k = \{197, 247, 297, 347, 397, 447, 497\}$. Plot a graph showing how r changes as k varies.

What to submit:

[2] Original dataset received from Dr. Jason Rennie http://qwone.com/~jason/

1. Part(a): Written solution to question (1) to (5).
2. Part(b): Written solution to question (1) to (3).
3. Part(c): The plot of r as a function of k and submit the source code via the electronic submission website.

3 Theory of k-means (20 points) [Justin, Robi]

In this problem, we study two different initialization methods for the k-means algorithm. First, let us setup the problem: we have a set $\mathcal{X}$ of n data points in the d-dimensional space $\mathbb{R}^d$. We are also given k, the number of clusters, and would like to choose a set of k centers $\mathcal{C}$ to minimize the cost function:

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

Then, the k-means algorithm for the above problem works as follows:

1. **Initialization**: Arbitrarily choose k initial centers $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$
2. $\forall\, i \in \{1, 2, \ldots, k\}$, set the cluster $C_i = \big\{x \in \mathcal{X} | \text{argmin}_{1 \le j \le k}\{||x - c_j||\} = i\big\}$
3. $\forall\, i \in \{1, 2, \ldots, k\}$, set c_i to be the center of mass of C_i, that is $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$
4. Repeat steps 2 and 3 until $\mathcal{C}$ no longer changes.

We first prove some basic facts about this algorithm.

(a)[5 points]

Assume S is an arbitrary set of points (in $\mathbb{R}^d$) with center of mass $c(S)$, and assume z is an arbitrary point (not necessarily in S). The notation $||\cdot||$ denoting the L_2 norm, first prove that:

$$\sum_{x \in S} ||x - z||^2 - \sum_{x \in S} ||x - c(S)||^2 = |S| \cdot ||c(S) - z||^2.$$

(b)[5 points]

Then, prove that, while $\mathcal{C}$ keeps changing, each iteration of the k-means algorithm decreases the cost, *i.e.* prove that for each iteration i (where $\phi_i = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}_i} ||x - c||^2$), we have $\phi_i \geqslant \phi_{i+1}$.

(c)[5 points]

Finally, prove that the during the k-means algorithm, the cost function always converges to a finite value. (This claims holds independently from the dataset and the initial clusters.)

(d)[5 points]

With an example show that with a bad initialization, the k-means algorithm may converge to a clustering that has an unboundedly larger cost than the optimal clustering (*i.e.*, the one with the optimal cost). In other words, given an arbitrary number $r > 1$, give a situation where k-means converges to a clustering whose cost is at least r times larger than the cost of the optimal clustering.

(Note: A hard-coded example or the output of a coding example that behaves badly would not receive full credit. We are asking for a formal example that can work for any arbitrary number $r > 1$, and a formal analysis of the performance of the k-means algorithm.)

What to submit:

(a) Proof that $\sum_{x \in S} ||x - z||^2 - \sum_{x \in S} ||x - c(S)||^2 = |S| \cdot ||c(S) - z||^2$

(b) Proof that the cost of k-means decreases at each iteration

(c) Proof that the cost function always converges

(d) Formal example that leads to non-optimal convergence and analysis

4 k-means on MapReduce (25 points) [Sumit, Anshul]

Automatic tagging and cross-linking of documents: Clustering algorithms are frequently used to automatically categorize documents by web services which receive a large number of new content items daily. These services need an automated way to find related items and to categorize and cross-link the documents as human intervention would be very expensive. One such example is a news aggregation website like Google Reader which processes tens of thousands of new documents per day.

We will explore this usecase in this problem.

(a) [10 pts]

k-Means clustering on Hadoop: Implement the k-means using Map Reduce where a single step of Map Reduce 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.

Run the k-Means algorithm on Hadoop to find the centroids for the dataset at: `http://snap.stanford.edu/class/cs246-data/hw2-q4-kmeans.zip`

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. `vocab.txt` contains the words used for generating the document vector. (For example the first word represents the first feature of the document vector. For document 2 (line 2 in `data.txt`), feature 3 "alkalemia" (line 3 in `vocab.txt`) has frequency `0.5`, so the third component of the document vector is `0.5`.)

3. `c1.txt` contains k initial cluster centroids. These centroids were chosen by selecting $k = 10$ random points from the input data.

4. `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).

Use Euclidean distance as the distance measure. Set number of iterations = 20 and number of clusters = 10. Use points in `c1.txt` for initialization.

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 a 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 no-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 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 :

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

Cluster Initialization strategies: The output of k-Means algorithm depends on the initial points chosen. There are many ways of choosing the initial points. We will compare two of them in the following parts.

(b) [5 pts]

For every iteration, compute the cost function $\phi(i)$ (as defined in the previous question). 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 for `c1.txt` and also for `c2.txt`.

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

(c) [3 pts]

Is random initialization of k-means using `c1.txt` better than initialization using `c2.txt` in terms of cost $\phi(i)$? Why? What is the percentage change in cost after 10 iterations?

Automatically tagging documents: One way of tagging documents is to assign word tags to each cluster. To get the top `k` word tags for a cluster, sort the features for its centroid by their coordinate values in descending order, choose the top `k` features and use words from `vocab.txt` to get the word tags for these top `k` features.

For e.g. if the indexes of top 5 features of the centroid for cluster 1 are `[1, 2, 6, 7, 10]` then the tags for the documents in cluster 1 are `[activator, decay-corrected, softer, mitomycin, spc]` which are words at line `1, 2, 6, 7, 10` respectively in `vocab.txt`.

Use points in `c1.txt` for initialization.

(d) [7 pts]

Which 5 tags would you apply to the second document? *For verification, the 6th tag is* `anserine`.

What to submit:

(a) Code (include a printout with your submission and also submit it online)

(b) Code (include a printout with your submission and also submit it online), and graph for initialization strategies

(c) The answer should include percentage improvement values

(d) 5 best tags