Netflix Recommendation System - Unsupervised Learning Module - Assignment 1

Maryam Zolfaghar

[1]Computer Science Department, UC Davis

1. Spectral Clustering

1.1. a) A short paragraph over-viewing the reasoning behind the approach

Spectral clustering is an unsupervised learning algorithm in which the similarity/affinity determines the clusters for data points. In the Netflix assignment, we are dealing with a sparse matrix. Spectral clustering does not make any assumptions on the form of cluster and can be implemented efficiently even for large datasets as long as the similarity graph is sparse. Then, after constructing a similarity graph we only need to sole a linear problem. However, it is important to choose a good similarity graph. For Netflix dataset, we could create either MovieXMovie or UserXUser similarity matrix. I have used cosine-similarity for the MovieXMovie matrix. Compare to other methods like k-means algorithms, spectral clustering require some additional steps to provide similarity matrix (preprocessing). Therefore, it does not apply on original data and similarity matrix could have more meaningful information compare to raw data. Here, for Netflix assignment, it is helpful to find similarity between nodes in graph (either users or movies) and use that information to predict ratings.

1.2. b) A diagrammatic summary of your approach

The main algorithm has the following steps:

- 1. Transform train.csv dataframe to either UserXUser or MovieXMovie matrix during some preprocessing steps
- 2. Construct a similarity matrix/graph
- 3. Determine adjacency, degree and laplacian matrix (W, D, L)
- 4. Calculate eigenvalues and eigenvectors of the laplacian matrix L
- 5. Using the first k eigenvectors corresponding the the first k smallest eigenvalues and apply k-means model to find centers for clusters, for tuning this k I have used both cross validation and eigengap trick.
- 6. Applying k-means on the first k eigenvecors and find centers and clusters that each movie is belonging to. For each missing data (or validation data in cross-validation or any other data that we want to predict the ratings for), calculate the average of ratings of other data points in the cluster where that missing data belongs to.
- 7. After being done with tuning parameters, set k to the finalized k (i.e, 4) to predict ratings for test data. For each entry in test.csv, I have first checked whether there was an existing rate for the specific (movie_id and cusotmer_id), if not, the cluster number where that movie_id belongs to has been restored and the average rating of similar movies ratings in that cluster assigned to that entry of test data.

1.3. c) An at most one page detailed writeup on your approach including how you tuned any hyper-parameters

- 1. I have created MovieXMovie graph, each node is one movie
  - function **utils/read_preprocess_data.py**
  - Read csv-datasets as dataframe in python
  - Create a UserXMovie dataframe and (mean) aggregate over date entry
  - Fill out NaN data by the mean of the columns (movies)
- 2. I have constructed a similarity matrix using pairwise cosine similarity between all users for a pair of movies. The corresponding matrix will have shape of movie# $\times$ movie#
  - This step has been done in function **utils/gen_similarity.py** that is generating similarity matrix (args.sim_method='cosine_similairty') for movies.
- 3. I have calculated the degree matrix (summation over the diagonal data of the similarity matrix), and then calculate the laplacian by $L = D - W$ formula. We know that if the graph W has k connected components, then laplacian matrix will have k eigenvectors with corresponding eigenvalues equal zero.
  - This step has been done in function **utils/calc_laplacian.py**
- 4. Calculate the eigenvalues and eigenvectors of the laplacian matrix
  - This step has been done in function **utils/calc_eig.py**
  - 1. This function is calculating eigenvaues and vectors of laplacian using "scipy.sparse.linalg.eigsh"package that gave me the k smallest to largest eigenvalues and corresponding eigenvectors
  - 2. I have checked if only the first eigenvalue was zero or a very small number close to zero, I threw it away.
  - 3. I have also calculated the **eigengap** to use for tuning the k. The results showed k equals 4 could be a good option since there is a huge perturbation after the first k eigenvalues.
  - 5. For tuning parameter k, in function **spectral_clustering.py**, for a number of epochs and for different number for parameter k, I have generated similarity matrix MovieXMovie, then calculate eigenvalues and vectors and laplacian for this matrix and only include the first k eigenvectors corresponding the first k smallest eigenvalue. For training data, I have randomly selected 25% of the existed ratings and set them to zero and then validate the algorithm on these test data. Mean squered error (MSE) and root of MSE (RMSE) for each epoch have been stored for different number of k. I have finally set k to 4 since around this number test data error increased where train error was still decreasing.
  - 6. I have applied sklearn.cluster.MiniBatchKMeans with k number of clusters on the first k eigenvectors for movies and saved labels/clusters for each movie. It seemed sklearn.cluster.MiniBatchKMeans is way faster than sklearn.cluster.Kmeans!
  - 7. For test data, I have read test.csv and for each entry, first checked if that (movie_id, customer_id) has been rated before in train data or not, if so assign that rating, if not, assign the average of ratings of movies in the same cluster that this movie_id belongs to.

2. Matrix Completion

2.1. a) A short paragraph over-viewing the reasoning behind the approach

In matrix completion we try to complete a missing values in a matrix. To do that we use matrix factorization which is breaking down a matrix into a product of multiple matrices. Singular value decomposition (SVD) is one way to do this factorization. SVD is an algorithm that decomposes a matrix A into the best lower rank approximation of A ($A = U\Sigma V^T$). In our assignment, **A** is UserXMovie rating matrix, U is the user feature matrix, Σ is the diagonal matrix of singular values, and V^T is the movie feature matrix. Both U and V^T are orthogonal. U shows how much users like each feature and V^T shows how relevant each feature is to each movie. To get the lower rank approximation, I kept the top k features of these matrices which is the k most important underlying high-level features. Here, with this bipartite graph, we have information from both movies and users.

2.2. b) A diagrammatic summary of your approach

- 1. Transform train.csv dataframe to either UserXUser or MovieXMovie matrix during some preprocessing steps
- 2. Using SVD function from scipy package
- 3. Making predictions from decomposed matrices and tuning parameter k using cross validation for a number of epochs
- 4. Set k to the finalized k, and predict ratings for missing values and entries in test.csv

2.3. c) An at most one page detailed writeup on your approach including how you tuned any hyper-parameters

- 1. Setting up the rating data during some preprocessing steps and filled NaN values with column means (e.g, for movieXmovie matrix, it is mean of rating of all users for each movie). Same function that I have used for the SpectralClustering (**utils/read_preprocess_data.py**).
- 2. I have decomposed UserXMovie matrix using scipy.sparse.linalg.svds 3. For tuning parameter k which is the dimension for $A_{n\times k}; \Sigma_{kk}; V^T_{k\times m}$, for a number of epochs, I have created train and test data. 25% of train data that has already a rating rate have been chosen and set to zero. SVDs then applied on train data and the MSE of the algorithm was calculated for the 25% missing data. In SVDs, then k for the low-rank matrix has been tuned according to different MSE for different k. MSE has been calculated as a difference between predicted ratings from reconstructed A^*_{ij} and true rating from A_{ij} (before setting that to zero for the validation set). I have used $k = 4$ at the end according to the result where test data error increased where train error was still decreasing.
- 4. After setting k to the finalized k (o.e, 4), for test data, I have read test.csv and for each entry, first checked if that (movie_id, customer_id) has been rated before in train data or not, if so assign that rating, if not, assign the round (np.ceil) predicted rating in $A^*_{movie-id,customer-id}$ to that entry.

2.4. Code and Jupyter Results

I have uploaded my code, report, and result for predicted test data in my **GitHub** for NetflixRecommendationSystem project.

3. Comparison

In Spectral clustering, during prediction we have lost the information since the similarity matrix has been created for either Movies or Users and not both of them together. So, for prediction, for a movie, averaged rating over all users has been used. However, for matrix completion, we have information of both User and Movie since we used bipartie graph. Matrix completion is way faster than spectral clustering method. Low-dimension matrix tries to capture the underlying features driving the raw data. The amount of preprocessing it needs before doing the SVD is less than spectral clustering. Therefore, if can also scales better to larger datasets faster. On the other hand ,using a low-rank matrix will loose some of high-level information. One good approach could be combining a couple of these algorithms to get the benefit of each. One basic future step is adding more informative information like genre of each movie as feature to the matrix that could be helpful in preprocessing steps. We also did not include bias difference for each user, for example, if we had genre feature, one user might be biased toward watching one genre more or giving higher rates to movies or one genre in general. It might be helpful to include this bias in learning or during the preprocessing. One easy and simple way might be normalizing each user by subtracting mean and dividing by the standard deviation of all ratings for all movies that user has been watched so far. Then later for the prediction part, add that to the predicted rate.