ECS 271 - Netflix - Spectral Clustering

Markham Anderson

April 12, 2020

1 Overview

I opted to make use of spectral clustering because it was the less familiar two me of the two methods presented. That said, the design of my system was quite spartan, and the motivation behind this strategy lay in my inexperience with spectral clustering. I address some design alternatives in the Approach section below, as well as implementation details for all points of this overview. Preparatory to clustering, I constructed a user graph and a movie graph, whose edges were a function of the user-movie ratings provided in the training data. From these graphs, spectral clustering yielded a set of movie clusters and a set of user clusters. I suppose that a cluster of users represents like-minded individuals who tend to have similar tastest in movies; a cluster of movies represents items of similar attributes (e.g. genre, production value, levity). Founded on the supposition that a given cluster of users will tend to award similar ratings for a given movie and that a cluster of movies will tend to receive similar ratings from a single user, my model infers a rating value for a given user-movie pair as the mean of (1) the mean rating from the pertinent user cluster and (2) the mean rating from the pertinent movie cluster. Details are provided in section **??**: Approach.

2 Diagrams

Fig **??** outlines the steps that lead from the input ($usr \times mov$) to the user-movie rating prediction. Fig **??** provides greater detail on the steps that lead from the clusters to the movie-related mean μ_m. (Calculation of the user-rated mean μ_u is done identically to μ_m except that the roles of users and movies are swapped.)

3 Approach

In **??**, you see that the input, taken from `train.csv` was organized into a matrix of $|movie| \times |user|$, which I denote M. This matrix is incomplete because not all possible user-movie ratings are given in `train.tsv`, so for the unknown matrix values, I used 0 as a placeholder. (0 is not a legal value for known cells.)

The two graphs on which spectral clustering was performed were created identically, one for users, one for movies. I'll describe the process for the movie graph here. I created a fully-connected graph of movies. Then for all movie pairs (m, n), I set the edge weight $w_{m,n}$ as follows.

$$w_{m,n} = \sum_{u \in Users} (1.5 - |M[m,u] - M[n,u]|) \times 1_{m,u} \times 1_{n,u}$$

$1_{a,b}$ is an indicator function which equates to 1 if $M[a,b] > 0$ and equates to 0 otherwise. Thus each user contributes to the edge weight between every pair of movies for which they have provided ratings (for both movies). If a user has rated two movies similarly, this contribution is positive; if a user has rated two movies dissimilarly, this contribution is negative. My choice of 1.5 as the boundary for similar versus dissimilar ratings was arbitrary, based on my supposition that if two movies have ratings that are different by a value of more than 1, I think they should not be connected in the graph.

I considered influencing edge weights with the difference between films' release dates, their genres, and the Levenstein distance between their titles. However, demands on my time convinced me to move ahead with the barebones design described thus far. Alternative to using these data, I might have constructed a constraining matrix which could be used later after the pattern of the one denoted Q in the class lecture.

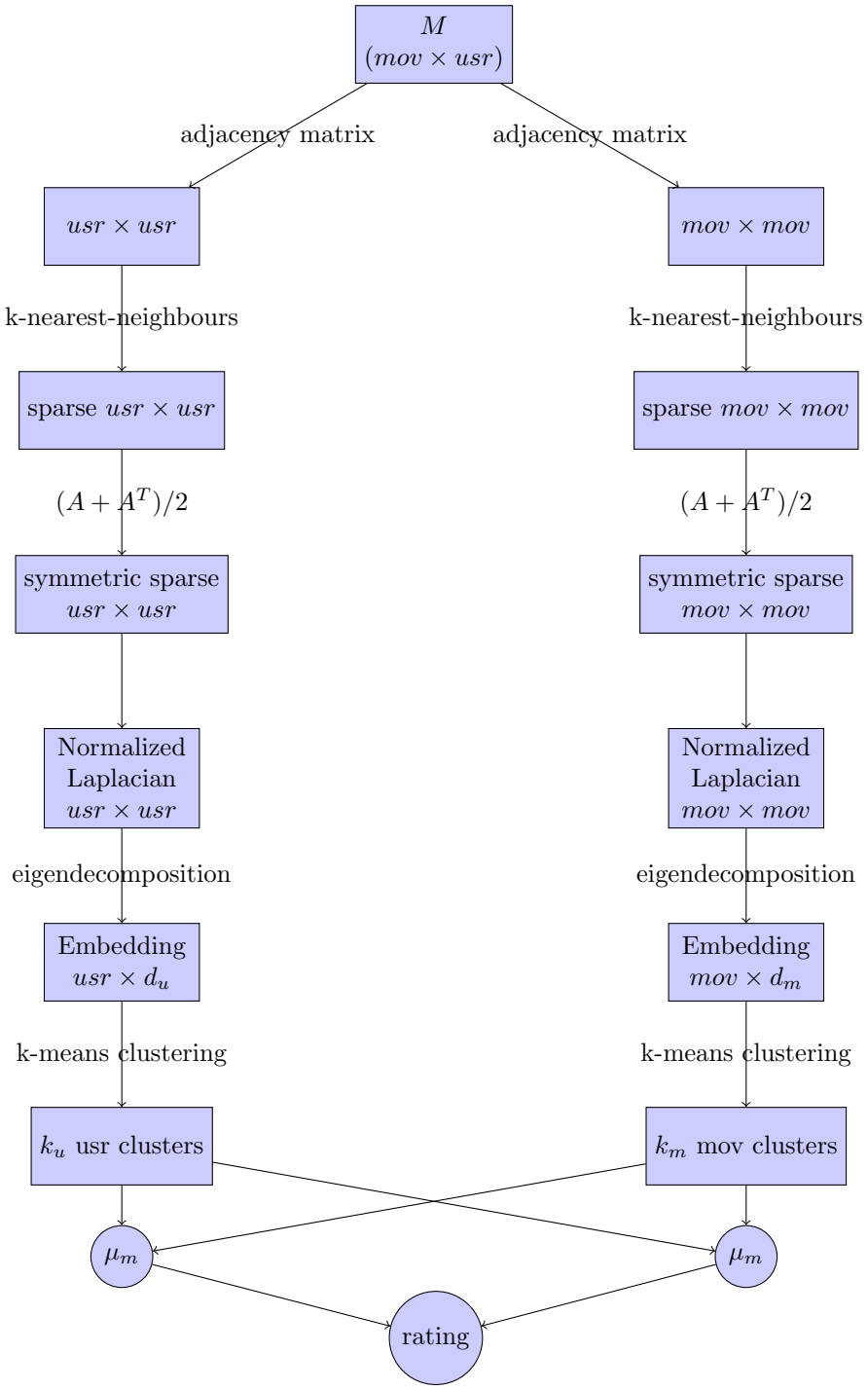

Figure 1: Soup to nuts

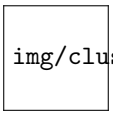

Figure 2: Movie mean for a given cluster of users.
IC for t-sne https://www.oreilly.com/learning/an-illustrated-introduction-to-the-t-sne-algorithm

After having thus constructed a fully-connected graph, I used k-nearest-neighbours to determine which edges to keep. Making the graph thus sparse would allow a complexity of $O(n^2)$ for the eigendecomposition to follow, however, the k-nearest-neighbours algorithm left my new adjacency matrix asymmetric, so I replaced it with the similar matrix $A := (A + A^T)/2$. I then constructed a Normalized Laplacian and its computed its first d eigenvectors.

I experimented with several values for k_{nn} and d. For k_{nn}, examined the distance between the eigenvalues to evaluate them. I observed that the graphs of the distances adopted a sharper curve as k_{nn} increased, so I proceeded with a high value of 128 for k_{nn}. For d, I began with the value where I saw the jump in the eigenvalues' distances take place, but through experimentation on the training set, I eventually settled the much lower value of 16.

The selected 16 eigenvectors provided a spectral embedding on which I applied k-means clustering. As before, experimentation on the training set dictated my choice for $k_{clusters}$, which in the end was 32.

Of course, k_{nn}, d, and $k_{clusters}$ could have had different values for users and for movies, and it might have been well to have done so because of the vast difference in the sizes of the two sets, but I kept them the same for most of my experiments in order to reduce the size of the hyperparameter search space.

Having completed the spectral clustering, I inferred a rating for each user-movie pair as the mean of two values $rating = (\mu_u + \mu_m)/2$. μ_u and μ_m are computed identically, except that one is primarily predicated on the user clusters, and the other is primarily predicated on the movie clusters. **??** shows how μ_m is computed for a given user-movie pair:

Let $C_u^{(usr)}$ be the user cluster in which user u resides, and let $C_m^{(mov)}$ be the movie cluster in which movie m resides. μ_m is computed as the mean of all (non-zero) ratings from all users $v \in C_u^{(usr)}$ for movie m. Formally:

$$\mu_m^{(u,m)} = \frac{1}{n_{nz}} \sum_{v \in C_u^{(usr)}} M[m, v]$$

$$n_{nz} = \sum_{v \in C_u^{(usr)}} 1_{m,v}$$

$$rating_{u,m} = \mu_m^{(u,m)} + \mu_u^{(u,m)})/2$$

When evaluating this method against the training data, it was necessary to omit the term $M[m, v]$ when $u = v$, as it would give to much weight to the true label for each example.

I experimented further with using only μ_u or μ_m instead of the mean of the two in order to produce a rating, but the mean performed better. I also experimented with including the training set's mean rating in my mean term, i.e. $(\mu_u + \mu_m \mu_r)/3$, but this still produced inferior results.

When computing $\mu_m^{(u,m)}$, in cases where a cluster produced no neighbours with ratings, i.e. no non-zero entry $M[v, m]$ s.t. $v \neq u \bigwedge v \in C_u^{(usr)}$, I opened up the calculation to use *all* $v \in Users$. Thus, I used a the mean rating for the film, not just the mean rating from the selected user cluster. When computing $\mu_u^{(u,m)}$ and finding cases where a cluster produced no neighbours with ratings, I took the corresponding measure, with the roles of users and movies swapped. In the event that there were no ratings at all for a given movie or user, the mean rating over the entire dataset (over all users and all movies) was used.