Towards a Movie Recommendation System

April 18, 2019

Ben Christensen, Benjamin Burt, Jane M. Cox

Abstract

Online movie-streaming platforms today are more popular than ever. Given the overwhelming amount of content available, users have come to rely on personalized recommendations for discovering new shows. It is clear then, that there is a large need for accurate algorithms to create those recommendations. The purpose of this project is to explore existing machine learning methods and their success in accurately predicting a movie to recommend. We focus on both clustering and classification algorithms. We test these models and their accuracy on the MovieLens data set. The aspects of using a sparse data set in machine learning methods are explored. The best recommendation from a single method came from using non-negative matrix factorization.

1 Introduction

With the introduction of online streaming in 2007 by Netflix (Lusted 2012), the experience of watching and selecting a movie to watch has fundamentally shifted. Instead of checking the newspaper's listing of available cinema, or browsing the stacks at Blockbuster, users today seem to have all of cinema at their fingertips. With the introduction of Hulu, also in 2007, and Amazon Prime's streaming service in 2011 (Cross 2014), there are now a surfeit of films accessible to viewers. With such an overabundance, the ability to accurately match a user with films he or she would actually enjoy viewing becomes paramount. As a result, all three of the above streaming services fiercely guard their algorithms.

Recommendation Systems take advantage of data collected about users and items. Netflix, for example, uses a combination of user data (including clicks/time spent viewing a product) and paid taggers who meticulously sort through movies and genres, to generate their recommendations (Gomez 2016). The amount of data that streaming sites generate for use by such algorithms is, understandably, huge.

Our purpose is not to compete with such massive data sets and algorithms. Instead, we aim to accurately predict film recommendations for a given user using simple machine learning algorithms. Our goal in this is to explore whether it is possible to accurately recommend movies using these methods, and if it is, what method or combination of methods is best to use. Our motivation for doing so is two-fold. First, given how massive the data sets are that streaming services use, we seek to understand if a careful application of algorithms can quickly and non-expensively recommend a movie. Second, we seek to understand if it is possible to accurately match a user with a recommendation when using only user-generated ratings and tags.

2 Data

For this project we consider data from the MovieLens online movie rating site. It is provided free of charge for research purposes at *movielens.org*. MovieLens advertises itself as giving users, "non-commercialized, personalized movie recommendations". To use it, users initially rate several movies, after which the site generates a user profile and recommends further movies to watch. Run by the GroupLens research lab from the University of Minnesota, MovieLens does not stream videos. As such, it is free from advertisements or commercial pressures, such as the push on streaming platforms to promote original content. Like nearly all data sets reliant on voluntary user ratings, it is very sparse. In fact, only .54% of possible ratings are filled. While this is indeed small, it is not at all uncommon to have sparsity like this in real world data sets, so this does not present a huge problem.

The MovieLens data set has a variety of data useful for recommendation systems. Most importantly, it contains users, movies, and individual user ratings for said movies, with ratings on a .5 scale ranging between .5 and 5. Additionally it contains user generated tags for each movie, as well as genre information for the movie. User generated tags may be similar to genre, such as the popular tag, "children's"; however seldom-repeated and unique tags also exist, such as the tag "circle-k". Genre information is culled from imdb.com. Lastly, there are scores for each movie's relevance to each user generated tag. These were found using a machine learning algorithm by the researchers who put together the data set.

The median number of ratings per film is 18. Many of the films with less than the median number of ratings have very few ratings, on average about 1 or 2. This may be because less popular or known films tend to receive less views overall, resulting in a smaller pool of potential users who can rate it. However without view data, such an assessment is impossible to prove. Due to this lack of ratings, we chose to drop all films with 17 or less ratings. This ensures that every film we consider has enough ratings to accurately predict user preferences. The final count of films we consider ratings for is about 13,000.

One challenge with working with the data arises from how ratings were distributed. Again, sparsity presents a problem. If a user did not rate a film, the user-rating entry for that film is in the form of a NaN. This is problematic due to the massive number of unknown ratings. Although we removed any movies with less than 18 ratings, movies still contain a considerable number of NaN's from the users who had not rated them. Nearly all of the algorithms we implement require a dense user/movie matrix, at least in their simplest implementation. In those cases, we fill the NaN's with the middle rating, 2.5. We initially considered filling them with zeros; however as the possible ratings ran from .5-5, this would have unfairly skewed our predicted ratings to be lower than they should be. Filling the unknown ratings from a user with the average of all their ratings also skews the data, as a user may love horror films but hate romantic comedies, and an overall average would not reflect that. Furthermore, given the sparsity of the data, genre averages for each user is not always feasible. By filling in the NaN's that we have with the median rating of 2.5, we obtain the smallest possible skew on the data.

3 Methods

We base the bulk of our methods off of user-collaborative filtering. This groups similar users together and use their ratings to recommend movies for a user in their group. There are several potential clustering algorithms appropriate for this task. We chose to implement K-Nearest Neighbors (KNN), K-Means, and Non-negative Matrix Factorization (NMF). Each algorithm takes different approaches to determine similarity between users so we decided to combine each algorithm in an attempt to improve the accuracy of our recommendation system. Following we will discuss our implementation of each algorithm, but first it is instructive to discuss the transformation of the data necessary in preparation for clustering.

The movie ratings provided by MovieLens include the user ID of the user giving the rating and the movie ID of the movie the user is rating. To compare users by their ratings we create a matrix with user IDs as the rows and movie IDs as the columns. But since users don't rate every movie, only 0.54% of this matrix is filled with movie ratings. This may seem extremely small. It is a small proportion, but even the best recommendation systems are working with 1%-2% density for this user-movie matrix (Sindwani 2010). In fact, the movies users choose to rate can tell us about the similarity of the users. In both KNN and K-means, we incorporate this information to improve our predictions.

In each algorithm we give the neighbors in the cluster voting power in determining the predicted rating for each user. We then take the prediction as the weighted average of each model's prediction. We use cross-validation to select the optimal number of clusters in each model and the weights for the combination of the models. The highest predicted rating for each user is the movie we recommend to the user.

Another way to predict user-movie ratings is content-based filtering by recommending movies similar to movies users enjoy. Two features MovieLens provides for movies are genres and user-generated tags. Movies with overlapping genres are likely to be similar. Likewise for tags. Using one-hot encoding, we create a movie-genre matrix similar to the user-movie rating matrix that will allow us to use the same clustering algorithms albeit with adjusted prediction functions. Instead of one-hot encoding tags as well, we use the relevance score each movie has for tags that was generated by researchers at MovieLens using machine learning algorithms. MovieLens calls these relevance scores "genome scores." [http://files.grouplens.org/data sets/movielens/ml-20m-README.html] On both the movie-genre matrix and the movie-relevance matrix we can cluster to find similar movies and then predict user-movie rating based on the average user rating for each of these movies.

A major obstacle to performing content-based filtering is that the density of these matrices is 100%. That means the content-based algorithms performs about 200 times the operations performed by the user-collaborative algorithms. Our current implementation of the clustering algorithms isn't efficient enough to handle all of these operations in a reasonable amount of time. We intend to address that in the future. With dimensionality reduction we may be able to reduce run time without major changes to the way operations are performed and predictions are generated. Next we will outline the similarity measures and prediction formulas used in the clustering algorithms.

For a baseline to measure our success against, we consider the mode rating given by users: 4. If we were to predict that every user rates movies that they haven't seen as 4, we obtain 56% accuracy. For clustering methods where a percent accuracy is not a reasonable measure, we

consider the mean squared error.

As mentioned above, We consider clustering methods and clustering methods within the framework of user clustering

3.1 Classification Methods

3.1.1 K Nearest Neighbors

The first classification algorithm we performed is the K-nearest neighbors (KNN) algorithm, a supervised machine learning algorithm. This algorithm is a good candidate for our data set of users and ratings for a couple of reasons. First, KNN is a non-parametric learning algorithm. That is, it makes no underlying assumptions about the distribution of the data. Second, it groups data points based on the data around them. For a user-based recommendation system, such a grouping means that users with similar ratings will be grouped together.

The basic metric for similarity in KNN is euclidean distance between users. One issue with this metric is for every movie that one user has seen, if another user hasn't seen it, the difference between their ratings is not defined. A better metric is cosine similarity. This is the same as Pearson Correlation with one adjustment: similarity decreases as the subset of movies seen by both users decreases. This way the users that watch (and rate) many of the same movies and have similar ratings for those movies are more similar than the users that have similar ratings for a few movies.

Cosine Similarity:

$$sim(u,v) = \frac{\sum_i (r_{ui} - \bar{r_u})(r_{vi} - \bar{r_v})}{\sqrt{\sum_i (r_{ui} - \bar{r_u})^2}\sqrt{\sum_i (r_{vi} - \bar{r_v})^2}}$$

With this metric, we perform KNN in the standard way. Then with the k-nearest neighbors we generate a prediction according to the following equation [Pham et. al]. Two helpful features of this method are the base prediction and the weights for the update to the prediction. The base prediction is the average user rating for the user. Since many users have average ratings of 4 or higher, this is a much better base prediction than 2.5 or 3.0. Then, the update function is constructed as the weighted average of neighbors' deviations from the cluster average. Those more similar to the user we are generating a prediction for have more weight in changing the prediction.

Prediction using cosine similarity (Pham et. al):

$P_{i,j}$: Prediction for user i's rating of movie j

$r_{k,j}$: User k's rating of movie j (Equal to $\bar{r}_k$ if user k hasn't rated movie j)

$\bar{r}_i$: User i's average movie rating

$w_{i,k}$: Cosine similarity between user i and user k

N_i : The set of user i's neighbors

$$P_{i,j} = \bar{r}_i + \frac{\sum_{k \in N_i} (r_{k,j} - \bar{r}_k) w_{i,k}}{\sum_{k \in N_i} \mid w_{i,k} \mid}$$

3.1.2 Support Vector Machines

The second classification method we consider is the support vector machine algorithm, or SVM's. SVM's work by creating hyper-planes in n-dimensional space that partition the data. Our data is

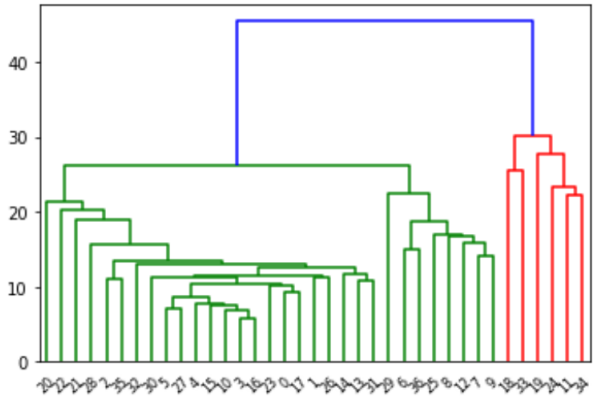
very high dimensional, due to how many movies and users we have. The kernel trick allows us to find separating hyper-planes even in high dimensions. SVM's are normally a binary classifier, so we must reformulate our problem to take advantage of that. Instead of predicting a numeric rating (e.g. 4.5), we predict a binary value of recommend (1) or do not recommend (-1). Thus the accuracy of the method is measured not in MSE but by % correctly recommended. SVM is a supervised learning method, so we create labels by interpreting a rating of ≥ 3 as "recommend", and a rating of < 3 as "not recommend". Our algorithm is set up to take one user at a time, and then classify every movie in the database as "recommend" or "do not recommend" for that user. One clear disadvantage of this method is that there is no ordering in our recommendations. With other methods, we would recommend a movie with a predicted rating of 4.5 before a movie with a predicted rating of 3.5. With this binary method, no such nuance is possible.

3.2 Clustering methods for user preferences

3.2.1 Hierarchical Agglomerative Clustering

The unique clustering algorithm is an hierarchical agglomerative clustering algorithm (HAC) for user clustering. Hierarchical clustering clusters points together by building a hierarchy of clusters that go together. The agglomerative refers to the bottom-up nature of the algorithm. HAC starts with the assumption that every observation is its own cluster, then at each step merges two clusters that have the least dissimilarity as it moves up the hierarchy. Below is the HAC hierarchy for user 51. As the hierarchy increases, the number of clusters decrease.

Dendrogram of Agglomerative Clustering



After hyper-parameter calculations, we choose to measure dissimilarity with euclidean distances for this method. With HAC there are several options for how we choose the distances to merge with, known as linkage. We chose to use the ward, which minimizes the variance within clusters. This prevents users with similar ratings in one area, but vastly different ratings in another, from being clustered together. Similar to our implementation of SVM, we treated this clustering method as binary: is the movie recommended to the user or not. Our user is again set up to take one user at a time, then classify every movie in the database as recommend or do not recommend.

3.2.2 K-Means

We first discuss the K-means clustering algorithm. K-means is loosely related to the k nearest neighbors algorithm discussed in the classification section, but fundamentally differs in implementation and goals.K-means clustering partitions our n users into k clusters, clustering each user based on the cluster with the nearest mean. This results in the data set being split into Voronoi cells, or clusters in which every point is assigned to the cluster which has the center closest to it.

Unfortunately, The standard K-means algorithm also does not work with blank values. Instead of altering the method for K-means, however we decided to fill the blank values with 2.5 before clustering. As discussed previously, we could have filled blank values with each user's average, but with a user-movie matrix density of .5%, 99.5% of the filled ratings would be that average. Then average rating would have overwhelming influence in deciding which users are similar – even if the movies they actually rated we very dissimilar.

The majority of our work with K-means is in determining how to use neighbor ratings to generate a predicted rating. In KNN the major factor in prediction user similarity. In K-means, we implement a 'popularity' measure. We create a prediction function similar to the function outlined above. The base prediction is the user's average rating. This is updated by the difference of the cluster's average rating for the movie and the cluster's average ratings over all the movies they've rated. This difference is weighted by movie popularity within the cluster. The more neighbors in a cluster who have rated a movie, the higher the weight for the prediction update. This works in both the positive and negative direction. Then if 30 users within a cluster rate a movie with an average rating of 4.98, that movie will have a higher predicted rating for the user of interest than a movie that only one member of the cluster rated with an average rating of 5.0.

Method for predicting user i's movie rating for movie j:
$P_{i,j}$: Prediction for user i's rating of movie j
$\bar{u}_i$: User i's average movie rating
$\bar{m}_j$: Cluster's average rating for movie j
$\bar{c}$: Cluster's average movie rating
C : Cluster corresponding to user i
M : All movies
$\mathbb{1}(r_{i,j})$: A function indiciating if user i has rated movie j
γ_j : $\frac{\sum_{i \in C} \mathbb{1}(r_{i,j})}{\sum_{i \in C, k \in M} \mathbb{1}(r_{i,k})}$

$$P_{i,j} = \bar{u}_i + (\bar{m}_j - \bar{c}) * \frac{\gamma_j}{\max_{k \in M} \gamma_k}$$

3.2.3 Non-negative Matrix Classification

The second clustering algorithm we consider is non negative matrix factorization, or NMF. NMF is a dimensionality reduction method. Given a user-movie matrix $\boldsymbol{X}$, It finds two matrices $\boldsymbol{W}$,$\boldsymbol{H}$ such that $\boldsymbol{X} = \boldsymbol{WH}$, with the added constraint that every entry of $\boldsymbol{W}$ and $\boldsymbol{H}$ must be positive. This is very useful in our application of movie ratings, where the entries of our matrix (the ratings) are always positive values. So it is very easy to intuitively interpret $\boldsymbol{W}$ and $\boldsymbol{H}$. The most important hyperperameter is r, which specifies the dimensions of $\boldsymbol{W}$ and $\boldsymbol{H}$. NMF is also a clustering algorithm, grouping common movies/users into r 'types'. If $\boldsymbol{X}$ is size m movies by n users, then $\boldsymbol{W}$ is size $m \times r$, and $\boldsymbol{H}$ is size $r \times n$. Each column of $\boldsymbol{H}$ contains scores for one user's affinity to each of the r 'types' of movies. Each row of $\boldsymbol{W}$ contains scores for one movie's affinity to each of

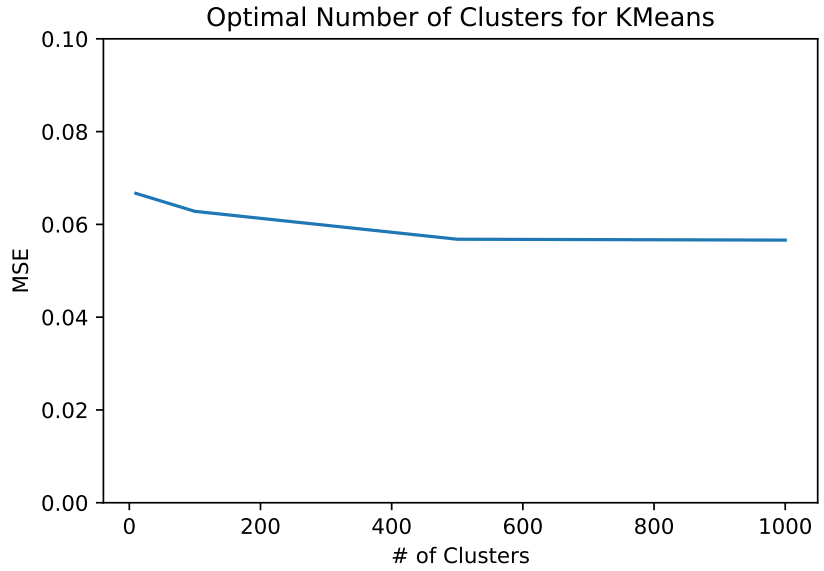
the r 'types' of movies, i.e. W_{ij} can be interpreted as the rating given to movie i by community j.

4 Results

We differentiate the results from our methods between methods that were binary-based and those that were not.

Of the binary-based methods, SVM out-performed HAC. SVM has an average accuracy of 80%, while HAC had an average accuracy of 76%. That is, SVM recommends a movie that a user would enjoy (rate higher than 2.5) 80% of the time, while HAC recommends one only 76% of the time.

For the non-binary based methods, we consider the mean-squared error. NMF performed with the highest accuracy for individual scores, with a .025 MSE. The next best-performing method was KMeans with .056 MSE. Pictured below is a graph that shows the relationship between number of clusters and MSE. MSE was already pretty accurate after 10 clusters, and after about 500 clusters, we found that MSE was not significantly reduced. Last was KNN with a MSE .0185. However, the last value was generated without cross-validation, which is necessary for us to be confident in its superiority.



As mentioned before, NMF clusters into r different types. After a Grid Search, we found that the a smaller amount of clusters (i.e. 10) worked best for predicting. Pictured below are 3 examples of the 10 groups we found. Listed are the top 10 movies that most exemplify that group (i.e. they had the highest relevancy score for that group). You can see that there are common genres in the group, but they are not restricted by the genre labels that came with the data set.

Top Movies for Several NMF Clusters

5 Analysis

As mentioned above, it is important to note that there are two ways of interpreting the recommendation system. One, a completely binary system, results in a fairly good accuracy. The results described above with MSE, however, are for a predicted system, where we estimate the exact rating a user would give particular movie, then base whether or not to recommend a film off of that score. The accuracy for this recommendation system is lower, in part due to the need for exact, accurate ratings. Ultimately, it is when we utilize NMF that we obtain the best results. This is probably due to the regularization we were able to do with NMF, which allowed us to avoid overfitting. The NMF is also very powerful in that it is able to find clusters, dimension reduce, and predict ratings at the same time.

6 Conclusion

We conclude that yes, it is possible to create a semi-accurate movie recommendation system using basic machine learning algorithm, at least for user-based filtering. Given the secrecy surrounding complex recommendation algorithms used by companies such as Netflix, the simplicity of implementing some of these methods is encouraging. Furthermore, user-based tags and user-based filtering can form the basis of an accurate recommendation system. We hope to expand our findings for content-based filtering.

References

[1] @booklusted2012netflix, title=Netflix: The Company and Its Founders: The Company and Its Founders, author=Lusted, Marcia Amidon, year=2012, publisher=ABDO

[2] @articlegomez2016netflix, title=The netflix recommender system: Algorithms, business value, and innovation, author=Gomez-Uribe, Carlos A and Hunt, Neil, journal=ACM Transactions on Management Information Systems (TMIS), volume=6, number=4, pages=13, year=2016, publisher=ACM

[3] @inproceedingssindhwani2010one, title=One-class matrix completion with low-density factorizations, author=Sindhwani, Vikas and Bucak, Serhat S and Hu, Jianying and Mojsilovic, Aleksandra, booktitle=2010 IEEE International Conference on Data Mining, pages=1055–1060, year=2010, organization=IEEE

[4] @articlecross2014streaming, title=Streaming film: How to serve our users, author=Cross, Cheryl and Fischer, Christine and Rothermel, Cathy, journal=Serials Review, volume=40, number=3, pages=154–157, year=2014, publisher=Taylor & Francis