Movie Recommendation Systems

Vivek Dalal[1], Raj Sankhe[2], Tej Sankhe[3]
INFO 6105 Data Science Engineering Methods and Tools
Fall 2018
Professor Nik Bear Brown

I. Abstract

There are inundated movies released every year and users also inhabit varied choice of movies. So, it is important that movie recommendation engines keep users engaged by recommending the movie of his/her choice. In this study, we have implemented and evaluated content based, collaborative based (Item-Item, User-Item, SVD, SVD++) filtering on Movielens data. Moreover, using content-based and SVD based filtering, we have built a hybrid model by stacking these models to increase the accuracy of movies recommended to the user. The study addresses the implementation and evaluation of models listed above.

II. Introduction

A. *Overview*

Significant dependencies exist between user and item-centric activity. A successful recommendation system explores this activity and produces relevant suggestions. For example, a user who is interested in a historical documentary is more likely to be interested in another historical documentary or an educational program, rather than in an action movie. In many cases, various categories of items may show significant correlations, which can be leveraged to make more accurate recommendations. A movie recommendation system is important due to its strength in providing enhanced entertainment and personalized user experience. Such a system can suggest a set of movies to users based on their interest or the popularities of the movies.

Companies are spending millions of dollars on building accurate recommendation model as this is a personalized campaign directed to targeted customers. The classic example of this is the Netflix Prize which was an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films. The competition was held by Netflix in 2006 and run for almost 3 years and the grand prize of US$1,000,000 was given to the team which bested Netflix's own algorithm for predicting ratings by 10.06%.

The most widely used recommendation systems are based on Collaborative Filtering (CF) and Content-based Filtering. Content based filtering leverages the similarities between movies. It assumes that if an user likes a movie then he will also like a similar movie. In the MovieLens dataset, we have movie genre information which we intend to use as the factor for finding similarity between movies. On the other hand, the Collaborative Filtering Recommender is entirely based on the past behavior and not on the context. They are of two types : Memory based and Model Approach.

B. *Dataset*

For this research project, we have used MovieLens 100K dataset. MovieLens datasets were collected by the GroupLens Research Project at the University of Minnesota. This dataset consists of:

- 100,000 ratings (1-5) from 943 users on 1682 movies
- Each user has rated at least 20 movies

The data was collected through the MovieLens website (movielens.umn.edu)

III. Methods

A. *Content-Based Filtering*

Content-based filtering, also referred to as cognitive filtering, recommends items based on a comparison between the content of the items and a user profile. The content of each item is represented as a set of descriptors or terms, typically the words that occur in a document. The user profile is represented with the same terms and built up by analyzing the content of items which have been seen by the user. In the context of movie recommendation system, we will consider genre as term of movie to see similarity. The concepts of Term Frequency (TF) and Inverse Document Frequency (IDF) are used in information retrieval systems and also content based filtering mechanisms (such as a content based recommender). They are used to determine the relative importance of a document / article / news item / movie etc.

Term Frequency (TF) and Inverse Document Frequency (IDF) TF is simply the frequency of a word in a document. IDF is the inverse of the document frequency among the whole corpus of documents. TF-IDF is used mainly because of two reasons: Suppose we search for **The rise of analytics** on Google, it is certain that **The** will occur more frequently than **analytics**. However, the relative importance of **analytics** is higher than **The** from search query point of view. In such cases, TF-IDF weighting negates the effect of high frequency words in determining the importance of an item (document).

We will consider genres as an important parameter to recommend user the movie he watches based on genres of movie user has already watched. To measure the distance or similarity between two movies, we use various distance measures. Here, we have used Cosine Similarity.

TF-IDF Score

$$TF - IDF\ Score = TF_{x,y} * IDF = TF_{x,y} * \log\frac{N}{df}$$

, where $TF_{x,y}$ is the frequency of keyphrase X in the article Y,
N is the total number of documents in the corpus.
df is the number of documents containing keyphrase X

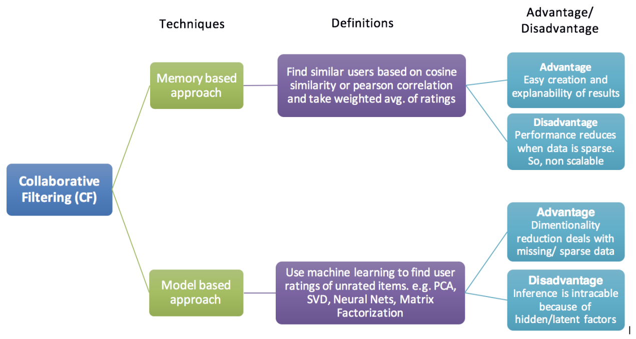
Cosine Similarity
The dot product between two vectors is equal to the projection of one of them on the other. Therefore, the dot product between two identical vectors (i.e. with identical components) is equal to their squared module, while if the two are perpendicular (i.e. they do not share any directions), the dot product is zero. The dot product is important when defining the similarity, as it is directly connected to it. The definition of similarity between two vectors A and B is, in fact, the ratio between their dot product and the product of their magnitudes.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

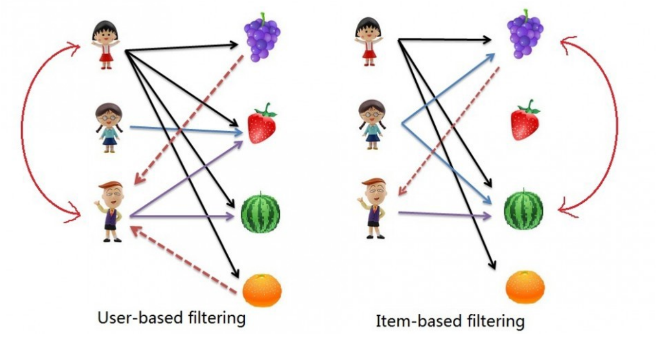
By applying the definition of similarity, this will be in fact equal to 1 if the two vectors are identical, and it will be 0 if the two are orthogonal. In other words, the similarity is a number bounded between 0 and 1 that tells us how much the two vectors are similar.

Evaluation
As the movie recommended by content-based filtering is based on genres, for evaluating model we have clustered movie based on groups of genres with the KNN classifier. The classifier label returned by KNN classifier to the movies recommended by the content based filtering is compare to the classifier label of the movie on which model recommends, the hit and error are calculated accordingly to measure accuracy.
Hit: true_count/total = **0.932**
Fault: false_count/total = **0.067**

B. Collaborative Filtering


Techniques
Definitions
Advantage/ Disadvantage
Collaborative Filtering (CF)
Memory based approach
Model based approach
Find similar users based on cosine similarity or pearson correlation and take weighted avg. of ratings
Use machine learning to find user ratings of unrated items. e.g. PCA, SVD, Neural Nets, Matrix Factorization
Advantage Easy creation and explanability of results
Disadvantage Performance reduces when data is sparse. So, non scalable
Advantage Dimentionality reduction deals with missing/ sparse data
Disadvantage Inference is intracable because of hidden/latent factors

*1) **Memory-Based Collaborative Filtering**:* Memory-based algorithms approach the collaborative filtering problem by using the entire database. Here we draw the similarity between User-User or Item-Item by finding out the distance between them. Distance is calculated by referring to some numeric value. For use case of movie recommendation, rating can be considered as a factor to calculate the distance.



User-Item Collaborative filtering In User-item filtering the distance between the items is calculated based on the users ratings (or likes, or whatever metric applies). When coming up with recommendations for a particular user, we look at the user that is closest to the chosen user and then suggest items which are liked by similar user but not watched by the chosen user. So, if you have watched and liked a certain number of movies we can look at other users who liked those same movies and recommend one that they also liked but which you might not have seen yet. Here the distance between users is calculated to infer the similarity between them. For a movie recommendation system, rating given by each user to the movie can be considered to create a vector for each user and then by using distance measures such as Euclidean Distance, Cosine distance, Pearson correlation or Jaccard Similarity we can find similarity between them.

Item-Item Collaborative Filtering Item-item collaborative filtering was originally developed by Amazon and draws inferences about the relationship between different items based on which items are purchased together. Here the distance between items is calculated to infer the similarity between them. For a movie recommendation system, rating given by each user can be considered to create a vector for each movie and then by using distance measures such as Euclidean Distance, Cosine distance, Pearson correlation or Jaccard Similarity we can find similarity between them.

Distance Measures

Cosine Similarity [III-A]

Pearson Correlation

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

Where

- cov is the Covariance

$$cov(X,Y) = E[(X-\mu_X)(Y-\mu_Y)]$$

- σ_X is the standard deviation of X
- σ_Y is the standard deviation of Y

Evaluation

A straightforward method of measuring the recommendation quality is to measure the following:

Mean Absolute Error (MAE) this method simply takes the mean of the absolute difference between each prediction and rating for all held-out ratings of users in the test set.

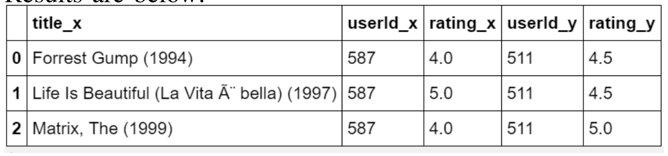
$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

Root mean squared error (RMSE) is a related measure that has the effect of placing greater emphasis on large errors: on a 5-star scale, the algorithm is penalized more for being 2 stars off in a single prediction than for being off by 1/4 of a star 8 times. It is computed like MAE, but squares the error before summing it.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T} (\hat{y}_t - y_t)^2}{T}}$$

Compare Ratings of similar users

Once User-User filtering gives similar users, common movies watched between them is found and rating is compared to see if our model has provided us similar users. Results are below:



	title_x	userId_x	rating_x	userId_y	rating_y
0	Forrest Gump (1994)	587	4.0	511	4.5
1	Life Is Beautiful (La Vita Ã¨ bella) (1997)	587	5.0	511	4.5
2	Matrix, The (1999)	587	4.0	511	5.0

Novel Approach using SVD

The movies recommended by the User-User and Item-Item filtering are taken and passed to SVD model to predict rating of them for given user. These ratings are further taken to calculate the hit ratio. If rating predicted by model is more than 3 than the recommendation is hit.

$$HitRatio = \frac{recommendedMoviesRating > 3}{totalNumberOfRecommendedmovies}$$

Hit ratio of model

Hit ratio of User-user collaborative filtering 0.7778
Hit ratio of Item-Item collaborative filtering 0.8888

Limitation of Memory based filtering

Memory-based collaborative filtering approaches that compute distance relationships between items or users have these two major issues:

- It does not scale particularly well to massive datasets, especially for real-time recommendations based on user behavior similarities which takes a lot of computations.
- Ratings matrices may be overfitting to noisy representations of user tastes and preferences. When we use distance based neighborhood approaches on raw data, we match to sparse low-level details that we assume represent the users preference vector instead of the vector itself.

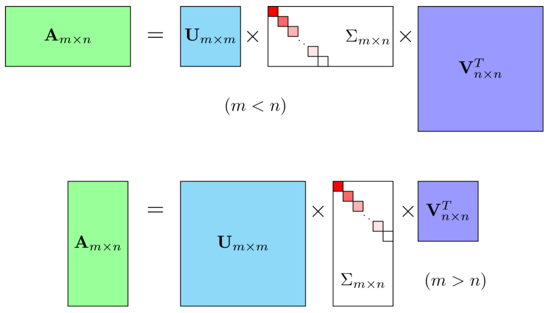
Thus, we need to apply Dimensionality Reduction technique to derive the tastes and preferences from the raw data, otherwise known as doing low-rank matrix factorization.

- We can discover hidden correlations/features in the raw data.
- We can remove redundant and noisy features that are not useful.
- We can interpret and visualize the data easier.
- We can also access easier data storage and processing.

*2) **Model-Based Collaborative Filtering**:* Model-based Collaborative Filtering is based on matrix factorization (MF) which has received greater exposure, mainly for dimensionality reduction and latent variable decomposition. Matrix factorization is widely used for recommendation systems where it can deal better with scalability and sparsity than Memory-based Collaborative Filtering. The goal of MF is to learn the latent preferences of users and the latent attributes of items from known ratings (learn features that describe the characteristics of ratings) to then predict the unknown ratings through the dot product of the latent features of users and items. When you have a very sparse matrix (The sparsity level of MovieLens100K dataset is 98.3%), with a lot of dimensions, by doing matrix factorization, you can restructure the user-item matrix into low-rank structure, and you can represent the matrix by the multiplication of two low-rank matrices, where the rows contain the latent vector. You fit this matrix to approximate your original matrix, as closely as possible, by multiplying the low-rank matrices together, which fills in the entries missing in the original matrix.

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$$

We have chosen models inspired by Singular Value Decomposition (SVD) to predict the user-item ratings matrix. SVD models are very popular primarily to their scalable nature and accuracy. SVD is a method of decomposing a matrix into three other matrices:



$\mathbf{A}_{m\times n} = \mathbf{U}_{m\times m} \times \Sigma_{m\times n} \times \mathbf{V}^T_{n\times n}$
$(m < n)$
$\mathbf{A}_{m\times n} = \mathbf{U}_{m\times m} \times \Sigma_{m\times n} \times \mathbf{V}^T_{n\times n}$
$(m > n)$

In the above figure, A is the input data matrix (users's ratings), U is the left singular vectors (user features matrix), Σ is the diagonal matrix of singular values (essentially weights/strengths of each concept), and V^T is the right singular vectors (movie features matrix). U and V^T are column orthogonal, and represent different things. U represents how much users like each feature and V^T represents how relevant each feature is to each movie.

To get the lower rank approximation, we take these matrices and keep only the top k features, which can be thought of as the underlying tastes and preferences vectors.

To build a robust recommender system, we need to develop models which factor in both explicit and implicit user feedback. For our Movielens dataset, a less obvious kind of implicit data does exist. The dataset doesn't only tell us the rating values, but also which movies users rate, regardless of how they rated these movies. In other words, a user implicitly tells us about her preferences by choosing to voice her opinion and vote a (high or low) rating. This reduces the rating matrix into a binary matrix, where 1 stands for rated, and 0 for not rated. Admittedly, this binary data is not as vast and independent as other sources of implicit feedback could be. Nonetheless, we have found that incorporating this kind of implicit data which inherently exist in every rating based recommender system significantly improves prediction accuracy. **SVD++** factors in this implicit feedback. Mathematically SVD++ can be represented as shown.

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right)$$

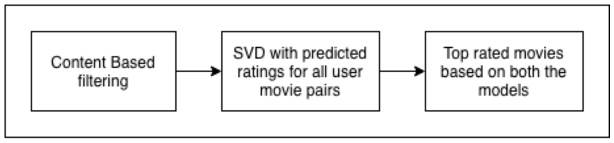
Where the y_j terms are a new set of item factors that capture implicit ratings. Here, an implicit rating describes the fact that a user u rated an item j, regardless of the rating value.

Evaluation

SVD and SVD++ models have been evaluated using RMSE. After fine tuning the important hyper parameters (number of factors, number of epochs, learning rate and regularization term) for SVD and SVD++, we got a RSME of **0.87** for SVD and **0.938** for SVD++.

C. Hybrid Model

To overcome shortcomings of an individual model, we have developed a hybrid model wherein we stack two different models. The resultant hybrid model gives higher accuracy and more relevant results. The movie recommended by Content-Based Filtering is passed to SVD model which predicts the rating the user will give to the recommended movie. Finally, we return the movies in the descending order of SVD predicted ratings.



Content Based filtering
SVD with predicted ratings for all user movie pairs
Top rated movies based on both the models

Stacked hybrid model

IV. Results

The SVD and SVD++ models are evaluated using RMSE, content based and collaborative based models are evaluated using hit ratio. In content-based filtering, we got model accuracy of **0.932**. In Memory based collaborative approaches, hit ratio of user-user filtering is **0.778** and for Item-Item filtering is **0.889**. In Model based collaborative approaches, the RMSE for SVD and SVD++ models are **0.87** and **0.938** respectively.

V. Code and Documentation

The code and dataset used for this project can be accessed at the below mentioned Github link
Movie Recommender System

The code is available under **MIT license** for use.

VI. Discussion

Nowadays, recommendation systems are increasingly gaining popularity due to their high number of applications. The users cannot manage all the information available on the internet. So, it is necessary for content providers to filter and customize the content shown to each user. Showing the right recommendations results in improved user engagement and in turn results in higher revenues. As per the algorithms discussed in this study, one recommendation system can have combination of algorithms as per different scenarios. So, if we have a metadata about the Movies i.e genre, artists, production house, then, on basis of that we can recommend movies to user. User-user filtering suffers from the cold start problem. The system needs to wait until the user makes some purchases and rates them. Only then similar users can be found and recommendations can be made. However, Item-Item filtering overcomes this problem by finding similarity between items and recommending relevant items to users Further, SVD and SVD++ approach the problem by Matrix Factorization and dimensionality reduction. These algorithms provide good predictions for ratings an user might give to movies which he hasn't watched. The Hybrid model we developed, filters the results further and provides accurate recommendation of movies.

References

[1] https://www.datacamp.com/community/tutorials/recommender-systems-python

[2] https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

[3] https://medium.com/@james_aka_yale/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223

[4] https://en.wikipedia.org/wiki/Singular_value_decomposition

[5] https://github.com/gpfvic/IRR

[6] https://surprise.readthedocs.io/en/stable/index.html

[7] https://www.quora.com/Whats-the-difference-between-SVD-and-SVD++

[8] https://blog.statsbot.co/singular-value-decomposition-tutorial-52c695315254

[9] https://www.quora.com/Whats-the-difference-between-SVD-and-SVD++

[10] https://medium.com/recombee-blog/machine-learning-for-recommender-systems-part-1-algorithms-evaluation-and-cold-start-6f696683d0ed

[11] http://www.awesomestats.in/python-recommending-movies/

[12] https://ieeexplore.ieee.org/document/8058367

[13] http://recommender-systems.org/content-based-filtering/

[14] https://hackernoon.com/the-fastest-way-to-identify-keywords-in-news-articles-tfidf-with-wikipedia-python-version-baf874d7eb16

[15] https://www.researchgate.net/publication/215470714