Personalize Movie Recommendation System
CS 229 Project Final Writeup

Shujia Liang, Lily Liu, Tianyi Liu

December 4, 2018

Introduction

We use machine learning to build a personalized movie scoring and recommendation system based on user's previous movie ratings. Different people have different taste in movies, and this is not reflected in a single score that we see when we Google a movie. Our movie scoring system helps users instantly discover movies to their liking, regardless of how distinct their tastes may be.

Current recommender systems generally fall into two categories: content-based filtering and collaborative filtering. We experiment with both approaches in our project. For content-based filtering, we take movie features such as actors, directors, movie description, and keywords as inputs and use TF-IDF and doc2vec to calculate the similarity between movies. For collaborative filtering, the input to our algorithm is the observed users' movie rating, and we use K-nearest neighbors and matrix factorization to predict user's movie ratings. We found that collaborative filtering performs better than content-based filtering in terms of prediction error and computation time.

Related Work

Content-based filtering makes recommendation based on similarity in item features. Popular techniques in content-based filtering include the term-frequency/inverse-document-frequency (tf-idf) weighting technique in information retrieval [1][2] and word2vec in natural language processing [3]. We move beyond the common use of tf-idf of finding similar movies to predict movie ratings and apply doc2vec, an extension of word2vec, to extract information contained in the context of movie descriptions. Other techniques include the Bayesian classifier [4][5], decision tree, neural networks, and so on [6].

Content-based filtering has the advantage of being able to solve the cold start problem when there hasn't been enough users or when the contents haven't been rated. However, it is limited to features that are explicitly associated with the items and requires extensive data collection process. In particular, automatic features extractions are difficult and expensive for multimedia data such as images, audio, and video stream [1][7]. Also, it does not distinguish between the quality of items. For example, a well celebrated movie and a badly received one are equally likely to be recommended if they share similar characteristics such as common phrases in movie descriptions.

Collaborative filtering addresses some of issues of content-based filtering – it recommends items that similar users like, and avoids the need to collect data on each item by utilizing the underlying structure of users' preference. There are two major approaches in collaborative filtering: the neighborhood model and latent factor models. The neighborhood model recommends the closest items or the closest user's top rated items. The latent factor model such as matrix factorization examines the latent space of movie and user features [8][9][10]. We modify this technique by introducing non-linear kernel and different update schemes and evaluate their performances. Matrix factorization can also be modified to incorporate time dependence in order to capture users' change in preference with time [11].

Many work has been done to combine the two techniques, giving rise to various hybrid approaches. One such approach is realized by incorporating content information in collaborative filtering, in which the content of a rated item is used to estimate the user's preference for other items [12][13]. This is particularly helpful when users rated only a small portion of the items population, as the information of the unrated items can be inferred from content-based filtering, instead of having just a missing value. Other hybrid methods include linear combination of predictions from both methods [14] or developing a unified probabilistic model [1][15].

Dataset and Features

We use the MovieLens dataset available on Kaggle [1], covering over 45,000 movies, 26 million ratings from over 270,000 users. The data is separated into two sets: the first set consists of a list of movies with their overall ratings and features such as budget, revenue, cast, etc. After removing duplicates in the data, we have 45,433 different movies. Table 1 is the top 10 most popular movies by their weighted score, calculated using the IMDB weighting [2]. We randomly split this dataset into an 80% training set and 20% test set for content-based filtering.

Movie Title	Avg Votes	Num Votes	Weighted Score
The Shawshank Redemption	8.5	8358.0	8.445871
The Godfather	8.5	6024.0	8.425442
Dilwale Dulhania Le Jayenge	9.1	661.0	8.421477
The Dark Knight	8.3	12269.0	8.265479
Fight Club	8.3	9678.0	8.256387
Pulp Fiction	8.3	8670.0	8.251408
Schindler's List	8.3	4436.0	8.206643
Whiplash	8.3	4376.0	8.205408
Spirited Away	8.3	3968.0	8.196059
Life Is Beautiful	8.3	3643.0	8.187177

Table 1: Top 10 most popular movies by weighted score

The other dataset used is the user-movie ratings, which contains user ID, movie ID and the user's 1-5 rating of the given movie. We represent this data as a matrix where one dimension represents users and the other dimension represents movies; the matrix is very sparse since most users have rated only a small portion of all the movies. Techniques such as nearest neighbor and matrix factorization are used to analyze this dataset.

We treat the task as a continuous ranking problem, rather than a classification problem of thumbs-up/thumbs-down, because it is more flexible and encodes more information. We can map ranking back to predicted rating, and at suggestion time, we will recommend the highest-ranked items to users. Due to computational constraints, we use a random subset of 100,000 ratings data in our project. We randomly split the set of user-movie rating pairs into an 80% training set and 20% test set. We minimize the sum of squared error of between the predicted ratings and the actual ratings.

Methods

We run the baseline of content-based filtering using TF-IDF, calculates the weighted similarity between bag of words. The importance of a term t is positively correlated with the number of times it appears in document d, but the frequency of term t among all documents is inversely related its ability to distinguish between documents. Thus we calculate the frequency of word t in document d, weighted by the inverse of frequency of t in all documents: $\text{tf-idf}(t) = tf(t,d) * idf(d) = tf(t,d) * \log \frac{|D|}{1+|d:t \in d|}$, where $|D|$ is the length of the document, and $|d : t \in d|$ is the number of documents where t appears.

We calculate the cosine similarities between movies based on movie features. We create two similarity matrix – one using movie descriptions and tagline, and the other using actor names, director names, and keywords. We remove the space between first and last names to avoid confusion on different people with the same first name. Movie descriptions and taglines are often long and extensive, whereas actor names are only one word; combining them into one word vector would overweight descriptions and underweight actor and director names. Thus we separately calculate two cosine similarity matrix and combined them using a weight that minimizes training set error.

TF-IDF looks for the frequency of the exact word in a document and could not pick up on synonyms or similar descriptions, so it produces very low similarity scores across all movies. Word2vec is based on a distributional hypothesis where words appear in the same context tend to have similar meanings. To take the context of a word into account, we use doc2vec[16], which is an extension of the word2vec model that added a document-unique feature vector representing the entire document. We use word2vec to calculate movie similarity based on movie descriptions, and preprocess the data by removing the punctuations and convert all the words to lower case to avoid confusion. We also remove stopwords such as "the", "and", "a", and "of"

[1] https://www.kaggle.com/rounakbanik/the-movies-dataset/version/7

[2] https://help.imdb.com/article/imdb/track-movies-tv/faq-for-imdb-ratings: $\frac{v}{(v+m)} * R + \frac{m}{(v+m)} * C$, where R is average movie rating (votes), v is the number of votes for the movie, m is the minimum votes required to be listed in the Top Rated list, which we set as 90 percentile of votes and roughly equals to 160, and C is the mean vote across all movies.

that do not provide information on the context. Movie names are concatenated as a single word. We use the movie feature similarity obtained from TF-IDF and word2vec to predict ratings of movie i using ratings of movie j and the similarity between movies i and j:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_j \text{sim}(i,j)(r_j - \mu_u)}{\sum_j \text{sim}(i,j)} \tag{1}$$

Next, we apply K-nearest neighbors as a baseline to collaborative filtering. In item-to-item collaborative filtering, we predict user u's rating of movie i to be the weighted sum of movie i's rating from the k nearest users based on their ratings similarity score [17]. Since some people might be more critical of movies, we adjust the predicted rating based on their average ratings:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u,v)(r_{vi} - \mu_v)}{\sum_{v \in N_i^k(u)} \text{sim}(u,v)}$$

where $\text{sim}(u,v)$ is the rating similarity between user u and v, r_{vi} is user v's rating of movie i, and $N_i^k(u)$ is the k nearest neighbors of user u's ratings. Similarly, we can predict user u's rating of movie i using KNN for item-to-item collaborative filtering, adjusted for a given movie's average rating:

$$\hat{r}_{ui} = \mu_i + \frac{\sum_{j \in N_u^k(i)} \text{sim}(i,j)(r_{uj} - \mu_j)}{\sum_{j \in N_u^k(i)} \text{sim}(i,j)}$$

Matrix factorization is another useful method in building a recommendation system[4]. Users' movie ratings form a sparse matrix, where single user only rates a tiny portion of the whole movie set. User ratings can be decomposed as $\hat{r}_{ui} = q_i^T p_u$, where entries in q_i^T measure the extent to which the movie possesses certain characteristics, and entries in p_u measure the extent of "interest" that user has in the movie on corresponding characteristics. To learn these two factor vectors, we minimize the regularized squared error based on the training set:

$$\min_{q^*,p^*,b^*} \sum_{(u,i) \in s} (r_{ui} - \mu - b_u - b_i - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2 + b_u^2 + b_i^2)$$

where s is the set of (u, i) pairs for r_{ui} in training set and λ controls the extent of regularization to penalize the magnitudes of learned parameters and thus avoid overfitting. $bias = \mu + b_i + b_u$ incorporates global average μ, bias from movie b_i and bias from user b_u.

We minimize the RMSE by stochastic gradient descent. The user and movie factors are initialized randomly from a uniform distribution, and baselines are initialized to zero. Then for each user-movie rating in the training set, we update each parameter according to the error between the predicted rating and true rating. For example, the user factor p_u is updated by $p_u = p_u + \alpha\big((r_{ui} - \hat{r}_{ui})q_i - \lambda p_u\big)$, where α is the learning rate. Similar rules apply to all other parameters. Note that the entire factor vector for user u is updated. After all the examples in the training set are seen, one "epoch" has finished, and we repeat the "epochs" until the change in training RMSE converges. Alternatively, instead of updating the entire factor vector for users and movies simultaneously, we first update the first entry, $p_{u,1}$ and $q_{i,1}$, by going through several epochs until convergence. Then we move on to $p_{u,2}$ and $q_{i,2}$, and so on. We can also change how the use factors and movie factors interact by kernelizing the term $q_i^T p_u$. The other kernels we considered are radial basis function kernel (rbf) and sigmoid function.

Throughout this report we use root mean squared error (RMSE)= $\sqrt{\frac{1}{|s|}\sum_{(u,i)}(r_{u,i} - \hat{r}_{u,i})^2}$ to evaluate the performance of the algorithm, where $r_{u,i}$ is the actual rating given by user u to movie i and $\hat{r}_{u,i}$ is the predicted rating.

Results and Discussion

For content based filtering, we use the movie similarity matrices generated by TF-IDF and word2vec to predict an user's movie ratings (Formula (1)). To combine the two similarity matrices from TF-IDF, we calculate their weighted sum. Evaluating the performance on a training set consisting of 80% randomly selected user-movie rating pairs, we see that the RMSE is smallest for the weight $w_1 = 0.7, w_2 = 0.3$ as shown in figure 1. We then run the prediction algorithm with these weights on the test set. Figure 3 illustrates the performance of our algorithm in predicting users' rating for movies. Each bin represents user-movie pair that has rating in the specific range. The blue bars represent the portion for which our algorithm's prediction is within ± 0.75 of the true rating, and the the green bars represent the portion for which our algorithm's prediction is outside of ± 0.75 of the true rating. We see that our algorithm performs well for user-movie pairs with ratings higher than 3, which constitute the majority of the data points. The RMSE on the test set is 1.052.

For doc2vec, we use the Distributed Memory version of Paragraph Vector (PV-DM) model and include some noise words to increase the overall robustness. We have experimented with various starting learning rates (Table 3), and $\alpha = 0.025$ minimizes RMSE to 0.927. The learning rate starts with 0.025 and linearly decrease to 0.001. Using this model, we could

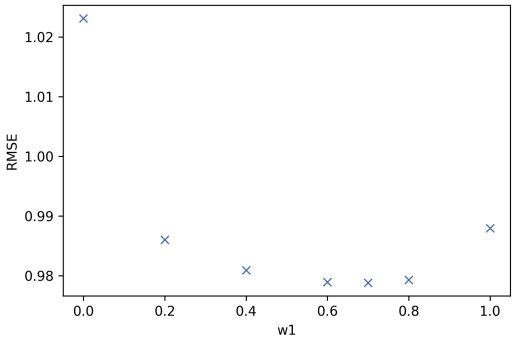

Figure 1: RMSE for different values of w1. (w2 = 1 - w1)

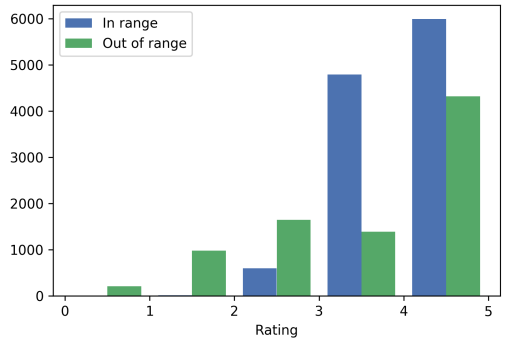

Figure 2: Histogram for predictions using TF-IDF similarity.

also query keywords or movie names for the most similar movies based on the similarity score calculated from the movie overview. Table 2 shows the top 5 most similar movies to Skyfall.

Similar Movies	Sim Score
Octopussy	0.8358
Transporter	0.8298
Safe house	0.8287
Unlocked	0.8106
Undercover Man	0.8062
Push	0.8033
Sniper 2	0.8007
Patriot Games	0.8005
2047 Sights Death	0.7952
Interceptor	0.7951

Table 2: Movies most similar to Skyfall

Learning Rate	RMSE
0.020	0.927403
0.025	0.927003
0.030	0.927009
0.035	0.927358
0.040	0.927994
0.05	0.929483
0.1	0.938572
0.2	0.950744
0.4	0.970215

Table 3: Learning rate with test RMSE

For collaborative filtering, we run item-to-item and user-to-user CF using KNN. We grid searched for the best k value (Figure 3) and found that the test RMSE overall decreases as k increases. The test RMSE stabilizes around $k = 20$, and it does not monotonically decrease. $k = 43$ yields the lowest test RMSE of 0.9079 for item-to-item CF and $k = 28$ yields the best test RMSE of 0.9203 for user-to-user CF, though other values of $k > 20$ yield the same results to three decimal places, so the results are fairly robust. With $k = 43$, the item-to-item CF training RMSE is 0.2900 and the test RMSE is 0.9079. With $k = 28$, the user-to-user CF training RMSE is 0.2830 and the test RMSE is 0.9203. The differences between training and test RMSE suggest some degrees of overfitting. We have tried increasing k up to 300, but the gap between test and train RMSE remains. Figure 4 and 5 show the rating prediction histogram from the KNN model. Again, the model performs well for ratings that are larger than 3 due to an imbalance in the ratings.

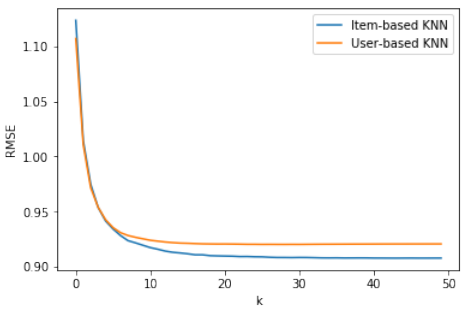

Figure 3: KNN training RMSE with different values of k

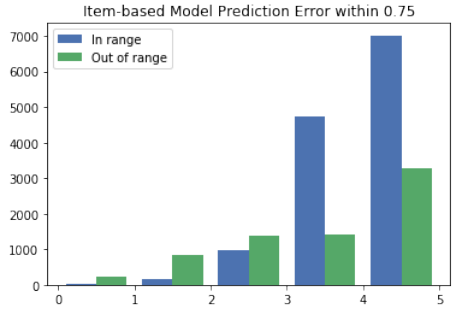

Figure 4: KNN Item-based Ratings Prediction Histogram

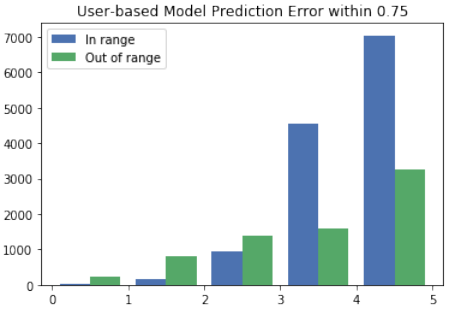

Figure 5: KNN User-based Ratings Prediction Histogram

For matrix factorization, we run the algorithm with various parameter settings to find the optimal values. Figure 6 shows

the RMSE for different number of factors used in the model. We see that using 14 factors is optimal. The optimal values for other parameters such as the regularization constant and learning rate are found in similar fashion. We have found that using 14 factors, a regularization factor $\lambda = 0.01$ and learning rate 0.001 achieves the lowest trainig RMSE of 0.832.

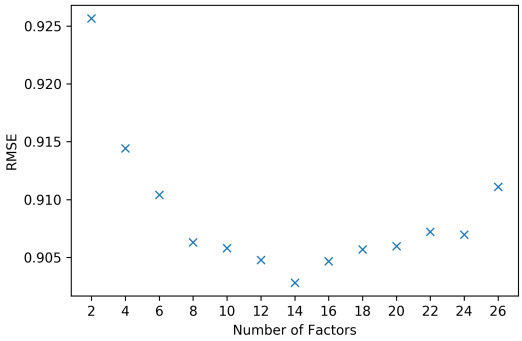

Figure 6: RMSE for different number of factors. 14 factors seems to be the optimal.

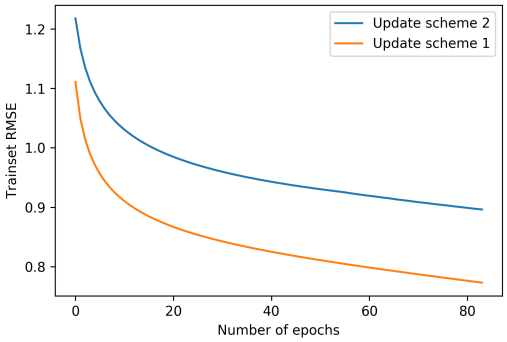

Figure 7: RMSE as a function of number of epochs for different update schemes

We also compare the two update scheme described in the method section. In the second update scheme, we minimize the error by updating one factor (individual entry of the user/movie vector) at a time. The RMSE is reduced significantly after the first few factors have completed updating. This shows that the most important characters of movie (or user preference) are contained within the first few factors. Figure 7 shows the RMSE as a function of number of epochs for the two update schemes. We see that the first update scheme, where all factors are updated simultaneously, converges faster and gives better result. We also compare the performance of the algorithm using different kernels, as shown in Figure 8. We thus choose to use the linear kernel for our model, and the test RMSE is 0.9039. From Figure 9, we see that our algorithm performs well for ratings that are larger than 3. However due to the small sample size of ratings smaller than 3, even when the true rating is low, the algorithm still tends to predicts high ratings and thus performs poorly.

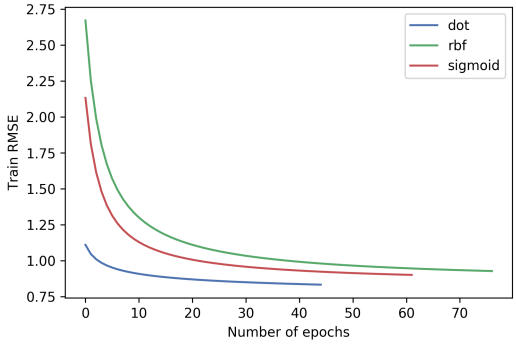

Figure 8: RMSE as a function of number of epochs for different kernels.

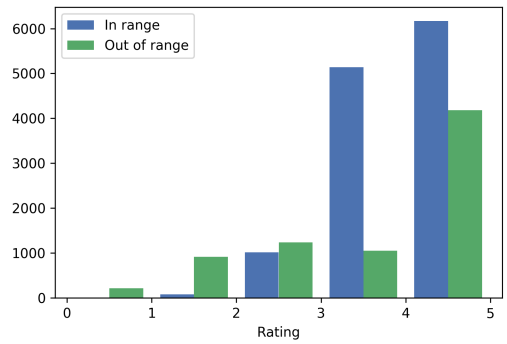

Figure 9: Histogram for predictions using matrix factorization.

Conclusion

We have explored both content-based and collaborative filtering for building the recommendation system. Collaborative filtering overall performs better than content-based filtering in terms of test RMSE. Additionally, content-based filtering is computationally more expensive than collaborative filtering, as it involves extensive processing of text features. Therefore collaborative filtering is preferred. All codes are available at Google Drive link.

For future work, we would like to address the skewed prediction caused by imbalance in the number of low ratings compared to high ratings. We would also explore ways such as regularization to address the overfitting issue in KNN. Additionally, our recommendation system can be improved by combining content-based filtering and collaborative filtering. Possible techniques include incorporating content features as additional features in collaborative filtering, or vice versa, decision trees, and neural network. We could also add in a time dimension to our model to capture the change in user preference over time.

Contribution

Lily cleaned the data, conducted literature review, and worked on KNN and doc2vec. Tianyi cleaned the data, reviewed literature, and worked on TF-IDF and matrix factorization. Shujia reviewed literature, coded up the matrix factorization algorithm with Tianyi, and made poster.

References

[1] A. Tuzhilin and G. Adomavicius. *Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions*. IEEE Transactions on Knowledge & Data Engineering, vol. 17, no.6, pp. 734-749, 2005.

[2] G. Salton. *Automatic Text Processing.* Addison-Wesley (1989)

[3] Caselles-Dupré, Hugo et al. *Word2vec applied to recommendation: hyperparameters matter*. RecSys. 2018.

[4] R. J. Mooney, P.N. Bennett, and L. Roy. *Book Recommending Using Text Categorization with Extracted Information*. Proc. Recommender Systems Papers from 1998 Workshop, Technical Report WS-98-08, 1998.

[5] M. J. Pazzani and D. Billsus. *Learning and Revising User Profiles: The Identification of Interesting Web Sites*. Machine Learning, vol. 27, pp. 313-331, 1997.

[6] M. Pazzani and D. Billsus, *Learning and Revising User Profiles: The Identification of Interesting Web Sites*. Machine Learning, vol. 27, pp. 313-331, 1997.

[7] U. Shardanand and P. Maes. *Social Information Filtering: Algorithms for Automating 'Word of Mouth'*. Proc. Conf. Human Factors in Computing Systems, 1995.

[8] D. Billsus and M. J. Pazzani. *Learning Collaborative Information Filters*. Proceedings of the International Conference on Machine Learning. 1998.

[9] Y. Koren, R. Bell, C. Volinsky. *Matrix Factorization Techniques for Recommender Systems*. Computer. 42, 8. 2009.

[10] Simon Funk, Netflix prize https://sifter.org/ simon/journal/20061211.html

[11] Y. Koren. *Collaborative Filtering with Temporal Dynamics*. Proc. 15th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD 09), ACM Press, 2009, pp. 447-455.

[12] M. Pazzani. *A Framework for Collaborative, Content-Based, and Demographic Filtering*. Artificial Intelligence Rev., pp. 393-408, Dec. 1999.

[13] I. Soboroff and C. Nicholas. *Combining Content and Collaboration in Text Filtering*. Proc. Int'l Joint Conf. Artificial Intelligence Workshop: Machine Learning for Information Filtering, Aug. 1999.

[14] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin. *Combining Content-Based and Collaborative Filters in an Online Newspaper*. Proc. ACM SIGIR '99 Workshop Recommender Systems: Algorithms and Evaluation, Aug. 1999.

[15] A. Popescul et al. *Probabilistic Models for Unified Collaborative and Content- Based Recommendation in Sparse-Data Environments*. Proc. 17th Conf. Uncertainty in Artificial Intelligence. 2001.

[16] Doc2vec documentation. https://radimrehurek.com/gensim/models/doc2vec.html

[17] Surprise Documentation. https://surprise.readthedocs.io/en/stable

[18] J. H. Lau and T. Baldwin. *An empirical evaluation of doc2vec with practical insights into document embedding generation*. arXiv preprint arXiv:1607.05368. 2016.