MovRec: Recommendation System for Movies

I. Introduction

Personalized recommendations have become an integral part of an user's online experience. Most of the major companies like Amazon, Google, Netflix, Etsy, Facebook, Tinder etc. employ multiple state-of-the-art recommender systems to introduce their users to new content and experiences that they might not come across on their own. These advancements have also been aided by rapid growth in consumer data collection and availability. These algorithms now account for a major chunk of these companies' revenue and users. According to a recent report, Netflix's recommender system drives around 80% of the traffic on their website and brings in about 1 billion dollars of revenue per year.

Over the years, recommender systems have been built with two strategies (or a combination of them) - *Content-Based* and *Interaction-Based*. In content-based approaches, user and item profiles are created to characterize them and their behaviours. For movie recommendations, the movie can be modelled by its genre, runtime, cast, language etc. and the user can be modelled by its age, demographic, gender etc. However, this is a very data-intensive process requiring a huge collection of data. The interaction-based approaches (*Collaborative Filtering*) only relies on the user-product interaction history to make recommendations. They recommend products which are either similar to what a user has interacted with earlier or are products that similar users have interacted with. As such, the amount of data needed is fairly reduced. In recent times, there have been hybrid approaches introduced like Factorization Machines or using learning-based approaches from Natural Language Processing, Deep Learning and Reinforcement Learning.

In this report, we provide a comparative study of various recommendation systems built for recommending movies on the Amazon Movie Reviews Dataset.[1] The dataset contains 1 million user reviews for around 350k users and 32k movies. We portray how various collaborative filtering techniques from similarity-based, latent factors, Factorization Machines to Neural Collaborative Filtering address previous shortcomings and improve recommendation performance. We focus more on Factorization Machines and Neural Collaborative Filtering.

[1]The dataset can be found at: https://snap.stanford.edu/data/web-Movies.html

II. Dataset and Tasks

A. Data Description

Amazon movie review dataset contains movie reviews from amazon for a period of 15 years that includes approximately 8 million reviews. Each data record has movie ID, user Id, movie rating and review text by user, review helpfulness, timestamp, and other fields like review summary and user profile name. For reporting all our results we make use of the first 1 million reviews due to computational limitations. Table I describes the statistics for the first 1 million interactions.

TABLE I: Data Statistics

Information Type	**Value**
Unique Users	329577
Unique Movies	28595
Min Timestamp	13-10-1997
Max Timestamp	10-25-2012
Mean Global Rating	4.08

Out of the first 1 million reviews, while only considering productId, userId and movie rating dataset contain 8420 duplicate entries which make 99.15 records as original that is used for modelling purpose. The last 100,000 records are kept for testing purposes for reporting metrics corresponding to different recommendation techniques. We also create a validation set with 100,000 samples to validate our model performance and stop overfitting.

B. Predictive Task Description

The dataset contains user reviews, each review has a rating which the particular user has given to a particular movie. We define our predictive task as - predicting a rating for a movie for a given user. We aim to predict ratings given a new user-movie pair. Once we are successfully able to predict ratings for a user, we can make use of these ratings to recommend movies to the user which we think he will rate highly.

To put it more concretely, we want to learn a function f such that the following equation holds

$$f(u,m) = \hat{r}_{u,m} \tag{1}$$

where $\hat{r}_{u,m}$ is the predicted rating for user u and movie m. Note, the actual rating $r_{u,m}$ may not necessarily be equal to $\hat{r}_{u,m}$. However, we want to minimize the difference between the predicted value $\hat{r}_{u,m}$ and the true value $r_{u,m}$. These errors can be quantified as the total loss that is incurred. The mean squared error between the true and predicted values is used as the loss.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(r^i_{u,m} - \hat{r}^i_{u,m})^2 \tag{2}$$

where N is the total number of interactions and i is the interaction being considered currently.

The loss function value also provides us with a tool to evaluate different methodologies and choose the best among them. The method which generalizes the best on unseen data i.e. which has the lowest MSE on the *test* set, should be chosen.

However, Mean Squared Error is not a robust metric for evaluating recommender systems prediction. This is because even if the squared errors are similar for multiple items, it is impossible to differentiate between higher-rated items and the items with lower ratings. Errors for higher-rated items are worse when the predicted rating is less and this distinction is hard to follow when the heuristic is MSE. Moreover, MSE assumes that the errors are normally distributed, which is not always the case. The more popular items tend to dominate the MSE calculation. Hence, the model with a lower MSE does not mean that it is a better recommendation model.

To circumvent the above issues, we use a different metric - Area under the ROC curve (AUC), as it is easier to optimize and comprehend. AUC is a ranking-based heuristic which weighs the models on how high purchased/positive items are being rated. The following equation helps us find the AUC for a recommender.

$$AUC(u) = \frac{1}{|I_u||I - I_u|}\sum_{i \in I_u}\sum_{j \in I - I_u} R_{ij} \tag{3}$$

where,

$$R_{ij} = \delta(Rank_u(i) < Rank_u(j)) \tag{4}$$

and I_u is the set of items the user u has interacted with, I is the set of all items and δ is the dirac-delta function.

Further, the AUC across all users U is,

$$AUC = \frac{1}{|U|}\sum_{u \in U} AUC(u) \tag{5}$$

where U is the set of all users.

In this work, we focus on Matrix Factorization methods (Factorization Machines and Neural Collaborative Filtering). We define multiple baselines to evaluate our model against. These include - naive mean predictor, similarity-based collaborative filtering methods and latent factor models. All these methods only rely on user interactions which saves us a considerable amount of feature engineering. However, since Factorization Machines allow us to incorporate additional features, we show that spending some time to incorporate relevant features might be worthwhile with respect to the final performance.

III. Related Works

Recommender systems have become especially popular today due to the increase in online shopping on websites such as Amazon, social networking and online entertainment on Facebook and Netflix, etc. Especially since the famous Netflix Prize competition commenced years ago, collaborative filtering (CF) techniques have been very successful in building recommender systems.

In [2], the authors have started out with the concept of latent feature representation to accurately try to predict the item ratings based on the user, and item interactions. The model introduced consists of user and item biases and latent features for both users and items. Taking inspiration from the results and reception of this paper we have introduced a similar model with separate regularizers for the biases and latent features. As in [2], a comparative analysis has been performed on multiple datasets, we rather pick one dataset - 'Amazon Movie Reviews Dataset' and try to improve our predictions.

Many CF-based approaches are built on matrix factorization [3], in which the user, item interaction matrix is broken down into two low-rank matrices corresponding to latent features of users and items. The inner product between the two user and item features is then used to estimate the rating which a user might give to an item.

In recent times, dense neural networks have received immense appreciation and success in domains like computer vision and natural language processing. However, deep neural networks have not been elaborately explored in depth and range in the domain of recommendation systems. In [8], the authors have replaced the dot product of latent features with a dense architecture which learns an arbitrary function from the interactions. They have presented a framework named Neural Collaborative Filtering (NCF). NCF is a general approach and expresses and generalizes the concept of matrix factorization in its framework. To deal with and model non-linearities, a multi-layer dense neural net is leveraged to learn the user-item interactions.

In [10], the authors have discussed the recent setting in recommendation systems where, more users, items and rating data are consistently being added to the model, causing many variations and shifts in the underlying relation between users and items to be rated/recommended. This problem is known as 'concept drift' or 'temporal dynamics in recommendation systems'. CF also includes methods like neighborhood models which utilize the collaboration of the ratings by similar users to come up with recommendations [9]. In general, a neighborhood model can adapt in order to model dynamic changes in Recommendation Systems over a period of time, by integrating either time-dependent algorithms [11], [12], [13], [14] or by making use of time-independent algorithms [15], [16], [17], [18]. In [10], the authors have presented several studies on time-dependent and time-independent algorithms for neighborhood-based RS. They have used popular methods/techniques like K Nearest Neighbours(KNN) and Long Short-Term Memory(LSTM) models. Taking inspiration from this, we have also managed to incorporate 'Timestamp' as a feature and were able to outperform all the other models on the basis of MSE and AUC scores.

IV. Exploratory Data Analysis

This section deals with demonstrating and exploring various facets of the data.

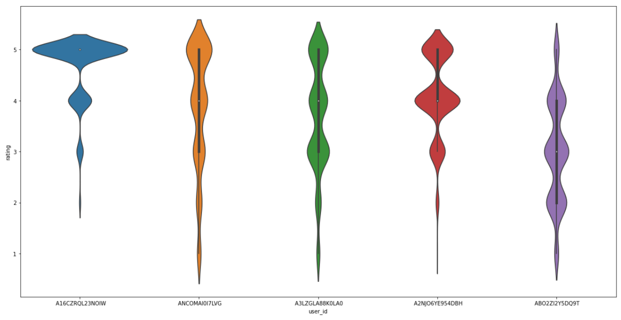

Fig. 1: Variation of rating for top 5 active users

Fig. 1 shows the rating distribution of the top 5 users (in terms of the number of reviews provided). From the violin plot graph, it is visible that the 1st user tends to rate most of the movies as 5 stars whereas 5th user in the plot evenly rates moves from 4 to 2 stars. Fig. 2 shows the variation of average rating at the year-month granularity across the complete data timestamps. The figure shows the high average rating for the time before the year 1999 and then a gradual decrease in average ratings till the year 2007 with an upward trend post that. Fig. 3 shows the variation of rating for different years each represented by a separate line plot across the 12 months dimension. A general trend of decrease in the monthly average rating is seen from May to August months which is visible in Fig. 3 for the years 2008 to 2012.

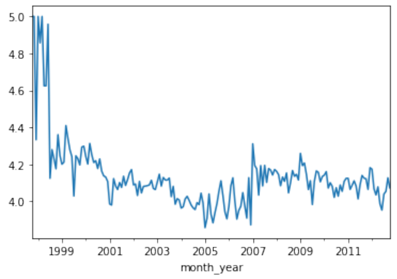

Fig. 2: Variation of average rating along year month timestamp

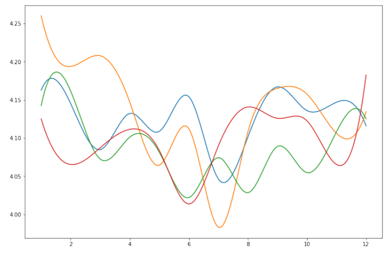

Fig. 3: Variation of average rating along month wise from 2008 to 2012

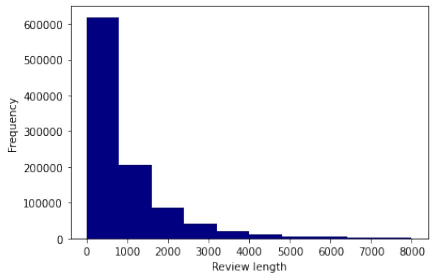

Fig. 4: Histogram of length of reviews

TABLE II: Top 5 users with the highest number of reviews

User Id	Reviews(Count)
A16CZRQL23NOIW	1174
ANCOMAI0I7LVG	1160
A3LZGLA88K0LA0	1135
A2NJO6YE954DBH	1096
ABO2ZI2Y5DQ9T	925

We also looked at the text of the review which was present with each interaction. Fig. 4 shows a gradual decrease in the frequency of reviews with an increase in review length. A time variation of the number of reviews is shown in Fig. 6 which suggests an increasing trend. The distribution of ratings is shown in Fig. 5 where five stars rating has the most number of counts and 2 stars rating has the least number of counts. If we look at the change in length of review with respect to ratings it is observed that 5-star ratings have higher outliers of lengthier reviews shown in Fig. 7 whereas on an average 5-star and 1-star reviews are shorter than others (Fig. 8). The same trend follows if we look at the average punctuation use across ratings (Fig. 8).

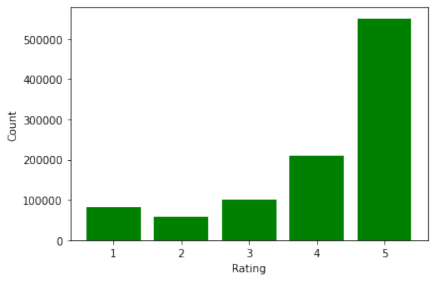

Fig. 5: Histogram of ratings

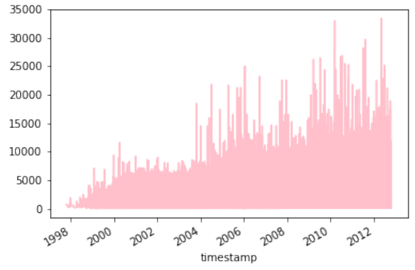

Fig. 6: Volume of reviews vs time

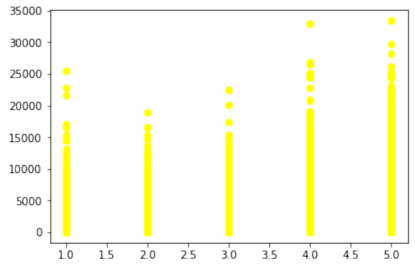

Fig. 7: Length of review vs rating

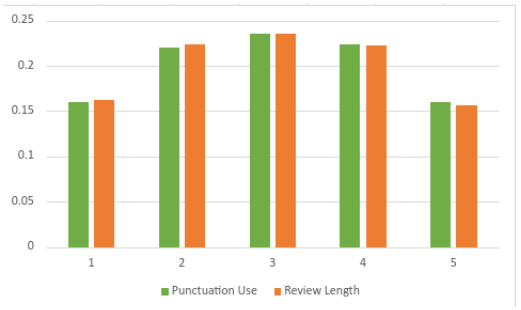

Fig. 8: Variation of punctuation use and review length vs rating

One of the interesting findings about the dataset (1st 1 million reviews) is that there are 34.16% of reviews (341632 out of 1000000 reviews) that are duplicates when considered in conjunction with timestamps which means many users provide the same review for different movies. Table III shows the top 5 users with the highest duplicate review percentages.

TABLE III: Users with highest number of duplicate reviews

User Id	**Duplicate Reviews**(%)	**Reviews**
A3NJF6DH8FETN1	95.45%	44
AY39CRKDV5L8S	95.31%	64
A2BG5QH12A411C	94.11%	17
A1BA9WTDTJQA2M	93.33%	15
A34RW2KH2ZQSNT	92.59%	27

V. Models

In this section, we describe how all our baselines were created in addition to talking about Factorization Machines and Neural Collaborative Filtering development.

A. *Always Mean Predictor (AMP)*

This is a naive predictor which always predicts the mean rating μ for any given user and movie pair. It is not dynamic with respect to the user or movie, hence it is the easiest to learn and leaves a lot to be desired in terms of performance.

$$\mu = \frac{1}{N}\sum_{i=1}^{N} r_{u,m}^{i} \tag{6}$$

B. *Collaborative Filtering*

Ideally, we want to account for users' preferences while recommending it movies. Collaborative filtering methods incorporate item similarity and user similarity to make their predictions. They are of two main categories

1) Item-Based: The item-based approach to collaborative filtering predicts results based on the similarity of the current item with all the other items a user has consumed. It will recommend movies or predict higher ratings for movies which are similar to movies a user has already consumed. We build an item-based collaborative filtering model (called *CF1*) which predicts ratings as follows

$$\hat{r}_{u,i} = \overline{R}_i + \frac{\sum_{j\in I_u\setminus\{j\}}(r_{u,j} - \overline{R}_j)\cdot Sim(i,j)}{\sum_{j\in I_u\setminus\{j\}} Sim(i,j)} \tag{7}$$

where I_u is the set of movies the user u has already rated, $\overline{R}_i$ is the average rating for the movie i and $Sim(i,j)$ is the measure of similarity between movie i and movie j. There are various well-known methods to define similarity like Jaccard, Pearson, Cosine etc. We use the Jaccard similarity which is defined as

$$Sim(i,j) = \frac{U_i \cap U_j}{U_I \cup U_j} \tag{8}$$

where U_i and U_j are the set of users who have interacted with movie i and j respectively.

2) User-based: We also create a user-based collaborative filtering model, *CF2*. This method predicts higher ratings for a given movie and user if that movie has been highly rated by users similar to the given user. This can be quantified as follows

$$\hat{r}_{u,i} = \overline{R}_u + \frac{\sum_{j \in U_i \setminus \{j\}} (r_{j,i} - \overline{R}_j) \cdot Sim(u,j)}{\sum_{j \in U_i \setminus \{j\}} Sim(u,j)} \tag{9}$$

where $\overline{R}_j$ denotes the mean rating given by a user j. We use the Jaccard similarity to find the similarity between users u and j

$$Sim(u,j) = \frac{I_u \cap I_j}{I_u \cup I_j} \tag{10}$$

Both *CF1* and *CF2* provide intelligent baselines which achieve great performance. However, they are not very scalable if the dataset is huge, because we need to calculate similarity for all users/movies and one new interaction changes the prediction for every other user-movie pair.

C. Latent Factor Models

Similarity-based models rely heavily on the similarity function and the heuristics used to define the rating. We can do better if we do supervised learning to model both user and item in a latent space, hence latent factor models. We learn the user behaviour and movie traits in a hidden small dimensional latent space. If the users and movie are similar in the latent space, we will predict a higher rating for them. This is inspired by the fact that singular value decomposition of the interaction matrix can help us learn low-dimensional representations of users and movies. Since performing SVD of a very large sparse partially observed matrix is computationally not possible, we learn the low dimensional representations using supervised learning. The latent factor models can be described as

$$\hat{r}_{u,i} = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i \tag{11}$$

where α is a constant, β_u and β_i are the bias terms associated with the user u and item i, γ_u and γ_i are k-dimensional vectors representing the user u and item i in the smaller k latent dimension. The dot product between γ_u and γ_i captures the similarity among them. We can learn these parameters by minimizing the regularized mean squared error defined as

$$loss = \sum_{u,i} (\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - r_{u,i})^2 +$$
$$\lambda \left[\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_u ||\gamma_u||_2^2 + \sum_i ||\gamma_i||_2^2 \right]$$

We can solve this by using alternating least squares or gradient descent. For our experiments we define two latent factor models

1) LF1: We create a latent factor model with only the bias terms for users and movies.

$$\hat{r}_{u,i} = \alpha + \beta_u + \beta_i \tag{12}$$

2) LF2: We also train a complete latent factor with all its terms. However, during the training process, we realize that the bias terms and the latent factor terms are differently scaled and as such should be regularized independently. So we modify our loss function to have different regularizers for both of them. The modified loss function looks like

$$loss = \sum_{u,i} (\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - r_{u,i})^2 +$$
$$\lambda_1 \left[\sum_u \beta_u^2 + \sum_i \beta_i^2 \right] + \lambda_2 \left[\sum_u ||\gamma_u||_2^2 + \sum_i ||\gamma_i||_2^2 \right]$$

The latent factor models though powerful suffer from the *"Cold Start"* problem, i.e. when a particular user or item has very few interactions, the models' predictions are poor. They are also not easily interpretable and don't account for temporal evolution in the data. Collaborative filtering methods also have the same drawbacks.

D. Factorization Machines

Factorization Machines are general-purpose techniques to extend the latent factor models to incorporate features. They not only allow us to model user-item interactions but also, interactions between any pair of features. To do this, we associate every feature with a low dimensional latent vector and then the interactions between the features can be modelled by the dot product of the features weighted by their dot product in the latent space. The following equation describes the model

$$\hat{r}_{u,i} = w_0 + \sum_{i=1}^{F} w_i x_i + \sum_{i=1}^{F} \sum_{j=i+1}^{F} < \gamma_i, \gamma_j > x_i x_j \tag{13}$$

where x_i and x_j are arbitrary features, w_i's are the weights associated with each features, γ_i and γ_j are the low dimensional representation of features x_i and x_j, and F is the total number of features.

Factorization Machines allow us to capture second-order interactions between various features. They are a much richer class of models and a generalization of the latent factor models. If we only have user interactions available we can create two features - x_1 which is the one-hot encoded vector of users and x_2 which is the one-hot encoded vector of items. However, if we have additional attributes like timestamps, movie genres etc. we can still include them in Factorization Machines unlike Latent Factor and Collaborative Filtering. We train two Factorization Machine models

1) FM1: We only make use of the user-movie interactions. There are only two features - one-hot encoded user and one-hot encoded movie.

2) FM2: We also incorporate the timestamp present in our data as an additional feature by one-hot encoding it on the month and year. If an interaction is made in a particular month and year the corresponding bit in the one-hot binary vector is turned on.

E. Neural Collaborative Filtering

In recent years, deep learning models have been used to achieve state-of-the-art performance in vision, language processing etc. He et.al in [1] presented a new approach to model user-item interactions. Instead of taking the dot product between the latent space representation of the user and the item, they proposed to learn the non-linear relations between the two using a feed-forward neural network. Thus more complex interactions between the user and the item can be learned from this approach.

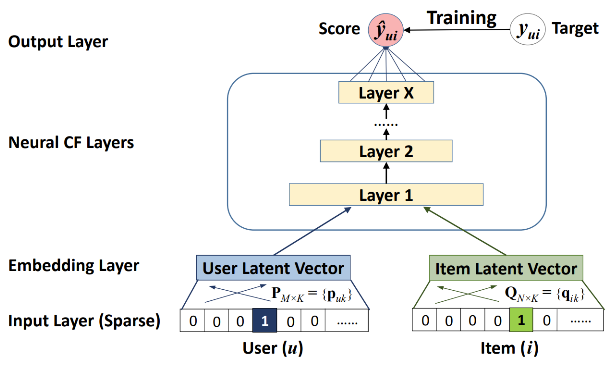

Fig. 9: Neural Collaborative Filtering

Fig 9 shows how the latent vectors of user and items are passed through a feed-forward neural network to learn non-linear relations and predict the value of ratings. We train a similar model using a single fully connected layer which takes the concatenated input of the user latent vector and the item latent vector. We also use the 'Relu' non-linearity as the activation function for that layer. We train the model using gradient descent and also use ridge regularization.

However, recent literature like Dacrema et al. [4] show potential problems with these approaches. They state that most of the neural network-based recommender systems can be outperformed by finely tuned simpler heuristic methods. They argue that added complexity of neural networks in terms of model parameters, training overhead, loss of interpretability etc. only amounts to modest or no improvements in performance. In our experiments too, we have noticed this trend. The *NCF* model does outperform *AMP*, *CF1*, *CF2*, *LF1* and *LF2*, but its performance falls short when compared to the factorization machines methods *FM1* and *FM2*.

VI. Experimental Setup

In this section, we describe the training process of these models. Building the *AMP* predictor is straightforward. The mean rating is calculated on the train set. For collaborative filtering *CF1* and *CF2*, we create the sparse interaction matrix by keeping only track of movies which the user has rated for every user. Similarly, for every movie, we only keep track of users who have rated that movie. This allows us to store all the interactions in memory. Once this is done, we can use the heuristics defined in Eq. 7 and Eq. 9 to make rating predictions.

For *LF1* and *LF2*, we define the gradients for each of the parameters and then use scipy to perform gradient descent to minimize the loss. The regularizer λ for *LF1* was 10^{-5} and for *LF2* the regularizers λ_1 and λ_2 were 10^{-5} and $5 * 10^{-5}$ respectively. The size of the latent dimension, $K = 4$ performed best on the validation set. Similarly, for *NCF* we train our model in Tensorflow using Adam as the optimizer. The regularizer value used was $5 * 10^{-2}$. The latent dimension size, K for *NCF* was increased to 20 as now we had the power to learn more complex interactions. The regularizers are hyperparameters which were chosen after running several experiments and choosing the value which had the lowest error on the validation set. To achieve this we had to tune the regularization constants very carefully. The performance on the validation set is indicative of the generalization performance of the model. Thus, choosing hyperparameters in such a way that minimizes the validation test error reduces the chances of overfitting the model.

The data preparation for *FM1* and *FM2* was slightly different. We need to create features for them. Since we have $329,577$, unique users for every user we create a one-hot encoded feature vector of size $329,577$ with all values to 0 except for the current user, which was set to

1. Similarly, we also create a one-hot encoded feature of each movie of dimension $28,595$, the number of unique movies in the dataset. We train *FM1* using these features and using 10^{-2} as the regularization constant for the weights and 10^{-3} for the latent factors. The latent space dimension, K used was equal to 3.

Additionally, for *FM2* we wanted to understand the effect of additional features. So, we bin the timestamp of each interaction into bins spanning one month. That is all reviews written in a month will fall into one bin. Since we have data from October 1997 to October 2012, we get 180 bins corresponding to 180 months. The one-hot encoded vector for the timestamp is then created by setting the corresponding month index to 1. Then this is used along with the user and movie vectors to learn a Factorization Machine model. The best validation performance is achieved for $K = 100$, the size of the latent dimension and using regularization constants 10^{-2} and $5 * 10^{-4}$ for the weights and latent factors respectively. This naive approach to temporal modelling outperforms all other methods thus highlighting the importance of well-tuned features. Recent methods in [7], [5] and [6] describe better alternative approaches to temporal modelling in recommender systems.

VII. Results

As mentioned in the above section, we have come up with several models - AMP (Always Mean Predictor), Collaborative Filtering, Latent Factor Models, Factorization Machines and Neural Collaborative Filtering to accurately predict how a user would rate an item. These models have been compared on the basis of Mean Squared Error (MSE) and Area Under Curve (AUC).

Table IV shows the comparative analysis of the performance for each model used on the test set. It is evident that *FM2* which incorporates an additional feature 'timestamp' outperforms all the others. The *NCF* model performs the second best. From the results, it is evident that there exists a non-linear relationship between user and movies because the models which learn a non-linear relation (*NCF*) or which makes use of an additional feature to model that non-linear relation (*FM2*) performs better than other. Similarity-based models like *CF1*, *CF2*, *LF1* and *LF2* are too simple to figure out such interactions.

TABLE IV: Comparison of different model performance

Model	**MSE**	**AUC**
AMP	1.6893	0.6859
$CF1$	1.5827	0.6914
$CF2$	1.5898	0.6903
$LF1$	1.5126	0.7037
$LF2$	1.4873	0.7125
$FM1$	1.2421	0.7435
$FM2$	1.0945	0.7684
NCF	1.1253	0.7526

Fig. 10 demonstrates how the value of K, the latent dimension size is chosen for a model. The graph shows the variation of mean squared error with the number of latent factors K on the validation set for *LF2*. The value of K for which the validation error was the lowest is chosen. A similar methodology was used to pick values of K and λ's for other models. Another interesting thing to note the figure makes it clear that the loss is non-convex in K.

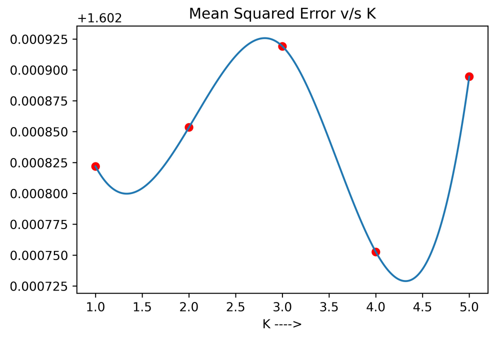

Fig. 10: Variation of MSE with $f$or *LF2*

A visual comparison of the learned predictions is presented in Fig. 11. The subplot on the left shows the spread of true ratings given by the 'Top 5 Users' (Users with the most interactions). The subplot on the right shows the spread of predicted ratings for the same users given by the *NCF* model.

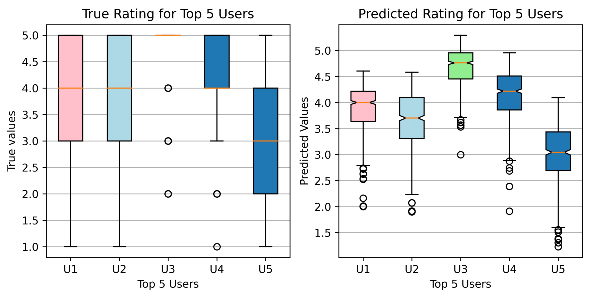

Fig. 11: True Ratings vs Predicted Ratings (for Top users)

VIII. Conclusions

It is evident that efficient modelling of user interactions is paramount for building good recommendation systems. The simpler methods of Matrix Factorization, with naive feature engineering, still outperform the complex machinery of neural networks. It enforces the fact that simpler methods of interaction modelling should be tested first before moving on to more complex methods.

References

[1] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In Proceedings of the 26th International Conference on World Wide Web (WWW '17). https://doi.org/10.1145/3038912.3052569

[2] McAuley, Julian John, and Jure Leskovec. "From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews." In Proceedings of the 22nd international conference on World Wide Web, pp. 897-908. 2013.

[3] Yehuda Koren, Robert Bell, Chris Volinsky, and others. 2009. Matrix factorization techniques for recommender systems. Computer 42, 8 (2009),30–37.

[4] Ferrari Dacrema, M., Cremonesi, P., Jannach, D. (2019). Are we really making much progress? A worrying analysis of recent neural recommendation approaches. Proceedings of the 13th ACM Conference on Recommender Systems.

[5] Liu, Kuan, et al. "Temporal learning and sequence modeling for a job recommender system." Proceedings of the Recommender Systems Challenge. 2016. 1-4.

[6] Chen, Tong, et al. "Sequence-aware factorization machines for temporal predictive analytics." 2020 IEEE 36th International Conference on Data Engineering (ICDE). IEEE, 2020.

[7] Y. Shi, "An improved collaborative filtering recommendation method based on timestamp," 16th International Conference on Advanced Communication Technology, 2014, pp. 784-788, doi: 10.1109/ICACT.2014.6779069

[8] He, Xiangnan, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. "Neural collaborative filtering." In Proceedings of the 26th international conference on world wide web, pp. 173-182. 2017.

[9] Liu, X.; Aberer, K. Towards a dynamic top-N recommendation framework. In Proceedings of the 8th ACM Conference on Web Science—WebSci '16, Hannover, Germany, 22–25 May 2016; pp. 217–224.

[10] Li, L.; Zheng, L.; Yang, F.; Li, T. Modeling and broadening temporal user interest in personalized news recommendation. Expert Syst. Appl. 2014, 41, 3168–3177.

[11] Escovedo, T.; Koshiyama, A.; Da Cruz, A.A.; Vellasco, M. DetectA: Abrupt concept drift detection in non-stationary environments. Appl. Soft Comput. 2018, 62, 119–133.

[12] Wang, Y.; Yuan, Y.; Ma, Y.; Wang, G. Time-Dependent Graphs: Definitions, Applications, and Algorithms. Data Sci. Eng. 2019, 4, 352–366.

[13] 121. Park, Y.; Park, S.; Jung, W.; Lee, S.-G. Reversed CF: A fast collaborative filtering algorithm using a k-nearest neighbor graph. Expert Syst. Appl. 2015, 42, 4022–4028.

[14] Vinagre, J.; Jorge, A.; Gama, J. Collaborative filtering with recency-based negative feedback. In Proceedings of the 30th Annual ACM Symposium on Applied Computing—SAC '15, Salamanca, Spain, 13–17 April 2015; pp. 963–965

[15] Dunlavy, D.M.; Kolda, T.; Acar, E. Temporal Link Prediction Using Matrix and Tensor Factorizations. ACM Trans. Knowl. Discov. Data 2011, 5, 1–27.

[16] Moghaddam, S.; Jamali, M.; Ester, M. ETF: Extended tensor factorization model for personalizing prediction of review helpfulness. In Proceedings of the WSDM 2012–the 5th ACM International Conference on Web Search and Data Mining, Seattle, WA, USA, 8–12 February 2012.

[17] Shi, Y.; Karatzoglou, A.; Baltrunas, L.; Larson, M.; Hanjalic, A.; Oliver, N. TFMAP: Optimizing MAP for top-n context-aware recommendation. In Proceedings of the SIGIR '12: The 35th International ACM SIGIR conference on research and development in Information Retrieval, Portland, OR, USA, 12–16 August 2012.

[18] Karimi, M.; Jannach, D.; Jugovac, M. News recommender systems – Survey and roads ahead. Inf. Process. Manag. 2018, 54, 1203–1227.