Massive Data Mining Report

Jin Sebastian (js2715), Kevin Wu (kw452)

Rutgers University

May 2023

1 Introduction

In modern times, recommendation systems have become increasingly prevalent in this data-driven era, seeing applications in advertising, e-commerce, media, and even banking. Extracting patterns in consumer data on a massive scale has allowed companies such as Amazon, Netflix, and Facebook to gain tremendous insights into their customers, Amazon with product purchases, Netflix with streaming recommendations, and Facebook with targeted advertising. These insights have enabled companies such as these to reap massive profits by better understanding their customer base, thereby making recommendation systems a core part of any modern business. Due to the integral nature of a recommendation system to a profitable enterprise, it is very important that they are accurate and are able to adapt to a dynamic business environment. In this paper, we will be exploring whether timestamps could be effectively used to predict consumer preferences and compare those results to more traditional methods of data mining such as item-item collaborative filtering.

2 Motivation and Previous Work

Using temporal dynamics within recommender systems and capturing concept drift has been a much research and popular topic, primarily starting due to the success of BellKor's Pragmatic Chaos in the Netflix Prize challenge and the subsequent Bell Koren papers detailing their solutions, in which collaborative filtering and temporal dynamics played a big part in. Concept drift broadly is the evolution of data that eventually invalidates the original data model as time passes (hence the drift). However, in the context of our recommender systems, a popular example would be the concept drifts of customer preferences as they either explore new products or as items regain or lose popularity over time. However, we need to find a balance between capturing important concepts over time as not all changes will be impactful on our predicted ratings. In addition, due to the many factors beyond what is reported in our data that may affect concept drift, such as a customer's mood affecting day-to-day rating or switches in users in a family account for a streaming service, it is critical to use methods that handle all sorts of concept drifts.

The basis of our recommender system and the timeSVD algorithm comes from Koren's 2009 paper Collaborative Filtering with Temporal Dynamics where he focuses largely on timeSVD++ as well as its variations with capturing different temporal effects (user and item bias, single day effect, linear vs spline modeling). However, there exists much more research such as using temporal factors on neighborhood models, application to e-commerce, and new algorithms using methods like Bayesian Tensor Factorization. In addition, the evaluation of time-aware recommender systems (and recommender systems in general) have also been heavily researched with emphasis on evaluation protocols and objective datasets across different studies as it has been proven that it is possible to manipulate training and testing datasets themselves to manipulate offline evaluation results.

3 Proposed Model

In implementing our algorithms, we use Python, the Python Surprise library, PySpark, and JupyterLab on Rutgers iLab machines for our development tools and environment. The Python Surprise library was primarily used for We also use additional packages with Python such as NumPy and Pandas. We use Surprise functions primarily to help with some parts of evaluation and data processing such as calculating RMSE and running cross validation, though we have programmed our own more specialized implementations as the library was limiting at times for example not supporting timestamps within its data structures. For the SVD algorithms, we used Surprise's framework (function definitions) for initialization and the two parts of fitting and estimating, but we programmed all of the bodies, essentially the actual implementation of these algorithms. Our implementation of top N recommendation was also relatively simple, using our predicted ratings from each of our algorithms, we simply ranked and choose the top N predicted ratings as our recommendations.

3.1 Item-Item Collaborative Filtering

Item-Item Collaborative Filtering is a model used in many e-commerce and media applications. Often it is chosen over User-User Collaborative Filtering models since characteristics of products are more stable than those of users. Users may have multiple preferences, and some of those preferences may change over time or even throughout the day, making the task of accurately recommending movies based on similar users difficult.

The main idea behind item-item collaborative filtering is given a user i and a movie j, we infer the rating that user i would give movie j based the k most similar movies to j that user i has rated. The similarity ratings between two movies x and y will be computed using cosine similarity which is defined as follows where r is a vector representation of a movie.

Cosine Similarity:

$$sim(x,y) = \frac{r_x \cdot r_y}{||r_x||||r_y||}$$

The vectors r can be obtained from the "genome-scores" CSV file which gives the relevancy score of each movie and tag pair. This data is then stored as a dictionary of vectors where the key is the movie id and the value is the vector of relevancy scores. The index of each score corresponds to a specific tag id. Once we obtain the similarity matrix, we take the weighted sum of k most similar movies by a specific user to obtain our prediction. Our predictions are then stored in a matrix.

Predicted Rating:

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} * r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

We then obtain for each user, the top k (10 by default) highest predicted rated movies that the user has not seen before as our recommendations. We will discuss the performance of this method in a later section.

3.2 SVD

The reasoning behind choosing SVD was primarily due to its simplicity, popularity, and the fact that this algorithm would be the basis for our algorithm that will include temporal context, timeSVD. Technically, the concept of SVD alone is not enough to be used for predicting ratings, but it is the concept of Matrix Factorization and latent factor models specifically that makes use of SVD as a very popular choice for recommendation systems. This was especially popularized by Simon Funk during the Netflix Prize where Funk's SVD is a great use of matrix factorization within latent factor models and applies SVD to use cases of sparse matrices (somewhere SVD normally wouldn't be able to be applied)[1]. In using SVD, we use this as our baseline algorithm that serves as the benchmark as to whether timeSVD has improved upon SVD or not (in terms of RMSE specifically though other comparisons may be made).

Another important note is our version of SVD includes biases and uses stochastic gradient descent for optimization on each epoch. While the original predictor equation would only have the dot product of the two matrices (the user and item factors), we also added two additional parameters, the user bias and item bias, noted as b_u and b_i respectively, owing to the fact that certain users on average may rate slightly higher or lower than the mean and the same goes for certain items being rated slightly higher or lower. These don't take into account user-item interaction which is why the dot product is also added in the final prediction. Hence, the final baseline predictor equation is

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

where u is the mean rating over all movies and q and p being the matrix factors of the user and movies. Stochastic gradient descent is a fast and popular algorithm for optimization especially for sparse matrices in order to reduce our prediction error and adjust the parameters of our predictor accordingly.

3.3 TimeSVD

As the name implies timeSVD is simply using a latent factor model with SVD and adding temporal context to the baseline predictor by incorporating it as separate parameters. We specifically use Koren's paper on Collaborative Filtering with Temporal Dynamics to draw heavy inspiration in terms of the temporal parameters to add to the predictor equation of SVD. For our specific implementation, we used our SVD implementation and built off it.

We first define timeSVD's baseline predictor equation, which again this is drawn from Koren's paper and will be briefly discussed here. We note that two of the major temporal effects on a rating prediction can be captured by both the user and item bias. We first look at the item bias b_i, where a movie may become more or less popular over time depending on situations like a sudden cult following or a new actor. This is captured with an additional term $b_i, Bin_i(t)$ where t is the day that the user rated. Luckily, item biases do not fluctuate heavily based on time, meaning a movie's rating often does not significantly change on a day to day basis. Thus, we are able to capture this by splitting item biases into time-based bins where each bin captures a distinct item bias. We balance the number of bins by choosing a number giving enough resolution as well as having an “adequate” number of ratings in each bin. In our case, we choose the number of bins to be 23, based on the number of years that the MovieLens dataset captured though further tuning is required. Thus the final item bias equation is $b_i(t) = b_i + b_{i,Bin(t)}$.

We then look at the user biases as over items a user may start to rate higher or lower over time, both in the short and long term, whether it be due to overall changes in preference of types of movies or unrelated context factors influencing the user in the short term. In order to capture this temporal user biases, there are actually a variety of functions that can be used. For the sake of simplicity both in implementation and explanation, we decided to focus on linear modeling choices. We first defined the mean date of rating for each user u by t_u and define the equation $dev_u(t) = sign(t - t_u) \cdot |t - t_u|^\beta$ for a user u rating on day t. The symbol B is set to .00005 as curiously having a higher Beta term resulted in worse RMSE ratings or even causing the error term to “blow up” and cause overflow in the function. Further tuning and research into this is required as the method for choosing the Beta term in Koren’s paper was just noted as through cross-validation though with no explanation as to its exact meaning. We introduce another learning parameter a_u for each user that we then add to the user bias equation to achieve the final user bias equation $b_u(t) = u_i + a_u * dev_u(t)$.

While with our new parameters we have captured gradual drift well, there are other effects that can be captured such as the single-day effect, which is a more sudden drift and spike or drop in a user’s rating on a day to day basis due to short-lived effects like a person’s mood, and spanning temporal effects on the user preferences (the p matrix). We have attempted an implementation as seen in our submitted codebase, but it does not properly function hence we left it for future work. Our final timeSVD baseline predictor equation combining all of the biases and dot product, $r_{ui} = u + b_i(t) + b_u(t) + q_i^T p_u$. In trying to solve for our parameters, we solve

$$\min \sum_{(u,i,t)\in} (r_{ui} - \mu - b_i(t) - b_{ui}(t) - q_i^T p_u)^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2 + a_u^2 + b_{i,Bin(t)}^2)$$

Specifically within our implementation, we use stochastic gradient descent and take the partial derivatives of this minimization function with respect to each parameter and adjust the parameter, call it θ, and define the equation to be, f_k, to $\theta = \theta - \alpha(\frac{\partial f_k}{\partial \theta})$.

3.4 Proper Evaluation Conditions

A main point when evaluating the timeSVD function especially is having proper evaluation conditions that would accurately represent the results of our algorithm especially when compared to existing academia and research. A paper by Campos conducted an analysis on evaluation conditions for offline evaluation systems, offline meaning using historical data sets, on time-aware recommender systems (TARS) across multiple papers, and it even only proved empirically the dramatic performance differences between various combinations of evaluation conditions. However, more importantly and what we primarily used from this paper, there were several guidelines to follow to avoid these methodological issues that arise from TARS and to select proper conditions. We will detail 3 of these guidelines, their reasoning, and how it was implemented within our code.

1. Guideline 1: *Use a time-dependent rating order condition for TARS evaluation*
   A time-dependent rating order condition specifically deals with the split of the training and testing split where it orders all ratings by timestamp and then takes data after a certain timestamp as the testing set. This prevents performance variations from inaccurate time information presented in the training set (information that came from the "future").

2. Guideline 2: *If the dataset has an even distribution of data among users, use the community-centered base condition. Otherwise, use a user-centered base condition in order to avoid biases towards profiles with long-term ratings.*
   While the paper does not specifically define what it means to have an "even distribution," we intuitively decided that there was not an even distribution of data due to the wide variety of the number of ratings associated with each profile. Using a user-centered base allows for us to take a certain proportion from each user rather than taking a chunk from the overall dataset, allowing all users to have a certain portion in both the training and testing sets.

3. Guideline 3: *Use a proportion-based size condition.*
   The reason for these guidelines is related to and very similar to the last provided guideline. This allows control of proportion of training and testing data split, and in the case of using a user-centered base condition, it prevents users from losing too much data to either the training or testing split, regardless of their size.

We implemented all of these guidelines when constructing our "proper" testing and training sets, separate from the random split we used for our initial testing. We first grouped the ratings dataset by user id (user groups), then sorted by the timestamp, and then took the top 20 percent most recent ratings of each user as part of the testing set, hence fulfilling the first 3 guidelines.

4 Results

We will be evaluating the effectiveness of our models using the metrics Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Precision, Recall, F-measure, and Normalized Discounted Cumulative Gain (NDCG). All of them describe the accuracy of our model in different ways. Both MAE and RMSE describe the accuracy of our predicted ratings in comparison to our actual ratings. The others describe the quality of our recommendations.

$$MAE = \frac{1}{n}\sum_{i=1^n}|p_i - a_i|$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1^n}(p_i - a_i)^2}$$

Starting with the random split set, timeSVD has outperformed SVD and Item-Item CF both in terms of RMSE and MAE. These results were on average after multiple (5) runs using cross validation. Now using the proper set with the training and testing dataset described from the proper evaluation conditions section, timeSVD still outperformed. These were also averaged after multiple (5) runs though cross-validation was not used in these runs

Metric	Item-Item CF	SVD	TimeSVD (Item)	TimeSVD (User + Item)
MAE	0.7088	0.6871	0.6777	0.6730
RMSE	0.9142	0.8908	0.8776	0.8740
Precision	0.8428	0.6901	0.6901	0.6804
Recall	0.6403	0.5321	0.5382	0.5328
F-measure	0.7277	0.6009	0.6047	0.5976
NDCG	0.9984	0.4441	0.4459	0.4466

Table 1: Results

and the same testing and training sets were used for every run.

There are a few interesting points:

1. There seems to be only a little difference between the addition of the temporal effects of the user bias and the timeSVD algorithm with just the item bias in both cases. There are multiple theorized reasons for this. The biggest factor seems to be the extremely low β set for our calculation of devu_t, however as noted before in our implementation, it seems to be necessary otherwise the term "blows up" and severely affects the predicted error and leads to a much worse RMSE. The spline model detailed in Koren's paper may possible solve this issue by having multiple time kernels rather than one average time for each user. Also, in general, our dataset may possibly be too small such that the temporal effect on user bias does not lead to significant changes

2. The proper set leads to a significant jump in the RMSE and MAE of all the algorithms as compared to the random split set. We theorize this is a result of the SVD algorithms no longer having access to data points set in the future of the training set, hence not giving extra temporal context/information to the algorithm.

3. The Precision, Recall, F-measure, and NCDG measurements for the SVD variations were all surprisingly extremely similar with no clear trends as to which had better accuracy measurements. Even with multiple (5) iterations, these were the average values and all factors were kept equal in terms of learning rates, number of epochs, factors, and the testing and training set data. The clear runaway was the Item-Item CF which had extremely high accuracy values, even though it had the worse RMSE and MAE values. Further research into this is required as the implementations for these accuracy calculations differed. However, it does raise the question of the general accuracy performances between latent factor models and neighborhood models as while RMSE was across the board higher for our latent factor models, it may not always necessarily translate to a better recommendation model.

We also note that we only calculated the SVD and timeSVD's RMSE and MAE with our proper datasets. This is intentional as firstly the creation of a proper set would not affect the top N recommendation predictions since we take the full dataset as our training set and the unknown values as our testing set. Secondly, the primary reason for creating these proper sets, as noted by the original paper, was to have objective accuracy measurements in the case that our implemented SVD algorithms were to be compared to other TARS in further research.

5 Conclusion and Future Work

In terms of minimizing RMSE and MAE, capturing temporal context seems to be extremely valuable, and while only proved for latent factor models, this could also be extended to neighborhood models as noted in Koren's paper. Each subsequent addition of a time parameter onto a bias improved our measures of prediction quality, and there were more concept drifts that were not captured but could be added to our predictor equation and implementation. While we added the user and item bias, there was also a possible parameter for the single-day effect in which there are sudden drifts due to a user often concentrating around a single value during a given day or session. Within the item bias, we also could have modeled periodic effects that would have caught more broad concept drifts, for example seasonal changes. The linear models themselves could have been upgraded to a spline as noted in our results section which likely would have resulted in a more accurate capturing over user biases. Lastly, in general, further tuning of our hyperparameters and testing with cross-validation especially as recommended within the guidelines of Campos' paper would be needed for optimization and further objective review of our algorithms.

The Precision, Recall, F-measure, and NCDG measurements for the SVD variations are largely inconclusive. The NCDG at least shows an improving trend with each time factor added, however measurements like the precision, recall, and F-measure show no clear pattern. The one extremely different algorithm was the Item-Item CF, but this could be largely experimental error due to differing implementations between the SVD and Item-Item CF especially considering the near perfect NDCG. Future work would hope to integrate the same measurement implementations for these algorithms, but regardless, the question of performance between latent factor models and neighborhood models definitely arises.

There are many more problems and aspects of this project we wish to explore for the future. In addition to targeting the aforementioned work, we wish to delve deeper into the performances between latent factor models and neighborhood models in the cold start problem. It would be interesting to see how our implementation of timeSVD performs in that situation especially relative to other CF algorithms. Another big problem we attempted to target but ultimately leave to the future are Shilling Attacks and detection of these attacks. There has been work in the attack methods and detections of Shilling Attacks in the Collaborative Filtering space as they especially are susceptible to nuke or push attacks from various shilling attack methods due to their (as the name implies) use of collaborative methods for rating prediction. We did preliminary research and decided on one paper (linked in References) specifically making use of group user credibility and time series for shilling attack detection, with the intuition that it will incorporate nicely into our algorithm due to us already implementing temporal context. This would help immensely in the robustness and anti-attacks of our algorithm though the implementation was more complex than expected and is left for the future.

6 Works Cited

References

[1] Campos, P.G., Díez, F. Cantador, I. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. User Model User-Adap Inter 24, 67–119 (2014). https://doi.org/10.1007/s11257-012-9136-x

[2] Koren, Y., Bell, R. (2015). Advances in Collaborative Filtering. In: Ricci, F., Rokach, L., Shapira, B. (eds) Recommender Systems Handbook. Springer, Boston, MA. https://doi.org/10.1007/978-1-4899-7637-6_3

[3] Koren, Y. (2009). Collaborative filtering with temporal dynamics. Knowledge Discovery and Data Mining

[4] Zhou W, Wen J, Qu Q, Zeng J, Cheng T. Shilling attack detection for recommender systems based on credibility of group users and rating time series. PLoS One. 2018 May 9;13(5):e0196533. doi: 10.1371/journal.pone.0196533. PMID: 29742134; PMCID: PMC5942815.