Temporal Factors in Beer Ratings
CSE 255, Winter 2014
Project 1

Alexander Xydes
axydes@ucsd.edu
A53046108

ABSTRACT

Predicting a user's rating of a product without taking how their preferences might change over time leaves out a lot of data that could be used to improve that prediction. Why not track that change in preferences and change the recommendations accordingly? As long as keeping track of that doesn't add significantly to the model there should be no reason not to. In this paper, I aim to do precisely this for user preferences towards beers. For beer in particular, there are multiple different timescales that could affect a user's preference and I add three different timescales into my prediction model. This model is then compared to a baseline collaborative filtering model based on the Pearson similarity ranking. The comparison is made using the commonly used Root Mean-Squared Error (RMSE) and precision@k metrics. The Temporal Factor model built in this work is demonstrated to perform better at both metrics.

1. INTRODUCTION

Reviewing products has become an everyday occurrence for many Americans, with most people expecting their reviews to help improved their use of the product in some way. Everything from movies, to clothes to alcohol can be reviewed and given a rating, usually on a scale from one to five. For alcohol in particular there are two smartphone applications, Untapped for beer and Vivino for wine, that have become popular. While Vivino does provide wine recommendations for its users, Untapped still does not as its focus is more on the social aspect of drinking. As an Untapped user myself, I would appreciate a feature that could provide recommendations for me based on a list of beers that I provide it to choose from. The other big beer rating sites (BeerAdvocate and RateBeer) also do not appear to provide beer recommendations.

While beer rating sites don't appear to provided recommendations, most other places that allow users to rate products (like Amazon, Netflix, Vivino) do provide them. A lot of the research that went into recommender systems was spurred by Netflix's competition to beat their own recommender system, which was started in 2006 [2]. By 2009, the last year of the competition the two front runner's were a mashup of various teams that had started the competition. One of those teams became an early leader by incorporating temporal dynamics into their recommender system. It makes intuitive sense that time should be a factor in recommender systems because user preferences change over time. A user could start their Netflix profile while they really like action movies, but a year later could hate action movies and really like dramas. This type of change is hard if not impossible to model without taking time into account.

This same type of change is also present in user preferences for beer. In fact, there might be other forms of temporal change in beer preferences such as seasonal favorites or even based on the time of day.

2. DATASET

The BeerAdvocate dataset from SNAP ([5]) was used as the basis for this work. Ratings are given between zero and five (inclusive) in increments of one half. The full dataset contained too much data to be used in a reasonable manner, so two smaller subsets were explored. One subset at 500000 entries and one at 750000 entries (Tbl. 1). The 500000 entry subset is the one used by this paper as it provides most of the users and beers that the larger subset had, a lot of the same variation in ratings while being smaller and easier to work with.

Table 1: Basic statistics about the BeerAdvocate dataset.

Statistic	Full Dataset	750000 Entries	500000 Entries
# reviews	1586259	750000	500000
# users	33387	25567	22374
# beers	66051	30009	19334
Timespan	1/98 - 1/12	1/98 - 1/12	1/98 - 1/12

Three timescales were chosen to see if the overall rating varied in any way over the course of each timescale. The day of the week was chosen because conceptually, it would make sense that people drink more on the weekends than during the week (Fig. 6). However, it proved to have less variation in the ratings than the other timescales. The hour of the day varied much greater than the day of the week (Fig. 7). The month of the year was chosen for a very similar reason as the day of the week; I hypothesized that people would drink more during some months less during others (Fig. 1). The year was chosen as well (Fig. 2). All three timescales show variation in the average overall rating given to beers (Fig.7, 1, 2).

The month timescale (Fig. 1) shows that beer preferences can change with the seasons. This make intuitive sense, as lighter, more refreshing beers may be preferred during the summer, and darker, heavier beers in the winter.

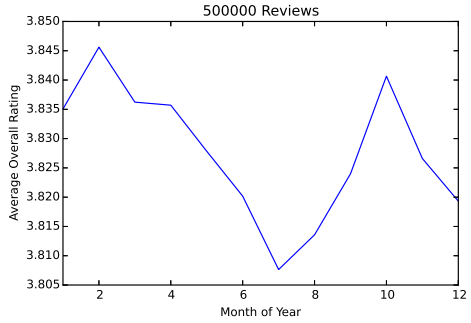

Figure 1: Average rating per month of year

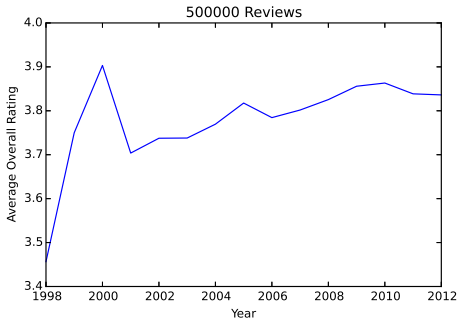

Figure 2: Average rating per year

The year timescale (Fig. 2) shows how the community of beer reviewers on BeerAdvocate has changed over the years. One can see more variation in average overall rating for the first few years, and then the average overall rating smooths out and starts following a generally positive slope. The first few years could demonstrate that the community was coming to a consensus about what scale their ratings should be on, i.e. what constitutes a neutral rating. This second trend could indicate a gaining of community expertise as time goes on.

The dataset with 500000 entries was chosen for this research as it provides the most reasonable amount of data to work with while still containing enough for potential for results and variation in the timescales explored.

3. PREDICTIVE TASK

The ultimate goal of this research is to have a model that can predict the top two or three beers for the current user and time from a given list. Therefore, the model developed will have to produce predictions of user ratings for beer. Two baselines will be used to compare this model against: a baseline that just uses the global average from the training data (baseline 1), and a baseline collaborative filtering algorithm (baseline 2). Performance is judged based on Mean-Squared Error (MSE), Root Mean-Squared Error (RMSE), R^2 and a comparison of precision@k rankings. The precision@k rankings for the test set is also used to judge the validity of the model's predictions at the ultimate goal.

The precision@k rankings use the following definition of relevant: label and predicted rating both above 2.5. This follows from the goal of predicting the top k beers, as it only matters if the predicted rating and subsequent actual rating are both high.

4. RELEVANT LITERATURE

The dataset used in this paper is described in Sec. 2, and was provided by [5]. However, it is no longer available as the website it originated on requested that the host remove access to it. It's been used in the past to cross-reference review text with different rated aspects of the items [7]. That paper also used it's models to summarize the review text which meant finding the best sentences that explain a user's rating. Another paper explores the effect of modeling a user's experience level on the effectiveness of predictive models [6]. This paper built two intertwined models, one for the user's experience level over time and one to predict the user's ratings. Neither of these papers explore the use of explicit temporal factors in a predictive model, those the second one ([6]) does approximate one temporal factor (user preference) with it's model of user experience.

One dataset similar to the BeerAdvocate one used in this paper that has been studied extensively is the Netflix Prize dataset [2]. This dataset consists of over 100 million timestamped ratings, where the ratings are integers between 1 and 5. This competition focused on reducing the Mean-Squared Error of model predictions. One of the early leaders of the competition focused on incorporating temporal dynamics into their model [4]. That model is what this work is based on, as it provides a straight-forward way of using temporal variation in ratings and preferences into a latent-factor model.

Another model that explores a different way of using temporal dynamics in the Netflix Prize rating data focuses on extending Probabilistic Matrix Factorization (PMF) [9]. Not only did they extend Probabilistic Matrix Factorization (PMF), they also used a fully Bayesian treatment to reduce overfitting and a Markov Chain Monte Carlo sampling method to integrate over the parameters which they based on [8].

5. FEATURES

Features based on for a global, user and item bias (μ, b_u, b_i) capture much of the variation in user ratings and go a long way towards accurately predicting those ratings on their own. Therefore, those bias features are used as the basis for the temporal model developed here. However, these basic bias features can't capture time-changing features that make up a realistic model of user preferences and item popularities.

To capture time-changing features, one must incorporate some sort of factors based on temporal features. As demonstrated above (Sec. 2), average ratings can change on many different timescales. Most of the following features are based on the work done by Koren in [4].

The popularity of items (beers) definitely changes over time. It would make sense that as a beer got older, it's popularity would change. It also makes sense that the season (month) of the year might also affect the popularity of a particular beer. Therefore, two features were added to track these possible affects: one to track the overall popularity of a beer and one to track it's seasonal popularity. The overall popularity is tracked by binning the time of the review such that there are 60 bins total for the entire dataset. Then each beer will have a bias factor for each bin ($b_{i,bin(t)}$, Eq. 3).

User preferences also change slowly over time, but they can also change much faster than beer popularity. Here three features were

added to track the different timescales that user preferences can change at. The first tracks general user preferences as they change over time (Eq. 6). According to [4], this fits a linear function to the gradual change of user preferences.

The second feature tracks the seasonal preferences of users in the same way as the feature that tracks the seasonal popularity of beers ($b_{u,month(t)}$). This is done because it's not clear which of the two effects is more prominent in this data, and including both in the model gives the model the flexibility to have one stronger than the other on a per user and per item basis.

The last feature for users tracks how their preferences change with the hour of the day. Based on the exploratory analysis, this seems like it might have an effect on the ratings of users (Fig. 7). This is done the same way the monthly features are tracked, i.e. one per hour per user ($b_{u,hour(t)}$).

In addition to the features mentioned above, the model includes latent features for both users and beers. These features will capture any significant relationship between users and beers that's not caught by the previous features. This is done via a vector of latent features for each user and each item (${q_i}^T p_u$).

5.1 Preprocessing

A fair bit of preprocessing goes into getting the features outlined above for all the entries in the dataset. First, the data downloaded from [5] is converted from plaintext into the JSON [1] format to make it easier to manipulate in python. Then, that data is split into three sets: train, validation and test sets. The validation and test sets get $75K$ entries each, and the train set gets the remainder of the data.

Then upon setup of the model, the following operations are performed. A list of users, and a list of items are created. For each entry in the train set, the following items are gathered: user, item, rating, timestamp, $bin(item, timestamp)$, and $dev_{user}(timestamp)$. The same information is gathered for the validation and test sets, except the ratings are separated and put into a list of labels for the validation and test sets.

6. MODELS

Two models were developed for this research, the baseline collaborative filtering algorithm (baseline 2) and the Temporal Factors model that is the basis of this research.

6.1 Baseline

The baseline collaborative filtering algorithm (baseline) uses user-user similarity when the current user is in the training set, item-item similarity when the current item is in the training set, and the global average of the training set when neither of those cases is true. All similarity comparisons are done using the standard Pearson correlation similarity measure (Eq. 1).

$$Sim(u,v) = \frac{\sum_{i \in I_u \cap I_v} \left(R_{u,i} - \overline{R}_u\right)\left(R_{v,i} - \overline{R}_v\right)}{\sqrt{\sum_{i \in I_u \cap I_v} \left(R_{u,i} - \overline{R}_u\right)^2 \sum_{i \in I_u \cap I_v} \left(R_{v,i} - \overline{R}_v\right)^2}} \quad (1)$$

6.2 Temporal Factors Model

The Temporal Factors Model is built from all the features mentioned in Sec. 5, and based on the work done by Koren in [4] on the Netflix Prize movie dataset [2]. The predicted ratings are based on a combination of the μ feature, the time-based user and item biases, and the user and item latent factors (Eq. 2). The length of the latent factors vectors p_u and q_i is set to 20 as that is a good starting point. A grid search over the validation set was performed on the length of these vectors over the following values: $10, 20, 50, 100, 200$, but no variation was observed in the resulting scores. This is most likely caused by an error in the code, unfortunately there was no time to look for the error.

$$\hat{r}_{u,i,t} = \mu + b_u(t) + b_i(t) + {q_i}^T p_u \quad (2)$$

This structure is commonly used as a basis for latent factor models with or without the temporal features, and is also used in [4]. Regular latent factor models use a single feature per user and per item, however this model bases the user and item biases on temporal features which can include many more features per user and per item.

The item bias (Eq. 4) includes one feature per item as usual, as well as more features that are based on time. There's one feature per month of the year and one that captures how item popularity drifts over time. This item popularity drift feature is based on the bin that the time of rating is placed into (Eq. 3). The number of bins is set to 60 as that yields bin sizes of about 10 weeks for this dataset and this bin size worked well in [4]. This yields a value of 85 days per bin for γ. Unfortunately, there was not enough time to do a grid-search on this hyper-parameter.

$$bin(t) = \frac{t - t_{min}}{\gamma} \quad (3)$$

$$b_i(t) = b_i + b_{i,month(t)} + b_{i,bin(t)} \quad (4)$$

The user bias (Eq. 5) includes a single feature per user as usual, but also includes more time-based features. One feature for each user per month of the year, and one feature per hour of the day. It also includes a feature that tries to capture how a user's general preference changes over time (Eq. 6). The hyper-parameter β is learned via grid-search for 30 iterations over the validation set and set to 0.1.

$$b_u(t) = b_u + \alpha_u dev_u(t) + b_{u,month(t)} + b_{u,hour(t)} \quad (5)$$

$$dev_u(t) = sign(t - t_u)\,|t - t_u|^{\beta} \quad (6)$$

These decisions keep the model relatively compact, while still capturing a lot of different possible temporal variations in the data.

With the equation for a predicted rating known, one can start learning what the parameters involved should be. This is done via solving Eq. 7 which optimizes the parameters and prevents over-fitting via the regularization terms Ω_u, Ω_i, and Ω_l.

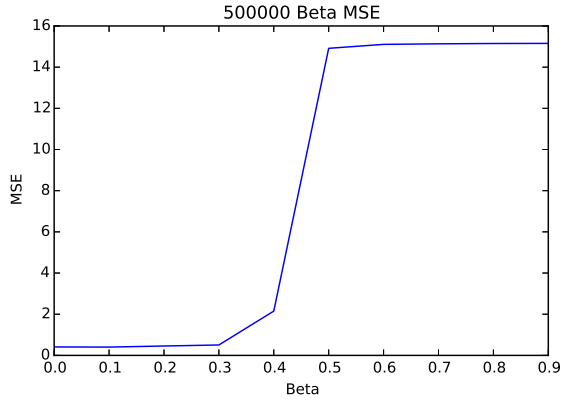

Figure 3: Grid-search results for β

$$argmin \sum_{u,i,t} (r_{u,i,t} - \hat{r}_{u,i,t})^2 + \lambda(\Omega_u + \Omega_i + \Omega_l) \quad (7)$$

$$\Omega_u = \sum_u b_u^2 + \sum_u \alpha_u^2 + \sum_{u,t} b_{u,month(t)} + \sum_{u,t} b_{u,hour(t)}$$

$$\Omega_i = \sum_i b_i^2 + \sum_{i,t} b_{i,month(t)} + \sum_{i,t} b_{i,bin(t)}$$

$$\Omega_l = \sum_u \|p_u\|_2^2 + \sum_i \|q_i\|_2^2$$

The hyper-parameter λ is learned via grid-search for 30 iterations as well over the following values: $0.1, 1, 10, 100, 1000$, and is set to 10.

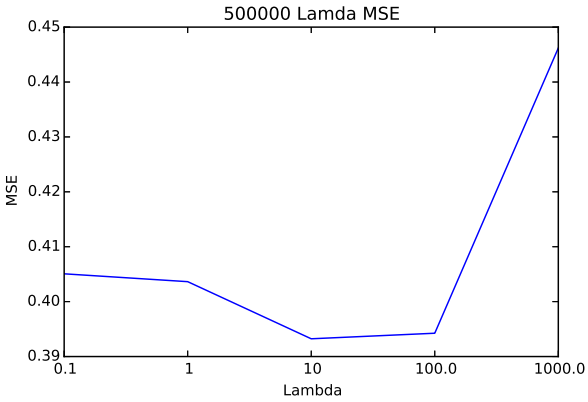

Figure 4: Grid-search results for λ

Learning is done via the L-BFGS algorithm provided by the SciPy [3] library and convergence takes 303 iterations to occur.

6.3 Scalability

Early on while implementing this model, I encountered some scalability issues as it was taking many minutes to complete one iteration of L-BFGS. After profiling the code in python, I realized that most of the slowdown occurred because I was getting the index of the user or beer in the corresponding list of users and beers every time I needed it. Since those lists are very large, these operations take a long time and doing it multiple times per entry caused the time needed to balloon. I realized I could just store the indexes of each user and item at the beginning during the setup of the training data and then use them directly. Along similar lines, I was calculating the results of Eq. 6 and 3 every time I needed them. This also caused significant slowdown in my runtime, and I could also store these values at the beginning during setup. Changing these lookups and operations into straight references caused my runtime for one iteration to sink to about 12 seconds.

6.4 Model Comparison

The Baseline 2 model is more commonly used in the real world, however the Temporal Model provides some clear advantages to that model. The Baseline 2 model is simpler to implement, which also means that it's easier to maintain and debug, both of which are highly regarded qualities in the professional world. It also requires no time to train the model and provides very nice explainabilty of its predictions. In other words, the Baseline 2 model can tell a user that it recommended item A because that user also like item B, C and D. However, this model also needs more data in memory (the entire rating matrix) than the Temporal Model. While the Temporal Model has the same performance as the Baseline 2 model on completely new data and has longer setup and training time it also needs less memory to run than the Baseline 2 model. The Temporal Model also can potentially achieve more accurate predictions because it is taking into account temporal variations in preferences and popularity that the Baseline 2 model cannot.

7. RESULTS

After running all the models on the test set, the Temporal Model performed best by a large margin over the two baseline models. This is best demonstrated in the gap in R^2 results between the models (Tbl. 2). The Temporal Model bests the baseline models by at least 0.22 in R^2.

Table 2: Comparison of results

	Baseline 1	Baseline 2	Temporal Model
Variance	0.510485	0.510485	0.510485
MSE	0.510493	0.509147	0.389147
RMSE	0.00260894	0.0026055	0.002277
R^2	-1.64×10^{-5}	0.0026206	0.237691

The difference between Baseline 2 and the Temporal Model is less drastic in the precision@k metric. This is because the Baseline 2 model does a good job of predicting which items a particular user might want and the Temporal Model doesn't have much room to improve. The similarity between the two also validates the predictions that the Temporal Model makes as it means that it's performing just as well as a simpler model. However, it does improve which you can see when k is between 8 and 12 especially (Fig. 5). Once k gets higher than that, the results are very relevant as most users will only look at the first few results.

Both of the latent factors vectors (p_u,q_i) are completely empty (all values are set to zero), which means that they didn't contribute to the model at all. This would explain why the grid-search over the lengths of these vectors turned up zero results, and also confirms that there is probably an error in my code. This also suggests that temporal features can capture a lot of the variation in user ratings that might normally go into the latent factors vectors if constructed correctly.

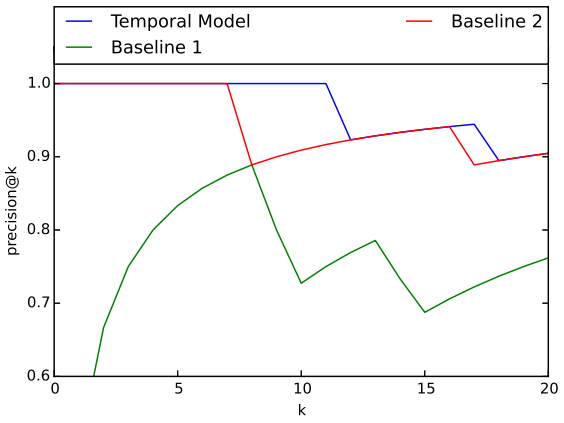

Figure 5: precision@k comparison

All of the other features have values that vary between -1 and 1 for the vast majority of users and items, with most of the variation occurring between -0.2 and 0.2. The variation in values for these features suggest that they are highly specific to the user and item.

8. CONCLUSION

Based on the results given here, adding temporal features to a latent factor model can provide much better results than a simple similarity based model in aggregate. It's not clear which temporal features added the most to the model for this beer review dataset, so exploring that would be the next step. Given the flexibility of the model, and the compactness when running it seems like a much better choice than a similarity-based model for a real-world prediction and recommendation system.

9. REFERENCES

[1] Introducing json. `http://json.org`, Jan. 2015.

[2] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.

[3] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2015-02-23].

[4] Y. Koren. Collaborative filtering with temporal dynamics. *Commun. ACM*, 53(4):89–97, Apr. 2010.

[5] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[6] J. J. McAuley and J. Leskovec. From amateurs to connoisseurs: Modeling the evolution of user expertise through online reviews. *CoRR*, abs/1303.4402, 2013.

[7] J. J. McAuley, J. Leskovec, and D. Jurafsky. Learning attitudes and attributes from multi-aspect reviews. *CoRR*, abs/1210.3926, 2012.

[8] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 880–887, New York, NY, USA, 2008. ACM.

[9] L. Xiong, X. Chen, T. Huang, J. G. Schneider, and J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*, pages 211–222, 2010.

APPENDIX

A. FIGURES

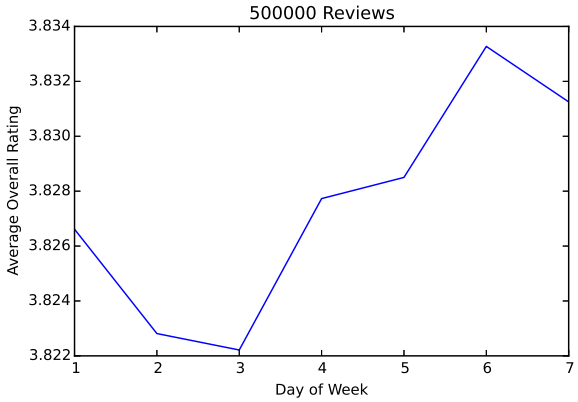

Figure 6: Average rating per day of week, Monday - Sunday

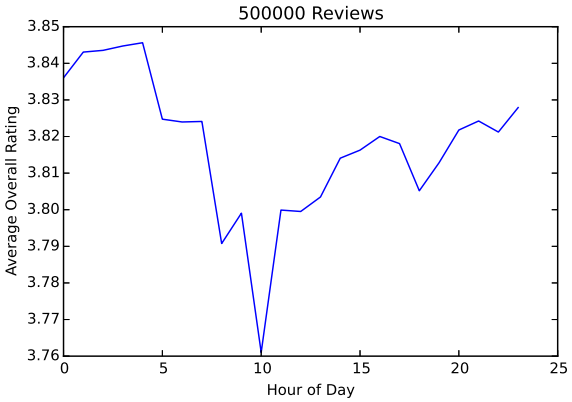

Figure 7: Average rating per hour of day