Rating Prediction of Fine Foods Reviews

Juston Lin
UC San Diego
jyl044@eng.ucsd.edu

Sheng-Yong Niu
UC San Diego
sniu@eng.ucsd.edu

Alexander Wu
UC San Diego
ajw016@eng.ucsd.edu

Abstract

In this paper, we focused on the predictions of customers' rating in Amazon Fine Food Reviews Dataset. We utilized sentiment analysis models such as Unigram-Bigram word count and TF-IDF model to predict rating score based on review summaries. Also, we applied models including latent factor model, linear regression, and naive alpha baseline model to predict the review rating. We compared above models by measurement of Mean Squared Error (MSE) in test set, and explain the possible reasons of their performance. In summary, we explored the detailed attributes of Amazon Fine Food Reviews dataset, and performed rating prediction with different models to help us build up a better recommendation system in the future.

Index Terms: Amazon Fine Food Reviews, rating prediction, latent factor model, sentiment analysis, linear regression

Ⅰ. Introduction and Related Work

User's rating score prediction is essential for recommendation system. Many companies such as Amazon, Netflix, and Airbnb utilize user rating dataset to predict user's preference and recommend their customers to potential product they may purchase. A great recommendation system can effectively implement companies' marketing strategies and build up a loyal community with groups of customers. In contrast, a bad recommendation system will disappoint potential customers because they are exposed by products they have no interested in.

In this report, we use the Amazon Fine Food Reviews dataset from all of the existing 568,454 food reviews Amazon users left up on the website from October 1999 to October 2012. The data was extracted by the Stanford Network Analysis Platform (SNAP), headed by Prof. Ana Pavlisic. One of the official paper that used this dataset was done by Prof. Julian McAuley. He and his co-author, Jure Leskovec, used this dataset 5 years ago in the paper [1], "From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews" in order to recommend products that a user would enjoy now while taking account that the user's tastes change over time. The main method that Prof. McAuley used was temporal dynamics. In other words, he was accounting for how user preferences change over time due to how humans change over time with new information. To do this, he and his co-author applied the techniques that we learned in class to the dataset in hopes of discovering a pattern that can follow the changes of people over time.

We came across this dataset through our search on Kaggle. We specified 2 criterias required to choose our dataset: required text and score rating. The Amazon Fine Food Reviews dataset was the first dataset we found on Kaggle that satisfied these results. Because Amazon forces users to rate from a system of 1 to 5 if they want to submit feedback, we did not have modify the dataset to account for missing information.

SNAP also has online reviews for beers, wines, movies, and all products on Amazon. The all-Amazon dataset was also used by Prof. McAuley to find the implicit tastes of each user and the properties of each product, as written in his paper [2] "Hidden Factors and Hidden Topics: Understanding

Rating Dimensions with Review Text". McAuley and Leskovec ran and interpreted the data through a latent factor recommendation system to take into consideration the user's experiences.

In this paper, we would like to explore various approaches on rating prediction. We compare different models, the advantages and disadvantages of them. Based on this research, we hope to be able to determine appropriate models for future applications on rating predictions.

Ⅱ. Data Analysis

Our dataset consists of 568,454 fine foods reviews from Amazon. The data includes:

- Reviews from Oct. 1999 to Oct. 2012
- 256,059 users
- 74,258 products
- Average score (rating) of 4.18

Column	Description
ProductId	Unique identifier for the product
UserId	Unique identifier for the user
ProfileName	Profile name of the user
HelpfulnessNumerator	Number of users who found the review helpful
HelpfulnessDenominator	Number of users who indicated whether they found the review helpful or not
Score	Rating between 1 and 5
Time	Timestamp for the review
Summary	Brief summary of the review
Text	Text of the review

Table 1: Data columns

To understand the general structure of dataset, we find out some general information and statistics for our feature development in the models. In the dataset, for each row, we have information including the product ID, user ID, profile name, helpfulness numerator, helpfulness denominator, rating score from the user, time stamp of review, summary of review, and the text of review from each user.

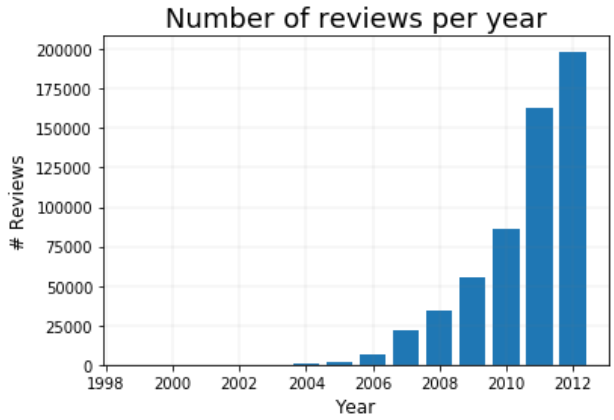


Figure 1: Number of reviews per year

Also, we plot the number of reviews over the time stamps to see if there is any special temporal meaning in our dataset. We can see the trend that the review number is higher is the last few years. We may need to confirm if there is any change in the rating rule in the last few years, or even need to adjust the rating weight and normalize them.

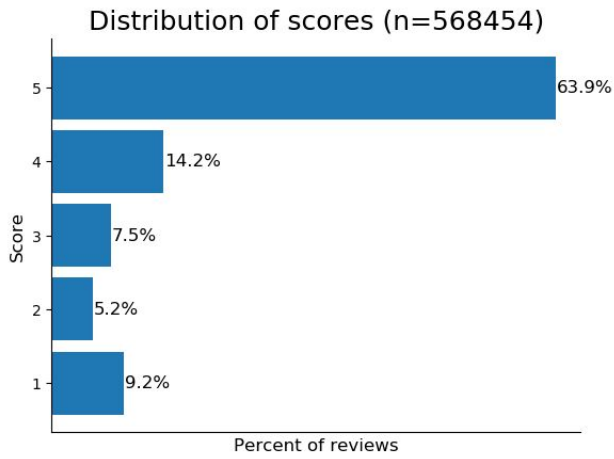
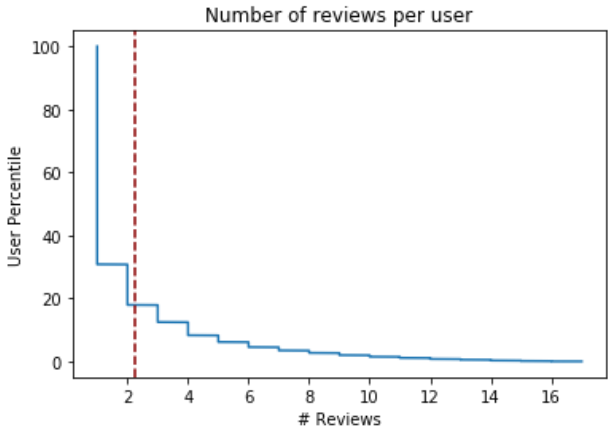


Figure 2: Distribution of ratings

When we look into the distribution of rating scores, we can find that the majority of reviews are rated 5, and average rating is 4.18. In general, we notice that

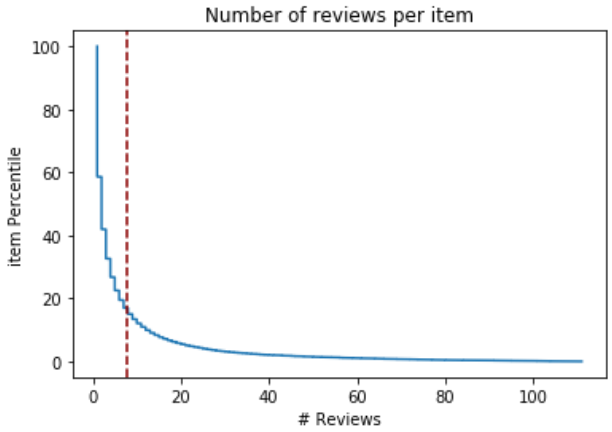
the average score is high and it may need to be considered in the baseline average model.



Min	Median	Mean	Max
1	1.0	2.220012	448

Figure 3: Number of reviews per user (red line is average)

In **Figure 3**, we sort the number of review of each person, and plot out to understand the user percentile and number of reviews. We can find that in average, each user writes 2.2 reviews depending on the dataset. Also, the median (50% percentile) of review given by users is 1. In general, we can find that most people don't give too many reviews, which also imply that our user-product matrix may be sparse, and we may need some factorization methods for modeling.



Min	Median	Mean	Max
1	2.0	7.655121	913

Figure 4: Number of reviews per item (red line=average)

Based on our experience of scrolling through reviews online, we expect that unhelpful reviews tend to be rated 1 star. This is because angry users, out of spite, want to give the lowest possible score. It is less often the case for a well thought-out review to be so critical and only see the negatives. Because of this correlation, we hypothesize that helpfulness of reviews can help us predict review rating. To explore this claim, we plot "HelpfulnessNumerator" (# users who found a review helpful) vs "Unhelpful" (# users who did not find a review helpful) in **Figure 5** for each of the possible ratings 1 through 5.

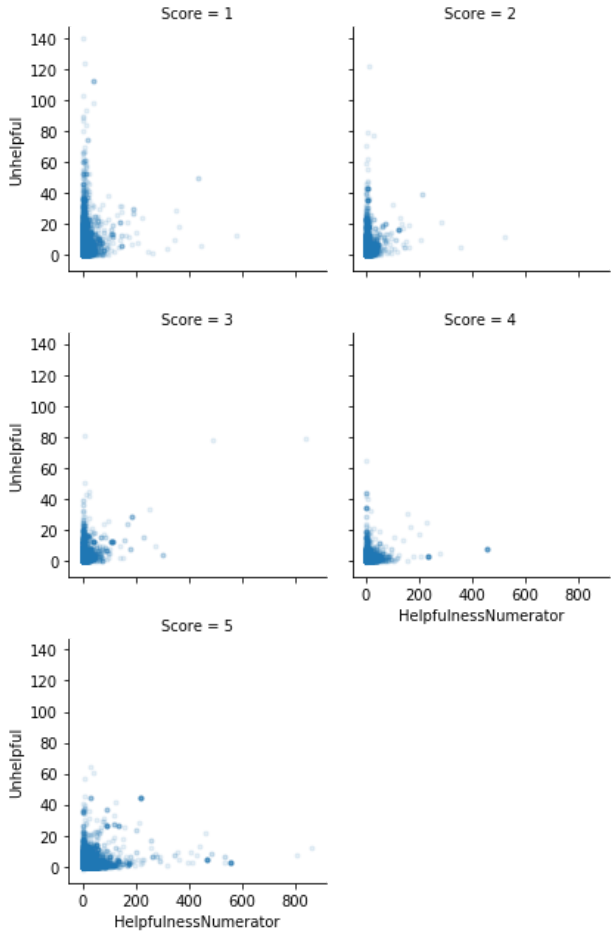


Figure 5: Helpful vs unhelpful reviews per score

Each point in **Figure 5** represents a review and the slope of the line which connects this point to the origin is interpreted as the fraction of users who found a review helpful. Therefore, a slope close to 0 represents a useful review and a sloper close to infinity (vertical) represents an unhelpful review. As

expected, we observe that when people tend to find a review super unhelpful, it tends to be the 1-star rated reviews. However, in a majority of reviews (points), we cannot discern a significant relation between score and helpfulness. Therefore, we do not expect these features to be predictive.

Most of the information in reviews come from the text fields. In **Figure 6** and **Figure 7**, we plot the distribution of word counts for summaries and review texts respectively.

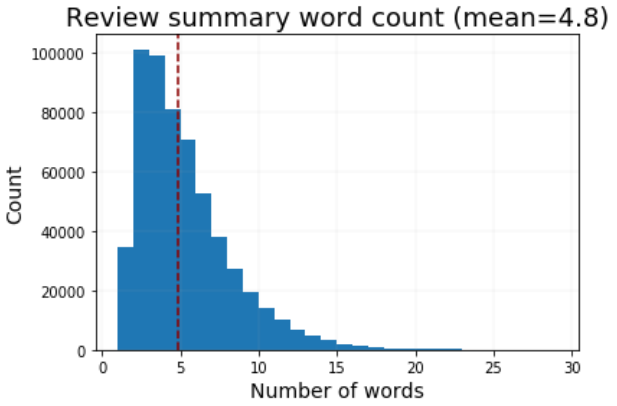


Figure 6: Summary text word count (include punctuation)

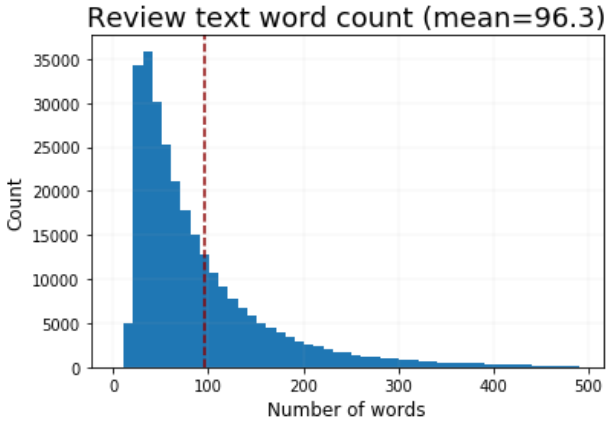


Figure 7: Review text word count (include punctuation)

Reviews appear to be quite rich in information.

We are interested to see the difference in language between reviews with low vs high score. In **Supplementary Table 1** (located in **Appendix**), we calculate the tf-idf matrix and return the top 10 terms for each rating. It is interesting to see that users use emphatic language in their 1-star and 5-star reviews; exclamation marks appear in these top 10 tf-idf weighted terms. In 1-star reviews, we see words like "disappointed"/"horrible" while in 5-star reviews, we see words like "great"/"excellent". In contrast, the language in review summaries of ratings 2-4 is characterized by reservations and milder language ("but", "ok", "..."). Since we are able to discern differences in text between different star ratings, it seems promising to apply text models here to predict review rating.

Ⅲ. Data Preparation and Methods

We split the Amazon dataset into training, validation, and test sets by a 50/25/25 split. We choose this split instead of using cross-validation because we have a lot of data and we want to save on computation time. The validation set is used to tune hyperparameters and the hyperparameter configuration that results in the lowest validation MSE is used to predict on the test set. We do this to avoid overfitting on the test set so that we can be confident about how our models perform on unseen data. We consider several models with varying degrees of success.

A. Baseline

The baseline model is the best possible constant predictor. Using this model, we predict the average score across training samples (4.18). This gives an MSE of 1.718. Other models we try should have an MSE of at most 1.718 to improve upon this baseline.

B. Linear Regression on Helpfulness

We first considered to model the data with a linear regression model. We applied the linear regression model using the same techniques from Homework 1 and 2.

$$\text{rating} \simeq \alpha + \beta_1 \times \text{"HelpfulnessFraction"}$$

We predicted the rating of each item based off their helpfulness ratio score (calculated by # users who found review helpful divided by total votes). We chose this feature based off our findings in our data analysis.

Through our trials leading up to the model above, we considered using 2 other features to model: the number of words in the review text and

the day of week the review was posted. For both models however, both of their MSEs went over 5, meaning that the models predicted worse than the baseline.

C. Latent Factor Model

We apply latent factor model with two different factorization methods, singular value decomposition (SVD) and non-negative matrix factorization (NMF). For SVD, we can write the user-product matrix as the form

$$SVD(A) = U\Sigma V^T$$

where A is an m x n matrix, U and V are m x m and n x n orthogonal matrices respectively. Σ is the m x n singular orthogonal matrix with non-negative elements. For NMF, we can write input data matrix X as the form of $X \approx PQ^T$, where $X \in \mathbb{R}^{N \times M}$, $P \in \mathbb{R}^{D \times N}$ and $Q \in \mathbb{R}^{D \times M}$. We apply SVD and NMF matrix factorization with latent factor model:

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

to minimize the objective:

$$\arg\min_{\alpha,\beta,\gamma} \sum_{u,i}(\alpha+\beta_u+\beta_i+\gamma_u \cdot \gamma_i - R_{u,i})^2 + \lambda\left[\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i \|\gamma_i\|_2^2 + \sum_u \|\gamma_u\|_2^2\right]$$

where α is the global average, βu represents "how much this user tend to rate things above the mean", βi represents "how much this item tend to receive higher rating than others?", γu stands for the preference of users, γi is item's properties, and R is the rating matrix between user and items. We choose hyper-parameters such as λ and K (dimension of latent vector) to optimize our result with alternating least squares method until convergence of MSE on validation set.

$$\alpha = \frac{\sum_{u,i \in Train}(R_{u,i} - (\beta_u + \beta_i + \gamma_u \cdot \gamma_i))}{N_{train}}$$

$$\beta_u = \frac{\sum_{i \in I_u} R_{u,i} - (\alpha + \beta_i + \gamma_u \cdot \gamma_i)}{\lambda + |I_u|}$$

$$\beta_i = \frac{\sum_{i \in U_i} R_{u,i} - (\alpha + \beta_i + \gamma_u \cdot \gamma_i)}{\lambda + |U_i|}$$

$$\gamma_{ukupdate} = 2\gamma_{ik}(\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i}) + 2\lambda\gamma_{uk}$$

$$\gamma_{ikupdate} = 2\gamma_{uk}(\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i}) + 2\lambda\gamma_{ik}$$

$$\gamma_{uk} = \gamma_{uk} - learningrate \cdot \gamma_{ukupdate}$$

$$\gamma_{ik} = \gamma_{ik} - learningrate \cdot \gamma_{ikupdate}$$

In the training process, we frequently get overfitting issue with our model. As a result, we keep tuning the hyper-parameters to get lower test MSE.

D. Ridge Regression on Text Features

Reviews come with summary text and review text. In the interest of time, we decided to predict rating using only the summary text because it is computationally less expensive (summaries are shorter in length) and we believe it to be more predictive. To extract features, we turn to a bag-of-words approach to vectorize our text to prepare for modeling. Specifically, we try combinations of word count and tf-idf weights on various N-grams (unigrams, bigrams, mixed). Then, we apply ridge regression on the outputted feature vectors to predict rating. We decide to use ridge regression to prevent overfitting.

When attempting to implement vectorization, we quickly realized that it was impossible to compute the complete feature matrix because there were simply too many reviews and possible words/tokens to fit into computer memory. Instead, we would need a sparse representation of this matrix which contains

the same information but uses far less memory. This sparse representation is possible because each review only uses a small portion of possible words/tokens. For this reason, we turn to the sklearn implementation of this bag-of-words vectorization, which uses this sparse representation of the feature matrix.

Tokenization is done using the "word_tokenize" function from the nltk library. Punctuation is kept but text is converted to lowercase before tokenizing. Using word count vectorization, our ridge regression predictor is

$$\text{rating} \simeq \alpha + \sum_{w \in \text{text}} \text{count}(w) \cdot \beta_w + \lambda \sum_{w \in \text{text}} \beta_w^2$$

The ridge regression rating predictor using tf-idf vectorization:

$$\text{rating} \simeq \alpha + \sum_{t \in \text{text}} \text{tfidf}(t, D) \cdot \beta_t + \lambda \sum_{t \in \text{text}} \beta_t^2$$

The sklearn implementation of tf-idf with terms (tokens) **t** and documents (reviews) **d** is as follows:

$$\text{tf-idf(t,d)} = \text{tf(t,d)} \times \text{idf(t)}$$

$$\text{idf}(t) = log\frac{1+n_d}{1+\text{df}(d,t)} + 1$$

- **tf(t,d)** = term-frequency (number of times term t occurs in document d)
- **idf(t)** = inverse document-frequency
- **df(d,t)** = document-frequency (number of documents d that contain term t
- **n_d** = total number of reviews

The rationale behind tf-idf is that we weight common words ("the", "is", etc.) less and weight "rare" (more meaningful) words more. Preliminary tests on our validation set reveal that applying tf-idf transformation performs better than using simple word count.

To prevent overfitting on the test set, we select the optimal λ in ridge regression based on the lowest MSE on the validation set. This way, we could get a more realistic estimate of how our model would perform on unseen data.

Lambda	Unigram Word Count	Bigram Word Count	Mixed Word Count
0.01	2.1046	2.8881	1.1518
0.1	2.1063	2.8432	1.1428
1	2.1430	2.9997	1.2243
10	2.3834	4.0388	1.6610
100	2.9347	5.8861	2.5012
Lambda	**Unigram tf-idf**	**Bigram tf-idf**	**Mixed tf-idf**
0.01	1.1263	2.1392	0.7945
0.1	1.1274	2.0896	0.7624
1	1.1986	2.4165	0.8392
10	1.6272	4.0616	1.3574
100	2.6610	6.9929	2.6965

Table 2: Select λ from lowest validation MSE (lowest MSE highlighted in green)

From the **Table 2**, we see that the mixed tf-idf model performs best with the validation MSE of 0.7624. We also notice that a lower λ (0.01) was better suited for unigram models and a higher value of λ (0.01) better suited the more complex models (bigram/mixed). This matches our intuition because ridge regression assigns a higher penalty to more complex models.

Ⅳ. Results

Model	Test MSE	Validate MSE
UnigramBigram TF-IDF	0.7217	0.7624
UnigramBigram Word Count	1.0391	1.1428
Unigram TF-IDF	1.1019	1.1263
Latent Factor Model with NMF	1.2181	1.2321
Latent Factor Model with SVD	1.3381	1.3270
Alpha Baseline	1.7175	1.7175
Linear Regression by Helpfulness	1.7197	1.7197
Bigram TF-IDF	1.9536	2.0896
Unigram Word Count	2.0747	2.1046
Bigram Word Count	2.6336	2.8432

Table 3: Model results

We apply a variety of models including prediction by global rating average (baseline), linear regression by Helpfulness ratio, linear regression of Unigram, Bigram word count and TF-IDF models, and combination of Unigram-Bigram word count and TF-IDF models, latent factor model with matrix factorization methods such as SVD (by python surprise package) and NMF. We first tune the model's hyper-parameters based on the MSE in validation set, and then combine both training and validation set to train with the hyper-parameters, and predict with the test set to obtain the Test MSE as shown in table. The Unigram-Bigram TF-IDF model out-performs compared with all other models, and models such as Unigram-Bigram word count model, Unigram TF-IDF, Latent factor model with NMF and SVD is significantly better than the alpha baseline. Other models such as linear regression with Helpfulness ratio, Bigram TF-IDF, Unigram and Bigram word count models did not perform better than the baseline.

As seen in our data analysis, the linear regression model gave an MSE of 1.7197, which is 0.0022 away from the alpha baseline. This tells us that the helpfulness ratio cannot predict the ratings of the Amazon Food Items and that a linear regression model was not the correct model to plot out the data. This aligns with the Stanford Paper, [4] "Evaluate Helpfulness in Amazon Reviews Using Deep Learning". In the paper, Bobby Nguy concluded that helpfulness ratings depended more on keywords and relevancy in the review texts and therefore is not a good predictor on total rating.

Ⅴ. Discussion

Out of all the models we tried, the test set confirms that the unigram-bigram tf-idf model performs best. This makes sense because it is the most expressive of the text models we tried. It has the benefits of both unigram and bigram models combined with the tf-idf weighting scheme to best extract sentiment from summary text.

We find that latent factor models with both SVD and NMF factorization methods perform better than the baseline method. The potential reasons are: (1) the features of User / Item preferences and properties are more useful than the pure Helpfulness ratio used by the linear regression, (2) latent factor models exploit the correlations between users and items, and (3) the model can prevent potential overfitting issue by adding λ for regularization. However, the disadvantage of the latent factor model is the "cold-start" issue, in which the model cannot predict rating for new users/products. Our group believes that the latent factor models perform worse than the best sentiment analysis model because there is more useful information in the summary of data than the user-product profiles for this dataset.

Our results are comparable to state-of-the art methods also used to predict rating from reviews. The conclusions from the [1] "From Amateurs To Connoisseurs: Modeling The Evolution Of User Expertise Through Online Reviews" (McAuley, Leskovec) show that modeling users' personal evolution allows the data manager to discover "acquired tastes" in product rating systems and when users acquire them. Using this method, they were able to improve upon the standard latent-factor model (with MSE of 1.425) and obtain a better model with a lower MSE of 1.1189 on the same dataset (Amazon Fine Food Reviews). This is better than our latent-factor model with NMF, from which we obtained an MSE of 1.2181. However, it appears our mixed tf-idf model performs better than this with an MSE of 0.7217. Of course, we need to consider that the model used by McAuley and Leskovec does not take review text into account. Also, our studies report MSEs from different test sets. In a future study, we may want to apply their method of modeling user experience on our test set to give a fair comparison.

Other studies have used the same dataset but analyzed it from a different perspective. The conclusion from the [2] "Hidden Factors and Hidden Topics: Understanding Rating Dimensions with

Review Text" paper by Prof. McAuley show that by using a HFT (Hidden Factors as Topics) model, we can accurately fit user and product parameters with only a few reviews, which existing models cannot do using only a few ratings and a bunch of user profiles.

In future work, we can run the same experiment on review text to see if this improves our results. Since review text is much longer than summary text, we may have to look into feature hashing [3] to speed up computation. We can investigate how stemming and removing stop words affects rating prediction in addition to considering other techniques like Recursive Neural Network for Multiple Sentences [5] that have been proven to give excellent results.

VI. References

[1] J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. WWW, 2013.

[2] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. RecSys, 2013.

[3] Kilian Weinberger; Anirban Dasgupta; John Langford; Alex Smola; Josh Attenberg (2009). *Feature Hashing for Large Scale Multitask Learning* (PDF). Proc. ICML.

[4] Nguy, Bobby (2016). Evaluate Helpfulness in Amazon Reviews Using Deep Learning (PDF). WWW

[5] Ji, T., & Wu J. (2016). Deep Learning for Amazon Food Review Sentiment Analysis. http://cs224d.stanford.edu/reports/WuJi.pdf

VII. Appendix

Score: 1		Score: 2		Score: 3		Score: 4		Score: 5	
term	**weight**	**term**	**weight**	**term**	**weight**	**term**	**weight**	**term**	**weight**
!	0.031084	not	0.041369	not	0.034689	good	0.04275	!	0.052481
not	0.027401	too	0.014854	but	0.028641	!	0.022078	great	0.033784
! !	0.0179	disappointed	0.014685	,	0.023831	great	0.021548	delicious	0.0185
?	0.012665	.	0.014497	good	0.021746	,	0.01952	! !	0.018046
.	0.012195	,	0.014073	ok	0.018659	but	0.01688	best	0.016471
disappointed	0.011431	the	0.013077	, but	0.016326	a	0.013812	love	0.01589
horrible	0.010623	...	0.012785	...	0.015416	tasty	0.013108	good	0.01523
terrible	0.010268	!	0.012449	.	0.014874	.	0.012402	the	0.014162
the	0.010194	but	0.011457	a	0.012992	coffee	0.012041	excellent	0.011976
n't	0.010092	taste	0.011289	the	0.012878	for	0.011877	product	0.011969
awful	0.009634	n't	0.010692	it	0.012608	very	0.011441	coffee	0.011914

Supplementary Table 1: Top 10 tf-idf terms for each rating (using Mixed Unigram-Bigram model)