Score Prediction of Amazon Video Game Reviews through Collaborative Filtering Techniques

Pranay Kumar Myana and Kashyap Tumkur

Abstract— **In this report, we examine the Amazon Video Game dataset [4], a corpus of review scores of video game products. We explore the characteristics of the dataset and use this to choose the task of predicting the scores of product reviews. We use collaborative filtering techniques to predict these review scores and compare the efficacy of naive, bipartite projection and latent factor models, and improve upon the results obtained by the bipartite projection approach in [1]. We end the report with a discussion of our observations and the lessons learned from the project.**

I. INTRODUCTION

The Amazon Video Game dataset is a corpus consisting of 463,668 product reviews spanning a period of 16 years from February 1997 to March 2013, made available by the Stanford Network Analysis Project [4].

In the first section, we analyze the data in an exploratory fashion to understand the available features of each review, the scope of each of these features, and the relevance of these features to each other. In the next section, we use this information to explore the tasks of predicting *helpfulness* of a review, as measured by the number of "helpful" votes a review has received as a fraction of the total number of the votes it received. We characterize this notion of helpfulness in several distinct ways and describe the effectiveness of these methods.

We next explore the task of predicting the score attached to a review, which is the focus of this report. We investigate the correlation of review score with other features such as helpfulness, the review text, or the product price, and decide to pursue this task for the purposes of this project.

The third section describes related work in the field, especially regarding the collaborative filtering techniques we use. We describe existing research we draw from, and the results therein that we improve upon.

For the task of score prediction, we begin with a naive linear model which serves as the baseline for further efforts. We then implement collaborative filtering techniques of bipartite projection and latent factor models and observe their performance on this task and further optimize these models based on our initial observations. We describe these models in the fourth section of this report.

Next, we draw conclusions from the results of our experiments and delineate the reasons for why our improvements worked (and did not). We also describe possible suggestions for improvement and future work in the next section.

Finally, we describe the lessons learned from this project and the important takeaways obtained from our experiments.

II. EXPLORATORY ANALYSIS

A. The Data

In this section, we examine the characteristics of the data and the various features of the dataset as a whole. The 463,668 reviews of the Amazon Video Game dataset occupy $463MB$ of space in uncompressed text format.

Each review contains the following fields:

- `'review/profileName'`: the profile name of the reviewer, usually in colloquial English.
- `'review/time'`: the UNIX timestamp of the time the review was written.
- `'review/helpfulness'`: provided as a fraction with numerator the number of helpful votes and denominator the total of all votes received by the review, e.g., "5/11".
- `'review/summary'`: a reviewer-provided summary of the review in natural language.
- `'review/userId'`: the ID of the user, which is unique across all users in the dataset.
- `'review/score'`: the score for the product assigned by the reviewer, which can vary from 1.0 to 5.0.
- `'review/text'`: the natural language text of the review.

The review data also provides the following information about the product itself, attached to each review:

- `'product/price'`: the price of the product in USD.
- `'product/productId'`: the ID of the product, which is unique across all products in the dataset.
- `'product/title'`: the title of the product in natural language.

Using the `'review/userId'` and `'product/productId'` fields, we observe 228,570 unique user IDs and 21,025 unique product IDs in the dataset. However, we observe that the `'review/userId'` field can have the value "unknown", and a total of 99,128 reviews across the 463,668 reviews in the dataset are attributed to user "unknown". As fitting a model to "unknown" does not amount to fitting a model for any particular user, we omit these reviews from the dataset, leaving us with a total of 364,541 reviews.

Further, for 3970 of the reviews, it is the case that another previous review for the same product exists by the same reviewer. For each of these cases, we simply take the latest review into account and omit the preceding reviews. This brings us to a total of 360,571 reviews in our dataset.

We now look at the distribution of reviews across the users. We notice that aside from two power users, ”A3V6Z4RCDGRC44” and ”AJKWF4W7QD4NS” with 750 and 529 reviews respectively, all the users have lesser than 300 reviews. The number of users falls even lower as we lower the number of reviews: there are only 132 of 228,569 users with at least 50 reviews, and the remaining users are shown in Fig. 1 (since this is much easier to visualize than across the entire range of reviews, which results in a far emptier graph). The spike at number of reviews equal to 1 corresponds to 180,139 of all users, which is almost 80% of all users. Fig. 2 depicts the entire distribution as a heat map.

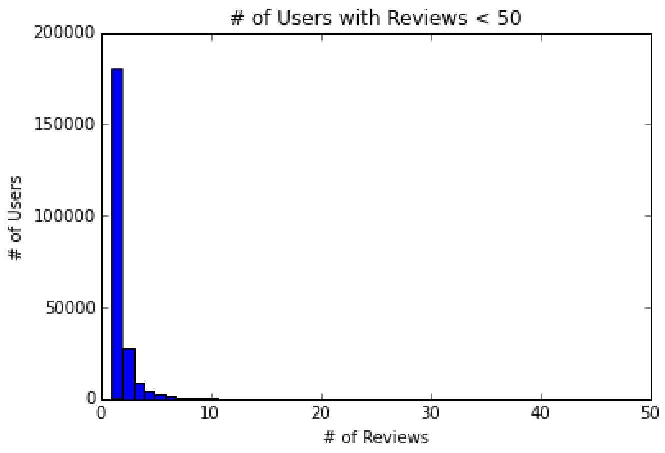

Fig. 1. Distribution of Reviews across Users

Similarly, we also look at the distribution of reviews across products. Four products, ”B000FKBCX4”, ”B000B9RI0K”, ”B000N5Z2L4”, and ”B0009VXBAQ” have between 2000 to 3300 reviews each, with all other products having far lesser than 2000 reviews. Again, we choose a handy threshold value of 200 reviews for visualizing this distribution. All but 331 of the products have reviews less than 200, which are shown in Fig. 3. In this case, there is a more gradual dropoff, with 3,925 products having only 1 review, 6,279 having at most 2 reviews, and 13,544 of 21,025 products having at most 10 reviews. Fig. 4 depicts the entire distribution as a heat map.

From the heat maps, we notice that the dataset is very sparse with information about specific users, with their opinion about only one product being known for a majority of the users. The information for products is more detailed,

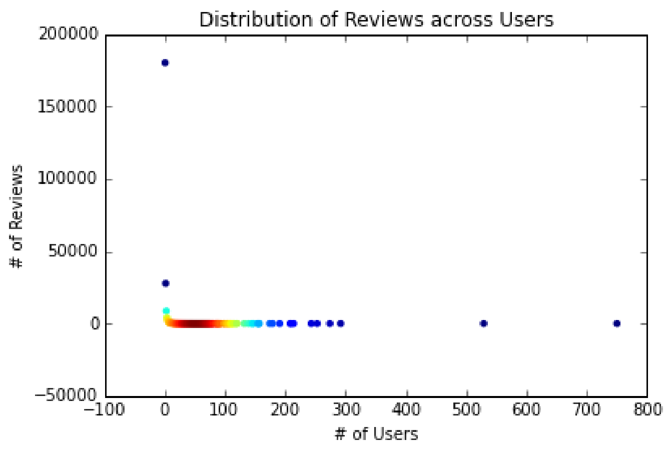

Fig. 2. Distribution of Reviews across Users

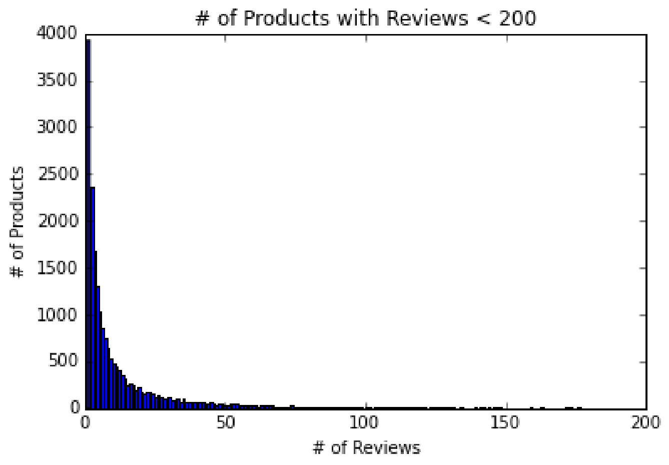

Fig. 3. Distribution of Reviews across Products

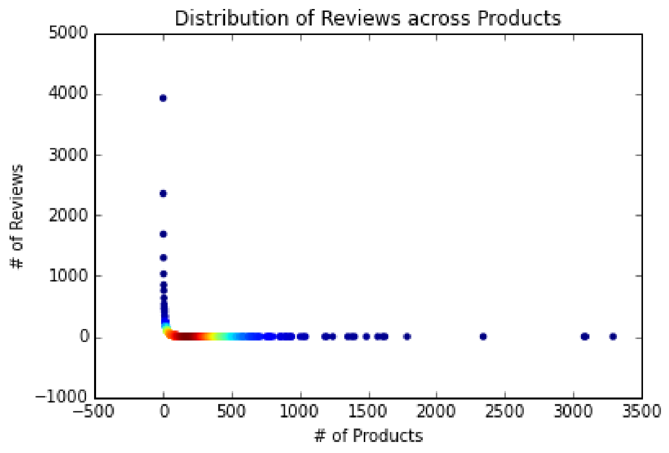

Fig. 4. Distribution of Reviews across Products

with products having an average of 22.05 reviews with a median value of 6.0.

B. *Helpfulness*

In this subsection, we examine the representation of *helpfulness* of a review. The dataset, by default, provides helpfulness as a fraction, $\#helpful/\#total$, where $\#helpful$ is the number of ”helpful” votes received by a review, and $\#total$ is the total number of votes received by the review (i.e., including the votes that did not rate the review ”helpful”.)

However, simply using the ratio of $\#helpful$ to $\#total$ renders the same weight of a review with $\#helpful = \#total = 1$ and $\#helpful = \#total = 100$. We would like to increase the weight given to factors such as the total number of votes cast, or the fraction of users who found the review more helpful than not. Therefore, given $\#helpful$ and $\#total$, we look at the following different possible metrics of helpfulness that can be used to represent the data better. In a later section, we fit models to each of these metrics to observe the advantages and disadvantages of each quantitatively.

- $\frac{\#helpful}{\#total}$: the simplest ratio. We plot the distribution of reviews with this metric in Fig. 5. We notice that

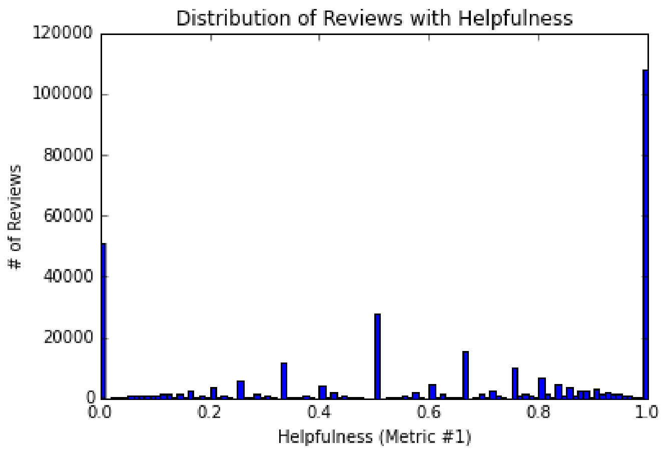

Fig. 5. Distribution of Reviews with Helpfulness

  there are unusually high spikes at Helpfulness = 0.0 and Helpfulness = 1.1. This indicates that it is likely that many reviews only have 1 vote (or similarly low values), and the heplfulness of that vote swings the entire ratio of helpfulness of the review. Our plot in Fig 6 corroborates this. Specifically, 56% of reviews have votes in the range of 1 to 5. Further, most of the votes allotted to these reviews with small numbers of votes are ”helpful”, resulting in the large spike at helpfulness = 1.0. However, this does not imply that the review will be helpful to the general public.
- $\frac{\#helpful}{\#total} * log(\#total + 1)$: which increases the weight given to the total number of votes, so that we do not have the issues seen with Metric #1. This distribution

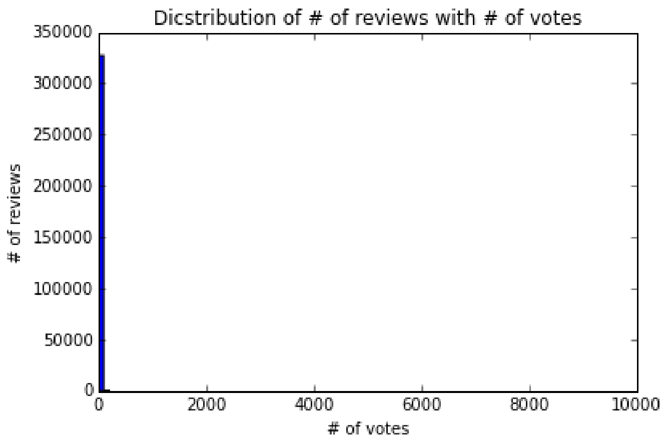

Fig. 6. Distribution of Reviews with # votes

  is shown in Fig. 7. However, this metric results in only a representation of ”helpful”, and no way to quantify that a review is ”not helpful”. We address this issue in Metric #3.

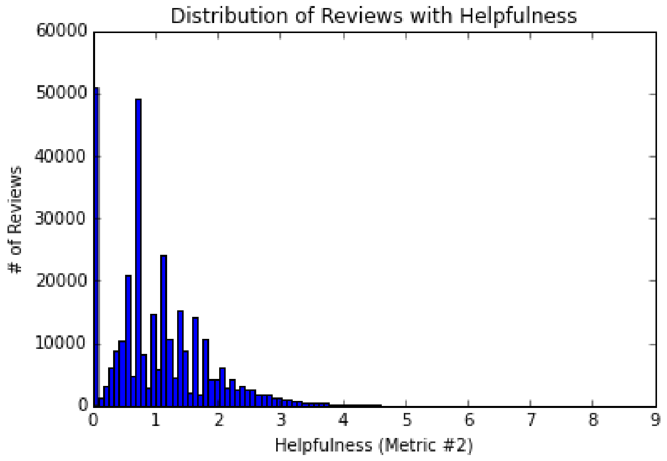

Fig. 7. Distribution of Reviews with Helpfulness

- $\frac{\#helpful - \#notHelpful}{\#total)} * log(\#total + 1)$: which adds weight to the degree by which a review is more helpful to users than not. Here, $\#notHelpful$ is simply equal to $(\#total - \#helpful)$. This spreads the distribution over the entire number range, with helpfulness scores below 0 indicating ”not helpful”, as shown in Fig. 8.

C. *Product Price*

Another numerical feature that we examine is the price of a product in our dataset, for all products that do not have the price field marked ”unknown”. As can be seen from Fig. 9, the majority of products have low prices (prices $<$ 50). Quantitatively, we find that 77% of all products have a price lesser than \$40.00.

We are interested in exploring the relationship of the price of a product to the number of reviews it gets. Presumably,

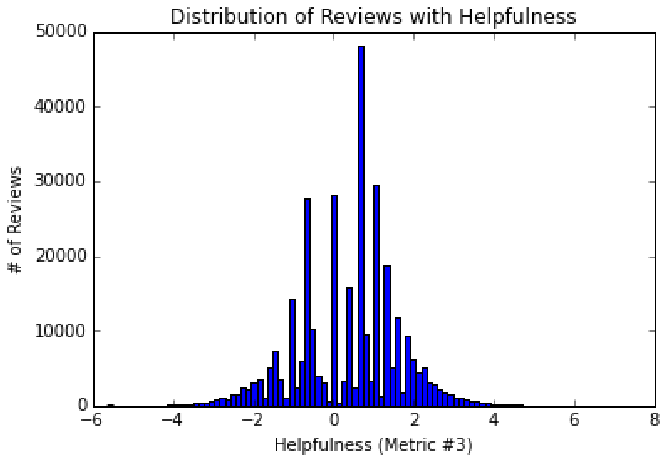

Fig. 8. Distribution of Reviews with Helpfulness

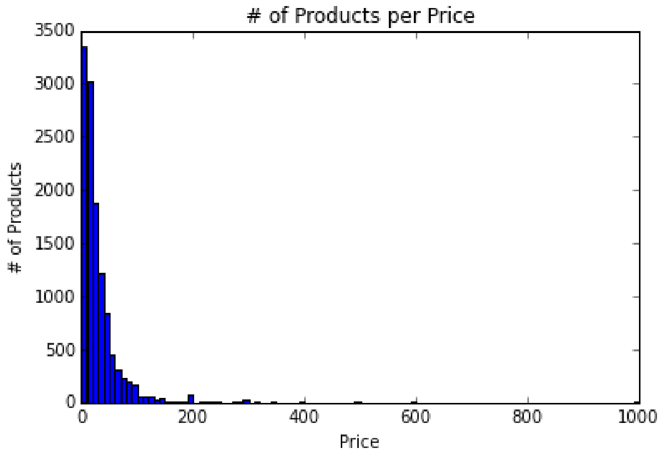

Fig. 9. Distribution of Product Prices

cheaper products are bought more often and by more people, and thus should receive a correspondingly higher number of reviews. Fig. 10 displays a heat map of the price of a product vs. the number of reviews received.

We can make the following observations from this heat map:

- There is a high concentration of products with very low prices and with very low number of reviews, which is primarily due to the high number of products with low prices.
- While there seems to be a trend of the number of reviews decreasing with the increase in price, it is also not possible to conclude that it is the case due to the extremely high number of low-priced prices against high-priced process.

D. Pairwise Correlations of Features

We explore the correlations between pairs of features that appear likely to be related. While the dataset does not lend itself to many correlations in data, one of the interesting

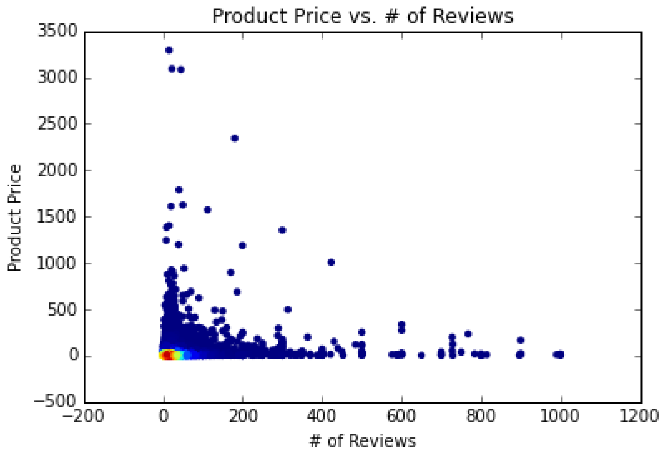

Fig. 10. Distribution of Reviews with Price

correlations was that of the length of the review text with the review score.

It is presumable that the more interested a reviewer is in a product, they are more likely to write longer reviews. However, from Fig. 11, we observe that there is a large concentration of reviews with high (4 or 5) scores with very little text at all (close to 0). This indicates that many reviews are perfunctory and that the reviewer found the score a sufficient means of expressing his (positive) opinion.//

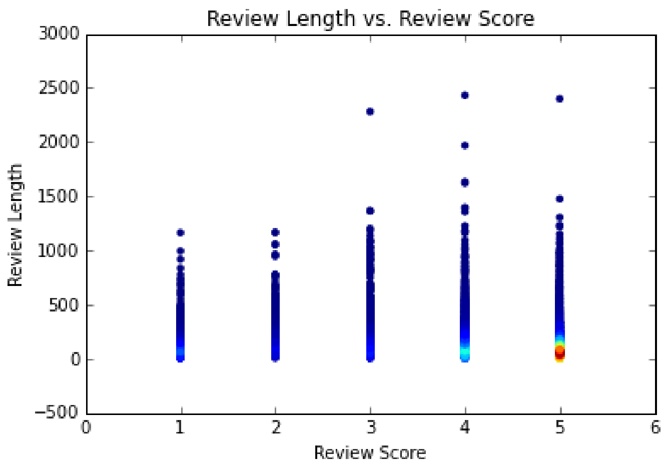

Fig. 11. Review Length vs. Review Score

In the following section, we use this information to analyze two predictive tasks and choose the task for the remainder of our report.

III. IDENTIFYING A PREDICTIVE TASK

In this section, we look at two possible predictive tasks and analyze these by fitting basic models to them. We then decide upon one as our focus for the remainder of the report.

A. *Helpfulness Prediction*

In this section, we consider various features of the dataset that are likely to affect the helpfulness of a review. We fit a linear regression model using these features and observe their performance. We perform this for each of the three metrics for helpfulness discussed in the previous section.

We first consider the impact of the length (number of words) of the review text on the helpfulness of a review with the consideration that it is likely that a review is more helpful the more detailed (and hence the more lengthy) it is. We observe the following results for each of the helpfulness metrics:

- A linear regression model fit to helpfulness represented as $\frac{\#helpful}{\#total}$ results in a Mean Squared Error (MSE) of 0.133885. The range of helpfulness scores in data ranges in [0.0, 1.0] when represented by this metric.
- When the helpfulness is represented as $\frac{\#helpful}{\#total} * log(\#total+1)$, the range of scores in the data also increases to [0.0, 8.372042]. The linear regression model achieves an MSE of 0.611337 on the data.
- We also represent helpfulness as $\frac{\#helpful-\#notHelpful}{\#total)} * log(\#total + 1)$. This increases the range of helpfulness scores in data to [-5.616637, 7.604703]. The linear regression model fit to this achieves an MSE of 1.593324.

Next, we introduce various features to the linear regression model with helpfulness represented by the third metric. We observe the following changes:

1) We introduce the review timestamp as a feature to the linear regression model. This improves the MSE of the model to 1.584711.
2) We next introduce the length of the review summary as well, which improves the MSE of the model to 1.580600.
3) Finally, we introduce the review score as a feature to the model, which turns out to be highly correlated with helpfulness. The MSE is improved to 1.51516.

Of these features, we observe that the most highly correlated feature with helpfulness is the review score. The weights for (intercept, review length, review timestamp, review summary length, review score) turn out to be (4.938945e-02, 1.903170e-03, -7.138889e-10, 2.576660e-02, 1.792664e-01) respectively, which correspond to the correlation of the feature with the helpfulness score. We also tabulate the MSE and ranges of fitted values in Table I.

TABLE I
RESULTS OF HELPFULNESS PREDICTION

Model	MSE	Range of Helpfulness Scores in Data
Metric #1	0.133885	[0.0, 1.0]
Metric #2	0.611337	[0.0, 8.372042]
Metric #3	1.593324	[-5.616637, 7.604703]
Metric #4 and all features	1.580600	[-5.616637, 7.604703]

B. *Review Score Prediction*

In this section, we consider the task of predicting the review score through the review features present in the dataset. We do so by fitting a linear regression model to the data and observing the performance of the fitted model.

We fit the following variations of the model:

1) We use the features of review length, review timestamp, and length of review summary to predict the score of the review. This results in an MSE of 2.036195 for the data, which contains scores in the range [1.0, 5.0].
2) We next introduce the helpfulness score, as represented by the third metric in the previous section, as a feature to the model. Once again, we observe a high degree of correlation of this feature with the review score, with the MSE improving to 1.951897.

The weights of the fitted model for (intercept, review length, review timestamp, length of review summary, helpfulness score) are (5.08290695e+00, -3.06708772e-04, -1.09816114e-09, -1.48011700e-02, 2.30938527e-01), which also indicates the relatively high correlation of review score with helpfulness. We also tabulate the MSE of the models in Table II.

TABLE II
RESULTS OF SCORE PREDICTION

Model	MSE	Range of Scores
Model #1	2.036195	[1.0, 5.0]
Model #2	1.951897	[1.0, 5.0]

Based on these results, we choose the task of predicting review scores due to the following reasons:

1) The task of review score prediction is challenging and not easily modeled by the other features in data, which suggests collaborative filtering techniques as an useful tool.
2) The task of helpfulness prediction depends on the metric used to represent it, which is ambiguous.

We choose to predict the review scores through collaborative filtering techniques, which do not require explicit features to be present in the data. Further, the neighborhood-based approaches of collaborative techniques do not require the tuning of any hyperparameters, which reduces the complexity in implementing and using the predictive models.

IV. RELATED WORK

We first approach the task of review score prediction using the neighborhood-approach collaborative filtering technique as presented in [1]. We use the approach used in [2] where this model is implemented on Yelp data as a reference. We modify the similarity metric to a more intuitive metric, which results in an improvement of our results over that of the algorithm in [1].

Latent factor models are a popular technique for explaining observed data by uncovering latent features in data **??**. There exist several well-known techniques such as matrix decomposition. However, such approaches do not apply in our case due to the sparsity of the dataset, with a majority of the $(user, product)$ pairs for rating scores missing from the dataset. Therefore, we opt for a representation of the problem as an optimization problem; as this turns out to be non-convex, we solve approximately using the technique of Alternating Least Squares **??**.

V. FEATURE SELECTION

As described in the previous sections, we looked at various features in the given data and how they correlate with each other. We were particularly interested in either predicting helpfulness or predicting review score. We chose to move forward with review score prediction.

Choosing features for review score prediction is not straight-forward. Most of the features made available in the dataset are various attributes of the review. There are only three features of the product (`price`, `productId`, `title`) and only one of these features seem to be of use, if any, in predicting review score for an unseen (user, product) pair. Review features cannot be used in the prediction, as these features (in fact, the review itself) will not be available for the given unseen pair. Hence, we chose to go with collaborative filtering models which do not need any features (of the user or the product).

VI. MODEL DESCRIPTION

A. *Naive baseline*

As a baseline, we used a naive model to predict the score for a given user-product pair. We model each user u with a single parameter δ_u and each product p with a single parameter δ_p.

Let μ be the average of all the review scores in the training data, and β_u and β_p be the average score of user u and product p respectively. Then δ_u and δ_p are the bias of u and p, respectively, over the average μ.

$\delta_u = \beta_u - \mu$ and $\delta_p = \beta_p - \mu$

δ_u quantifies the tendency of user u to rate products above the mean while δ_p quantifies the tendency of product p to receive higher scores than others.

The naive prediction model is

$$\hat{r}_{u,p} = \alpha + \delta_u + \delta_p \tag{1}$$

Given a user-product pair this model naively predicts the score by adding the user and product bias to the average score.

We observe that all the scores in the dataset are integers, as shown in Fig. 12. The scores range from 1 to 5. This can also be verified from Fig. 11. Based on this observation, we round the score given by the prediction models (naive baseline as well as the bipartite projection model and its variations) to the nearest integer (using `numpy.round`).

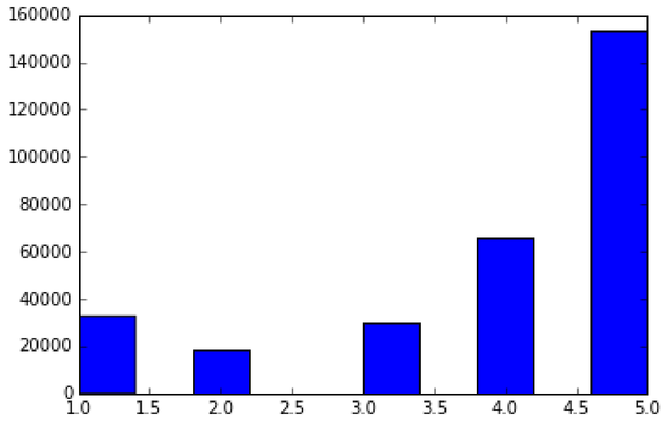

Fig. 12. # Reviews vs. Review Score

B. *WEIGHTED BIPARTITE PROJECTION*

In this section, we describe the algorithm employed in our collaborative filtering model [[1]].

Collaborative filtering (using neighborhood approach) is defined as - "a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue x than to have the opinion on x of a person chosen randomly"[[3]].

This definition, in particular, defines user-based collaborative filtering method. Item-based collaborative filtering method can be defined in a similar way. In this report, we focus only on user-based collaborative filtering approach.

Prediction of the score of user u towards product p is given by:

$$\hat{r}_{u,p} = \beta_u + \kappa \sum_{v=1}^{N} sim(u,v)(r_{v,p} - \beta_v) \tag{2}$$

Recall that β_u is the average review score of user u. N is the total number of users in the data.

$sim(u,v)$ is the similarity measurement between user u and user v. It is the the way we quantify this similarity, is what differentiates the numerous collaborative filtering models (using neighborhood approach). *Cosine Similarity*, *Pearson Correlation Similarity* are some of the well-known similarity measures in use.

1) Algorithm - Recommendation power: The similarity metric used in this algorithm is termed *Recommendation Power*. Recommendation power of v with respect to u, $RP(u, v)$,can be interpreted as - "how likely user u is to choose user v to recommend a product for u". From this definition, we can see that $RP()$ is not commutative, i.e., $RP(u, v) \neq RP(v, u)$.

We use weighted bipartite projection to evaluate $RP(u, v)$. If we create a graph with all the users and products as the vertices, and we have an edge from user u to product p if and only if u has rated p. This graph is constructed from the training data. RP values are calculated from this graph, by projecting the products onto the user-space (hence the name). User u gives some recommendation power to user v, if they've reviewed the same product p, and the value is dependent on the scores ru, p and $r_{v,p}$. We can consider these scores as weights of the corresponding edges in the bipartite graph.

Interpretation: User u likes to distribute recommendation power to users (so that they can recommend products to u), based on the scoring pattern (the graph). But users are not connected directly in this bipartite graph as we have edges only across users and products. So u distributes the power across all the products he has reviewed (depending on his review score towards each product) and each product, in turn, distributes the power it received from u across all the users that reviewed the product (depending on the review score of each user towards this product). This explains how $RP(u, v)$ is evaluated, for every user v.

$$RP(u, v) = \sum_p \frac{r_{u,p}}{R_u} * \frac{r_{v,p}}{R_p} \tag{3}$$

R_u is the sum of all the review scores given by user u and R_p is the sum of all the ratings received by product p.

C. VARIATIONS

Looking at the definition of recommendation power in the above section, $RP(u, v)$, due to product p, is directly proportional to $r_{u,p}$ and $r_{v,p}$. Suppose we have $r_{u,p} = 1.0$, which means that u clearly does not like the product p. Consider to other users v and v' such that $r_{v,p} = 5.0$ and $r_{v',p} = 1.0$. From the above definition, p gives more recommendation power of u to v than to v'. But we can clearly see that both u and v' do not like p whereas v likes p. It seems more intuitive to give more recommendation power of u to v' than to v, as v' is more similar to u. Thus, the above definition of recommendation power (similarity) seems counter-intuitive.

For this reason, we try different variations of recommendation power to capture similarity, which, we feel, are more intuitive. And the results show that this is indeed the case.

We realize that users are more similar when their review scores for the same item are similar. We experiment with several variations that are consistent with this idea.

- Variation 1:
$$RP(u, v) = \sum_p (5.0 - \frac{|r_{u,p} - r_{v,p}|}{C_p}) \tag{4}$$
where C_p is the number of reviews for product p.
- Variation 2:
$$RP(u, v) = \sum_p (5.0 - \frac{|(r_{u,p} - \beta_u) - (r_{v,p} - \beta_v)|}{C_p}) \tag{5}$$
The idea is - instead of using the actual scores, we consider how strong the users' preference (like or dislike) is towards the product, which is captured in $r_{u,p} - \beta_u$.
- Variation 3:
$$RP(u, v) = \sum_p \frac{|(r_{u,p} - \beta_u) + (r_{v,p} - \beta_v)|}{C_p} \tag{6}$$
This variation tries to capture the similarity in the preference (like or dislike) of the users u and v towards the product. Preferences of the same sign (+/like - /dislike) result in a larger magnitude than preferences of different signs. One of the issues of this model is case 1: $r_{u,p} - \beta_u = 3$, $r_{v,p} - \beta_v = -1$ has the same score as case 2: $r_{u,p} - \beta_u = 1$, $r_{v,p} - \beta_v = 1$ which does not seem appropriate.
- Variation 4:
$$RP(u, v) = \sum_p \frac{\exp\{-|(r_{u,p} - \beta_u) - (r_{v,p} - \beta_v)|\}}{C_p} \tag{7}$$
Here, we use the inverse of the exponential of the absolute difference in the preferences of u and v. It achieves the maximum when the preferences are the same. It also eliminates the issue raised in variation 3, and also provides a better representation than variation 2.
- Variation 5:
$RP(u, v) =$
$$\sum_p \frac{\exp\{-|(r_{u,p} - \beta_u) - (r_{v,p} - \beta_v)|\} * |(r_{u,p} - \beta_u)|}{C_p} \tag{8}$$
In this variation, the recommendation power given to user v via product p is weighted by the magnitude of the preference of u towards p. If u has a strong preference towards p, it gives a strong signal about u's preferences and it has to be sufficiently accounted for. Later, we see that this variation results in the best performance.

Note: In all these variations the recommendation power values are normalized such that, $\sum_v RP(u, v) = 1, \ \forall u$.

The results of naive baseline model, weighted bipartite projection and all its variations are presented in TableIII.

From the above table, we can see that bipartite projection model performs better than the naive model we have for baseline. The variations we tried improve the performance

TABLE III
RESULTS OF BIPARTITE PROJECTION MODEL AND VARIATIONS

Model	MSE	MAE
Naive Model	0.8627	0.4429
Bipartite projection	0.8363	0.4141
Variation 1	0.8360	0.4142
Variation 2	0.8359	0.4139
Variation 3	0.8429	0.4193
Variation 4	0.8350	0.4134
Variation 5	**0.8349**	**0.4132**

of the model by a small margin. Variation 5 gives the best performance, and it is not surprising as this model seems the most appropriate intuitively.

D. LATENT FACTOR MODELS

Latent factor models are a well-known technique for explaining observed data by uncovering latent features [5]. In this subsection, we aim to fit a latent factor model to the observed review scores without taking into account any of the features associated with a review other than the user and product IDs.

We achieve this by extending the naive model described above:

$$\hat{r}_{u,p} = \alpha + \delta_u + \delta_p \tag{9}$$

With a latent factor model, we aim to represent the users and products in the same K-dimensional space, and use the dot product of these representations as an additional factor to the model. Each user u is represented by a K-dimensional vector, γ_u, and each product p is represented by a K-dimensional vector, γ_p. The dot product of the vectors, $\gamma_u.\gamma_p$, is an additional scalar factor to the model as follows:

$$\hat{r}_{u,p} = \alpha + \delta_u + \delta_p + \gamma_u.\gamma_p \tag{10}$$

We represent this in the objective function as follows. Then, we fix α, δ_u and δ_p as in the naive model, and attempt to learn γ_u and γ_p for each user u and each product p respectively. We learn these values using Alternating Least Squares [6], wherein we first fix γ_u for all u, learn the values of γ_p for all p, and vice versa. We perform this step repeatedly until a stopping condition (either when the absolute difference in the errors of two consecutive iterations falls to 10^{-7}, or a maximum number of iterations) is satisfied.

$$\arg\min_{\gamma_u,\gamma_p} \sum_{u,i} (\hat{r}_{u,p} - R_{u,p}), \tag{11}$$

where $R_{u,p}$ is the known review score of the (user, product) pair.

The actual updates of the γ_u and γ_p for all u, p are performed as follows. This also indicates the need for fixing α, δ_u and δ_p values for all u, p.

1) We fix the values of γ_u for all u. To learn the values of γ_p for a particular product p, we go through all the instances of the training data that involve p. For each of these instances (u', p) with all values of u', we set $\gamma_{u'}$ as the features and $R_{u',p} - \alpha - \beta_{u'} - \beta_p$ as the target to be predicted. Thus, we obtain the feature matrix X and targets y, which gives us a linear matrix equation $X\gamma_p = y$ for which we find the least-squares solution. The solution thus obtained is the value of γ_p. We perform this step for all products p.
2) Similarly, for learning γ_u for a particular user u, we first fix the values of γ_p for all p. We then construct the feature matrix X as the values of $\gamma_{p'}$ for all training instances (u, p'). The respective targets are obtained as $R_{u',p} - \alpha - \beta_u - \beta_{p'}$, as Y. By solving the equation $X\gamma_u = y$, we obtain the values of γ_u. We then repeat this procedure for all values of u.
3) We perform this procedure repeatedly until the stopping condition is met.

To avoid overfitting the data too closely, we also perform $L2$ regularization, which modifies the objective function to the following:

$$\arg\min_{\gamma_u,\gamma_p} \sum_{u,i} (\hat{r}_{u,p} - R_{u,p}) + \lambda(\sum_u \|\gamma_u\|_2^2 + \sum_p \|\gamma_p\|_2^2), \tag{12}$$

where λ is the regularization parameter.

In addition to learning γ_u and γ_p, we must tune the values of the hyperparameters K and λ. We perform this through grid search over various combinations of values of λ and K as follows:

1) We divide the dataset into four sections: a training set of 180,000 reviews, a validation set of 60,000 reviews for tuning the value of λ, a validation set of 60,000 reviews for tuning the value of K, and a test set of 60,000 reviews for evaluating the performed of the final model.
2) We vary the value of λ across the range of values in (0.0001, 0.001, 0.1, 1.0, 100.0, 10000.0). We then choose the value of λ that obtains the best performance on the validation set.
3) For each value of λ, we vary the value of K across the range of values in (1, 2, 4, 8, 16) and observe the errors.
4) We fix λ and K to the learned values and evaluate the performance of the model on the test set of data.

We observe the errors on the training scores as λ varies, depicted in Fig. 13. As expected, the error on the training set increases with an increase in the value of λ. The error on the validation set also initially drastically decreases as regularization increases, and then stabilizes after $\lambda = 0.1$.

We also observe the errors on training scores while tuning K to ensure that the regularization performs as expected, as depicted in Fig. 15. However, the validation errors while tuning K do not change very much, as depicted in 16. Thus,

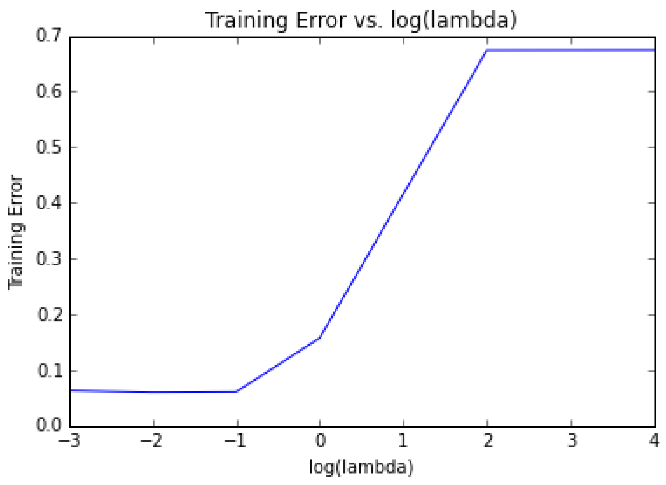

Fig. 13. Training Error vs. log(λ)

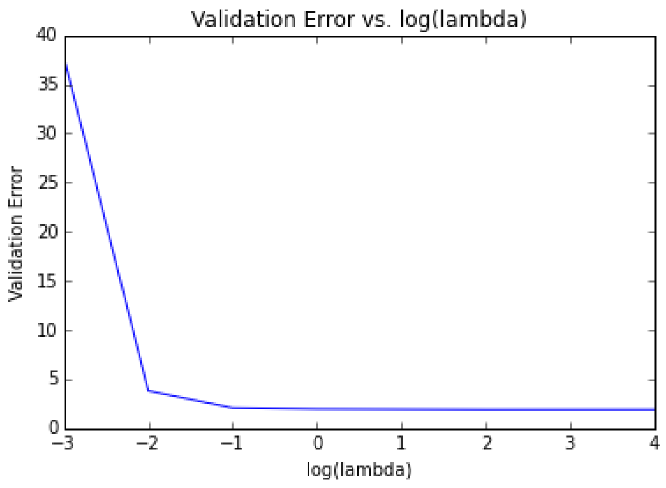

Fig. 14. Validation Error vs. log(λ)

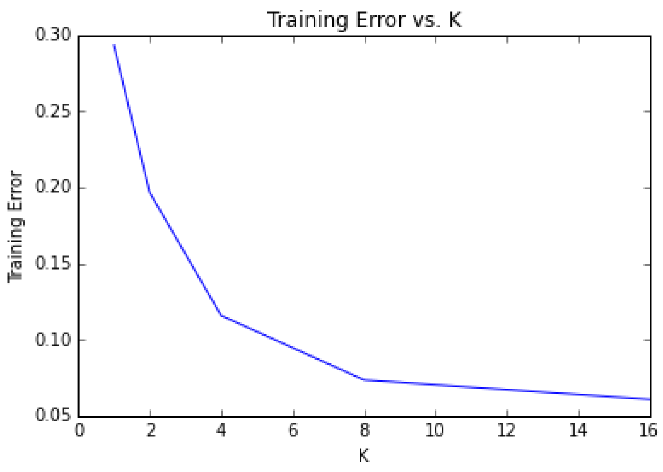

Fig. 15. Training Error vs. K

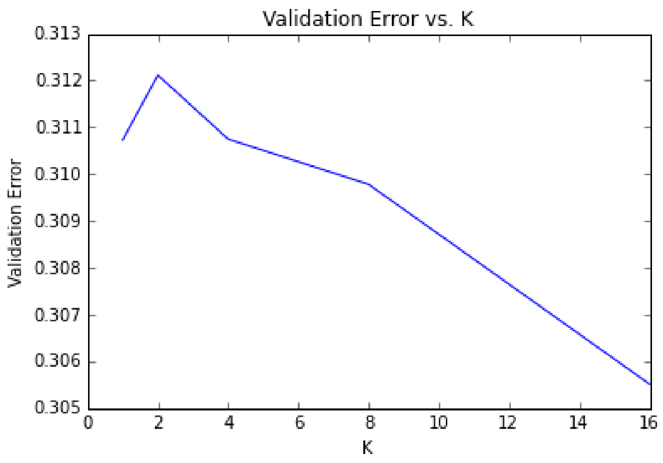

Fig. 16. Validation Error vs. K

while the latent factor model performs better than the naive baseline, increasing the number of latent factors do not significantly affect the performance.

With the tuned values of $\lambda = 0.1$ and $K = 16$, we evaluate the performance of the latent factor model on the test set. However, we notice that the results display a large amount of variance, ranging from 0.3 to 1.6, when training and testing on different splits of the dataset. This is due to the fact that this dataset lends itself to the ”cold-start” problem [7], where there exist many users in the test set with only that review, and about whom nothing else is known. Further, for users with a low number of reviews (e.g., less than K) in the training set, the γ_u fits the data perfectly and does not generalize well to new data.

Therefore, as compared to this method of modeling user preferences through an optimization problem as above, we achieved better results using the neighborhood approach of collaborative filtering described above.

VII. CONCLUSIONS

We consider “Amazon: Video games” dataset and initially perform exploratory analysis on the dataset with focus on review helpfulness and review score. We pick a predictive task - predict the review score using collaborative filtering models (that does not use any features).
We construct a naive prediction model to be used as a baseline and show that weighted bipartite projection model (neighborhood approach) can be used to improve the performance.

We change the similarity metric used in this model, experiment with various metrics which we feel are intuitively more appropriate. We also show that using these metrics also leads to improvement in the results.

Later we also perform Latent Factor Modeling for the chosen predictive task and show that it can provide results but it requires more data and also involves tuning a couple of hyperparameters (k and λ).

VIII. FUTURE WORK

In addition to the work that we have presented in this report, there are various additions and different approaches that can be pursued. We briefly discuss some of them in this section.

Weighted bipartite projection algorithm presented above can be improved in couple of different ways. Given that the data is very sparse, we can first cluster the nodes (users and the products) and dissolve all the nodes in a cluster into one single cluster and perform bipartite projection on this reduced graph. The predicted for the cluster of the node (user / product) is assigned as the score for that node. One more extension of the model is to consider multiple hops instead of a single hop that we used for the 'projection' step. The recommendation power of user u is distributed across other users via the products reviewed by u. Instead of stopping here, every user v can distribute the power of u received by him to the products reviewed by v and so on. All these extensions are described in [2]

In the Latent factor model that we have presented here, we can also train β_u and β_p (along with γ_u and γ_p) by making only slight modification to the optimizing algorithm. Basically, we have to add an intercept term to the feature matrices of each of the ALS steps.

Finally, we can evaluate our models on a different dataset that is not very sparse for a fairer comparison of the models.

REFERENCES

[1] Shang M., Fu Y., Chan D., *Personal Recommendation Using Weighted Bipartite Graph Projection*, 2008.

[2] Sumedh Sawant, *Collaborative Filtering using Weighted Bipartite Graph Projection - A Recommender System for Yelp*, 2013.

[3] Wikipedia, *Collaborative Filtering*.

[4] *Stanford Network Analysis Project*, http://snap.stanford.edu/data/web-Amazon-links.html.

[5] Ricci, F., Rokach, L., Shapira, B., Kantor, *Recommender Systems Handbook*, 2011.

[6] , Y. Hu, Y. Koren, C. volinsky, *Collaborative Filtering for Implicit Feedback Datasets*, IEEE International Conference on Data Mining (ICDM) 2008.

[7] A. Schein, A. Popescul, L. Ungar, D. Pennock, *Methods and Metrics for Cold-Start Recommendations*, Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2002).