

# PREDICTING PRODUCT RATINGS FROM WEB BROWSING DATA

MATH-122A Prof. Tyler Maunu  
Final Project Report

## GROUP MEMBERS:

Zachary Boroda

Julia Dai

Zuhaeer Islam

David Mehlhop

Ruth Rosenblum

## Abstract

The application of numerical methods to big data has revolutionized the ability to accurately predict consumer behavior. In the age of targeted advertising, this ability to draw trends from massive data sets and utilize their predictive value has become increasingly important in companies' quest to reach their desired markets. In this paper, we explore 6 such methods of data analysis to solve a simple problem: given a matrix of 4500 users' web data on 100 sites (450000 data points) and a sparse (incomplete) matrix of 3000 of those users' ratings for 75 products (approximately 30000 data points), complete the full prediction matrix of all 4500 users ratings for the 75 products. We begin by describing the initial preprocessing of the provided data including normalization, the testing/training split of our data, and describing the cost function that was used to evaluate our predictions. We decided to build our models using a methodology derived from the standard matrix completion problem. We explain how we used 3 different approaches for processing the user web data (dimensionality reduction using PCA, dimensionality reduction using PCA on top of 3 clusters derived from k-means clustering, and using the unmodified web data) and 2 different approaches for the matrix completion problem with descent by stochastic gradient descent (using the webdata as the initial features for the matrix completion problem and performing matrix completion with randomized initial features on the matrix consisting of the processed user web data concatenated to the ratings matrix). Having tested each of these 6 models, we found that, with an average squared test error of 0.80983304, using the top 15 singular values from a PCA of the history data as row features for a matrix completion problem is the best model for completing a product ratings matrix. Finally we present `predictions.csv` as our final ratings predictions for the 4500 users and the 75 products.

# Table of Contents

<b>Abstract</b>	<b>1</b>
<b>I. Data Division</b>	<b>3</b>
<b>II. Decisions on Methods</b>	<b>3</b>
<b>III. PCA</b>	<b>4</b>
<b>IV. K-means and PCA</b>	<b>5</b>
<b>V. Matrix Completion</b>	<b>5</b>
Using <code>user_history</code> as a feature matrix	6
Concatenating <code>user_history</code> to the training set	6
<b>VI. Testing on Different Approaches</b>	<b>7</b>
Result Matrix	7
Analysis of the Results	8
<b>VII. Conclusion</b>	<b>9</b>
Generating the Final Outcome	9
Final Thoughts	9
Acknowledgements	10

# I. Data Division

The data that was provided:

- `user_history`: A complete matrix which lists the historical engagement of 4500 users with a collection of 100 different websites.
  - `USER ID`: the numerical ID for each user.
  - `website 1 - website 100`: the historical engagement of a user with a website.
- `user_ratings`: ~33,000 ratings of 3,000 users on 75 products, where each row represents a user's rating on the corresponding object. The data could be converted into a sparse rating matrix with the rows being the users and the columns being the products.
  - `USER ID`: the numerical ID for each user, which corresponds to the IDs in the user history matrix.
  - `PRODUCT`: the name of the product that the user gave a rating for.
  - `RATING`: the rating that the user gave to a product.

In order to test the performance of our model, we divided the ratings data into a training set and a testing set, making sure that the testing data was not used in the training. Since the matrix was already sparse, we needed to maintain enough data with which to train, so we used 90% of the data for training and 10% for testing. We split the melted `user_ratings` matrix rather than the pivoted one so that it would be likely that each user and each product appears at least once in both the train and the test subsets. We generated the following files that store the training/testing split:

- `train_rating.csv`: 90% of `user_rating`
- `test_rating.csv`: the remaining 10% of `user_rating`

## II. Decisions on Methods

Next, we needed to consider how our model should be trained to accomplish the overall objective of completing the ratings matrix by predicting a rating for each user-product pair. However, unlike in Homework 6, where the only available data was the sparse matrix itself, here we were provided with the additional information of each user's web browsing data. In order to utilize this information in the most effective way, we investigated two possible methods of modifying the user history data: reducing its dimensionality by selecting the top principal components and clustering the data into categories and then reducing the dimensionality of each cluster separately. A direct utilization of the full matrix was applied as a comparison as well.

Additionally, we compared different methods of using the web browsing history in our matrix completion algorithm. A detailed description of the actual data adjustment and matrix completion is presented in Sections III, IV, and V.

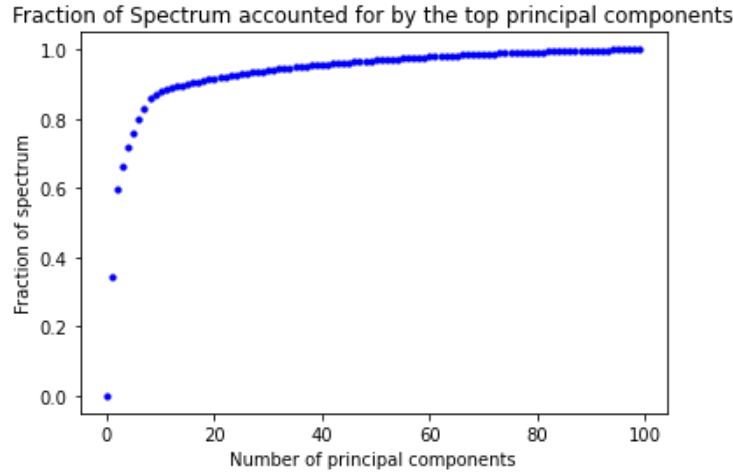
In order to evaluate the performance of each method, we kept track of the test loss and train loss, defined as the mean squared error of the predicted ratings:

$$\mathcal{L} = \sum_{(i,j) \in \Omega} (r_{ij} - (\sigma(p_i^T q_j) + \mu))^2$$

where  $r_{ij}$  is the rating of the  $i$ -th user of the  $j$ -th product,  $p_i$  is the feature vector for the  $i$ -th user,  $q_j$  is the feature vector of the  $j$ -th product,  $\Omega$  is the set of user-product index pairs,  $\sigma$  is the standard deviation of the train data before normalization and  $\mu$  is the mean of the train data before normalization.

### III. PCA

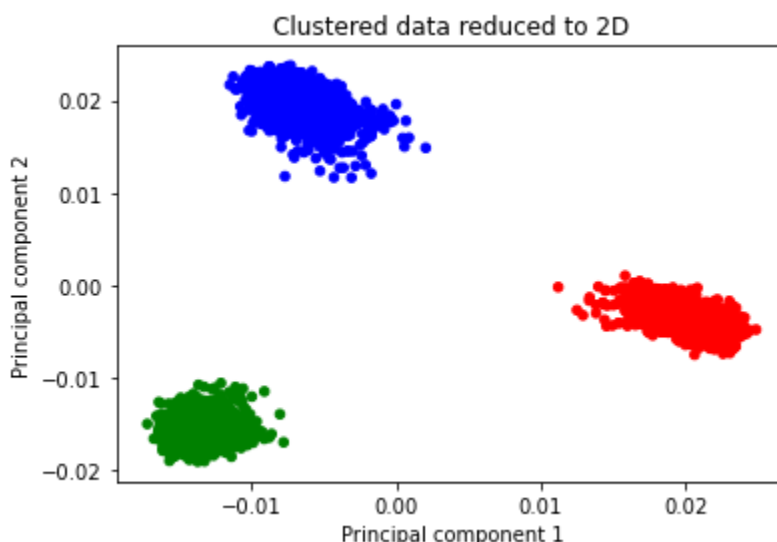
We decided to use PCA to reduce the dimensionality of the `user_history` data to the directions of its maximum variance. We hypothesized that these directions would then prove to have the most prediction value for the `user_ratings`. We performed PCA by taking the SVD of the normalized `user_history` matrix and then took the top 15 singular values, which was the minimum required to preserve at least 90% of the variance (as shown in the figure below), to be our features for the matrix completion model.



### IV. K-means and PCA

As an alternative approach, we decided to use K-means clustering and then PCA on the individual clusters to not only reduce the dimensionality of the `user_history` data to the directions of maximum variance, but also capture the fact that there is likely to be a correlation

between groups of users who have similar web browsing history and those who would rate products similarly. As in the pure PCA approach described in Section III, we hypothesized that these directions would then prove to have the most prediction value for the `user_ratings`. We used the K-means algorithm to cluster the normalized `user_history` matrix into three clusters, as shown in the figure below. Then, for each cluster independently, we performed a PCA using the SVD and took the singular vectors whose singular values made up 90% of the spectrum (which were computed to be 50 for each cluster) as the features for the matrix completion algorithm. The main downside to this method is that the matrix completion should then be performed separately on each cluster, since the features represent different singular directions for the different clusters.



## V. Matrix Completion

Considering the amount of data we have, using full gradient descent and matrix multiplication on the data would be too computationally expensive. Thus, we decided to use **stochastic gradient descent** (SGD) in this completion process to reduce the runtime and memory usage. We found that it was much easier to do matrix completion using `numpy` rather than `pandas`, so we converted the `pandas` dataframes into `numpy` arrays. We then built a Python dictionary to keep track of the indices so we would still be able to easily access a particular entry of the data.

In order to make the training process easier, we normalized the training set and recorded the mean and standard deviation so that we could scale and bound the predicted ratings data back to the original range of 0 to 10. We also normalized the `user_history` data so that it would be more consistent with the normalized ratings data, thereby adjusting to the scale and range of the initial state.

In the matrix completion process, we trained two feature matrices,  $P$  and  $Q$ , such that the prediction matrix will be  $R = P^T Q$ . Since we were given the `user_history` data, we compared the two approaches described below.

## Using `user_history` as a feature matrix

Instead of randomly initializing  $P$  and  $Q$ , we could substitute `user_history` directly for  $P$ , where rows correspond to users and columns correspond to the features. In this case, we have to keep track of the number of columns in the `user_history` to make sure  $Q$  has the same number of rows. Because for each of the methods above (full matrix, PCA, K-means), we are going to use different versions of the `user_history` which will have different dimensions, the alignment of `user_history` and  $Q$  is crucial.

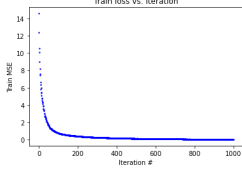
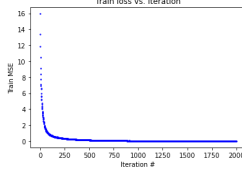
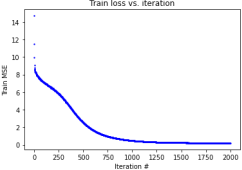
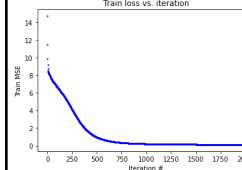
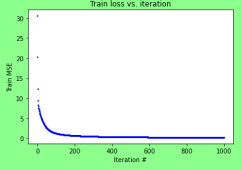
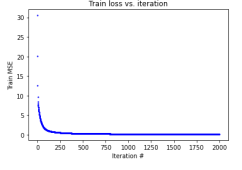
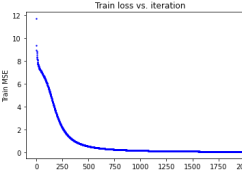
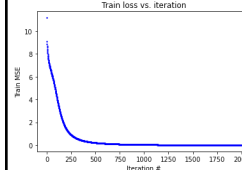
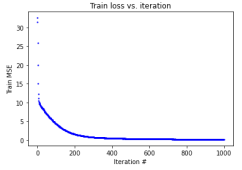
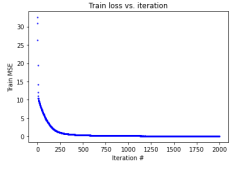
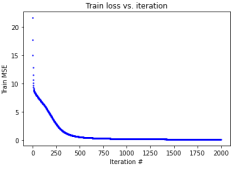
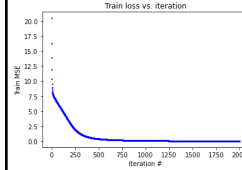
## Concatenating `user_history` to the training set

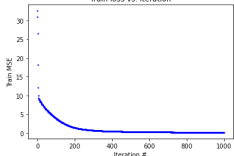
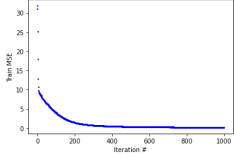
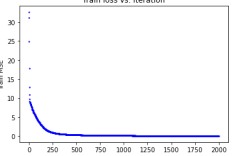
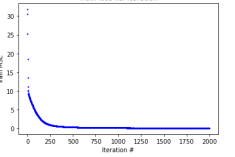
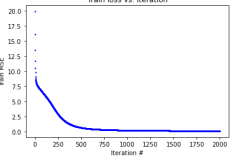
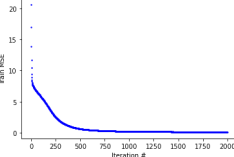
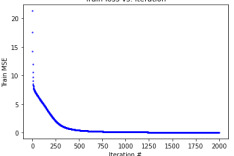
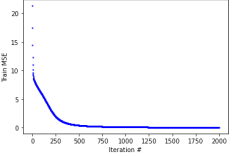
Because the initial rating matrix is very sparse, which causes our SGD algorithm to have fewer entries to sample, we considered appending the web history data to the rating matrix. Since we are assuming that both the web browsing history data and the product ratings are features of users, it seems reasonable to concatenate them as long as they are normalized. We then used the `melt` function to convert the web data into the same format as the ratings data (a three-column dataframe with columns corresponding to `USER_ID`, `PRODUCT`, `RATING`). We then performed matrix completion on the newly constructed matrix. In this way, the computed predictions matrix had a size of  $4500 \times 175$ , with the first 75 columns corresponding to products and the last 100 columns to the web browsing data. Because there are more samples to choose from, we decided to use a larger batch size in our SGD in order to make our prediction more precise.

# VI. Testing on Different Approaches

## Result Matrix

The table below shows the mean squared error (MSE) on the train and test ratings data for each approach used. The trials summarized below were all performed with a learning rate of 0.005.

	Using History as Features		Concatenating History Matrix	
All Features	<p>epochs = 1000 batch size = 1000 train MSE: 0.02561487 test MSE: <b>0.84050212</b></p> 	<p>epochs = 2000 batch size = 1000 train MSE: 0.00133722 test MSE: <b>0.88915143</b></p> 	<p>epochs = 2,000 batch size = 10,000 k = 15 train MSE: 0.20013220 test MSE: <b>1.03019212</b></p> 	<p>epochs = 2,000 batch size = 10,000 k = 30 train MSE: 0.12017014 test MSE: <b>1.45091582</b></p> 
PCA	<p>epochs = 1000 batch size = 1000 train MSE: 0.21827406 test MSE: <b>0.81144860</b></p> 	<p>epochs = 2000 batch size = 1000 train MSE: 0.17752113 test MSE: <b>0.84486256</b></p> 	<p>epochs = 2,000 batch size = 10,000 k = 15 train MSE: 0.07380016 test MSE: <b>2.99543863</b></p> 	<p>epochs = 2,000 batch size = 10,000 k = 30 train MSE: 0.00537404 test MSE: <b>6.07286182</b></p> 
K-means and PCA	<p>cluster 0: epochs = 1000 batch size = 300 train MSE: 0.15309807 test MSE: 1.48244558</p>  <p>cluster 1: epochs = 1000 batch size = 300 train MSE: 0.14324800 test MSE: 1.40712201</p>	<p>cluster 0: epochs = 2000 batch size = 300 train MSE: 0.06125466 test MSE: 1.43204846</p>  <p>cluster 1: epochs = 2000 batch size = 300 train MSE: 0.06067332 test MSE: 1.40612665</p>	<p><u>k = 15</u> cluster 0: epochs = 2,000 batch size = 3,000 train MSE: 0.12383935 test MSE: 1.76168608</p>  <p>cluster 1: epochs = 2,000 batch size = 3,000 train MSE: 0.12334286 test MSE: 1.59697402</p>	<p><u>k = 30</u> cluster 0: epochs = 2,000 batch size = 3,000 train MSE: 0.03027050 test MSE: 3.39924704</p>  <p>cluster 1: epochs = 2,000 batch size = 3,000 train MSE: 0.02843997 test MSE: 3.90478099</p>

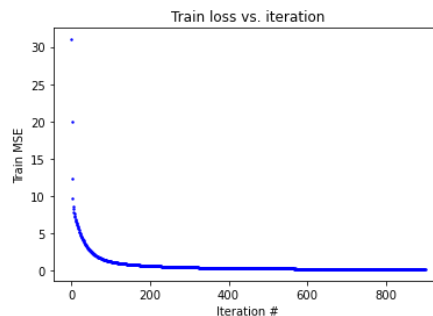
	 <p>cluster 2: epochs = 1000 batch size = 300 train MSE: 0.13745048 test MSE: 1.33676862</p>  <p>average train MSE: 0.14459885 average test MSE: <b>1.40877874</b></p>	 <p>cluster 2: epochs = 2000 batch size = 300 train MSE: 0.06019729 test MSE: 1.27877305</p>  <p>average train MSE: 0.0607084233 average test MSE: <b>1.37231605</b></p>	 <p>cluster 2: epochs = 2,000 batch size = 3,000 train MSE: 0.11621658 test MSE: 1.76880303</p>  <p>average train MSE: 0.12113293 average test MSE: <b>1.70915437</b></p>	 <p>cluster 2: epochs = 2,000 batch size = 3,000 train MSE: 0.02725822 test MSE: 3.79338081</p>  <p>average train MSE: 0.02865623 average test MSE: <b>3.69913628</b></p>
--	--	--	---	---

## Analysis of the Results

By performing a grid search on the results matrix above, we could conclude that the best model for predicting the ratings is using the top 15 singular values from a PCA of the history data as a feature matrix, with 1000 epochs and a batch size of 1000.

In order to further optimize the model, we tried learning rates, numbers of epochs, and batch sizes. While modifying the learning rate and batch size did not improve accuracy, we found that training the model for 900 epochs resulted in a slight performance improvement (test MSE of 0.80983304), as shown below.

learning rate: 0.005  
epochs = 900  
batch size = 1000  
train MSE: 0.22862033  
test MSE: **0.80983304**





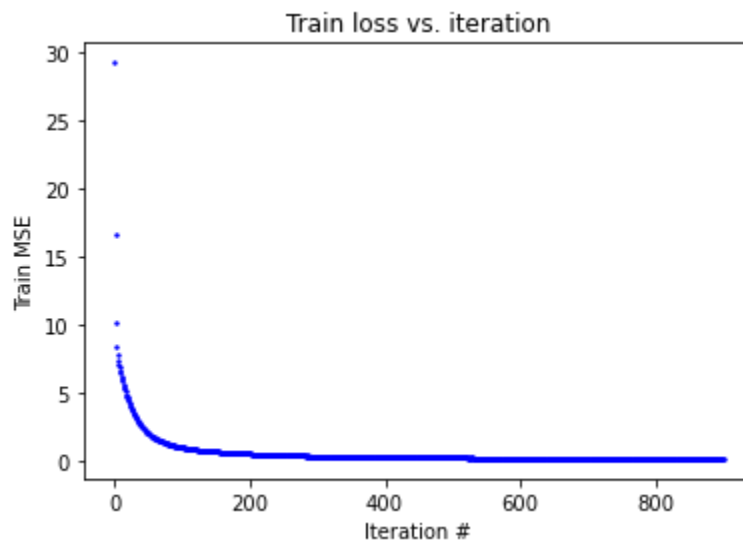
## VII. Conclusion

### Generating the Final Outcome

In the model choosing stage, we only used 90% of the data. However, for the final predictions, we used all of the ratings data to train the model in order to maximize our accuracy.

After training our chosen model on the full `user_rating` matrix, the predictions we generated were based on all of the rating data that we had. Then, we appended the `USER ID` to the `pandas` dataframe `predictions`, and melted `predictions` to produce the final three-column dataframe, which we stored in `predictions.csv`.

Even though we did not have a testing set to test our final outcome, the final train loss was 0.23185156, and the MSE over our iterations is shown in the graph below.



### Final Thoughts

Throughout working on this project, there were a few assumptions that are key to data analysis in general that we were forced to make. The first and most crucial assumption is that the ratings data provided is representative of the user ratings as a whole – more concretely, that the mean and standard deviation of the missing predictions are the same as those of the subset of the data on which we trained our model.

Furthermore, in order to make use of the web browsing history data, we had to assume that there is some correlation between a user's web browsing behavior and the ratings that they would give to the products in question.

Despite these assumptions, we were able to make predictions with remarkable accuracy, achieving the testing error of less than 1. This suggests that our predictions on average fall within one rating point of the actual prediction.

## Acknowledgements

We are thankful to our wonderful Professor Tyler Maunu for providing the guidance and teaching us how to approach tasks such as the one we were assigned for this paper. We are all blown away by how much we learned this semester.

We would also like to thank Professor Ruth Charney for bringing most of our group together under the pretext of a combinatorics group.

Last but certainly not least, we must thank Julia's chinchilla, Shiliu, for providing us with inspiration.