The Steam Engine: A Recommendation System for Steam Users

Barry Plunkett, Brandon Lin, Stephanie Shi, Chris Painter

eplu@sas.upenn.edu, {branlin,stephshi,cpainter}@seas.upenn.edu

Department of Computer and Information Science, University of Pennsylvaniau

Abstract

Steam is a video game distribution platform. We employ neighborhood, matrix factorization, and mixed collaborative filtering (CF) methods to predict the number of hours Steam users will play games. We also adapt a regression boosting framework for matrix factorization CF algorithms and apply it to the prediction task. We find that neighborhood methods outperform matrix factorization methods, and a mixed approach outperforms both. Additionally, we find the boosting framework did not meaningful improve performance. To improve predictions, future research should incorporate user friendship networks.

1 Introduction

With over 67 million active monthly users and a library of more than 700 million games [10], Steam is the world's largest video game distribution platform. However, developers cannot expect users to happily troll this virtually limitless trove of content in search of new games to purchase and play. Users would grow bored, disengage from the platform, and make fewer purchases. To tackle this problem, we seek to predict the number of hours a user will play a new game [8].

2 Related Work

Researchers at Brigham Young University conducted comprehensive exploratory data analysis of the Steam Gaming Network [8], but no public effort has been made to apply collaborative filtering techniques to Steam users and games. Research on CF techniques was popularized by the Netflix Challenge, in which researchers competed to predict user ratings for movies using a data set of movie ratings. The challenge also stimulated research into more sophisticated mixed methods [3], as mixed algorithms, which feature elements of both factorization and neighborhood based approaches, outperformed the simpler models [4].

3 Data Set

We use the publicly available Steam data set compiled by the iLab at Brigham Young University [7]. The data, collected using the Steam Web API, gives the number of hours Steam users have played the games in their library and includes more than 130 million observations over 109 million users and 316 million games (99.6% sparse). We constructed and worked with a sub-sample of this data consisting of hour counts associated with 516,000 unique users.

4 Problem Formulation

For our collaborative filtering algorithms, we use the number of hours a user has played a game as a proxy for how much a user likes a game. We removed users who have never played games, games that have never been played, and observations of users playing games for more than 80000 hours.[1] This yields a data set containing playtimes for about 5.5 million user-game pairs. Given the high variability in the number of hours a user could have played a game, we used two approaches to shrink the range of the "ratings" variable in question:

- **Binning:** A list of all possible numbers of hours played from the data set was generated, along with the 20th, 40th, 60th, and 80th percentiles. These percentiles determined 5 disjoint intervals, and each game-rating pair was assigned a rating (from 1 to 5) corresponding to their respective interval.

- **Normalizing:** For each user, hours played was replace by its z-score within the user:

$$r_{iu} \leftarrow \frac{r_{iu} - \frac{1}{m}\sum_{i=1}^{m} r_{iu}}{\sigma/\sqrt{\#\text{ of games played by } i}}$$

Substituting z-scores for raw playtime helps account for inherent user biases [5], i.e. certain users may play few or many hours of games relative to the rest of the population. This substitution also helps control for the distortionary effects of inherent user playtime variance, i.e. some users may express preference for games with fewer hours played varying between the games they played the most and those they played the least. Thus, throughout our experiment, we used the normalizing technique and report results in terms of user z-scores. The z-scores were truncated outside the interval $[-3, 3]$, and 3 was added to each score, so data points are in the range $[0, 6]$.

Formally, we define our problem as two inputs: a subset of user-game tuples $\Omega \subsetneq \{(i, j) \mid i \in [m], j \in [n]\}$ and observed implicit ratings $r_{ij}, \forall (i, j) \in \Omega$ (formulated above) for each user-game pair. As output, we predict the implicit ratings (hours) $\widehat{r}_{ij}$ for user-game pairs.

All algorithms were implemented in Python. Due to the size of the data, all had to be implemented by hand except SVD, for which we used the Surprise package.

5 Algorithms

5.1 Baseline

For our baseline, we implemented a model that assumes a Gaussian prior on the data. To make a prediction for the rating a particular user u gives to item i, the model makes a draw from $\mathcal{N}(\mu_u, \sigma_u)$, where μ_u is the mean of the ratings associated with all items u has rated in the training set and σ_u is the corresponding standard deviation.

[1] Steam was released in late 2003, so playtime may not exceed 15 years. Assuming users spend 8 hours a day away from their games yields a maximum playtime of about 8000 hours.

5.2 Neighborhood Algorithms

We implemented a *user-based* neighborhood algorithm, which predicts for a particular user u and item i:

$$\widehat{r}_{ui} = \mu_u + \frac{\sum_{v \in P_u(i)} \text{Sim}(u,v) \cdot (r_{vi} - \mu_v)}{\sum_{v \in P_u(i)} |\text{Sim}(u,v)|}$$

where $P_u(i)$ represents the "closest" users to u that have played game i. The model is based on the idea that user u will likely rate item i like users who are similar to u have rated i. We implemented two common similarity measures: Pearson Correlation and Raw Cosine Similarity [2].

5.3 Latent Factor Models

Latent factor models assume users and items can be represented in the same low-dimensional space of latent factors. Intuitively, these latent factors represent attributes of the users and games, such as first-person and preference for first-person. SVD is among the highest performing algorithms in this class. For a given user u and item i, SVD predicts:

$$\widehat{r}_{ui} = q_u^\top p_i + b_i + b_u + \mu$$

In this formulation, the $q_u \in \mathbb{R}^k$ is a representation of user u in latent-factor space and $p_i \in \mathbb{R}^k$ is a representation of item i in latent-factor space. b_u and b_i represent bias terms, and μ gives the global mean. We use stochastic gradient descent to learn parameters which minimize squared loss under this function and update all parameters after every prediction. This optimization was implemented in the Surprise package.

We attempt to enhance this SVD algorithm using an adaptation of a regression boosting technique, AdaBoost.R [9]. Although we believed SVD factorizations could be conceived of as weak learners that minimize squared loss like most regression techniques, we could not find any published paper that evaluated the performance of SVD in an Adaboost framework. As a result, we evaluate the performance of SVD as a learner in AdaBoost.R and submit the results as a novel contribution. On each iteration of the boosting algorithm, we initialize weights on the training data, associate these weights with a probability distribution over the data ($p_{(i,j)} = w_{(i,j)} / \sum_{(i,j) \in R} w_{ij}$), draw a sub-sample of the data according to this distribution, use this sample to learn representations of users and games as factor vectors and associated biases with SGD, calculate probability-weighted squared loss for each example in the full set of data, calculate a measure of confidence for the factorization based on total loss, update the weight of each example based on its loss, such that the weights examples with the highest loss increase and those with lower loss decrease, relatively.[2] To classify a new point, the boosting algorithm predicts the confidence weighted median of the predictions of the weak learners.

5.4 Mixed Methods

Finally, we implement neighborhood factorization, an algorithm which combines factorization and neighborhood approaches. The intuition for this approach is that neighborhood approaches can leverage localized patterns encoded in ratings particular users have given to the close neighbors of

[2] Details in citation.

an item, and factorization approaches capture the global structure of the data. As a result, the two should complement each other well when integrated carefully. To integrate both approaches, for a particular user u and item i, we predict:

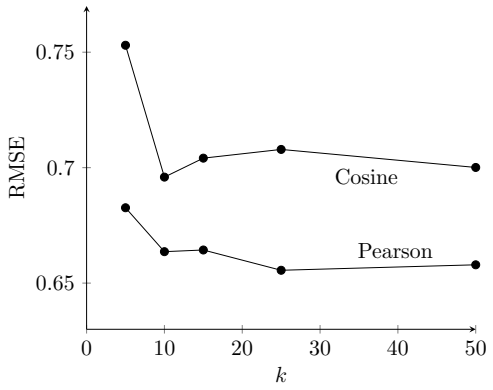
$$\widehat{r}_{ui} = b_u + b_i + \mu + |R^k(i,u)|^{-1/2} \sum_{j \in R^k(i,u)} (r_{uj} - b_{uj}) w_{ij} + q_u^\top \cdot p_i$$

Here, we define we define $R^k(i,u)$ as the subset of the k games most similar to i, which user u owns. $b_u, q_u, b_i, q_i, w_{ij}$ are parameters that we update using SGD for the squared loss under the prediction rule. Note that b_{uj} is constant, and it is given by $b_u + b_j + \mu$ for the initial values of b_u and b_j. We employ gradient descent to update parameters.

6 Experimental Design and Results

We split our subset of data into train and test sets by randomly assigning 10% of user-item pairs to the test set (T) and leaving the remaining examples in the training set (R). Performance was evaluated by computing root mean squared error on the test set: $RMSE = \sqrt{\sum_{(i,j) \in T} (r_{ij} - \widehat{r}_{ij})^2}$, as in the Netflix Challenge [1]. For all SGD usages, we use $\eta = .005$ as the learning rate.

6.1 Memory-Based



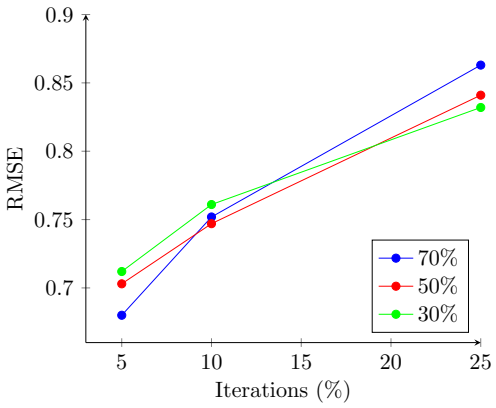
k	**Pearson RMSE**	**Cosine RMSE**
5	0.6826	0.7530
10	0.6636	0.6959
15	0.6643	0.7041
25	0.6556	0.7079
50	0.6579	0.7000

We learned models for $k = 5, 10, 15, 25, 50$. Increasing k on average led to a lower RMSE. We can see that the Pearson correlation lead to a lower RMSE as compared to raw cosine, which is expected since the raw cosine does not account for bias adjustment effect of mean-centering.

6.2 Latent Factor Models

For SVD, we have 4 regularization parameters: λ_{b_i}, λ_{b_u}, λ_{q_u}, λ_{p_i}. For simplicity, we set these equal to each other, and treat the regularization parameter as a single hyper-parameter. We also have a hyper-parameter for the number of latent factors (k). Using the surprise package, 5-fold cross validation on the training set for all combinations of $\lambda = [.001, .01, .02, .05, .1]$ and $k = 25, 50, 100, 200$ with 20 epochs of SGD.[3] We find $k = 100$ and $\lambda = .02$ yields the lowest mean

[3]Parameters tended to stabilize at this point.



Model	Best RMSE
Baseline	1.0724
SVD	0.7056
SVD w/ Boosting (5 weak learners)	0.6800
Neighborhood Factorization	0.6200

cross-validation error. The matrix factorization model learned with these parameters yields .7056 RMSE after 20 epochs.

For the boosted SVD, we do not cross-validate over k or λ due to the high cost of training. Instead, we use the optimal number of parameters k and regularizer λ from the simple SVD for boosted SVD. Given these hyper-parameters, we compare the performance of boosted SVD models with three sub sample sizes: 30%, 50%, and 70% and several different committee sizes: $5, 10, 25$. The results of these experiments are summarized in the plot below. In general, the subsample size did not introduce a great deal of variance into performance, increasing iterations increased training error, and the best boosted model ($i = 5$ and $s = 70\%$) barely outperformed standard SVD.

For the neighborhood factorization algorithm, we initialize user and game vectors as the output of SVD trained with $\lambda = .02$ and $k = 100$. For computational efficiency, we choose $n = 10$ for the number of neighbors, since there is low return to increasing neighbors [6]. We calculate the similarity of games using Pearson correlation multiplied by a shrinkage factor: $sim(i, j) = \frac{n_{ij}}{n_{ij}+\beta} \cdot p_{ij}$, where n_{ij} gives the number of users who own both games i and j. We use $\beta = 100$, a common choice [6]. This shrinkage reflects the idea that we are more confident about similarities when more users share the games. We initialize the game-game weights to 1. We initialize b_u and b_i by minimizing $\sum_R (r_{ui} - \mu - b_u - b_i)^2$ via SGD. The model has three regularization parameters: λ_b for bias terms, λ_{qp} for factor vectors, and λ_w for the game-game weights [6]. The model is extremely expensive to train because we can't store the sets of similar games in memory ($R(i; u)$), so we don't attempt to optimize hyper-parameters. Instead, we borrow parameter values from Koren: $\lambda_b = \lambda_{pq} = .005$ and $\lambda_w = .001$, ,and we subsample 50000 samples for each epoch [6]. Before training, the model has RMSE of .72 on a subsample of 50000 instance in the test set. After 30 epochs, RMSE falls to approximately .62.

7 Conclusion and Discussion

Our results have shown that we are able to predict Steam user game play times to within .62 standard deviations of a user's playtime on average. By showing each user u only games for which our best model predicts high ratings, Steam can bolster customer satisfaction and engagement.

As expected, the mixed method outperforms neighborhood and factorization approaches. Interest-

ingly, neighborhood methods slightly outperformed factorization methods. The reasons for this are unclear, but the use of hours played as a heuristic for player preference, as opposed to explicit user ratings of games, could contribute. For example, users may play games that aren't well aligned with their preferences because their friends are playing them or because they are popular. Since we use hours as a heuristic for enjoyment, we can't distinguish well between instances of popular games users actually enjoy and those they don't.

Boosting is able to improve the SVD error slightly by a margin of 0.02; however, as the number of weak learners increased, we observed that the RMSE increased. This could indicate overfitting of the training data with a higher number of models. Therefore, we conclude that boosting SVD in the AdaBoost.R framework is not effective in this case for the subsample sizes we experimented on. Future research should consider alternative boosting frameworks and careful exploration of hyperparameters, especially smaller subsample sizes.

References

[1] Frequently asked questions. https://www.netflixprize.com/faq.html.

[2] C.C. Aggarwal. *Recommender Systems: The Textbook*. Springer International Publishing, 2016.

[3] Yu He Andrey Fuerverger and Shashi Khatr. Statistical significance of the netflix challenge.

[4] Edwin Chen. Winning the netflix prize: A summary.

[5] Bracha Shapira Francesco Ricci, Lior Rokach. *Recommender Systems Handbook*. Springer, 2 edition, 2015.

[6] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. *Proceedings of the 14th ACM SIGKDD*, pages 426–434, 2008.

[7] Internet Research Lab. Steam dataset, 2016. https://steam.internet.byu.edu/.

[8] Justin Wu Mark O'Neill, Elham Vaziripour and Daniel Zappala. Condensing steam: Distilling the diversity of gamer behavior. https://steam.internet.byu.edu/.

[9] Greg Ridgeway, David Madigan, and Thomas Richardson. Boosting methodology for regression problems. May 2001.

[10] Steam. Steam: Game and player statistics. store.steampowered.com/stats/.