Steam Game Recommender
EECS 349: Machine Learning | Professor Doug Downey

Kapil Garg, Aaron Karp, Yuxuan Xiao

Abstract

Our task is to generate predictions for games a user on Steam might like based on the games they currently own. Currently, there is not a reputable, automated recommendation system for video games; users primarily buy games based on reviews of friends and review sites. We believe that this can be improved by using player data from Steam, the largest gaming platform for PC users.

Our data set includes the Steam libraries of around 10,000 users. Each instance has 2,283 binary attributes, either 0 or 1, representing a user's ownership of a specific game. We used a random 80/20 split to generate our training and test sets, on which we trained and evaluated our classifiers. As some of the games had few players, as little as 1% of the total size of the data set, we supplemented our accuracy results with precision, recall, and f-measure, which better reflects the quality of a classifier.

The classifiers we evaluated include kNN, Naive Bayes, decision trees, and Bayes Net. As a baseline, we used ZeroR. Apart from kNN, which we implemented in Python using bit strings for speed, the other classifiers were from Weka. We tried each classifier with a set of 500 games as target attributes. The first 50 are the first 50 games in the data set, which have many players due to their age. The other 450 games were picked with a pseudo-random number generator to easily reproduce the results between runes. Since games varied widely in the number of owners, the most important measure of success were the f-measures for both owning and non-owning the class. ZeroR always has an f-measure of 1 for one attribute value and 0 for the other. Better classifier had high f-measures for both possible attribute values (0 or 1).

Though the classifiers we found tended to vary in accuracy and f-measure depending on the number of owners of a game, looking at the results for the f-measure for owning the game over the 500 games showed that decision trees was the best classifier, with the highest average f-measure of 0.474 and the highest upper and lower quartiles from 0.303 to 0.638. Please refer to the Appendix for full data table and visualization of data.

1 Full Report

Companies like Netflix and Amazon have great recommender algorithms that help you decide your next TV show or potential purchase. However, video game recommendations are still, for the most part, done peer to peer or by a salesperson at a store. We believe it is feasible to create a recommendation algorithm that takes what games you already play and recommend games that you may also like.

We attempt to predict whether or a not a player would like a game based on their owned games on the largest community and platform for PC gaming, Steam. We chose Steam because of their easily accessible public API and the popularity of the platform; many PC gamers own the majority of their games through Steam.

No data set of games and users was readily available, so we had to create our own. Steam's public API allows an API key holder to get the library of any Steam user who has not set their profile to "private" or "friends-only." Without a comprehensive of Steam users, though, we resorted to scraping usernames from the Steam community user group, a general purpose Steam community group for discussion and questions about all things Steam. Though joining the group itself only takes one click and requires no further interaction, even this action may bias our data towards

users who are more involved with Steam and may own more games. We obtained over 10,000 usernames this way. After filtering out users with no games or private profiles, we were left with over 9,000 users' data.

We pulled the games owned by each of these users then only included games with a minimum number of 100 users in our data set. We used a random 80/20 split to generate our training and test sets. This allows us to quickly and effectively evaluated our classifiers. Because some games had few users after the split, accuracy would not be a good measurement of the performance of a classifier, as ZeroR was nigh unbeatable, as it always claimed no users owned the game. Instead, we used measures of precision, recall, and f-measure. For games with few users, ZeroR will always have an f-measure of 1 for choosing users that don't own the game, but will always have an f-measure of 0 when evaluating users owning the game.

The classifiers we evaluated include kNN, Naive Bayes, decision trees, and Bayes Net. We also attempted to try SVMs, linear regression, and neural nets, but the nature of the data, with the large number of attributes, made some of the classifiers infeasible to run. We encountered the issue of how to choose games to evaluate our classifiers on, as different games in the data set may have vastly differing amounts of users. Our data was sorted by the SteamID of the game, an arbitrary ID assigned by steam which directly relates to how early the game was published. A lower SteamID indicates a game was published earlier, which roughly correlates with the number of users. This correlation only holds well for a limited number of SteamIDs, after which it become much weaker. In order to guarantee we would include games with many users in our testing, we decided to run each classifier with 500 different games as target classes. Of the 500, the first 50 are the first 50 games in our attribute set, ordered by SteamID. This includes extremely popular games like Counter-Strike and Team Fortress 2. The remaining 450 games were chosen by a pseudo-random number generator initialized with the seed "ML349", so that we could produce the same 450 games each time.

Our next problem was that Weka's version of kNN was extremely slow to evaluate. The distance between two users was defined as the number of differing games in their libraries, equivalent to Hamming distance. Weka likely represents the data with an array or similar data structure, increasing the time needed to calculate the distance. In order to reduce the run time of kNN, we implemented a custom kNN classifier in Python that stored the data as bit strings and obtained the distance with a simple bitwise XOR. In addition, our implementation uses multi-threading to evaluate multiple games at once.

For the classifiers directly from Weka, it would have been prohibitively time-intensive to run three different classifiers 500 times each. Instead, we used a package called PyWeka to automate running the classifiers and saving the data in a custom format. Our code is written such that any Weka classifier and its command line attributes can be provided and run over the same 500 games.

Though the classifiers we found tended to vary in accuracy and f-measure depending on the number of owners of a game, looking at the results over the 500 games showed that decision trees was the best classifier. The accuracy and f-measure for not owning the games differed little between the classifiers. All classifiers achieved $> 90\%$ for both. Thus, we discounted these as measures of quality for our classifiers. Instead, we focused on the f-measure for owning a game. Decision trees emerged as a clear winner here, with a mean f-measure of 0.474, far higher than the closest competitor, Naive Bayes, with 0.345. In addition, the f-measure for decision trees higher values for both quartiles than any other classifier. Though we expected kNN to perform the best for a problem like this, decision trees may have won because of bundles or natural pairs/groups of games. For example, if a user has Half Life, they are very likely to also have Half Life 2. In other cases, games are often sold in bundles, so users are likely to own all of the games in a bundle. This would allow for decision trees to perform well by splitting on those related games. Please refer to the Appendix for full data table and visualization of data.

In the future, we may want to reduce the size of our attribute set, which currently stands at over 2,000 attributes, in order to be able to run more algorithms in a reasonable amount of time. We may also want to weigh games by the number of hours a user has played or by the games

that a user's friends play. Our data currently only reflects ownership of a game, which does not necessarily mean a user likes it. We may also want to weigh games based on the reviews a user has given.

2 Appendix

Algorithm	F1 (1) Mean	F1 (1) Q1	F1 (1) Median	F1 (1) Q3	F1 (0) Mean	F1 (0) Q1	F1 (0) Median	F1 (0) Q3
ZeroR	0.012	0.000	0.000	0.000	0.952	0.963	0.983	0.990
Bayes Net	0.343	0.188	0.319	0.496	0.926	0.912	0.936	0.959
Decision Tree	0.474	0.304	0.475	0.635	0.966	0.965	0.983	0.991
Naive Bayes	0.344	0.188	0.316	0.497	0.926	0.912	0.937	0.959
kNN (k = 5)	0.293	0.062	0.247	0.510	0.968	0.969	0.985	0.992

Table 1: F1 Scores for various algorithms and classification (1 = game owned, 0 = game not owned)

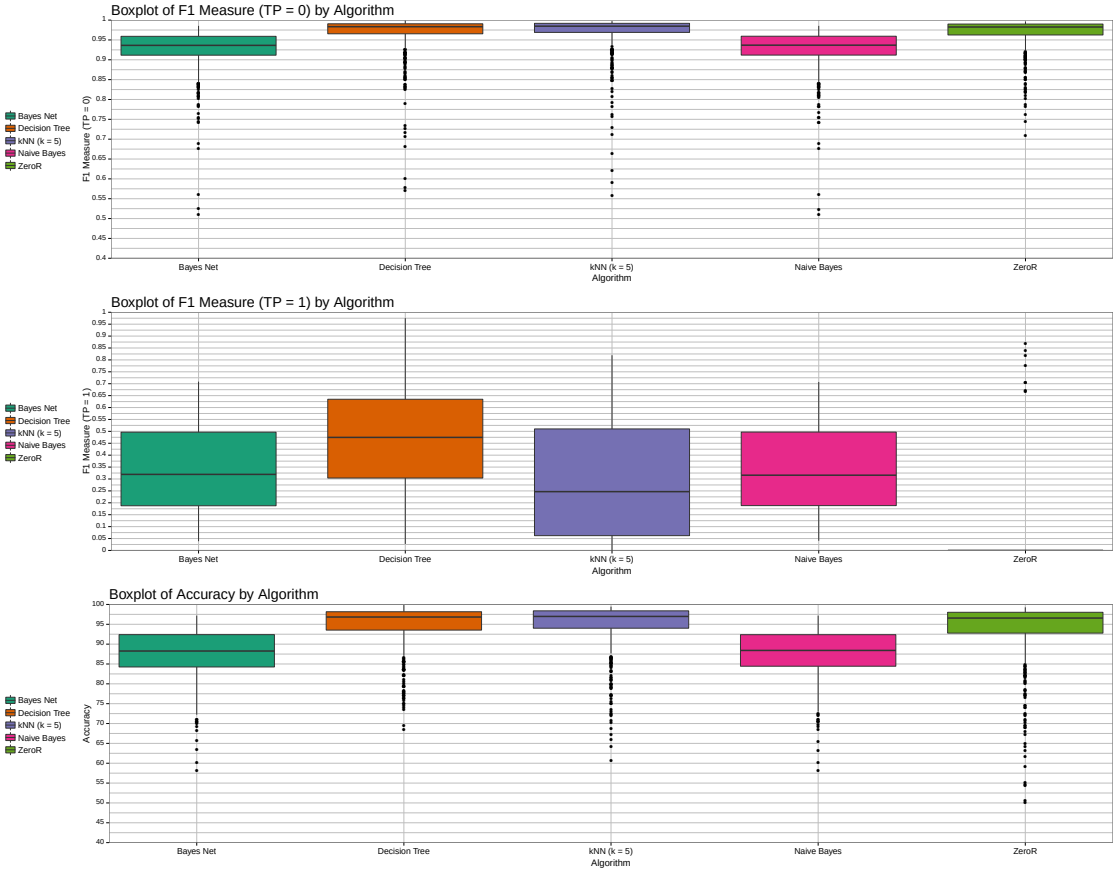

Figure 1: Table 1 Visualized as Boxplots