PC game play time estimation based on Steam data and reviews

Introduction

The Steam platform (http://store.steampowered.com/) is currently the largest PC game online distributor in the world, and has accumulated a vast amount of player and game data. One interesting metric that we want to study is how long a player will spend actively playing a new game after the purchase (i.e. expected total play time). This metric is a major factor influencing the players' buying decisions when a new game is released. It is also important to the developers who have to plan the support cycle for the game.

In this project, we have created various models that predict the average total play time of a single-player game over all of its owners on Steam, based on the information about the game that is available on Steam within a reasonable period of time after its initial launch (including user reviews). Our most accurate prediction model uses a random forest regression.

Related Work

While there is an abundancy of research on predicting users' response to a product, but we have not found any publication on predicting this particular metric (play time) for single-player PC games. However, prior research focusing on making product quality related predictions using text features extracted from user reviews [3] would still provide useful insights on how we should parse and analyze similar features in this project.

Data Acquisition and Cleaning

We have obtained the input data from two sources:

1. Steam provides a web API (https://developer.valvesoftware.com/wiki/Steam_Web_API) for developers to programmably access publicly available data regarding each individual player and each individual game, including play time and player reviews.
2. Steam Spy (https://steamspy.com/about) is a third-party site that regularly runs sampling and aggregation using Steam Web API to provide aggregated statistics on Steam games. It offers useful information like play time distribution. It also ranks and categorizes steam games according to user reviews and tags. This site also provides an API for programmable access.

Since the games available on Steam changes everyday, we took a snapshot of all applications on Steam on 11/04/2017 and all further data crawling was based on this list of games. Next, we obtained the detailed information (developer, genre, price etc.) for each of the application from Steam, and did a preliminary filtering on the application list to leave only entries that are of type “game” (vs. trailers, DLCs, or other software) and are currently available to all users (no early access). Afterwards, we programmatically crawled all user reviews for all the games in the aforementioned list. The average total play time for all Steam games was obtained from Steam Spy, which obtains this information by sampling user data because playtime information is only available on user level on Steam. Finally, we performed some data preprocessing to exclude games with null values in certain fields, fill 'publishers' with 'developers' when 'publishers' is unavailable, and exclude games published after 8/10/2017 so as to have stable playtime data. We also noticed that the sampling method employed by steam spy has very large uncertainties for games with small numbers of players, so we also filtered out games with fewer than 1000 owners.

We make one important assumption about the metric to be predicted, average total playtime for each game: we assume that average playtime stabilizes 3 months after its release on Steam, because players usually play a game over a short period of time and then move on to others. This assumption allows us to treat this metric as constant for games released 3 months before our data was collected.

Features

The features incorporated into our model can be divided into two types (summarized in Table 1). The first type is meta information about each game, such as number of owners, initial price, publishers and genres, which we obtained from Steam's official APIs. The features category, publisher and genre are categorical, to which we applied one-hot encoding with an 'others' column. Note that each game can be considered to belong to multiple categories/genres or be produced jointly by multiple publishers, more than one column for each feature can be 1 for a game. These numerical features were not normalized because linear regression and tree-based ensemble methods that we tried did not require features to be normalized.

The second type of features are from the text of user reviews. After tokenizing the sentences in all reviews written for a game, and lemmatizing and removing stop words in the text, we tallied the most popular 1-grams, 2-grams and 3-grams. We then hand-picked 62 popular such n-grams that we thought could be informative about the play time, such as 'long-time', 'waste', 'make-season-2', and 'keep-come-back'. Each of these n-grams forms a column, where the feature is the count of this n-gram's appearance in all reviews on this game, scaled by the total count of 1-, 2- and 3-grams in these reviews.

The final data matrix has 256 columns/features and 9334 rows/examples, which we divided 80:20 as training and test set. We did not need a separate validation set as we used the out-of-bag scores to tune the hyper-parameters in the tree-based ensemble regressors.

Feature (256 columns)	Processing
Category	One-hot encoded with an 'others' column
Publisher	
Genre	
Initial price	Raw numerical input (unscaled)
Number of up-votes	
Up-votes to down-votes ratio	
Number of owners	
Achievements to unlock	
Review text features	Weighted frequency

Table 1 Model features and encodings

Method

We applied linear regression to the features as a baseline model, followed by a random forest regressor, which is a tree-based ensemble method that gives a good trade-off between model flexibility and prediction variance. This is because regression trees can model complex interaction among the features, while the randomness introduced de-correlates the trees in the forest, curbing variance. We used 200 estimators (trees). Two hyper parameters were tuned: minimum node size (optimal 3) which controls the depth of the tree, and number of variables used to decide each split (optimal 40, out of 256 columns). A validation set for parameter tuning was not required because random forest uses bootstrap samples for each tree and the out-of-bag scoring was used to tune the parameters. The random forest algorithm for regression is as follows [2]:

For b = 1 to 200:

Bootstrap sample of training set size is drawn from the training data

Grow a tree to the bootstrapped data, selecting 40 variables at a time and pick the best variable to split among these, until the minimum node size 3 is reached (when further splits would result in leaf nodes with fewer than 3 examples)

Take the average of the 200 regression trees.

We developed our models using Python, using linear_model and RandomForestRegressor as well as helper functions such as train_test_split and GridSearchCV from Scikit-learn.

Gradient boosting tree [1] was explored as another ensemble method, but the metrics did not improve and it seemed to do worse on games with very long/short play time.

Experiments and Results

Cost Function Selection

As we can see from Figure 1, the distribution of playtimes is very skewed: a fraction of games has very long play time, while the majority of games have around 250 minutes of play time. Thus if we use the mean squared error or mean absolute error as the cost function during model training, it will likely be dominated by a few games with very long play time. This led us to experiment with using the natural log of play time as the output instead of raw play time. Using the logarithm of play time has its own problems: a disproportionate emphasis is given to games with very short play time (in the 10 - 50-minute range) due to the shape of the logarithmic function. We therefore tried out a combination of cost functions and applying log on output or not.

Experiments

Because of the difficulty in choosing a good cost function, we experimented with a combination of using MSE and MAE as cost function and taking the log of the output play time using the three methods described above.

Discussion of Results

The training and test error associated with our experiments and models are presented in Table 2. Instead of quoting a total or average mean squared error, we consider the errors of each model as the percentage of games for which our prediction error was more than $\pm$100%, $\pm$50% and $\pm$30% of the actual playtime. The reasons for presenting the training and test results as such are

1. that it is unfair to compare any particular metric associated with a particular cost function that we used during training (e.g. MSE, MAE) across the models when they were optimized against different cost functions
2. that the play time is very skewed so percentage error rather than absolute error is a better indicator of prediction accuracy, and
3. that there are a handful of outliers in the prediction error distribution, which distorts the overall performance indicator

We experimented with three different models and four different cost functions, as well as various combinations of features. In Table 3, we summarize a representative selection of the results, including two benchmarks (predicting using the average of the training set and the median).

Model	games with more than 100% error		games with more than 50% error		games with more than 30% error	
Data set	test	train	test	train	test	train

Predicting average	55%	47%	80%	75%	87%	85%
Predicting median	18%	19%	47%	48%	62%	64%
Linear Regression (MSE)	36%	37%	60%	60%	76%	75%
RF (MSE)	26%	22%	48%	38%	63%	55%
RF (MAE)	18%	14%	40%	29%	58%	46%
RF (MSE), log(y)	13%	9%	38%	26%	57%	44%
RF (MAE), log (y)	12%	9%	37%	27%	56%	44%
RF (MAE), log (y), text	**12%**	**9%**	**36%**	**26%**	**56%**	**44%**
GB (least square), log(y)	14%	15%	40%	40%	59%	59%

Table 2 Training and test error for different models/cost functions combinations

We note that predicting the median is quite a reasonable model, giving lower errors than linear regression and random forest using MSE. This is because the modal play time is around 217 minutes or the median and there is a large concentration of games with play time around that value. Using the logarithms of the play time as cost function during training instead of the raw play time improved performance significantly. Although our optimal model is the one the text features from reviews, the effect of including these text features is minimal given our accuracy criteria (however these features are among the more informative ones – discussed below).

Looking more closely at the performance of our optimal model, random forest using MAE as criterion with logarithm of play time and including all features detailed above, we note that the most informative features according to random forest's feature scoring are initial price, number of owners, number of up-votes, achievements to unlock, up- to down-vote ratio, genre (Indie), 1-gram ('minute'), genre (Strategy), genre (Action), genre (RPG), 1-gram ('waste'), publisher (others) and 2-gram ('10 10'). In Figure 1, we overlaid the predicted play time on top of actual play time of each game in the test set. We can see that our model does captures the skewed distribution of the play time, but underestimates at the higher end and overestimates at the lower end.

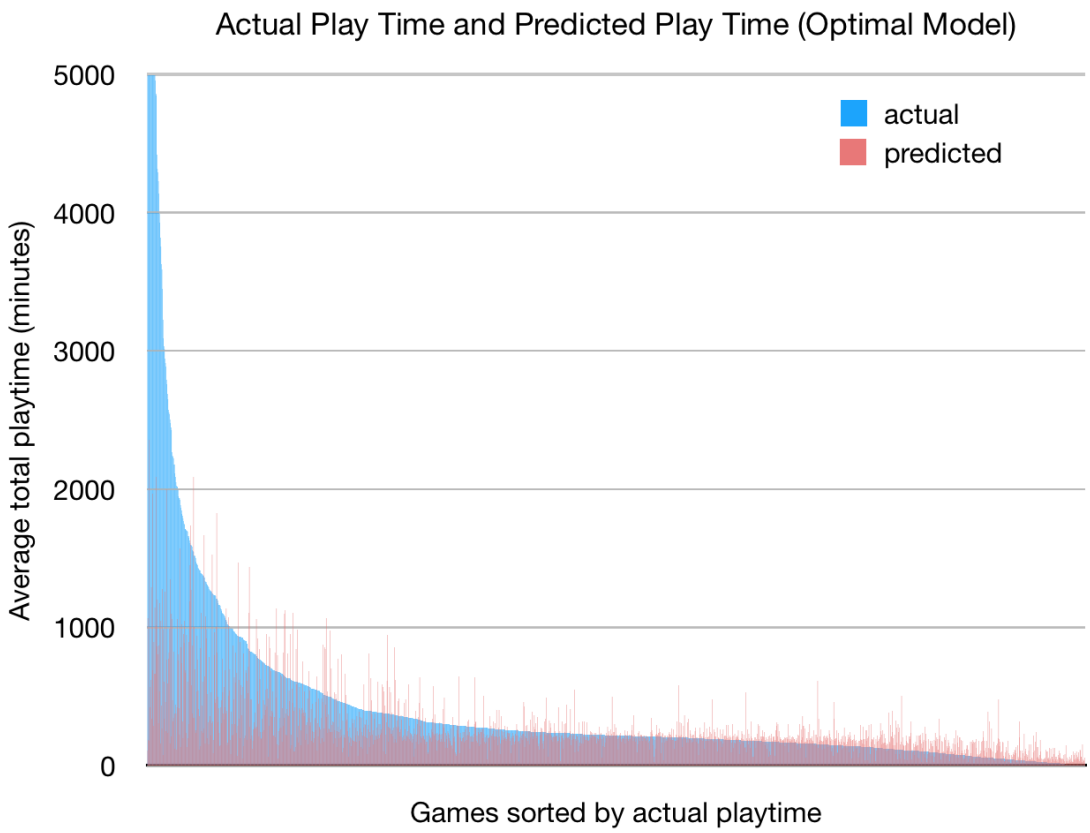

Figure 1 Actual play time in test set and corresponding predicted play time w/ optimal model

A particularly interesting and challenging set of games to predict for are those with high play time (more than 1000 minutes). While Figure 1 indicates that our model does not predict very well for these games, we can see from Table 3 that for games that are well-known, we can achieve good estimation of the playtime. This could be because the publishers of such well-known games are major studios, which have their own columns in the encoded feature vector, as opposed to indie games that are lumped into the 'others' column, providing more information to the regressor.

Game	Actual (minutes)	Predicted (minutes)	Percentage error
Age of Empires II HD	2791	2001	28%
WWE 2k16	1921	1576	18%
Watch_Dogs® 2	1632	1194	27%
Assassin's Creed® Unity	1613	1452	10%
Watch_Dogs™	1596	1273	20%
Final Fantasy IX	1390	1109	20%

Far Cry® 2: Fortune's Edition	584	543	7%

Table 3 Playtime predicted for selected well-known games

Conclusion and Future Work

While our features and models were able to capture some of the variation in play time among the games, the errors remain quite large for games with very long play time. To further the work we have done, more informative features and a more flexible model are required to improve prediction performance. We would also like to better understand how to tune the gradient boosting tree and see if it can improve on the performance of random forest.

Reference

[1] T. Chen, C. Guestrin, "XGBoost: A Scalable Tree Boosting System," presented at the 22nd ACM SIGKDD International Conference, pp. 785–794, New York, USA, 2016.

[2] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning. Springer New York Inc., New York, NY, USA, 2001.

[3] M. Yu, M. Xue, and W. Ouyang, Restaurants Review Star Prediction for Yelp Dataset. https://cseweb.ucsd.edu/~jmcauley/cse255/reports/fa15/017.pdf