# Steamy Reviews

Brian Guo (bg379), Harrison Unruh (hju3)
Nitya Kasturi (nsk43), Paul Chesnais (pmc85)

March 29th, 2017

## 1 Introduction

### 1.1 Goal

The goal of our project is to create an application to aggregate game reviews left by users on the online platform, Steam. Textual reviews will be scraped from the Steam website, along with useful information such as: whether or not the reviewer liked the game, how many hours the reviewer has played the game, and how many other users found the review helpful. Our objective is to tap into the wealth of information provided by users in game reviews and provide a quick, accessible metric for users to both find new games and decide if they would like to buy a game.

### 1.2 Information Need

The information need we are addressing with this application is the need of finding similar games based on games users enjoy. Steam already does this based on the tags that users can provide for each game, but our goal is to do this through the reviews users leave on each game. The motivation to have an application that does this through comments is because comments and user reviews are a better way to get a general idea of how users are perceiving games.

### 1.3 Inputs and Outputs

Any game (or games) listed on the Steam library is considered to be a valid input. Other titles will be rejected on the grounds that we do not have data to include them.

The application will return a list of keywords and phrases, including notable gameplay elements and common emotional sentiments, which can then act as our own personal tags for games. Game recommendations will be presented as another list, based on similar Steam tags or tags we have generated ourselves.

### 1.4 Use Cases

Our application serves a similar role to the "Recommended Games" feature in Steam; a means for users to quickly identify games that interest them based on those they have already played. A typical use case is therefore a person looking for a new game to play based on their existing Steam library. The objective of the application is to remove the need for a user to read through a game's reviews to determine if the purchase is worthwhile, instead applying review data to the game recommendation identification process. Examples of how the application works with a user are below in the Examples section.

### 1.5 Information Retrieval Component

The information retrieval component of the application is the process of scraping reviews from the Steam web platform and processing those reviews to extract relevant information. As a whole the application is an information retrieval system because there is an information need. Users want to identify games of

interest, and the system returns a list of relevant games. We will use some of the text processing concepts (de-duplication, etc.) discussed in class to process reviews. To quickly get game recommendations, we will maintain a vector space model and use cosine similarity to evaluate similar games.

### 1.6 Social Component

Review data from Steam is inherently social and is the core source of data for the application. Steam is a social platform for gamers to leave reviews about games they play, and provides data on how long each reviewer has played the game they are reviewing as well as how helpful other users found each review. We plan on (optionally) allowing users to connect their own Steam account to their application to provide recommendations based on their game library. We plan to accept feedback on games that were recommended by the system, namely to say if the recommendation was good. We will use this information to better guide future queries.

### 1.7 Machine Learning Component

The machine learning component of the application is its ability to determine similar games based on an existing set of 'good' games. These are games that are either present in the user's library (if they connect the system to their Steam account) or which they input manually. Each game will be represented by a vector with weights determined by various features. Tags for each game will come from both the existing list of tags provided by Steam and tags generated by processing reviews. Weights will be greater for tags with a higher degree of uniqueness (those which do not occur frequently in the dataset). We will generate a combined tag vector using the set of games supplied by the user, and check all games to search for the minimum distance to other vectors on the n-dimensional unit sphere. Our training labels for our algorithm will include the Steam-generated tags.

We recieved the most feedback from our TA on the machine learning component. He suggested that determining similarity between games by minimizing vector distance on an n-dimensional unit sphere would be a good approach for the project.

## 2 Data

### 2.1 Data Source

Our data is going to come both directly from Steam and from two aggregators: SteamDB and steamspy.

#### 2.1.1 Steam

There exists an endpoint on the Steam website that directly returns reviews. We originally found this endpoint in this repo, but ended up rewriting the adapter entirely. This endpoint is going to be the core endpoint, as it will provide us with all the reviews for every game, provided we know the game's ID. Steam also aggregates the number of positive and negative reviews as an overall rating where it can be one of *Overwhelmingly Positive*, *Very Positive*, *Positive*, *Mostly Positive*, *Mixed*, *Mostly Negative*, and *Negative*. The existence of a *Overwhelmingly Negative* rating has been theorized, but never observed. Since Steam provides us with this rating, we will be using them as ground truth for when we train our algorithms.

#### 2.1.2 SteamDB

SteamDB is the website that we scraped to get all the IDs of each Game on the Steam store. SteamDB also aggregates interesting metrics like recently popular games or recently updated games. All in all, it served best as the source for every possible game and its ID, so we likely will not be using it as much as Steam directly in the Future, now that we have all the IDs (except for when we want to refresh our list of IDs in the future).

#### 2.1.3 steamspy

steamspy has a nice aggregator for Tag metrics. They expose a table that contains how many games are in each tag, their average approval and more. We will be checking back to this website regularly to get the updated data, much like with SteamDB.

### 2.2 Data Format

#### 2.2.1 Steam

The Steam reviews endpoint returns a JSON containing 25 reviews in pre-rendered HTML. We have to use BeautifulSoup to parse out the relevant features in each review, and it seems that the data itself is quite regular. We stumbled upon the occasional less than zero value in the number of hours played, but otherwise there were no issues getting the data for this milestone. Getting the overall rating is more difficult, as there is no raw endpoint (as far as we could tell), so we ended up having to scrape Steam directly. We are going to have a database of reviews that is going to have important necessary columns, such as how many people found it helpful/funny, whether it recommended the game or not, and of course, the reviewID, user, and the text.

#### 2.2.2 SteamDB

SteamDB provides the total list of every game on Steam in a single table, so no issues there extracting the game IDs and titles. There is the slight concern that SteamDB has a no-scraping policy, but we are not crawling over their entire website, and will only do so very occasionally. We will also consider using steamDB to update our database of games, because the most recent games get added to the front page of steamDB.

#### 2.2.3 steamspy

Most of the tables exposed by steamspy can be downloaded as raw CSV files, so very little pre-processing is required on our end for us to be able to use the data.

### 2.3 Statistics

#### 2.3.1 Number of Hours on Record

Each review on Steam comes with how many hours the reviewer has played on the game. It would not be unreasonable to think that the more hours the reviewer has played, the likelier it is that the review is going to be helpful. For a random sample of 25,000 reviews, Figure 1 is a histogram of the number of hours played against percentage of people that found the review helpful (i.e. the number of people that found the review helpful over the number of people that rated the review). As we can see, the trend does not hold well enough for us to be able to extrapolate such a relationship across all reviews. The fact that this does not hold does not mean that we cannot still weigh reviews with more play time more heavily when we are doing our own ranking.

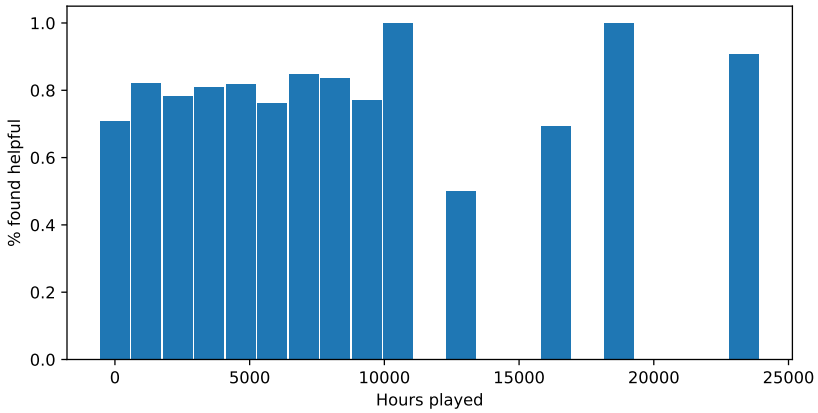


Figure 1: Hours Played Versus Helpfulness

### 2.3.2 Number of Reviews

The other statistic that is exposed by Steam for each review is how many reviews that reviewer has made. Much like above, it does not seem unthinkable that more experienced reviewers would leave more helpful reviews! Figure 2 is a histogram plotting the number of reviews by the reviewer versus how helpful they were (same helpfulness metric as above). But, unfortunately, no dice. It looks again like how experienced the average reviewer does not correlate very well with how helpful the reviews are. Interestingly though, it's possible for some Steam users to become *Curators*. Curators aggregate games based on certain qualities like overall gameplay, but also by genre or style. These users tend to leave more fleshed out reviews, and we plan on trusted reviews (such as those from PC Gamer) to compare.

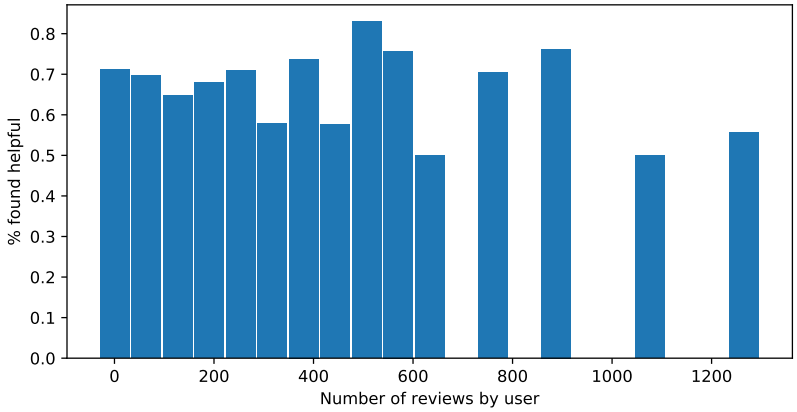


Figure 2: Number of Reviews Versus Helpfulness

## 2.4 Average Rating Per Tag

Another statistic that we have considered is the average rating given to a game by the customers. We've aggregated this value by taking the average of all the ratings of the games in a given tag and using this value to represent the rating of the given tag. Figure 3 is a histogram that plots the number of games that have a certain rating for a given rating number. The way that we established rating numbers is by taking a given qualitative assessment and assigning a numerical value tho that assessment. For instance, "Overwhelmingly positive" means that the game was regarded by a great number of reviewers as positive. We assigned this

value a score of 8, and each subsequent lower rating was a value of 1 lower. We don't believe that it is at all surprising that this graph looks very much like a normal distribution; we can assume that with a large number of games, player sentiment towards games approaches a normal distribution.

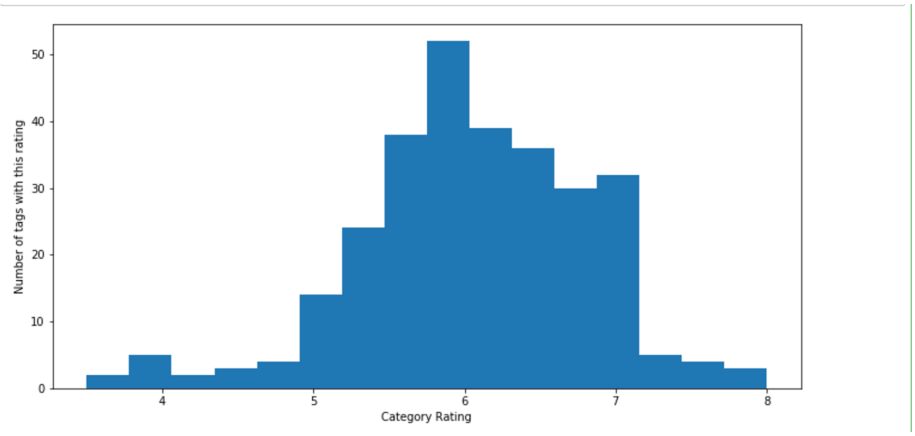


Figure 3: Average Rating Per Tag

### 2.5 Challenges

A major challenge is that our dataset is extremely large. Some games have up to 700,000 comments alone. Our goal will be to find ways to minimize the amount of data we need stored in our database. One way of accomplishing this is by preprocessing the data and saving the data through data structures such as game-tag sparse matrices.

## 3 Weekly Plan

- *March 31*: Find all necessary datasets/scrapers. Fix any bugs in scrapers and ensure that datasets are robust and accessible.
- *April 7*: Spring break. Get the database in which we will store the tag matrix running and populate it using the base set of tags provided by Steam. Develop a preliminary preprocesser for reviews. Start a cron job for periodically pulling new reviews and games.
- *April 14*: Finish the review processor and develop the algorithm for generating tags based on review data. Incorporate features such as hours played and review helpfulness into the review evaluation metric.
- *April 21*: Implement searching between games and calculation of similarity between games. We will most likely look at review similarity and a large portion of this part will likely be computing cosine similarity or at least some form of matrix similarity between the comments of games.
- *April 28*: Add user logins using Steam and incorporate additional data gathered from a user's account, setup a basic front-end. We will add functions to grab a user's games and compute the games that are most similar to the user's current library.
- *May 1*: Prototype completed. Tackle any remaining bugs and polish the user interface. Make sure search bar works, run tests against a number of outputs and save the results onto a file and do a brief scan to make sure they're correct.
- *May 8*: Last week of classes. Final product completed and presentation ready.

*We plan to meet with our TA every Monday.*

### 3.1 Division of Work

- Brian: Natural Language Processing, Database aggregations, Flask and Heroku deployment
- Harrison: Front-end design and implementation
- Nitya: Basic machine learning algorithms and scikit-learn
- Paul: Steam scraping adapters, data acquisition and storage and Flask/Heroku related issues.

## 4 Evaluation

### 4.1 Testing

A major component of testing our application will be comparing our comments-generated tags with the Steam user-generated tags. This will be an effective way of checking to see if our tags are close to what users believe the tags should be. We will also do some own personal user tests to see if recommended games make sense to other people. We will make sure to do smoke and sanity tests after every deployment, and to run our builds against a suite of unit tests. Load/stress testing is not really necessary and will only be done at the very end if at all. In other cases, we might simply just find a list of similarity rankings of games, and make sure that our algorithm at least returns these games somewhat near the top in similarity rankings against each other.

### 4.2 Literature Review and Comparison

Currently, Steam allows users to choose tags for the games they play, and that is how the each game is tagged. Steam does have a decent amount of tags, amounting to a total of 334 tags. The difference between Steam's already generated tags and the tags we hope to generate are that our tags will take the input of everyone who has commented on the game, while Steam's tags only take the input of the users who choose to provide a tag. We will have the opportunity to actually see if some of Steam's tags are correct for certain games.

## 5 Examples

Below is an idealized example of how our application will work:

A user will type in a game title as their query. This query will then be used to retrieve the AppID of the queried game. This AppID will be used to pull the game-tag sparse vector for this specific game. Before the application is released, we plan on initializing this matrix by pulling the comments from popular games, generating our own tags from the comments, and saving these tags into a game-tag sparse matrix that will be used in search. If the game is not found in the game-tag sparse matrix, we will then calculate the vector for this specific game, and append the vector to the game-tag sparse matrix. We hope to update these tags frequently as new comments and new games are constantly added. From here, we will then calculate cosine similarity or use the nearest neighbors our TA said we could use as a possibility. Instead of calculating similarity every time, we could have a matrix (a game-game matrix) to store similarity, so we only have to compute similarities for games that have not been computed. We then return similar games.

Using a similar logic, we hope to allow for multiple games in the query.