A Recommendation System for the Steam Online Game Store

Using Text Classification Model

**Andrew Greensweight**

**CpE 646 Pattern Recognition and Classification**

**Summer 2022**

**ABSTRACT:** Steam serves as the largest digital distribution platform for Personal Computer (PC) gaming, with approximately 120 million monthly users. To efficiently serve its users a unique and tailored storefront, Steam, like many large online storefronts, deploys machine learning to recommend users products they may be interested in. The objective of this project was to explore an offline recommendation system design using machine learning and the Term Frequency-Inverse Document Frequency (TF-IDF) statistical measure to utilize features extracted from a third-party Steam Application Programming Interface (API) called SteamSpy. The scope of the project included the following three phases: data acquisition using SteamSpy API, data cleanup and formatting, and data processing including the implementation of the TF-IDF algorithm. The third phase also included a user interface (UI) for a user to enter a game title they currently enjoy, which resulted in ten similar game titles to be recommended. The datasets of 1,000 game entries of the platform Steam and the final recommendations provided were similar to what the Steam storefront recommends for each title.

Introduction

In modern times, consumers are regularly exposed to recommendation systems designed to increase user engagement with an application. Recommendation systems utilize user data to provide a tailored service or product selection pool for a user. This recommendation helps to ensure each user has a good experience using a service or application. To name a few, the following examples deploy an underlying recommendation system within their application: YouTube, TikTok, and Netflix recommend videos and/or movies for you to watch, Amazon recommends products you may be interested in purchasing, and Yelp recommends restaurants. This project aims to explore personalized recommendations which are automatic in nature and are based on a user’s preferences – in this case, a Steam game title the user enjoys.

The primary types of recommender systems include collaborative filtering, content-based filtering, and hybrid filtering. Firstly, the collaborative filtering approach utilizes user details, ratings, and reviews from all users to build recommendations [1] [2] [3]. This approach leverages the data from other users with similar preferences to provide recommendations. A simplified example is if user ‘X’ likes video game ‘A’ and user ‘Y’ likes video game ‘A’ and ‘B’, with the collaborative filtering approach, the model may recommend game ‘B’ to user ‘X’ as users ‘X’ and ‘Y’ have similar preferences. To choose other users to compare to, this filtering selects all users that are neighbors to the current user using similarity measures in the available data related to personal information, cosine metric, and jaccard coefficient binary data [4]. Following this, the k-nearest neighbor classification method is used to generate a prediction value that a user may want what other users have. Alternatively, other techniques that can be based in collaborative filtering may include web mining algorithms, decisions trees and support vector machines [5].

The next primary type of recommender system uses content-based filtering to analyze products and establish a similarity metric with a user to recommend products. This approach relies heavily on information retrieval to make recommendations [5]. Content-based in this sense refers to instances where content can be read or analyzed such as in articles, movies, and anything with metadata attached to it. The content is assigned weighted labels to determine how well they describe the source. Similarly in this approach, the labeled content can be vectorized, allowing clustering algorithms to be used to make recommendations to the user [6]. Algorithms such as k-nearest neighbor Bayesian, and neural networks may be used to make the recommendations once the labeled content has been vectorized. Moreover, information retrieval and filtering systems can analyze large amounts of data to generate user attributes and ultimately a user profile. For this project, the source comes from the text used to describe the genre of the game and relevant keywords or “tags”. The attributes contained in the created user profile are compared with keywords making up the recommendation. Each keyword is weighted using term frequency-inverse document frequency (TF-IDF) to assign a perceived importance [1]. As such, this allows the user profile to be compared to a product profile for the similarities needed to make a recommendation.

The final primary type of recommender system uses hybrid filtering to provide its recommendations. Hybrid filtering provides a mesh of collaborative and content-based filtering for the purpose of avoiding problems from each individual filtering approach [1]. In this type, both filtering systems are implemented separately then their results are combined. This requires two recommender systems to be built and fielded. The recommendations will be generated separately then combined linearly [7]. Additionally, these recommendations will be weighted per each individual user based on the user’s preferences. The next step in this process is to identify which users share a similar profile to the user in question, as opposed to identifying which product shares a similar profile to a user per the content-based filter process. In other words, a common approach for hybrid filtered systems is to use collaborative filtering with content-based user profiles generated through a content-based filter approach [1].

The project was taken on from the perspective of someone who was just contracted by Steam to develop a recommender system to take a new user’s preference and return a set of game titles they may be interested in. The goals of this project were to download, process and analyze a data set of Steam games from the Steam store, and provide recommendations for similar game titles based on a user’s input. The project served as an opportunity to download, process, and analyze real-life datasets in order to provide a service. In the broader context, this project lends itself to the work a data engineer may conduct in their daily line of work and the decisions they may need to make along the way when deploying a machine learning based system. Although this concept is not necessarily novel in nature, the recommendation accuracy varies drastically based on the design choices executed such as the features chosen for content-based filtering and the dataset manipulations performed.

Steam was selected for this project because it is one of the largest video game digital distribution platforms and has a large of amount of data available publicly for data scientists to experiment with.

For this project, the content-based filter approach was implemented to design a Steam video game recommendation system. This approached was coupled with the TF-IDF statistical measure for the purpose of assigning weights to attributes for each game and then a cosine similarity metric was calculated to determine which ten vectors were nearest each other. TF-IDF was the statistical measure of choice because it is widely used in content-based recommender systems and because its application is best suited for situations in which the features are structured and easy to parse and analyze [1] [9]. The TF-IDF method can be described by the equation below, where “w” corresponds to the weight for each keyword within a document [10]:

Text

Description automatically generated

Figure 1. TF-IDF Method

TF-IDF was invented for document search and information retrieval. It works by increasing proportionally to the number of times a word appears in a document and is adjusted based on the number of documents that contain the word. Essentially, this causes common words that you may find in every document such as “the” to rank low even though they may appear many times, since they do not appear to mean much in a particular document [9]. However, if a word appears many times in a single document, but does not appear many times in the other documents.

The significance of TF-IDF in machine learning is that text needs to be converted into numbers for machine learning algorithms to be applied. The process of converting text to numbers is known as text vectorization. TF-IDF provides a vehicle for representing each word with a numerical value to represent how relevant the word is in the document. Following this, the documents with similar words will have similar vectors which can be evaluated using machine learning algorithms [10].

One of the drivers for using content-based filtering specifically was because the process of selecting an individual Steam user is tedious due to Steam not providing access to usernames on their site. To elaborate, the ability to compare one user’s preferences to other users’ preferences is not publicly available in an outward facing database. The third-party SteamSpy service was chosen over the main Steam API because SteamSpy gathers many of the same useful metrics and these metrics are more easily accessible through its API (see [8] for API documentation).

Method

Design Approach

A three-phase approach was taken for this project. The three phases consisted of data acquisition, data clean up, data processing and analysis. A high-level block diagram of the process is presented in Figure 1.

Diagram

Description automatically generated

Figure 2. Block Diagram.

The tools for this project included:

* Python (version 3.9) in the Spyder IDE
  + Import requests and sys
* Pandas – a Python software library for data analysis
* SteamSpy API [8]
* Scikit-Learn – a Python machine learning library

***Data Acquisition***

This phase began by using the SteamSpy API to retrieve data. A general function to process requests from an API when given a website URL and set of parameters was coded to interface with the SteamSpy API. Below is the JSON data that can be retrieved for each game [8].

Text

Description automatically generated

Figure 3. Features available from SteamSpy

Each game on the Steam store has a unique app ID. SteamSpy captures a subset of these features when you request ‘all’ from the API, however, to acquire the full set of data, the information per specific app ID would need to be requested individually. The first step was to request the app ID for the first 1000 game entries (the enumeration of the app ID did not imply release data, i.e. extracting the first 1000 app IDs captured most played games in the database). The next step was to iterate through the app IDs and request individual app data from the servers. This step consisted of coding a function to process the data retrieval in batches and to save the index so that if the download is interrupted, it may be resumed at the same index. It was also coded to not poll the API too quickly. This process of the project was the most time-consuming as this was the investigator’s first time using an API. Once downloaded, the SteamSpy data was converted to a Pandas data frame and was written to a Comma-Separated Value (CSV) file (raw\_steamspy\_data.csv) for further processing.

***Data Cleaning***

The goal of this phase was to parse the features of interest. The raw dataset from SteamSpy has columns for the features listed in Figure 2. Upon a cursory look, it was determined score\_rank returned not-a-number (NaN) for every single value, so this was immediately dropped. In similar fashion, the temporal features for average play time within the last two weeks and the median play time within the last two weeks tended to yield “0” for a handful of game entries. These two columns were also dropped. Moreover, the Pandas library has convenient tools for identifying specific values or null values within features. This check was conducted to eliminate any entries with crucial missing values such as for “name”.

The next step was to pre-process and format the “tags” feature column which would serve as the backbone for the content-based filtering approach. It was determined that the tags were not necessarily always JSON arrays. In some instances, they were dictionaries or lists and would need to be converted for easy manipulation later in the pipeline.

The final step was to drop the temporal related play time, user score, price, and discount columns were dropped. The rationale for these design choices is that the objective is to create an “offline” recommender that should not depend on the trends for the last two weeks. Additionally, for this specific project, the dollar value related features should not play a role in recommending a game.

***Data Processing***

The phase started with a pre-process function to remove games that do not support English and to perform additional formatting of the text-based features that will be trained on the model.

As mentioned in the introduction of this report, the text needs to be converted into numbers to apply machine learning algorithms. The chosen text for this project comes from the “genre” feature and the “tags” feature as these keywords summarize each video game well. The “genre” tag is assigned by the developer of the game, while the “tags” are voted on by the Steam user who play the game. These two features were combined to form each document, with the distinction being made to use the top 10 highest voted “tags” for each game. Additionally, duplicate words within these document strings were removed during the pre-processing to lower weight assigned to each word. Below is a sample of the pre-processed features for each game entry.

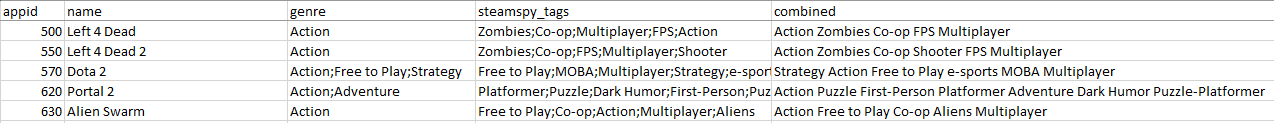


Figure 4. Sample combination of Features

The scikit-learn library was implemented to carry out the text vectorization via the command TfidfVectorizer. For the pre-processed Pandas data frame, this converted the raw collection of the “combined” features column into a matrix of TF-IDF features. Next, each row of the TF-IDF matrix is multiplied with the entire matrix using a linear kernel for the reasons discussed later in this report. This produced a similarity value between 0 and 1 for each of the games against the rest of the games. By sorting the similarity matrix by index in reverse order, we can extract the top 10 most similar games for each game.

Results

Overall, from the initial 1000 games that were downloaded from the SteamSpy API, 968 remain after the pre-processing evolution. Below are figures comparing the results yielded from this project against the actual values in the Steam store, with the caveat that the final dataset 968 has its limitations for relevant titles captured.

Text

Description automatically generated

Graphical user interface, website

Description automatically generated

Text

Description automatically generated

Graphical user interface, website

Description automatically generated

Figure 5. UI of the Project compared to the actual Steam storefront

Additionally, the similarity matrix and the derived similar game titles are provided below.

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated

Figure 6. Similarity Matrix and Corresponding Games.

All datasets from project will be available in the submitted folder.

Discussion

Although the project yielded good recommendations compared to a factual subset shown on the Steam storefront there is room for improvement. In order to reap the full benefits of the TF-IDF statistical measure, the video game descriptions should be retrieved from the Steam storefront. This would allow the weighted document-term matrix to grow significantly, which may produce more accurate recommendations as there are more links to be formed between game titles. Additionally, this project could have been made more robust by retrieving more game titles from the SteamSpy API. This would allow for more data points to be used in the text vectorization process.

The linear kernel was good because there were a substantial amount of features under the microscope. When using a linear-kernel, the mapping of data to a higher dimensional space does not improve the performance [6]. In text classification, both the number of instances (document) and features (words) are large which produces the optimal situation to use a linear kernel [1].

***Future Work***

A large portion of time was allocated to downloading and pre-processing the data retrieved from SteamSpy. As I continue to work on this project, I would like to normalize the other numerical values and combine them with the similarity matrix from. From here, I can determine the cosine similarity between vectors generated from each game. Additionally, it may be worthwhile to read in the game description, then perform the text vectorization.

Conclusion

The objectives for this project were met by the successful download, cleaning, and processing of data from the Steam platform. With the aggregated data, TF-IDF provided the vector values of wording similarities between video games on Steam needed to make a comparison between

*Build Instructions*

The tools are listed in the “Methods” section herein.

To arrive at the endpoint exactly as I did from start to finish, follow the steps listed below.

Ensure that the proper directories are created at each step:

1. Run **data\_acquisition.py** to create “**raw\_steamspy\_data.csv**” to newly created directory **“.Cpe646\_Project**”. Note: this may take approximately 10 minutes.
2. Run **data\_cleanup.py** to create “**steamspy\_data\_clean.csv**” to directory “.**Cpe646\_Project**”
3. Run **data\_processing.py** to start UI. Enter the game title exactly how it may appear on Steam. Three random samples are provided to try. You can uncomment lines 145 and 146 to output files for the similarity matrix and TF-IDF weighted document-term matrix.

Another approach is to download **steamspy\_data\_clean.csv** and ensure it is in the file path **‘.Cpe646\_Project/steamspy\_data\_clean.csv'**. This should be in the same location as the python files.

References

[1] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," Knowledge and Data Engineering, IEEE Transactions, vol. 17, no. 6, pp. 734-749, 2005.

[2] A. N. Regi and R. Sandra, "A Survey on Recommendation Techniques in E-Commerce," vol. 2, no. 12, 2013.

[3] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in 20th International Conference on Very Large Databases, Santiago, 1994.

[4] S. Sivapalan, A. Sadeghian, H. Rahanam, and A. Madni, (2014) "Recommender Systems in E-Commerce," DOI: 10.13140/2.1.3235.5847

[5] C.-P. Wei, M. Shaw and R. Easley, "A Survey of Recommendation Systems in Electronic Commerce," National Sun Yat-Sen University, Kaohsiung, 2001.

[6] M. Angrosh, S. Cranefield, and N. Stanger, (2013). “Context identification of sentences in research articles: Towards developing intelligent tools for the research community.” Natural Language Engineering.

[7] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes and M. Sartin, "Combining content-based and collaborative filters in an online newspaper," in ACM, Berkeley, 1999.

[8] <https://steamspy.com/api.php>

[9] G. Salton, Automatic text processing: the transformation, analysis, and retrieval of information by computer, Boston: Assison-Wesley Longman Publishing Co., 1989.

[10] Robertson, S. (2004). "Understanding inverse document frequency: On theoretical arguments for IDF”. Journal of Documentation. 60 (5): 503-520. DOI:10.1108/00220410410560582.