Generating Game Recommendations and Analyzing Feature Importance from Steam Dataset

Akshay Elavia
Software Engineering
San Jose State University
San Jose, USA
akshay.elavia@sjsu.edu

Akshay Orpe
Software Engineering
San Jose State University
San Jose, USA
akshay.orpe@sjsu.edu

Mukesh Mogal
Software Engineering
San Jose State University
San Jose, USA
mukeshbhausaheb.mogal@sjsu.edu

I. Introduction

Steam is the largest online game distribution platform for Personal Computing games. Steam has around 75% of the market share in 2013 [1]. Currently, there are more than 34,000 games with over 95 million active users present on the Steam platform. There are two ways customers can buy games from Steam, one way is to buy an individual game and another option is to buy Steam Bundles. Steam Bundle allows user to purchase multiple products (games) together at a discounted rate. Currently, Steam provides two types of bundles – 1. Complete the Set: this bundle allows user to pay for the games which they don't own already, 2. Purchase Together: This Bundle will include the relevant games together.

In this project, we are going to focus on the Games recommendation as well as Bundle recommendation. Having a great recommendation platform is a key part of the business for online Game Distribution Platform like Steam.

II. System Design and Implementation

A. *System Architecture Diagram*

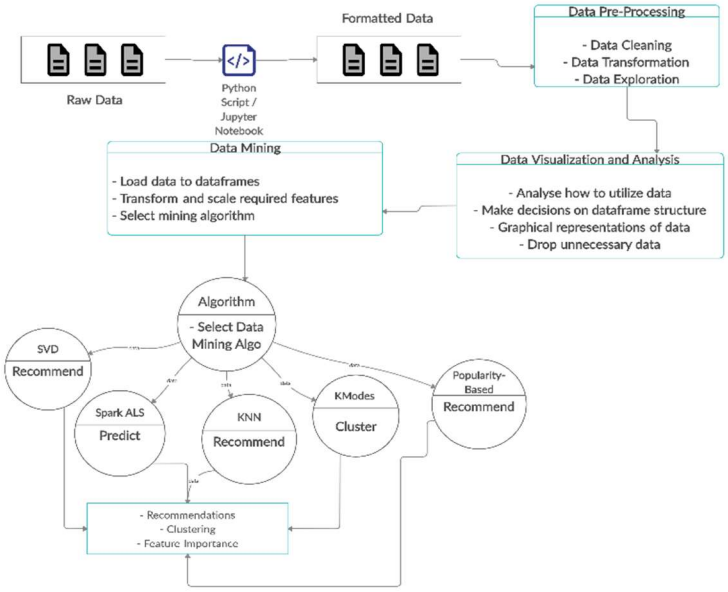

Figure 1 System Architecture

B. *Methodology*

We have decided to implement multiple methods of clustering and recommendation on the existing data. We will compare the results of these algorithms and describe how they can be combined to create a robust recommendation system for Steam games.

a. **Popularity-based Recommendations:** Using popularity-based recommendation is a very simple yet powerful idea. Popularity-based implementation is critical for new users as many times recommender systems such as collaborative filtering, content-based filtering might face cold start problems. This implementation will always be able to suggest the most popular games to the user irrespective of the user data collected by the system.

b. **Game Bundle Recommendations using SVD:** We predict ratings of users for unseen games using collaborative filtering. Using these predictions, a classifier is made to determine whether to recommend an unseen game to the user. Finally, for each user, the model is executed each item in a game bundle and a cumulative rating of the bundle is calculated for that user. Game bundles with cumulative rating above a threshold are recommended to the user.

c. **Spark ALS:** Spark ALS is one of the most popular algorithms provided in spark MLib. Alternating Least Squares is a matrix factorization algorithm. ALS converts high dimension user-item matrix into two low dimension matrices. In first, user matrix, rows indicate users and columns indicate latent factors. In Second, item matrix, rows are latent factors and columns represent items.

Using ALS, even less-known items can have rich latent representations as much as popular items have, which improves the recommendation system's ability to recommend less-known items. Spark ALS provides options to run Implicit and Explicit feedback.

d. **Feature Importance using Random Forest:** We intend to identify game attributes which impact a user's playtime. Using Random Forest, we will extract

the importance of each feature and visualize this using graphs.

e. **Clustering using KModes:** This analysis attempts to define clusters of games based on game metadata. As every feature in the game metadata is a categorical feature, KModes is a better choice than KMeans because it works better for categorical values than continuous values. Once these clusters of games are created, they are compared against actual bundles of games to see till what degree they match the clusters created by the KModes algorithm. A match score is computed for each bundle to signify the confidence of matching that bundle to a "cluster".

f. **Game Recommendations using KNN:** Using the clusters generated using KModes algorithm and the vectors generated as part of feature importance analysis, we will find the nearest neighbors to recommend games to a user who has at least one purchase. We will use game genre and specifications as the primary parameters for this.

III. TECHNOLOGIES & TOOLS USED

Multiple Tools and Technologies are used to achieve the best possible for results. Below mentioned are the Technologies and Tools primarily used for this project.

A. Programming Language: Python

Python has rich set of Data Science Libraries such TensorFlow, surprise, Keras, SciPy along with simple syntax and general-prepose nature python was a clear choice for this project. Python 3.7 was used for Development along with multiple libraries, few of important libraries are mentioned below.

B. Notebooks: Jupyter Notebooks and Google Colab

The Jupyter Notebook is an open-source web application used for developing the code, visualization, and narrative text. Jupyter notebooks are very interactive and improve efficiency for developing Data Science related tasks. Google Colab Notebooks are very similar to the Jupyter Notebooks but they are hosed on the cloud. The Cloud infrastructure makes Colab powerful tool for processing large scale data.

C. Distributed Computing Framework: Apache Spark

Apache Spark is a unified analytics engine for large-scale data processing. Along with capability of distributed computing Apache Spark provides a large number of Machine Learning tools. Apache Spark library MLib has many recommender systems libraries present out of the box.

For this project Apache Spark (Pyspark) was used for Popularity based suggestions and Collaborative filtering. Performance provided by Pyspark module is significantly better than simple python programs.

IV. EXPERIMENTS

A. Dataset Details

The Steam Dataset contains multiple files pertaining to game metadata, purchases of users, game reviews, and game bundles:

Dataset	Rows	No. of Features	Size
Users	2,567,538	9	553 MB
Items	15,474	14	21 MB
Reviews	7,793,069	10	4.27 GB
Game Bundles	615	12	834 KB

B. Data Cleaning

The raw data files are not invalid JSON format. They have single quotes instead of double quotes, and text fields like "item_name" and "review" also have inconsistent single and double-quotes. This rules out simple string substitution, so specific scripting is done for conversion to valid JSON. For each line: i) replace the single quotes with double quotes ii) Find indexes for the text fields, and replace the double quotes within the field values with single quotes iii) Remove escape characters

C. Data Preparation

i) **Reccomendation dataset**

To predict ratings for items, we need a dataset of the following format:

user_id,item_id,rating

As the user dataset does not have an explicit "rating" value, we will use implicit ratings to estimate the user's likeliness towards the game. Each user has a "playtime_forever" value, which we use to estimate the rating the user would give this game. For example, if the user has been playing a particular game for a long time, we can confidently assume the user would rate this game highly. The "playtime_forever" values varies a lot between users, so we cannot use the raw value as it would skew the results. For example, if a user's average playtime is 100 hours, and they have spent 5 hours playing a game, then the rating shouldn't be that high. But if a user's average playtime is 10 hours, and they have spent 5 hours playing a game, then the rating should be high. We normalize the rating between 1 and 100 by taking into consideration the minimum playtime and maximum playtime of each user. The playtime is scaled to a rating between 1 and 100 using the formula:

scaled_playtime = (1-(playtime-min_playtime)/(max_playtime-min_playtime)) + 100*((playtime-min_playtime)/(max_playtime-min_playtime))

Another decision made is to ignore items if the user has a playtime of 0 for that item. Because there can be several reasons the playtime is empty. For example, if the user has only recently purchased the game, or the user has not gotten

around to playing this game yet. Since we don't have any extra information to infer such reasons, it's better to not include 0 values as it would significantly skew the scaled rating. This prepared dataset has 3,252,105 rows and 3 columns with size 125MB.

b) **Game attributes dataset**

To analyze feature importance and further generate content-based recommendations, we had to leverage the game metadata. We will transform the item metadata in the following format:
user_id, item_id, title, publisher, developer, genre_1. . .genre_22, spec_1. . .spec_31
The raw data for each game is presented such that the "genre" and "spec" attributes are a list of categorical values. Hence, direct one-hot encoding using existing libraries was not possible. We followed a two-step process in order to convert categorical values into binary:

- *Step1:* Scan the raw data to extract all possible unique genres and specifications.
- *Step2:* Iterate over the formatted raw data and generate binary vectors representing multiple genres and specs for each item.

The resultant item vector looks as described above. We have 22 genres and 31 specifications of the games listed in the item metadata provided. This data was then loaded into a data frame for further analysis.

*D. **Algorithms Implementations***

a. Popularity-based recommendation: For this experiment, first data is needed to be loaded into Spark RDD. Even after loading data into RDD, it is not possible to find most popular games as data is stored in complex JSON structure. So, the next step needed to be done is normalizing the data into spark DF so that analytics can be performed on data directly. Once data is normalized, we need to group by all the playing hours for each game and then show the results by ordering games playtime in descending order. For future implementation, these popular games can be divided into different genres for genre-wise recommendation.

most_popular_games	totalPlayTime
Counter-Strike: Global Offensive	785040461
Garry's Mod	448342370
Terraria	154941260
The Elder Scrolls V: Skyrim	136652734
Warframe	123992479
Counter-Strike: Source	112604138
Left 4 Dead 2	102182773
PAYDAY 2	99755652
Sid Meier's Civilization V	82375981
Rust	81117861
Borderlands 2	80421808
Arma 3	67327273
Grand Theft Auto V	59667071
Unturned	50953605
Fallout 4	45244297
War Thunder	43565307
PlanetSide 2	41441669
Arma 2: Operation Arrowhead	40464456
Mount & Blade: Warband	39609531
Call of Duty: Black Ops II - Multiplayer	38639035

Figure 2 Popularity Recommendations

b. Game Bundle Recommendations using SVD: SVD algorithm is used to extract information from latent factors and predict ratings of a user for an unseen game. Following hyperparameters were tested:

Hyperparameters	Best Performing	RMSE
n_factors: [0,1,2] n_epochs: [10, 50, 60] lr_all: [0.0005, 0.0007] reg_all: [0.01, 0.09]	0 60 0.0005 0.09	15.1188
n_factors: [5,7,9] n_epochs: [60,70,90] lr_all: [0.0001, 0.0006] reg_all: [0.12, 0.15]	9 60 0.0001 0.12	15.6827

SVD model is trained using the best performing hyperparameters. This ratings predictor model is extended to recommend games as well as game bundles to user. For performing that task, we need to predict whether to recommend a particular game to a user or not. Using the prepared recommendation dataset, we predict the rating of a user for a game. If the predicted score is above 50, the game is recommended to the user. Following is the confusion matrix for this classifier:

	Actual True	Actual False
Predicted True	4324	4416
Predicted False	20941	620739

Now, given a user id, all bundles' cumulative score is calculated by predicting the rating of the user for each item. If the cumulative score is above a threshold, the whole bundle is recommended to the user.

c. Spark ALS: Implicit train gave the best results for the ALS. Below parameters were tested which gave the best possible results.

Ranks	**Iterations**	**Alpha**	**RMSE**
5	5	0.01	18.98680
5	5	2	18.84006
10	5	2	18.82714
10	5	0.01	18.99780

d. Feature Importance using Random Forest: SVD and Spark ALS will take into consideration a user's "playtime_forever" before recommending new items. It is important to identify other features which have an impact on a user's playtime. As discussed in "data preparation" section, genres and specifications were identified as features which could impact a user's decision when selecting a new game. Random Forest algorithm can help us extrapolate the importance of each feature with reference to the total playtime. Random Forest Classifier will generate a large decision tree at training time while evaluating it against the output

feature. Items with extremely low frequency are rejected by Random Forest as they cause clutter in the result. We need to eliminate all items below the frequency threshold. In our case, we have used 2 as the threshold which means that any item which occurs only once based on the number of play hours will not be considered for this classification.

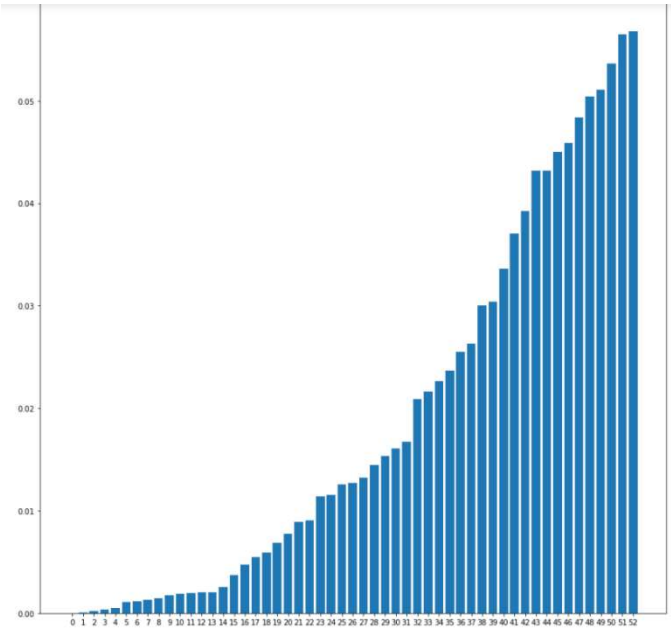

Figure 3 Feature Importance

The above graph describes the importance of each genre and specification with respect to the playtime. Our result shows that the following features are most important:

Feature	Importance
genre_action	0.057
spec_downloadable_content	0.056
genre_adventure	0.053
spec_steam_tradng_cards	0.051
genre_indie	0.050

This table gives us a good idea about the best-selling game themes and proves that these can be very useful when recommending new games based on past genres that a user has interacted with.

e. **Clustering using KModes:**

The KModes clustering algorithm has two modes for calculating distance between the vectors: Huang and Cao. Cluster compute costs for both these types are calculated for different number of clusters.

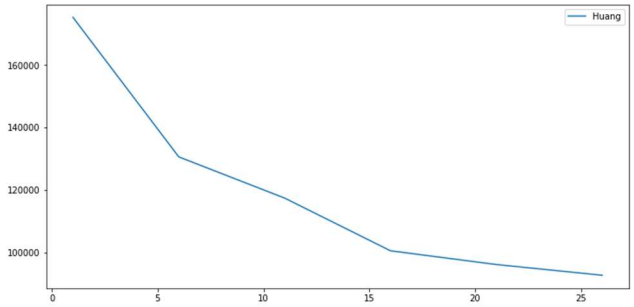

Figure 4 KModes Elbow Graph

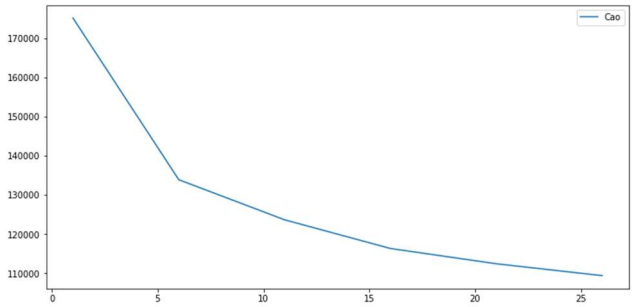

Figure 5 KModes Elbow Graph 2

Both Elbow Curve graphs suggest number of clusters as 7. Clustering using Huang mode gave more equitable distribution of clusters. Following graph shows the distribution of a feature "downloadable_content" across different clusters.

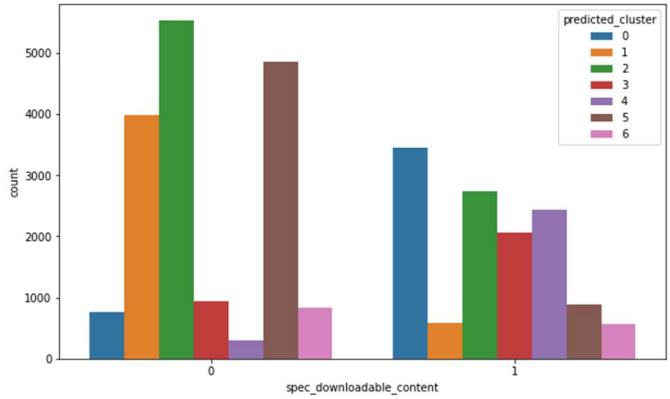

Figure 6 Feature distribution across clusters

A sample cluster with its associated features is represented as follows:

cluster 0:

genre_indie
genre_action
spec_full_controller_support
spec_steam_achievements
spec_shared/split_screen
spec_multi-player spec_single-player
spec_steam_trading_cards

We combine the dataframe with these cluster predictions with the games attributes dataframe. Next, for each item in the bundle, we get which "cluster" the item would belong to. Then we check what percentage of items in the bundle belongs to the same "cluster". The following graph plots the frequency of the cluster match_score for bundles:

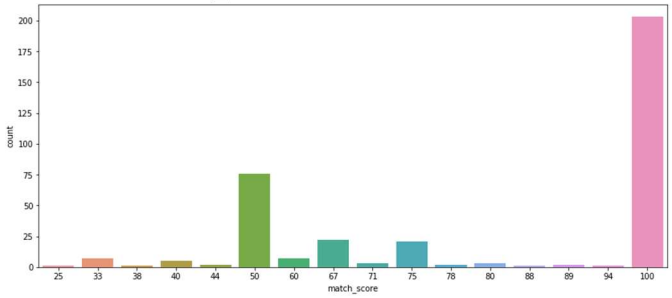

Figure 7 Bundle similarity with cluster

f. Game Recommendations using KNN: We have converted item metadata and combined it with user-item data to obtain binary vectors for each user-item pair. The process for this has been described in section III. This data can now be used to provide content-based recommendations to a user. A plotting of clusters obtained using KModes algorithm illustrated how the popularity of items correlated with its cluster.

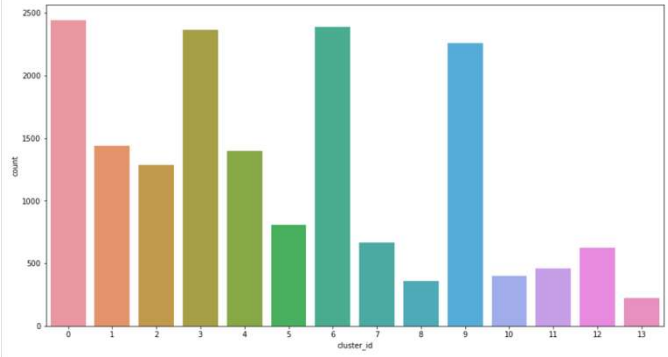

Figure 8 Cluster vs Items

The above graph shows the distribution of items across all clusters. According to this graph, cluster 0 seems to be the most popular closely followed by clusters 3,6 and 9. However, when you plot the same against the playtime for each cluster, it that the features belonging to that cluster have a huge impact on the popularity of items.

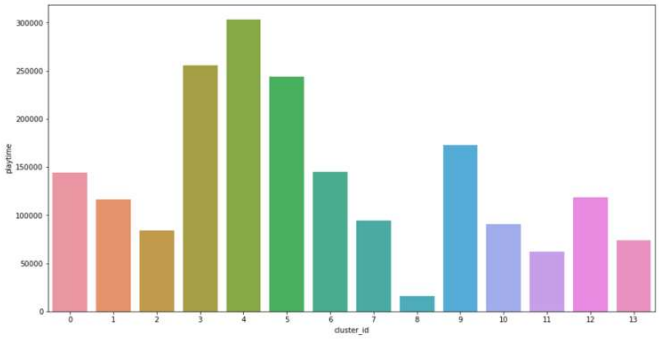

Figure 9 Cluster vs Playtime

As illustrated above, cluster 4 is the most popular cluster based on the playtime it has received. At the same time cluster 0 which has the maximum number of items is moderately ranked. When an existing user tries to access the Steam website, recommendations can be made by finding his nearest cluster.
A sample cluster is represented as follows:

cluster 4:
genre_action
spec_full_controller_support
spec_steam_cloud
spec_steam_achievements
spec_multi-player
spec_downloadable_content
spec_single-player
spec_steam_trading_cards

The above list of features indicates the combination preferred by most gamers when selecting a game.

*E. **Figures and Tables***

*a. **Project Responsibilities:***

Task	**Responsibility**
Finding Dataset	Mukesh Mogal
Project Proposal	Everyone
Data Pre-Processing	Everyone
Popularity Based Recommendation	Mukesh Mogal
Random Forest	Akshay Orpe
SVD	Akshay Elavia
Spark ALS	Mukesh Mogal
KModes Algorithm	Akshay Elavia, Akshay Orpe
Recommendation Dataset	Akshay Elavia
Game Attributes Dataset	Akshay Orpe
Documentation	Everyone

CONCLUSION

We have tried generating various metrics by using the above-mentioned algorithms. From our analysis, game bundles show a high resemblance to the clusters generated using KModes algorithm. While playtime is a great indication of the popularity of a game, the genre and specification provide a unique dimension in categorizing games. The popularity-based algorithm is helpful in overcoming the cold start problem which other algorithms will encounter. We can conclude that these results can be further enhanced by acquiring more details of interactions between a user and a game. As an extension of the feature clustering, we can apply graph mining on this dataset to connect multiple users interested in the same cluster of features.

References

[1] https://en.wikipedia.org/wiki/Steam_(service)

[2] https://partner.steamgames.com/doc/store/application/bundles

[3] https://jupyter.org

[4] https://spark.apache.org

[5] https://medium.com/@davidmasse8/unsupervised-learning-for-categorical-data-dd7e497033ae

[6] https://www.kaggle.com/ashydv/bank-customer-clustering-k-modes-clustering

[7] https://towardsdatascience.com/steam-recommendation-systems-4358917288eb