Game Recommendation System

CIS * 4020 Data Science

Darren Chay Loong

University Of Guelph

ID: 1049254

9 December 2021

Abstract

This report aims to delve into various techniques which could be used to recommend video games to people by making use of the information available from games, such as genre, descriptions, reviews, etc., and finding which approach would be the best. Furthermore, this report will also perform an in-depth analysis of the various features that are available and attempt to find which features will be most relevant for training the various models. Finally, after observing and analyzing all the different models, a conclusion will be made to assess which model will solve the problem the most accurately, as well as any further improvements that could have been done.

1. Introduction

1.1 Defining the problem

With video games now becoming the norm, especially during this pandemic where everyone had to stay at home, people can start having trouble finding which next game to play. Therefore, this report will attempt to find a method for recommending games to people based on the games. However, nowadays, since there are so many games and various platforms, such as PC, Mobile, PS4, Xbox, Nintendo, etc., I will narrow down the scope by focussing only on PC games for this report.

There are various features that define a game, such as whether the game is multiplayer, the story of the game, who developed the game, etc. Indeed, this could greatly increase the dimensionality of the project when we try and take all those different features, and therefore, a proper analysis of which features affect what makes a game 'good' and 'bad' is necessary. This also brings another question to the table: what makes a game good or bad? Since there is no real way of objectively defining this, again, we will need to find a way to measure how good or bad a game is (for e.g, the sales of a game).

Finally, since we are dealing with real-life data, there won't be any labels or training data that can be used to train our models, and therefore, we would most likely need to apply unsupervised learning algorithms when trying to implement a solution to our problem and find a way to test whether our solution was good or bad.

1.2 Gathering of information and data

Since there are several different platforms, such as Steam and Epic Games, that people can use to buy games, we are going to further narrow down our scope of data to games found on Steam only. This will help in analyzing the data more concisely and reduce the time taken to collect and process all the data.

After going over various games on Steam, we can see that each game has common features such as their categories, price, categories, etc. (see Fig.1). A cleaned dataset was found, created by Nik Davis, which will be used to train our models.

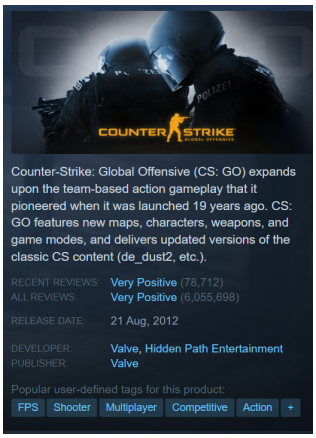

Fig.1

A dataset that contains the reviews for several games was also found. This dataset contains a list of reviewers and the games that they reviewed, as well as whether their review was positive or helpful. This can also be used to further improve the recommendations in our solution.

Some research was also done on some possible solutions that could be used when coming up with a solution for this project. In doing so, an interesting solution was found for a movie recommendation system that implemented several methods for recommending movies based on various features (such as Demographic Filtering). Since the movie recommendation problem is similar to this one, some of those methods can be adopted, implemented, and adjusted to suit this current problem as well.

1.3 Success Criteria

In order to assess whether the solution was properly implemented, we can manually check the recommendations for a specific game and see whether they are acceptable or not. Furthermore, we can also check the recommendations that Steam provides and compare those results to ours as well.

2. Data & Feature Analysis

2.1 Cleaning and Formatting of Data

2.1.1 Steam dataset

Fortunately, not a lot of cleaning had to be done for the steam dataset since the data had already been cleaned. However, some formatting has been done in order to better analyze the data and prepare it for training.

In order to cluster the data appropriately, some formatting has been done on some features, such as the genres column (which contained a list of genres per game), where they were one hot encoded to have a specific column for each element. This also helped identify the number of genres that were present in the dataset as well. The same thing was also done for the steamspy_tags (which has 339 tags) and the categories (which has 29 categories). The genres found after hot encoding were:

```
['Action', 'Adventure', 'Animation & Modeling', 'Casual',  'Design & Illustration', 'Early Access', 'Education', 'Free to Play', 'Gore', 'Indie', 'Massively Multiplayer', 'Nudity', 'RPG', 'Racing', 'Sexual Content', 'Simulation', 'Software Training', 'Sports',  'Strategy', 'Utilities', 'Video Production', 'Violent', 'Accounting', 'Audio Production', 'Documentary', 'Game Development', 'Photo Editing', 'Tutorial', 'Web Publishing']
```

The rating of a game was also calculated using the percentage of positive ratings out of all the ratings. This feature will help in giving a better representation of how good a game is as compared to positive ratings and negative ratings only.

2.1.2 Reviews dataset

For the reviews dataset, there were quite a few rows that did not have a game name associated with the review and some that didn't have a review text. Since having a game associated with the review is important, those rows were removed from the dataset.

Note: See the 1_steam_data_processing.ipynb and 1_weighted_score.ipynb file for more information on the cleaning and formatting for this section.

2.2 Analysis of the data and Feature Engineering

2.2.1 Steam dataset

As there were many features present in the steam dataset, only some features were thoroughly analyzed. First, the features were mainly described using 2 categories: features that gave more information on the type of game (genre, category, steamspy_tag, developer, price, etc) and features that gave an idea of how good a game is (positive and negative ratings, average and median playtime, and the number of owners). These categories will be called category 1 and 2. Any other features were considered unimportant features.

Some of the less important data such as the required age and language were also briefly analyzed, and since for a majority of those types of data, the distribution was heavily skewed (for example non-English games only made up of <2% of the data), this shows that those features do not really contribute much in describing the game (category 1) and were therefore discarded.

When observing the category 2 type features, I noticed that all the features in that category were heavily skewed to the left (see Fig.2), which could imply that there are outliers present in the data. Upon further investigation, those outliers were indeed very popular games, such as Counter-Strike, and should therefore not be excluded from the dataset. Furthermore, the attributes average playtime and median

playtime showed a strange behavior where the games with the highest averages might be outliers. After going through this steam website, we can see that the highest average playtime was around 1700 hours (as of December 2021) and none of the top 10 games found in the dataset feature on the website. Also, the median and average either have a big disparity or they are exactly the same, which is uncommon. Dota 2 which is considered one of the most played games today has a very large disparity between its average (23944) and median playtime (801). Unfortunately, the author of the dataset has not given more information on how these values have been retrieved. However, since the games that do have outliers in terms of playtime are still very popular games, they won't be removed from the dataset.

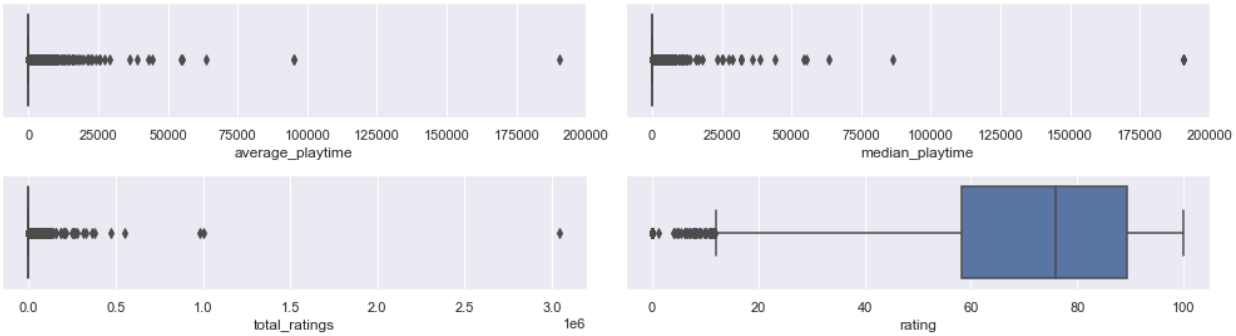

Fig.2 - Box plots of some category 2 features

We can also see through Fig.3 that there is a correlation between both average and median playtime and the number of owners of a game (although a bit hard to see) since the number of owners generally increases as playtime also increases (this is much more visible through average playtime).

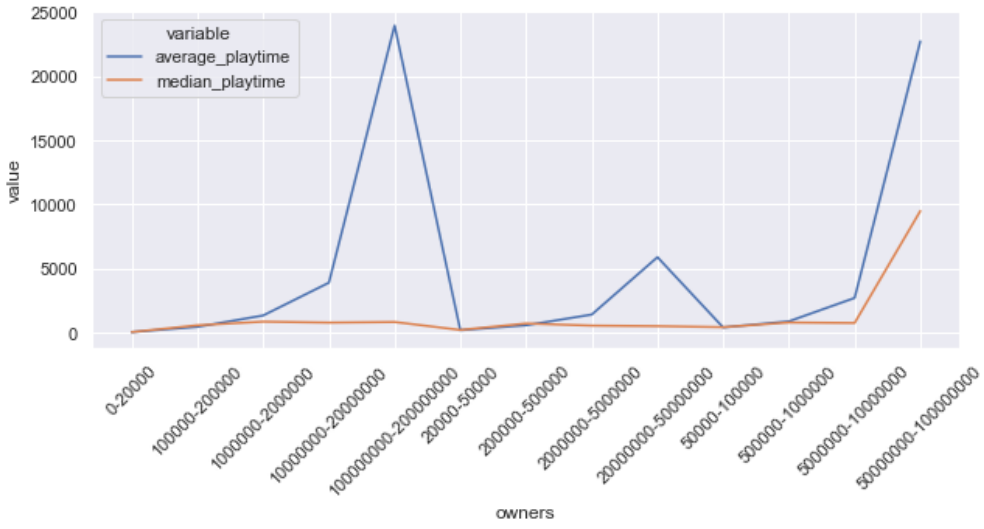

Fig 3 - Number of owners vs playtime (average and median)

After analyzing the category 2 type of features, I have concluded that the 2 features that give the best representation of how good a game is are the number of owners (since this implies how many people bought the game) and the rating of the game (since that indicates whether people liked the game or not).

Out of the various category 2 features, the ones selected for further analysis were: categories and genres.

For genres, each genre was one hot encoded to their own column, and genres that had less than the 25th percentile of the overall counts were removed since they barely add any information. The genres that ended up being removed were: `['Accounting', 'Audio Production', 'Documentary', 'Game Development', 'Photo Editing', 'Tutorial', 'Web Publishing']`. This feature will be one of the main features used when clustering the data later on as this is the one that gives the best description of the type of game it is.

As for categories, some of the most common categories were identified (multiplayer, singleplayer, co-op, and local) out of the 29 categories and were analyzed to see if they gave a good description of the game. This was done by plotting a bar graph for each category against the number of owners. We can see through Fig.4 that there are a lot more single-player games than the other 3 categories and that although skewed to the right, there is variance in the distribution of sales for those 4 categories and they, therefore, do affect the sales of the game.

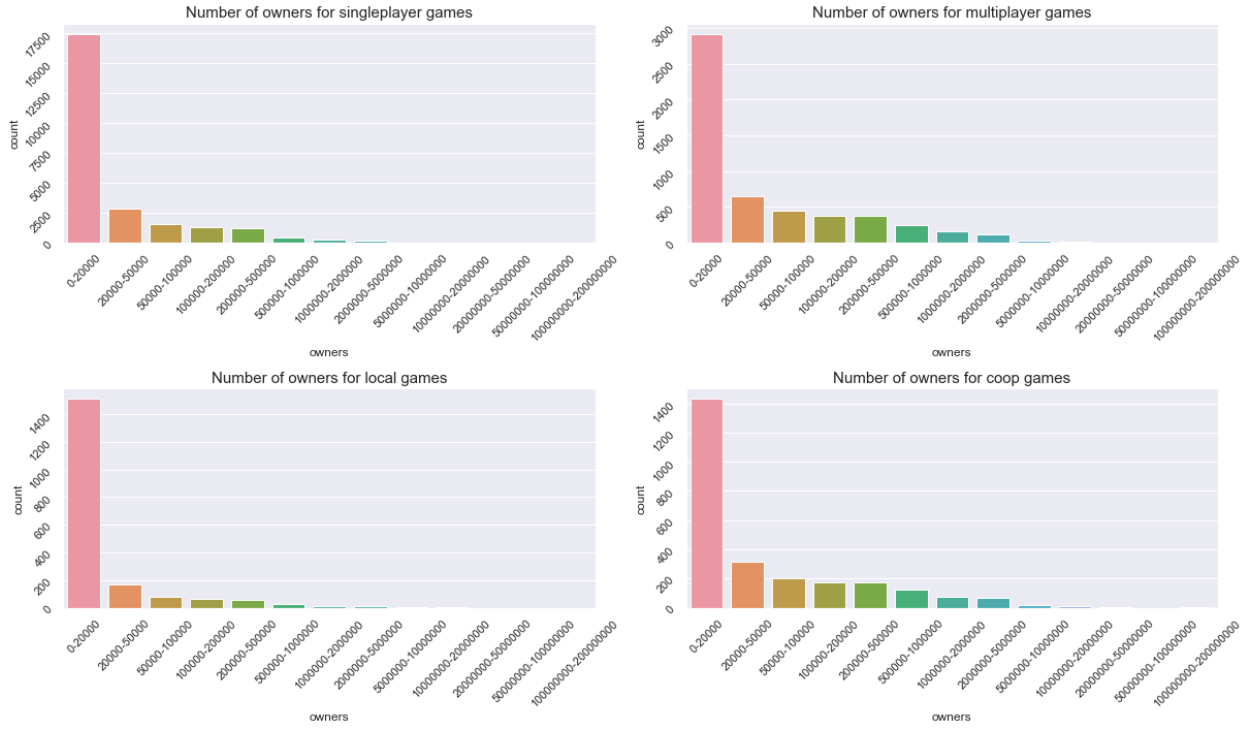

Fig.4 - Categories Bar Plots

Other features such as publisher and developer do give meaningful information on the game (for example a highly known publisher could imply better sales) and could be further analyzed in the future if more research were to be done on the topic. Similarly, although steamspy_tags does give relevant information about the game since it has 339 tags, this would massively increase the size of the dataset if hot encoded, and since genres and categories seem to capture fairly similar information, this feature was not further analyzed for this project. Furthermore, while average playtime gives a good indication of whether a game was good or not, it could also be a good feature to help categorize the games when creating our recommendation system.

A word cloud was also generated for the short descriptions of each game to see if there are any words that appear often in a description. As seen in Fig.5, it is not surprising that 'game' was one of the most common words, as well as 'world'. These short descriptions could also be used in creating recommendations.

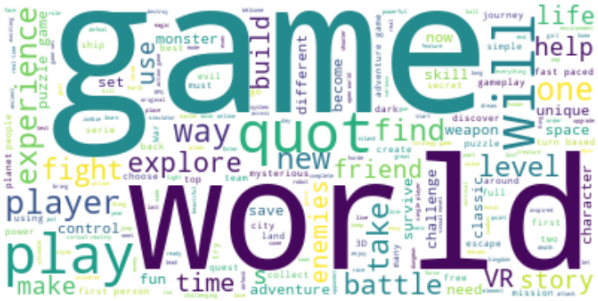

Fig.5 - Word cloud of short description feature

2.2.2 Reviews dataset

In order to give a proper score to a game using the reviews, it would not be appropriate to just calculate the average (since in the case where an average of 1.0 across 1 review is against an average of 0.95 across 100 reviews, the latter would be considered better). Therefore an appropriate formula needs to be chosen to calculate this. After doing some research on possible algorithms that could be used to create a weighted score for each game, an algorithm created by Torn (2017) was chosen to calculate the weighted score. This formula (1) was chosen since it takes into account the number of reviews a game has, which helps prevent instances as mentioned above from happening.

$$\begin{aligned} TotalReviews &= PositiveReviews + NegativeReviews \\ ReviewScore &= \frac{PositiveReviews}{TotalReviews} \\ Rating &= ReviewScore - (ReviewScore - 0.5) * 2^{-log_{10}(TotalReviews+1)} \end{aligned} \tag{1}$$

After calculating the weighted score for each game, the distribution of those scores was visualized, as well as the positive rate (sum of positive reviews divided by the total reviews) and average scores (Fig.6).

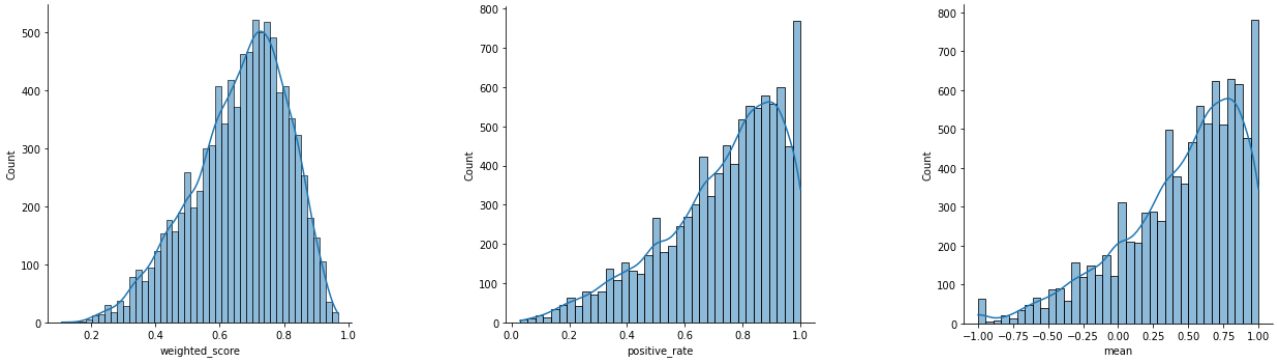

Fig.6 - Distribution of weighted score, positive rate, and average scores

It is clear that unlike the positive rate and mean, which are both skewed to the right, the weighted score provides a more unbiased score for the games and is also more normally distributed, which also shows that this score is better than the other 2. Therefore, the weighted score will be used when training our recommendation system.

Note: See the 1_steam_data_processing.ipynb and 1_weighted_score.ipynb file for a more detailed analysis of the features mentioned in this section, as well as 2_count_vectorization_sol.ipynb for the word cloud visualizations.

2.3 Summary of findings

After analyzing and processing the features, the ones that will be most useful in creating our recommendation system are genres, categories (multiplayer, single-player, coop, and local), short description, average playtime, and the weighted score.

3. Implementation Methods & Algorithms

3.1 Choosing the implementation methods

The solutions that will be implemented will follow a similar rule: pick the games that are most similar to the game we picked and rank them from most similar to least similar. This will be done using 2 different methods: Clustering and Content-Based Filtering.

Since this is done using unsupervised learning, choosing several clustering algorithms seems sensible. By grouping the games into different clusters, we can find out which games are similar to each other (by being in the same cluster) and we can then choose the nearest neighbors to the game in that cluster. The 2 clustering algorithms that were chosen to train our models were Kmeans clustering and Hierarchical Agglomerative Clustering (HAC). Kmeans was chosen because it tries to find the cluster centers for each center in a simple and fast to compute way. HAC was also chosen as another clustering algorithm because, unlike Kmeans, the shape of the clusters and the sensitivity of the cluster centers are not an issue. Therefore, we can compare the results of both clustering algorithms to check if one is performing better than the other.

After reading through Ahmed's (2020) Notebook on Movie Recommendation Systems, I decided to adopt and experiment with the content-based filtering method, since the problems are pretty similar in nature and the data is also quite similar.

Finally, a hybrid model which makes use of both methods will be implemented to give a final recommendation.

3.2 Clustering Method

Before any training is done, the reviews dataset was merged with the steam dataset so that the weighted scores can also be used when training using the different clustering algorithms. After clustering the data, we can then input a game, and the cluster for that game is recorded. The 10 nearest neighbors in that cluster were found for that game and were then sorted from closest to farthest. Those were the recommendations using a clustering algorithm. In order to find out the nearest neighbors, the euclidean distance (2) from each point in that cluster to the game is calculated, and the ones with the shortest distance are picked.

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \qquad (2)$$

3.2.1 Kmeans

In order to find out the best value for *k*, the number of cluster centers to initialize, the elbow method, and the silhouette coefficient method were both used with and without the weighted score as the weight for each game. For the elbow method, the sum of square distances of each point to its assigned cluster center is calculated for a range of values of *k* (in this case, 1-15). Those values are then plotted on a graph and the point of inflection on the curve (that looks like an elbow) is the best value for *k* (this was also tested using an online library called kneed that mathematically found the elbow of the curve). As for the silhouette score, the silhouette coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is (b - a) / max(a, b) (scikit-learn-developers, 2021). The silhouette coefficients were then plotted against *k* to see which *k* has the highest silhouette coefficient. As seen in Fig.7, the best value of *k* for this dataset was 4.

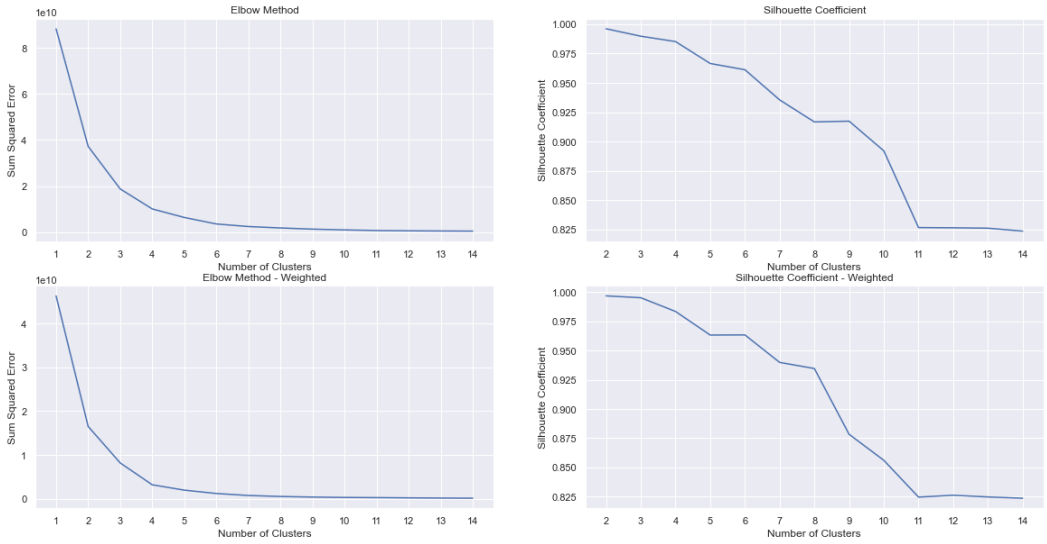

Fig.7 - finding the best value of *k*

However, after training the model, the results obtained did not look very satisfactory. For example, 'Gang Beasts', a multiplayer party game, had 'Unreal 2: The Awakening', a single-player, sci-fi, FPS game as its best recommendation when trained with and without the weights. Therefore, instead of using only 4 clusters, 22 clusters were used (the number of genres) instead, in an attempt to see if it could better cluster the data.

Although this did yield slightly better results than having only 4 clusters, the recommendations were still pretty mixed. For example, 'Gang Beasts' still didn't have any good recommendations but 'Counter-Strike' had a couple of good recommendations, such as 'Team Fortress 2', in the list of recommendations.

3.2.2 Hierarchical Agglomerative Clustering (HAC)

Similar to Kmeans clustering, the best value for *k* must be determined first. This was done by plotting a dendrogram (Fig.8) that shows how the clusters were joined together and deciding on a threshold for the maximum number of clusters. The number of clusters that were chosen was 13. This time, the euclidean distance (2) was used as the distance metric and single-linkage (3) was used to merge the clusters together.

$$sim(c_i, c_j) = \max_{x \in c_i, y \in c_j} sim(x, y) \qquad (3)$$

Unfortunately, the results using HAC were almost exactly the same as the results obtained from Kmeans and were therefore not very accurate.

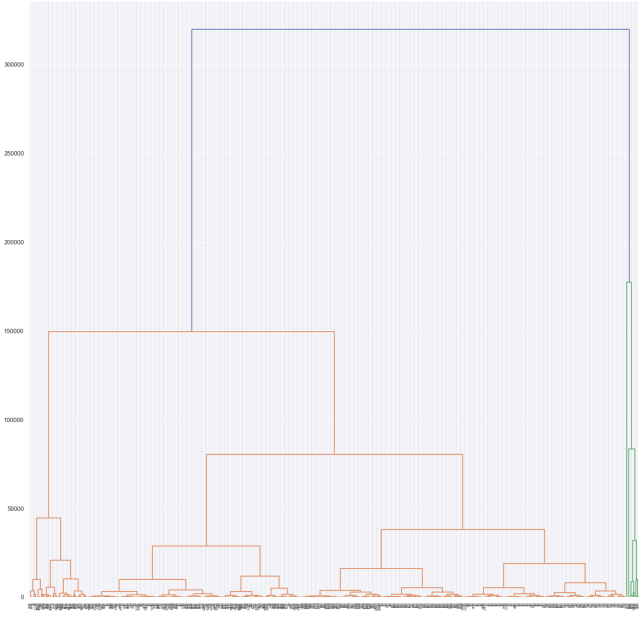

Fig.8 - Dendrogram created after applying HAC to the dataset

Note: For this section, refer to 2_clustering_sol.ipynb for more details on the clustering algorithm solution implementation

3.3 Content-Based Filtering Method

For this method, the number of times a word appears in a group of words (we are going to call that a word soup) will be used to measure how similar a game is to another game. For example, if 2 games both had the words 'action' and 'RPG' in their word soup, they could be similar to each other. However, if a 3rd game had the words 'action' and 'horror', this game would be the least similar out of the 3 games. Therefore, after choosing which features to add to the word soup, the word counts for each game will be calculated using Count Vectorization (by creating vectors that have the same size as our vocabulary in our word soup) and the similarity between each game will be calculated and sorted from most to least similar.

The metric that will be used to calculate the similarity between 2 games will be cosine similarity (4), which measures the cosine of the angle between 2 vectors projected in a multi-dimensional space. In this case, our 2 vectors will be the vectors created from count vectorization for 2 games.

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \quad (4)$$

In this solution, the main 3 features that will be used are genres, categories, and steamspy_tags. However, since steamspy_tags gives a more complete description of the game as compared to genres and categories, this feature will be added twice to the word soup, therefore increasing the weight of those tags. Also, the short description of the games was also added as one of the features since games that are pretty similar will have similar keywords in their description (for example, a game about 'saving the world' would both most likely have those words in its description). Therefore the 2 combinations of word soups are:

```
[genres + categories + steamspy_tagsx2] and [genres + categories + steamspy_tagsx2 + short description]
```

Both of the word soups yielded pretty similar results, with only the order of the games being different for each word soup (for example, 'Stick Fight: The Game' had 'Pummel Party' as 2nd for the one without the short description and 3rd for the one with the short description) and the similarity score being generally lower for the word soup with the short description. This is as expected since we are adding more words to the soup, increasing the dimensionality of the word vector, which would increase the distance between vectors. Furthermore, games that had sequels always appeared for both word soups, which is to be expected since they would have almost identical words in their word soups.

Note: For this section, refer to 2_count_vectorization_sol.ipynb for more details on the content-based filtering implementation

4. Results of the Algorithms

4.1 Success of each algorithm

4.1.1 Clustering Algorithms

As mentioned above, the results for the clustering method were pretty poor. After testing out several games and checking the recommendations for each of them, the games that were being recommended were not similar for most games. Furthermore, none of the games that steam recommends were the same as the ones that were recommended by the clustering method.

4.1.2 Content-Based Filtering

Unlike the clustering method, the results for the content-based filtering method were pretty good. Most of the games that were being recommended were the same types of games and some of the games were similar to the games being recommended on steam (for example, 'Gang Beasts' recommended 'Pummel Party' on steam and with the content-based-filtering method).

However, when comparing the results of the word soups for each game, I noticed that the sequels were generally listed higher for the one with the short description in the soup (for example, 'Overcooked! 2' was first for the word soup with the short description whereas it was 2nd for the one without the short description for the game 'Overcooked'). This is expected since sequels would generally have similar words in their description (for example the same character names). However, this might not always be the best result since someone who enjoyed a game would generally already know that they would enjoy the sequels for that game and it would therefore not be very useful to recommend that game. Thus, the word soup that gives a better result is the one without the short description.

4.2 Summary of Testing Results

After implementing both methods, it is quite clear that the content-based filtering method is much better than the clustering method and therefore, only the content-based filtering method will be used to implement a hybrid solution that makes use of both the weighted review scores and the algorithms that were already implemented.

5. Implementing a Hybrid Solution

5.1 The Hybrid Model

In order to have more accurate recommendations that are not solely based on the contents of the game, but also whether people liked the game or not, the weighted review scores that were calculated in section 2.2.2 will be used to add weights to each similarity score.

The first equation that was used to find a hybrid score was (5). Since we don't want multiplication by 0, a constant was added to each score (in this case it was 1). Furthermore, since the similarity score is more important than the weighted review score, this value was squared.

$$(1 + Similarity\ Score)^2\ x\ (Weighted\ Score + 1) \tag{5}$$

The second equation that was used was (6). This time, instead of squaring the similarity score, a weight was added to both values and they were then added together to get the hybrid score. For this solution, the weights that were chosen for both scores were 0.8 and 0.2 respectively.

$$0.8(1 + Similarity\ Score) + 0.2(1 + Weighted\ Score) \tag{6}$$

Therefore, in order to get the recommendations, the first 100 games with the highest similarity scores were taken and the hybrid scores were calculated for each. Those games were then sorted by their hybrid scores and the top 10 games were the recommendations for the game.

5.2 Results of Hybrid Model

After testing out this model with several games and for both equations by going over the games recommended and assessing whether it was a good recommendation or not, the results for both equations showed pretty similar results and were both good results. Comparing the results to the recommendations on Steam were also pretty similar for some games.

However, unlike (5), (6) could be further optimized by adjusting the weights associated with each score to give a more accurate result since these initial weights were arbitrarily chosen without too much research and analysis being done on those values. Therefore, this equation is a better equation that can be applied in the future.

Note: For this section, refer to 2_count_vectorization_sol.ipynb (2nd half) for more details on the hybrid solution

6. Conclusion

After a thorough analysis of the data to find the features that were the most important, and testing and experimenting with different solutions, and combining the best ones to produce a hybrid recommendation system, a game recommendation system was created that was able to give fairly good recommendations. In the end, the recommendation system made use of count vectorization on the genres, categories, and steamspy_tags features and calculated the similarity scores using (4) for each game. Then, those scores were combined with the weighted review scores, derived using (1), by using (6) to create a hybrid score, which was used to find the best games to recommend.

The recommendations that were given were pretty similar to the game in terms of genre and category, which is what is desired when giving a recommendation. Furthermore, this recommendation system is not biased to games that are very popular. For example, 'Counter-Strike' was ranked 8th in the list of recommendations for 'Team Fortress 2'. This could also be a good way to explore games that might be less popular but still enjoyable based on the game that was input.

However, this solution is far from perfect. Although an attempt was made to try and minimize the number of sequels that appeared in the list of recommendations, they were still present in the recommendations. Also, those recommendations are not personalized. Anyone who inputs the same game would get the same recommendations which might not be the best outcome if we are trying to find the best recommendations for everyone.

7. Going Forward

Therefore, if this project were to be further developed, there are several things that can be improved:

- The recommendations could be further improved by taking into account the games that a user has played and those games could be taken into consideration when creating the hybrid recommendation system.
- This project focussed only on steam data and the reviews, however, more features can be taken into consideration when making recommendations. For example, the popularity of the game on online platforms such as Twitch and Youtube. By adding those features into the mix, this recommendation system could be further improved by giving results that are currently trending and relevant today.
- Steamspy_tags could have been used instead of genres in the clustering method, alongside a dimensionality reduction technique, such as PCA, to reduce the time taken to fit the data.
- Further investigation could have been done in order to find out why the clustering method gave poor results so that this method could be improved in the future so that it can be used in the hybrid model as well.
- Finally, more research and analysis could have been done on what would be the best way to test the results since they were tested manually this time and there was no real metric that defined how good or bad the recommendations were.

References

I. Ahmed, “Getting started with a movie recommendation system,” *Getting Started with a Movie Recommendation System*, 16-May-2020. [Online]. Available: https://www.kaggle.com/ibtesama/getting-started-with-a-movie-recommendation-system. [Accessed: 08-Dec-2021].

SteamDB, “Introducing steam database's new rating algorithm · steamdb,” *Introducing Steam Database's new rating algorithm*, 27-Sep-2017. [Online]. Available: https://steamdb.info/blog/steamdb-rating/. [Accessed: 08-Dec-2021].

scikit-learn-developers, “Sklearn.metrics.silhouette_score,” *sklearn.metrics.silhouette_score*, 2007. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html. [Accessed: 08-Dec-2021].