Content Based Game Recommendation System with Steam API: Association using Cosine Similarity Algorithm

Richard Andrei B. Suñga
BSCS 3-C
Technological University of the Philippines
Malabon, Philippines
richardandrei.sunga@tup.edu.ph

Ella Monique T. Gonzales
BSCS 3-C
Technological University of the Philippines
Pangasinan, Philippines
ellamonique.gonzales@tup.edu.ph

Rick Gavin A. Dela Cruz
BSCS 3-C
Technological University of the Philippines
Cavite, Philippines
rickgavin.delacruz@tup.edu.ph

ABSTRACT

The project aims to create a python program that aids and guides people who seek to find new games that are related to the ones they play. The project will mostly target the young demographic and the data that is needed will be collected from Steam API. The csv files that will be retrieved will be used for its genre contents and so, if that condition is met (the csv file has the game and genre columns, we can use it), which means the inflow of data is flexible. As for the algorithm, we chose Cosine Similarity Algorithm for the similarity between two inner product space vectors measured by this algorithm. It checks whether two vectors are roughly pointing in the same direction by measuring the cosine of the angle between them. In text analysis, it is frequently used to assess document similarity, this is where the genres of the games the user plays enter.

Keywords: Cosine Similarity, Algorithm, Vector, Text Analysis, Steam, Python.

I. INTRODUCTION

Imagine knowing whether a game is suitable for your tastes and genre. Being able to recommend a person with a prediction for the similar games they can play would significantly help people find new games they can freshly enjoy and immerse themselves in. In accordance with user needs, we will be analyzing these data to some stretch and determine its extent. The analysts will strictly abide by the procedure and will follow the logics behind it without bias and personal decisions

All of the members of this team, like many other young people, are interested in video games, particularly computer games. As a result, the purpose of the project described in this project is to create a computer game recommender system.

These days, recommender systems are routinely utilized to suggest goods that consumers might enjoy. Collaborative filtering, content-based filtering, and hybrid recommender systems are the three basic types of recommender systems.

The program we have made uses the description of the items in order to recommend items similar to what a user like. A recommender system's data can be explicit, such as reviews or ratings, or implicit, such as behavior and events such as order history, search logs, clicks, and so on.

Implicit data is more difficult to process since it's difficult to tell which information is important and which isn't, but it's also easier to obtain than explicit data because the user doesn't have to do anything other than use the website or app as usual.

This project's main goal is to create a recommender system that will suggest games to users based on their likes and gaming patterns. In order to put in place, the greatest recommender system we can, we selected an algorithm that best suits text analysis.

We used data from Steam, one of the largest video game digital distribution services for computer games, for this study. We'll be working with a filtered game dataset.

II. REVIEW OF RELATED LITERATURE AND STUDIES

Text Mining

In [4], the practice of extracting high-quality information from text, also known as text data mining, is akin to text analytics. It entails "automatically extracting information from various written resources" and "discovering new, previously unknown information via computer."

Websites, books, emails, reviews, and articles are examples of written resources. Statistical pattern learning is commonly used to provide high-quality information by generating patterns and trends. We can distinguish three main views of text mining, information extraction, data mining, and a KDD (Knowledge Discovery in Databases) approach [3].

The process of structuring the input text (usually parsing, with the addition of some derived linguistic features and the removal of others, and subsequent insertion into a database), deriving patterns within the structured data, and finally evaluating and interpreting the output is what text mining is all about [1]. In text mining, 'high quality' usually refers to a combination of relevance, uniqueness, and interest. Text classification, text clustering, concept/entity extraction, generation of granular taxonomies, sentiment analysis, document summarizing, and entity relation modeling are all common text mining activities (i.e., learning relations between named entities).

Information retrieval, lexical analysis, pattern recognition, tagging/annotation, information extraction, data mining techniques such as link and association analysis, visualization, and predictive analytics are all part of text analysis.

Cosine Similarity

In the findings of [8], cosine similarity is a measure of similarity between two numerical sequences used in data analysis. The cosine similarity is defined as the cosine of the angle between the sequences, which is defined as the dot product of the vectors divided by the product of their lengths. As a result, the cosine similarity is determined by the angle of the vectors rather than their magnitudes.

Each word is assigned a separate coordinate in information retrieval and text mining, and a document is represented by a vector of the number of occurrences of each word in the document. Cosine similarity is a useful metric for determining how similar two publications are likely to be in terms of topic content, regardless of their length/irrespective of the document size.

How does cosine similarity differ from the number of common terms as a metric of similarity? The cosine similarity captures the orientation (the angle) of the texts rather than the magnitude when plotted on a multi-dimensional space where each dimension corresponds to a word in the document [9]. Calculate the Euclidean distance instead if you want the magnitude. The advantage of cosine similarity is that even if the Euclidean distance between two comparable texts is large (for example, the word 'cricket' appears 50 times in one document and 10 times in another), they can nevertheless have a lesser angle between them. The greater the resemblance, the smaller the angle.

Recommender Systems

The idea behind this recommender system is to connect users to material or things that they have enjoyed or purchased. Users' and products' characteristics are significant in this case. For example, we analyze features like genre, developers, popular tags, publisher, and others to detect similarities between games while making game suggestions.

Recommendation systems are based on the content's or the content's users' resemblance to each other. The resemblance of two objects can be assessed in a variety of ways [7]. This similarity matrix is used by the recommendation systems to suggest the next most similar product to the user. We will create a machine learning system in this article that will propose games depending on the user's favorite games. Cosine Similarity would be the foundation of this machine learning model.

III. METHODOLOGY

The research design of the paper is descriptive quantitative. The subject of the study is about game recommendation using the cosine similarity algorithm which is said to be effective with text analysis. The researchers utilized the steam api and procured a csv file for the raw data which was obtained from Kaggle.

An overall 5000+ games is used for the cosine algorithm model and another 40k reviewed games is used for the recommendation

	A	B	C	D
1	name	percentage	review_qualification	all_reviews
2	DOOM	92	Very Positive	Very Positive,(42,550),- 92% of the 42,550 user reviews for this game are positive.
3	PLAYERUNKNOWN'S BAT	49	Mixed	Mixed,(836,608),- 49% of the 836,608 user reviews for this game are positive.
4	BATTLETECH	71	Mostly Positive	Mostly Positive,(7,030),- 71% of the 7,030 user reviews for this game are positive.
5	DayZ	61	Mixed	Mixed,(167,115),- 61% of the 167,115 user reviews for this game are positive.
6	EVE Online	74	Mostly Positive	Mostly Positive,(11,481),- 74% of the 11,481 user reviews for this game are positive.
7	Devil May Cry 5	92	Very Positive	Very Positive,(9,645),- 92% of the 9,645 user reviews for this game are positive.
8	Human: Fall Flat	91	Very Positive	Very Positive,(23,763),- 91% of the 23,763 user reviews for this game are positive.
9	They Are Billions	85	Very Positive	Very Positive,(12,127),- 85% of the 12,127 user reviews for this game are positive.
10	Warhammer: Chaosbane	44	Mixed	Mixed,(904),- 44% of the 904 user reviews for this game are positive.
11	For The King	83	Very Positive	Very Positive,(4,600),- 83% of the 4,600 user reviews for this game are positive.
12	Danganronpa V3: Killing H	84	Very Positive	Very Positive,(3,547),- 84% of the 3,547 user reviews for this game are positive.
13	TERA	78	Mostly Positive	Mostly Positive,(14,184),- 78% of the 14,184 user reviews for this game are positive.
14	Call of DutyÃ®: Modern V	51	Mixed	Mixed,(1,118),- 51% of the 1,118 user reviews for this game are positive.
15	Stonehearth	75	Mostly Positive	Mostly Positive,(5,484),- 75% of the 5,484 user reviews for this game are positive.
16	Clone Drone in the Dange	94	Very Positive	Very Positive,(1,901),- 94% of the 1,901 user reviews for this game are positive.
17	GOD EATER 3	77	Mostly Positive	Mostly Positive,(1,945),- 77% of the 1,945 user reviews for this game are positive.
18	War Robots	44	Mixed	Mixed,(1,797),- 44% of the 1,797 user reviews for this game are positive.
19	Halo Wars: Definitive Edi	88	Very Positive	Very Positive,(2,442),- 88% of the 2,442 user reviews for this game are positive.
20	Call of DutyÃ®: Black Ops	84	Very Positive	Very Positive,(4,190),- 84% of the 4,190 user reviews for this game are positive.
21	Phoenix Wright: Ace Attc	97	Very Positive	Very Positive,(382),- 97% of the 382 user reviews for this game are positive.
22	Team Sonic Racingâ„¢	70	Mostly Positive	Mostly Positive,(487),- 70% of the 487 user reviews for this game are positive.
23	Titan Quest Anniversary E	90	Very Positive	Very Positive,(9,757),- 90% of the 9,757 user reviews for this game are positive.

Figure 1. Screenshot of sample data from user reviews

The content-based recommender system gives recommendation-based on the similarity between the game a user already has and other games.

In order to build the recommender system, we need to prepare the data and build the algorithm. In order to do so, we first need to preprocess the game dataset described in the dataset section with all the useful information to give as input to the recommender algorithm, formatted in a simpler way.

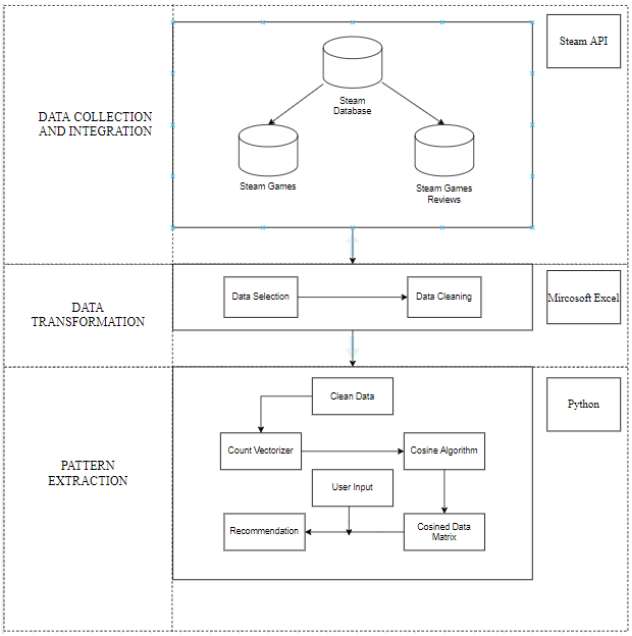


Figure 2. The Framework of the Predictive Model

Based on figure 1, in the Data Collection and Integration Stage, the data comes from the Steam Database which produces two datasets, the Steam Games and the Steam Games Review. After that, only the relevant data are selected and retained such as the developers, price of the game, number of reviews, etc., to improve the quality of the data for more accurate recommendations. And then in the Implementation Stage,

the cleaned data is transformed to token counts which is a vector data type, to be used for computing the similarity of the games through the cosine algorithm.

The percentage from the reviews are extracted since they are used. Then, we implement a function used to give game recommendations similar to another game. Lastly, we produce recommendations for all user based on the games users already possess.

To prepare the data for the content-based recommender, we start by selecting the information we think would be the most useful to find similar games. We read the useful columns from the game dataset by using the following code:

```
dataGames = read_csv(locationGamesFile,
                     usecols=["name", "genre", "game_details",
"popular_tags", "publisher", "developer"])
```

Figure 2. Code for gathering useful columns in dataset

We decided to only keep relevant games ranging to only 5000 in the game dataset simply because there are certain games that are not significant enough to consider recommending to the user. Also, our algorithm for creating a matrix for the user would take too much memory if we included all the games that are in Steam.

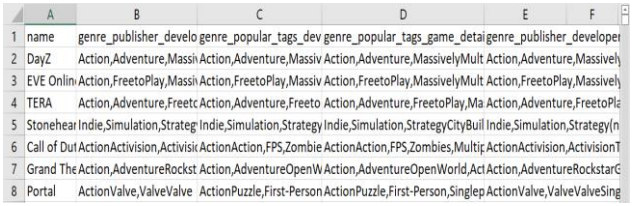
We eliminate the spaces from the useful columns we choose to use with the new smaller game dataset. We avoid a match between 'Steam Achievement' and 'Steam Cloud', for example, because they both contain the word 'Steam.' The names 'SteamCloud' and 'SteamAchievement' will now be unique to each of them.

Finally, using the code below, we create some new columns by merging numerous columns to obtain the optimal combination of information to offer us the best potential recommendation system.

```
# create some column containing a mix of different
information
usedGames["genre_publisher_developer"] =
usedGames['genre'] + \
    usedGames['publisher'] + usedGames['developer']
usedGames["genre_popular_tags_developer"] =
usedGames['genre'] + \
    usedGames['popular_tags'] + usedGames['developer']
usedGames["genre_popular_tags_game_details"] =
usedGames['genre'] + \
    usedGames['popular_tags'] +
usedGames['game_details']
usedGames["genre_publisher_developer_game_details"]
= usedGames['genre'] + \
    usedGames['publisher'] + usedGames['developer'] +
usedGames['game_details']

usedGames.drop_duplicates("name")
```

Figure 3. Function used to create mixture of different informations



	A	B	C	D	E	F
1	name	genre_publisher_develo	genre_popular_tags_dev	genre_popular_tags_game_detai	genre_publisher_developer	
2	DayZ	Action,Adventure,Massi	Action,Adventure,Massiv	Action,Adventure,MassivelyMult	Action,Adventure,Massively	
3	EVE Onlin	Action,FreetoPlay,Massi	Action,FreetoPlay,Massiv	Action,FreetoPlay,MassivelyMult	Action,FreetoPlay,Massively	
4	TERA	Action,Adventure,Freetc	Action,Adventure,Freeto	Action,Adventure,FreetoPlay,Ma	Action,Adventure,FreetoPla	
5	Stonehear	Indie,Simulation,Strateg	Indie,Simulation,Strategy	Indie,Simulation,StrategyCityBuil	Indie,Simulation,Strategy(n	
6	Call of Dut	ActionActivision,Activisi	ActionAction,FPS,Zombie	ActionAction,FPS,Zombies,Multip	ActionActivision,ActivisionT	
7	Grand The	Action,AdventureRockst	Action,AdventureOpenW	Action,AdventureOpenWorld,Act	Action,AdventureRockstarG	
8	Portal	ActionValve,ValveValve	ActionPuzzle,First-Person	ActionPuzzle,First-Person,Singlep	ActionValve,ValveValveSing	

Figure 4. Screenshot of result from merging different columns

With the code below, we can create a cosine similarity matrix for the recommender system. To begin, we generate a frequency matrix for each game and each word in the chosen column (column name). The cosine similarity matrix is then created using the frequency matrix.

```
count = CountVectorizer(stop_words='english')
count_matrix =
count.fit_transform(dataGames['popular_tags'])
cosine_sim_matrix = cosine_similarity(
    count_matrix, count_matrix)
```

Figure 5. Code for transforming the column and computing the cosine matrix

$$\text{Cosine similarity}(|X,Y|) = \frac{x \cdot y}{||x||\ ||y||} \qquad (4.12)$$

where $x \cdot y$ is the dot product of the x and y vectors with, for this example,

$$x \cdot y = \sum_{i=1}^{n} x_i y_i \text{ and } ||\mathrm{x}|| = \sqrt{x \cdot x}$$

$$x \cdot y = \sqrt{1\times5+2\times0+0\times0+0\times6+3\times7+4\times0+0\times0}$$

$$||x|| = \sqrt{1\times1+2\times2+0\times0+0\times0+3\times3+4\times4+0\times0}$$

$$||y|| = \sqrt{5\times5+0\times0+0\times0+6\times6+7\times7+0\times0+0\times0}$$

$$\text{Cosine similarity}(|x \cdot y|) = \frac{x \cdot y}{||x||\ ||y||} = \frac{5.1}{5.5\times10.5} = 0.08$$

Figure 6. Formula of cosine similarity

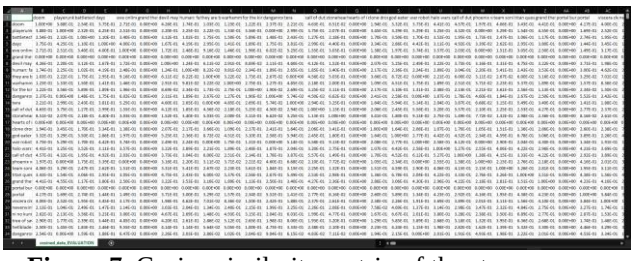


Figure 7. Cosine similarity matrix of the steam games

	A	B	C	D	E	F
1		doom	playerunk	battletech	dayz	eve onli
2	doom	1.00E+00	3.88E-01	2.54E-01	3.75E-01	2.71E-0
3	playerunk	3.88E-01	1.00E+00	2.32E-01	4.25E-01	2.51E-0
4	battletech	2.54E-01	2.32E-01	1.00E+00	1.10E-01	3.40E-0
5	dayz	3.75E-01	4.25E-01	1.10E-01	1.00E+00	4.00E-0
6	eve online	2.71E-01	2.51E-01	3.40E-01	4.00E-01	1.00E+0

Figure 8. Sample values of the cosine similarity matrix of the steam games

For games with a score of 1, it means that it was compared to itself, resulting in this format. As one can see, the format is in the likeness of a multiplication table.

In the code below, the variable 'indices' is used to calculate the cosine similarity matrix's index for each game. It's a map that uses the name as the key.

```
# Construct a reverse map of indices and game names
        dataReviews['name'] =
dataReviews['name'].str.lower()
        indices = Series(
            dataGames.index,
index=dataGames['name']).drop_duplicates()
```

Figure 9. Code for lowering the characters of the names and dropping duplicates

The function 'get recommendations' is used to produce recommendations for each game. The function takes a game title as a string and the cosine similarity matrix established before as inputs. The result is a list of game titles that are recommended in order of resemblance.

```
def get_recommendations(title):
            global cosined_data
            global listDesc

            if title not in listGames:
                return []  # for blank

            # Get the index of the game that matches the
name
            idx = indices[title]

            # if there's 2 games or more with same name
(game RUSH)
            if type(idx) is Series:
                return []  # for duplicate

            # Get the pairwise similarity scores of all games
with that game
            sim_scores = list(enumerate(cosined_data[idx]))

            # Sort the games based on the similarity scores
            sim_scores = sorted(
                sim_scores, key=lambda x: x[1],
reverse=True)

            # Get the scores of the most similar games
            # (not the first one because this games as a score
of 1 (perfect score) similarity with itself)
            x = int(n_recommendation/len(userGames))

            sim_scores = sim_scores[1:x + 1]

            # VECTOR DATA OF N
RECOMMENDATIONS

            df = pd.DataFrame(data={"vectors":
sim_scores})
            df.to_csv("./closest_vectors.csv",
                      sep=',', index=False)

            # Get the games indices
            movie_indices = [i[0] for i in sim_scores]

            # Return the top most similar games
            dataGames['developer'] = (
                dataGames['developer'] + " " +
dataGames['original_price'].astype(str))

            listDesc =
dataGames['developer'].iloc[movie_indices].tolist()

            #
print(dataGames['developer'].iloc[movie_indices])

            return
dataGames['name'].iloc[movie_indices].tolist()
```

Figure 10. Function for getting suggestions for each user games

The variable 'listGames' contains a list of all games found in both the user dataset and the game dataset. We check that 'idx' is not a Series; this can happen when two games with the same title (for example, two games with the name 'RUSH' in our dataset). Then, using the cosine similarity matrix, we calculate the similarity score for each recommended game and rank them from most similar to least similar.

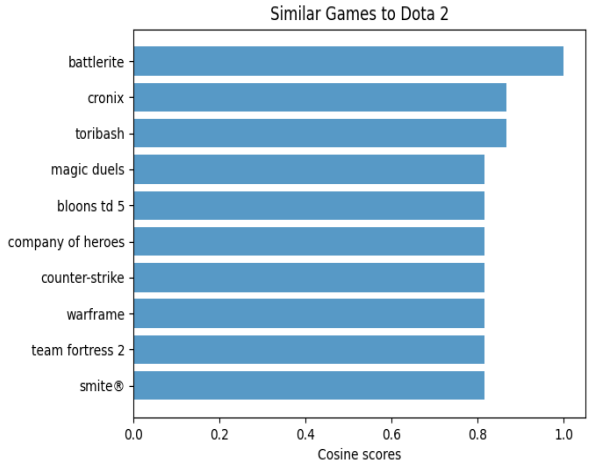


Figure 11. Bar graph of similar games to Dota 2

As shown in Figure 12, the scores of the games returned by the cosine algorithm is ranging from 0 to 1 with 1 being the most similar. In this example, the top 10 similar games to Dota 2 are Battlerite, Cronix, Toribash, etc., with Battlerite being the most similar having the score of 1. The process is repeated to other games if the user entered more than 1 input to ensure that there's at least 1 most similar game per each input.

IV. IMPLEMENTATION

To get recommendations for each user, we utilize the 'make recommendation for user' function, which is shown below. This function combines the recommendations generated by the 'get recommendations' function for each game, maintaining the recommendations with the best reviews (extracted from the game dataset). This function requires the following inputs: a list of

suggestions for the specified user (the previous 'get recommendations' function is applied to all the games a user entered, and the results are provided as a list with all the recommendations).

```
def make_recommendation_for_user(user_id, game_list, game_user_have):
        if type(game_list) is not list or len(game_list) == 0:
            # return empty one
            return DataFrame(data=[[user_id] + [""] * n_recommendation], columns=col_names)

        # get reviews of game recommendation, remove the games the user already has and order them by reviews
        recommendation_reviews = dataReviews.loc[dataReviews['name'].isin(
            game_list)]
        recommendation_reviews = recommendation_reviews.loc[~recommendation_reviews['name'].isin(
            game_user_have)]
        recommendation_reviews = recommendation_reviews.sort_values(
            by="percentage_positive_review", ascending=False)

        global recommendedGames
        recommendedGames = []
        i = 0
        for games in recommendation_reviews["name"]:
            recommendedGames.insert(i, games)
            i = i+1
        if len(recommendation_reviews.index) < n_recommendation:
            return DataFrame(data=[[user_id] + recommendation_reviews["name"].tolist() +
                                   [""] * (n_recommendation - len(recommendation_reviews.index))],
                             columns=col_names)
        else:
            return DataFrame(data=[[user_id] + recommendation_reviews["name"].tolist()[0:n_recommendation]], columns=col_names)
```

Figure 12.. Function for getting the recommendation from suggested games

A DataFrame without suggestions is returned if the list of recommendations is empty (which could happen if none of the games the user already owns are in the game dataset) or invalid. If the recommendation list is legitimate, the returned DataFrame contains the recommended game titles along with their reviews (percentage of positive review). Games that the user already owns are removed from the list (no need to recommend a game the user already purchased). Using reviews to sort the suggestions guarantees that all users think the recommended games are decent in general.

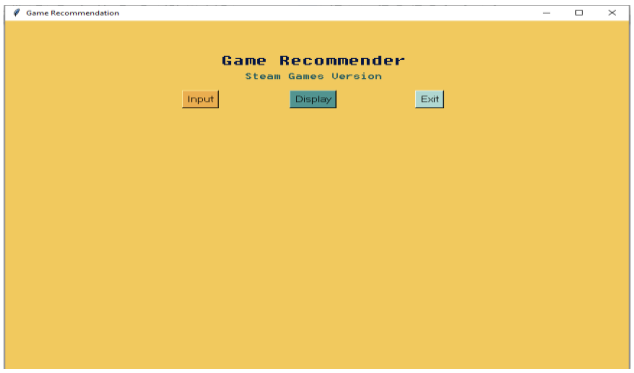


Figure 13. Screenshot of initial interface of the application

This is the initial interface of the program we created in python. It is very simple, before the user can display recommended games, the user must press the input button first and a window will appear.

There is a dropdown button at the top of the window which will indicate the number of inputs the user can choose from.

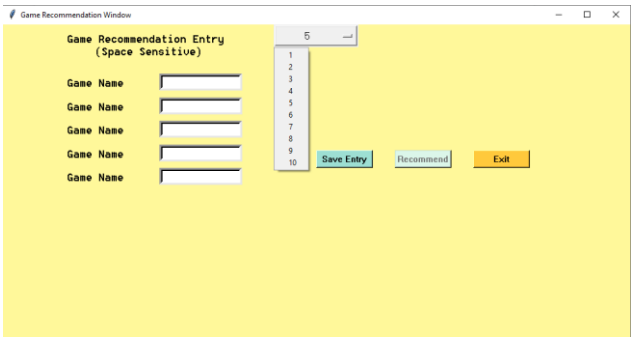


Figure 14. Screenshot of UI for user inputs

In this example, we chose 5 inputs. The user must then enter 5 valid inputs (non-case sensitive) but is highly sensitive to spaces. The recommend button is locked until the user presses the button that says “Save Entry”.

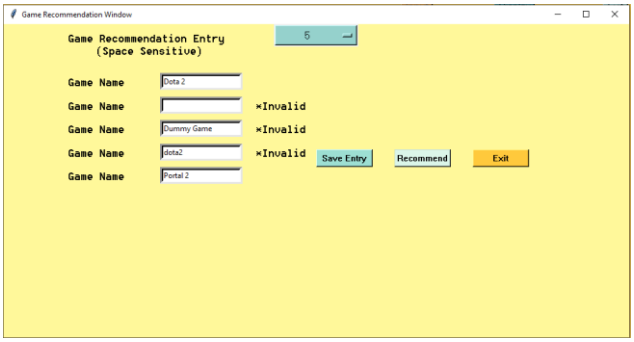


Figure 15. Screenshot of scenario when user entered invalid inputs

As one can see, the user will be notified if their input is invalid (has invalid spaces, duplicates or the game not being in the dataset used).

Figure 16. Screenshot of application showing the recommended games

When the inputs are valid, after pressing the recommend button, the user can now press the display button which will recommend a number of games based on the inputs the user put in the input window that was closed. These games are the games from the cosined data csv. The program will return the games which has the highest/top values in terms of similarity.

V. RESULTS AND DISCUSSION

The similarity between two inner product space vectors is measured by cosine similarity [5]. It checks whether two vectors are roughly pointing in the same direction by measuring the cosine of the angle between them. In text analysis, it is frequently used to assess document similarity.

Thousands of characteristics can be used to characterize a document, each of which records the frequency of a specific word (such as a keyword) or phrase in the document. As a result, a term-frequency vector is used to represent each document as an object.

For example, we see a document containing five instances of the word FPS, while GORE does not occur at all. The word GORE is absent from the entire document, as indicated by a count value of 0 and FPS has a value of 5. Such data can be highly asymmetric. If the document has a set number of words to be counted which we can name as N, then we can convert the values of the occurrences of these words and make an interpretation of it using the cosine similarity algorithm.

The cosine similarity is a number between 0 and 1 and is commonly used in plagiarism detection [6]. The number of unique words in the documents in issue is converted to a vector, where n is the number of words in the documents in question.

The number of times a word appears in the document is associated with each element of the vector, and the value represents the number of times that word appears in the document. The cosine similarity between the two documents is then calculated.

If both documents employ a lot of the same words, there's a chance you'll get a false positive. This can cause the calculation to be skewed, giving the impression that the two papers are linked when they aren't. In our case, that wouldn't be a problem since we would like to associate the selected column for reference in getting the results we want. So if the results return, the top most similar results will have a cosine similarity value close to 1, 1 being disregarded since it is the exact data we were comparing to other data.

It is possible to construct a more informational vector with the number of occurrences in the document, instead of just 1 and 0 [10]. Document datasets are usually long vectors with thousands of variables or attributes. For simplicity, consider the example of the vectors with X (1,2,0,0,3,4,0) and Y (5,0,0,6,7,0,0). The cosine similarity measure for two data points is given by:

VI. CONCLUSION AND RECOMMENDATION

To our conclusion, the cosine similarity algorithm proved to be useful in associating documents, in our case, csv files— one user data set from the input, one game data set from the steam api csv and one csv from game reviews of that api. In testing and pre-processing our first dataset that contains 40,000 games, the powerful hardware we had was still not enough to cosine all of the data so we had to perform extensive filtering techniques that employed fetching only the games that has high reviews.

We further narrowed it down to less than 6000 games to get relevant games and also pre-process it in a much more optimal time. Due to the nature of the project, we could say that this recommendation system has a potential to be a base program for a more improved recommendation application if enough time was given.

Many improvements can be done if sufficient utilities can also be accessed, this includes unrestricted knowledge of programming that involves actual creation of applications.

We conclude that the application has areas that are lacking in terms of functionality and design but in light of that, the recommendation system can be used by users who are in need of crude recommendations or just simply want to have a pocket recommender at their disposal.

If the chance persists, we will make an integration of the recommender that will be a hybrid recommender as our recent investigations prove that it is more plausible to use multiple algorithms in recommending products so as to have many evaluation metrics. Particularly, we would like to put apriori algorithm that is also under the association rule for collaborative filtering.

REFERENCES

[1] Feldman, R. and Sanger, J. (2007). The text mining handbook. Cambridge University Press. New York. https://www.cambridge.org/core/books/text-mining-handbook/0634B1DF14259CB43FCCF28972AE4382

[2] Geordan J., Audrey G., Doo H. (2019). Recommendation System for Steam Game Store: An overview of recommender systems. https://github.com/AudreyGermain/Game-Recommendation-System

[3] Hotho, A., Nürnberger and Paaß, G. (2005). Statistical Pattern Learning, "A brief survey of text mining". In Ldv Forum, Vol. 20(1), p. 19-62

[4] Marty Hearst (2015) What is Text Mining? https://people.ischool.berkeley.edu/~hearst/text-mining.html#:~:text=In%20text%20mining%2C%20the%20goal,interesting%20patterns%20from%20large%20databases.

[5] Jiawei Han and Jian Pei(2012), the similarity between two inner product space vectors is measured by cosine similarity https://www.sciencedirect.com/topics/computer-science/cosine-similarity

[6] Leigh Metcalf and William Casey (2016). The cosine similarity is a number between 0 and 1 and is commonly used in plagiarism detection. https://www.sciencedirect.com/topics/computer-science/c

[7] Mahnoor Javed (2020). Using Cosine Similarity to Build a Movie Recommendation System https://towardsdatascience.com/using-cosine-similarity-to-build-a-movie-recommendation-system-ae7f20842599

[8] Singhal, Amit (2001). Cosine similarity is a measure of similarity between two numerical sequences. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 24 (4): 35–43.

[9] Selva Prabhakaran (2018). The cosine similarity captures the orientation (the angle) of the texts https://www.machinelearningplus.com/nlp/cosine-similarity/cosine-similarity

[10] Vijay Kotu and Bala Deshpande (2019). Data Science Concepts and Practice (Second Edition). https://www.sciencedirect.com/topics/computer-science/document-vector