CMPT 353 Project Report

Prepared For:

Greg Baker and TAs

By:

Dingshuo Yang (301356845)

Ali Nikan (301538298)

Mohammad Parsaei (301449545)

August 4, 2023

1. Introduction

Video games have been getting more popular day by day, and they could even reshape the way we interact with the world. With the rise of platforms like Steam, there is a tremendous amount of data available about these games. For this project, we found two datasets, one is related to Steam Games and their different data, and the other one is a list of games with their playtimes. With these datasets, we decided to create a sophisticated machine learning application that offers two primary functionalities: a game recommendation system and a game popularity prediction model.

Our game recommendation system helps users discover new games they're likely to enjoy, based on their input. By inputting a game they like, users receive a list of top 5 recommended games that share similar features such as popular tags, details, and genres. The model operates on a Nearest Neighbors algorithm, trained on these features to find the most similar games. Each recommendation is accompanied by an explanation detailing the common features between the input game and the recommended games.

The second functionality, game popularity prediction is a well trained machine learning model that can predict if a video game will likely be popular given some information that is available during its development. This includes comprehensive data preprocessing and training, refinement through iterations of study. The process also involves detailed analysis of features like genre, price, and release date... etc, reflecting the multifaceted nature of game success.

Our application offers valuable insights for game developers, marketers, and gamers by providing an understanding of what attributes might contribute to a game's popularity.

In the following sections of this report, we will delve into the technical details of our project, discussing our data, analysis, results, limitations and potential future study.

2. Data Collection

For the purpose of ... , we collected two datasets. The first dataset ,“steam_games.csv”, was retrieved from Kaggle[1] . It contains about 40k records of video games available for sale on the widely used game digital distribution platform, Steam.

Since we think that the first dataset may not contain features that are relevant enough to create an informative model for predicting the success of a video game, we acquired a second dataset from Kaggle as well[2]. The second dataset, “steam-200k.csv”, includes about 200k records of Steam users and their playtimes for various games, which provides a potential enhancement for our model creation, as we believe the amount of time players spent in a game is an important factor to predict a game’s overall success.

[1] the first dataset was taken from kaggle.com/datasets/trolukovich/steam-games-complete-dataset
[2] the second dataset was taken from https://www.kaggle.com/datasets/tamber/steam-video-games

3. Feature Engineering

We performed feature selection based on domain knowledges, and during the process, the following columns were identified as unnecessary and thus removed from the dataset:

1. **"types"** column: This column represents the type of the selling objects on Steam. Since our study focused on "app" records, which represent video games, we removed it since the "types" column would not provide any additional information after the cleaning.

2. **“url”** column: This column contains the URL links for each game on the Steam platform. However, as our analysis did not involve any form of web scraping or content extraction from these URLs, this column was considered unnecessary and subsequently removed.

3. **"recent_reviews"** column: This column contains information about the recent reviews of a video game. As it represents more about the quality of a game's constant updates and maintenance, which was irrelevant to our study, we removed it and used the "all_reviews" column as our dependent variable, which is a better indicator of a game's overall reviews.

4. **“release_date”** column: This column was an important column as it contains strings of dates in the format of “month day, year”. We believe that the year and the month value may be informative to predict a game’s popularity, as we hypothesized that video games that are published during the pandemic years and holiday months may be more successful. Thus, we extracted the year and month to be two new features “release_year” and “release_month” and removed the original “release_date” column.

5. **“publisher”** column: This column contains information about the game's publisher. While publishers could definitely influence a game's marketing and distribution, and also, its popularity, our project did not focus on this aspect. Therefore, we decided to remove this column from our dataset.

6. **“minimum_requirements”** column: This column included the minimum system requirement to run a game. Although minimum system requirements can affect popularity of a game by limiting the people who could play the game. Due to not using this column to predict popularity or recommending a game in our project, we removed the “minimum_requirement” column.

7. **“recommended_requirements”** column: we removed this column from the dataset due to the same reason as the “minimum_requirement” column.

More feature engineering is discussed in the Model Training part later in this report.

4. Data Cleaning

A series of data cleaning steps were performed on the dataset. The main dataset contains not only video games, but also game bundles, DLCs (downloadable contents), soundtracks and also in-game purchase currencies.

This project's objective was to clean up a dataset that contained details on games that were accessible through the Steam platform. The dataset was first made available as a "steam_games.csv" CSV file. The main responsibilities were to get rid of unnecessary columns, deal with missing data, and filter out certain game types, unknown characters or prices in some particular columns. We also did some additional cleaning. For instance, in the 'game_description' column, we removed the redundant phrase "About This Game" that was present at the start of many descriptions.

One of the crucial columns we addressed is the price column, as it acts as an important factor to the sales of a video game.The dataset presents price data in various forms, for example, the majority of the price values are in the form of “$number”. However, there are also values like “1.02” which possibly stands for 1.02 k$ as its price value after discount is $906.48, which is relatively high but still reasonable. However, considering domain knowledge and common sense, it was concluded that this price value is highly unlikely to be the price of a single video game.Consequently, such entries were removed as we are mainly focusing on standard video games. After removing invalid price formats, such as prices missing $ or invalid numbers and NA prices, there were also games that had prices much higher than other games. Entries with game price more than $250 were removed from the dataset as they were deemed to be outliers.

In the “name” column of the first dataset, there were games that had non valid names. For instance, some of the games only had emojis as their names. There were also game names that were written in foreign language characters. In our cleaned dataset, we kept the games that had English valid characters in their names.

Our dataset was cleaned as a result of this process, and it is now suitable for additional analysis or modeling. The cleansed dataset no longer includes unnecessary entries like bundles or DLCs and only contains rows with complete data in critical columns. Now that the dataset has been cleaned, it can offer more accurate and useful information about the games that are available on Steam.

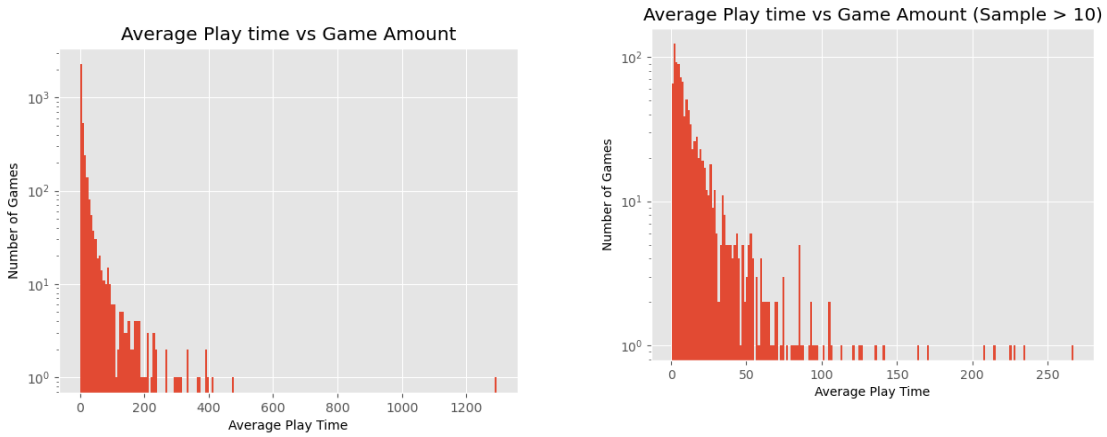
5. Outliers

We applied mainly two methods to remove outliers from the datasets. The first method is outlier removal by Z-score, and the second one includes manual analysis on certain outliers to find the cause and mitigate their effect with domain knowledge.

The main dataset contains mainly text data and a few numerical data. We performed Z-score outlier removal on the numerical data, and we used mutual information score feature elimination during the specific training steps to eliminate

potential outliers in text data that appeared as typos and rare categories which will be explained later in this report.

In the second dataset, which contains information about steam user's play time on various video games, we plotted the histogram of the average play time of games.

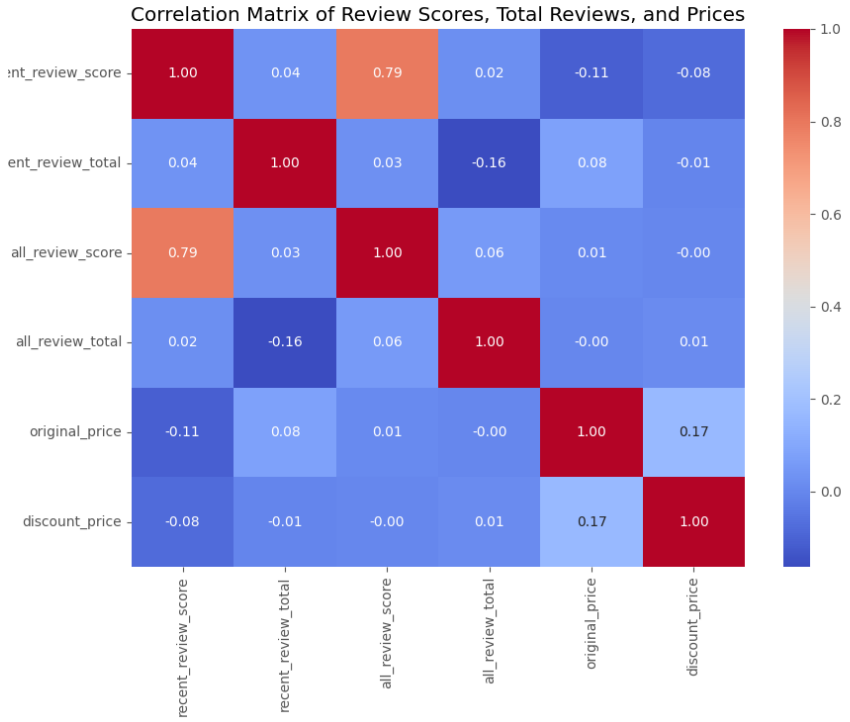


As shown in the first plot above, we can observe a game that has an average play time of over 1200 hours. This is very unusual as it is very far from the majority of the data, and rare in real life. Thus, we manually investigated the dataset, and found that this anomaly was based on a single record from one user within the 200000 records in our dataset. As implied by this outlier, we realized that it is necessary to only look at games with a reasonable sample size to ensure the validity of the average play time. Therefore, we filtered the data so that only video games that have more than 10 records before average grouping are left for later analysis. This again shows the importance of manual data checking and domain knowledge. As such problems are not obvious when only applying automatic outlier removal methods. The result plot is shown on the right side above.

6. Analysis

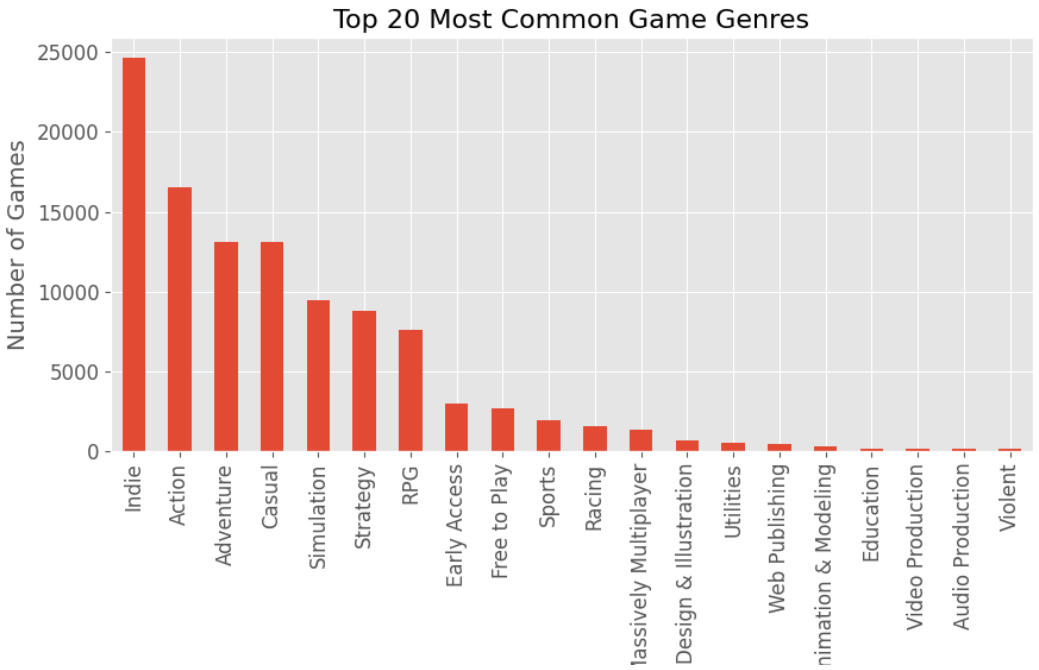
At the start of the project, we performed some data analysis on the raw data to understand its structure, content, and potential features that could be used for further analysis and model development. This initial analysis helped us in shaping the direction of the project, even though not all visualizations and analyses were eventually included in the final report. Some of these analyses were not considered since they needed more advanced methods, and some of them were not considered since they were not relevant to our project's goals.

For instance, we attempted to analyze the correlation between review scores, total reviews, and prices. This analysis could be seen in the picture below. However, it was not successful due to the complexity and size of the data. Nonetheless, this initial attempt informed the decision to use more advanced methods later in the project to handle this dataset.

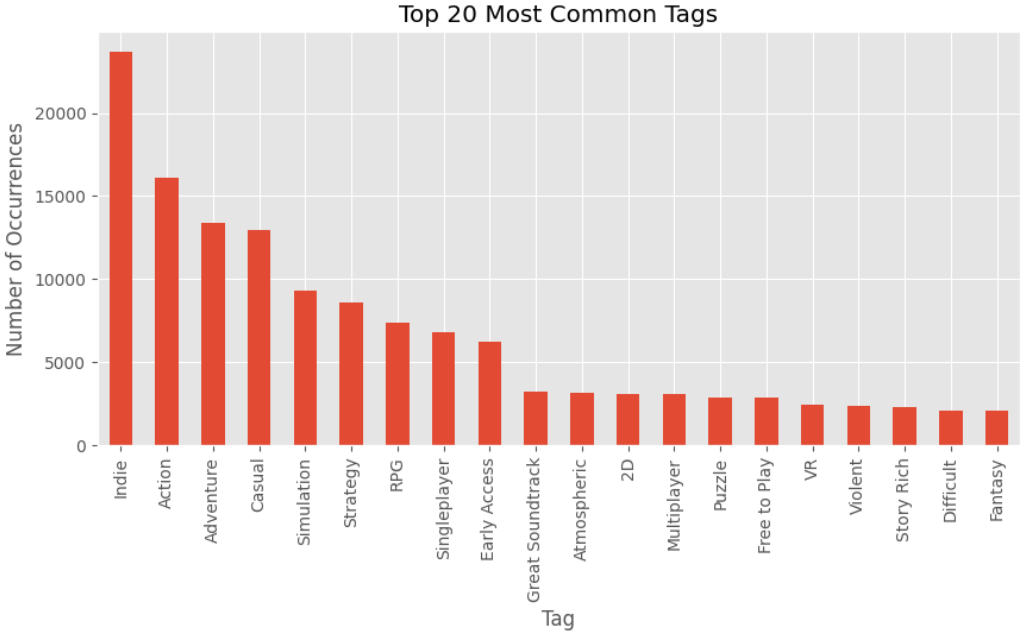


Another example of our initial unsuccessful analysis could be the analysis of pandemic impact. We tried to see if COVID-19 had any effects in the number of games getting published during the pandemic. However, it was decided that this analysis would not be included in the final report since it was not aligned with our goals in this project.

However, we also had some successful analyses in our initial data analysis that led us towards better goals for this project. For example, we did some analysis on the game genres. Our analysis and visualization show the 20 most common game genres. This was very helpful since it helped us have a better understanding of different game genres, leading to a better and more accurate Game Recommender system that we made later in this project. The visualization can be seen below:



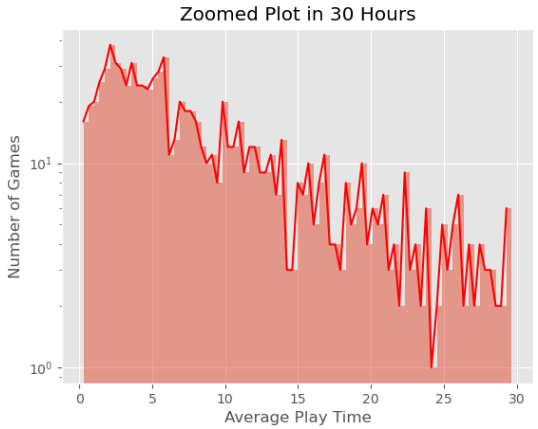
We also did the same analysis on the game tags. Tags are a vital aspect of game categorization, helping to identify specific elements of a game that players might be interested in. Therefore, this was also very helpful and crucial in making the Game Recommender. The visualization can be seen below:



In our initial code for data analyses, we did around 10 analyzations. Some of these analyses were unsuccessful due to the limitations of the methods used, or

were excluded because they were not relevant to the project's objectives. However, this first analysis offered essential lessons and insights that affected the project's approach and direction.

For the user play-time dataset, we plotted the average play-time of games, and as shown in the plots in the outlier section, we can observe that the average play-time tops at 2.5 hours, and then gradually decreases. This finding shows support for an interesting hypothesis that the first 2.5 hours of gameplay is crucial for determining whether a player will keep playing the game or not. Although It is plausible that some games are designed to be finished within 3 hours. It is also important to consider that according to various reports and video game stats websites, average video games' length has seemed to be increasing since 2008 from 8 hours to 30 hours in 2021. This notable increase in game length, far surpassing the three-hour mark, suggests that players are more likely to actively quit playing after the initial 2.5 hours because of being bored rather than the game is finished.



7. Game Name Matching

We decided to develop a script that reads the CSV file and prompts the user to enter the name of a game. The script then compares the user input to the closest game name in the dataset, taking into account any potential errors like typos or missing names. The user is then prompted by the script to confirm that the match is accurate. If the user says it's not accurate, the script searches for the subsequent closest match. The script continues this process five times before concluding that the user is probably typing a game name that isn't listed in the dataset.

We used Python's 'fuzzywuzzy' string matching package for this task. The Levenshtein Distance is used by this library to determine the differences between sequences (in this instance, the sequences are strings).

Applying the fuzzywuzzy library's procedure, the game name matching script's first implementation was straightforward. To get the most accurate match to the user's input, we used the extractOne() function.

However, a warning related to the fuzzywuzzy library was found when the script was run: "Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning." To remove this warning and potentially speed up the fuzzy matching process, the python-Levenshtein library should have been installed. Therefore, we used pip install python-Levenshtein (or pip3 install python-Levenshtein).

8. Game Recommender

We decided to design a game recommender system based on our dataset of Steam games. The recommender system would suggest five games similar to a given game based on the game's popular tags, genre, and game details. This system is using our previously done system (guessing game names from user's input) to find the game from the dataset. The guess.py script was designed to be imported into this game recommender script, and the find_game function from guess.py was used to ask the user for the game's name.

For this game recommender, we decided to use the sklearn.neighbors.NearestNeighbors model. Also, The system first preprocessed the game features (popular tags, genre, and game details) by converting them to lower case, stripping whitespace, splitting on commas, and filling NA values with 'unknown'. These preprocessings are **specifically** for our game recommender.

We also used sklearn.preprocessing.MultiLabelBinarizer to transform the features and labels into binary vectors. A binary vector was created for each feature column, resulting in three encoded DataFrames: df_encoded_tags, df_encoded_details, and df_encoded_genre. These DataFrames were concatenated along the columns axis to form the final df_encoded DataFrame.

We used NearestNeighbors model and then fitted it to df_encoded. Then, our system was used to find the five games most similar to a given game based on Euclidean distance. The Euclidean distance serves as a measure for the actual straight-line distance between two points in a plane. It is an easy-to-understand distance measurement, thus we applied it in our Nearest Neighbours model. It provides us with a straightforward method to calculate how similar two games are: the closer two games are to each other in our feature space, the smaller their Euclidean distance.

To provide explanations for the recommendations, we found the common features between the input game and each recommended game. We did it by using the 'set' data structure and the 'intersection' method to find common elements.

During this task, we encountered several challenges:

1. The game features were categorical and needed to be encoded into numerical form for use with NearestNeighbors. We solved this by using MultiLabelBinarizer to convert each feature into a binary vector.

2. At one point, the system was asking the user for the game name twice. This was because the find_game function was being called twice - once in guess.py and once in the game recommender script. We fixed this by removing the call to find_game in guess.py.

3. We received a warning from sklearn about X not having valid feature names. We suppressed this warning using the 'warnings' library.

After fixing these, the game recommender system was successfully implemented. It was able to accurately guess the game name from user input, recommend similar games, and provide explanations for the recommendations.

9. Model Training

One primary objective of this project is to develop a machine learning model capable of predicting video game's success. These predictions are based on various factors, such as genre, price, release date, and others available during a video game's development phase, so that developers can have an insight of whether they are going in the right direction or not, and if they are more likely to produce a successful video game.

1. Class selection

The class label selection was one of the most important steps of this project, as we have many columns that can be used to be or extract the target class from.

In the original dataset, there is a column 'all_reviews' which contains the rating category, review amount and the percentage of positive reviews. There are many approaches to extract an interesting target class. For example, we could directly use the review category, or use the number of positive reviews that is calculated by the total reviews and the positive percentage. A problem we encountered during this step is that games have different amounts of ratings. For example, there may be a very popular game with 1 million reviews but only 60% of them are positive, and there may also be a game with only 1000 reviews but 90% of them are positive. In this case, the calculated number of positive reviews is a great way to show a game's popularity as only positive reviews contribute to it.

The target class selection was a critical step in this project due to the multitude of information that could potentially be a useful result. For instance, the 'all_reviews' column in the original dataset contains rating categories, review counts, and the percentage of positive reviews. We could use the review category directly or use the number of positive reviews, computed from the total reviews and positive percentage. The difficulty here is that games have varying amounts of ratings. For example, a popular game with 1 million reviews might have only 60% positive reviews, while a less-known game with 1,000 reviews might have 90% positive reviews. In this case, it is debatable to tell which game is better as popularity and game quality are two totally different things. In this scenario, the calculated number of positive reviews effectively indicates a game's popularity as only positive reviews contribute to this value, and it effectively makes use of the percentage and the amount of reviews which indicates a game's popularity at the same time.

Thus, we decided to train a classification model on the rating categories, which contains labels shown in figure, and also a regression model that predicts the percentage of positive reviews so that we can then combine the prediction of these two models together based on the percentage to category match shown in the

- 95 - 99% : **Overhwelmingly Positive**
- 94 - 80% : **Very Positive**
- 80 - 99% + few reviews: **Positive**
- 70 - 79% : **Mostly Positive**
- 40 - 69% : **Mixed**
- 20? - 39% : **Mostly Negative**
- 0 - 39% + rew reviews: **Negative**
- 0 - 19% : **Very Negative**
- 0 - 19% + many reviews: **Overwhelmingly Negative**

plot, as an ensemble model to produce a more accurate result.

However, despite our intense data processing and effort, our multi-class model only achieved a prediction accuracy of 59% for the rating categories.

In order to improve this score, we combined the average play time data from the second dataset mentioned above, so that we could have a feature on how much time players spend in each game on average. This feature was supposed to improve the score as according to domain knowledge, the time a player spends in a game should be correlated to how good a game is. However, after applying the extra column, the score decreased by about 3%. We believe that this is due to the decrease in the size of our data after the combination of the two dataset. As the second dataset did not contain the average play time for all games in the main dataset, we removed all games that did not have the average play time column after the combination. This process reduced our data size from around 1800 to 1000, it might be the reason why our score decreased unexpectedly, and it showed that we did not have a sufficient amount of data to perform multi-class prediction.

To investigate this issue further, we also performed a random forest regressor analysis on the data, where we used the amount of “positive reviews” we mentioned earlier above as the dependent variable. However, the results were similar with a very low r-square, which means the prediction was not reliable. Our model was not good enough to provide a reliable prediction on the positive reviews nor the multi-class game rating categories.

After careful consideration, we realized that the regression model failed because the amount of positive reviews depends on too many factors in real life, and requires more features, while our data after processing was also not big enough to produce a reliable result for multi-class prediction. we concluded that our dataset with the features we extracted, even those we derived, were insufficient to produce a model that can predict the multiple labels reliably.

Thus, we decided to change our hypothesis. Instead of predicting multiple rating categories, we shifted our focus to a binary classification model, distinguishing between "popular" and "unpopular" games. This change simplifies the prediction without sacrificing any information, as the multi-category game rating might not be useful in practice from the first place, as developers may only care about whether their games are popular or not. Thus, We grouped all games that have a positive review percentage over 70% to be “popular”, with the rest to be “unpopular”. This classification also aligns with Steam’s official rating categories, as all games under the 70% positive percentage are classified as Mixed or negative. As mentioned later in this report, this decision proved successful, and led us to produce a reliable model to predict if a game is popular or not with a f1-macro score of 73%.

2. Feature Engineering

In data processing, we performed one-hot-encoding on several columns as a moderate amount of our features are categorical data. We applied one-hot-encoding

to the following columns: “language”, “release_year”, “release_month”, “developer”, “genre”, “game_detail”.

As one-hot-encoding on specific features significantly increased the dimensionality of the training data, making it hard to analyze and may result in lower accuracy, we performed feature selection in a combination of two methods: manual feature selection, and an automatic features selection based on mutual information score.

The “developer” column is an example of an instance where we performed manual feature engineering. The “developer” column contains a great number of companies, and most of them are small infamous companies that only appear once in the dataset, which make the one-hot-encoding produce over 1400 new features where each feature is just a unique company name. To reduce dimensionality and remove useless features, we believe that the company name data is only useful if it occurred multiple times in our dataset in different game records. Thus, we grouped all developers that only have one record, and grouped them under the same label ‘small_company’. This approach drastically decreased the dimensionality and only informative company names are preserved.

Besides the manual feature selection, we also implemented an automatic feature selection process based on the mutual information score to remove uninformative features resulting from one-hot-encoding from features such as “genre” and “game_details”. For example, some games may have unique or irrelevant genres that are not helpful for the prediction. After some tuning and manually checking the selection result and prediction score, we believe that a threshold of 0.01 was a great choice to remove useless features but also preserve useful ones. This method combined with our manual checking greatly reduced the dimensionality of the data and resulted in a more accurate and efficient model.

3. Algorithm Selection

Initially, we used an ensemble model consisting of Naive Bayesian, K-Nearest Neighbors (KNN), and Random Forest algorithms to predict the popularity of games. However, after iterations of testing and analyzing the result, we realized that some algorithms were not working as expected in our model.

Specifically, The Naive Bayesian model assumes that features are independent. However, in our dataset, certain features are related. For example, in the genres, games with the genre “Co-op” are more likely to have the genre “multiplayer” as well. The Naive Bayesian algorithm was not able to capture these relationships between features, leading us to exclude it from the model.

We then explored other algorithms such as the Linear Vector Machine (SVM), Decision Trees and Neural Networks… etc. After comparing their performance and the time they take to train, we concluded that the combination of SVM, KNN and Random Forest produced the most robust result in a fairly short amount of time. Therefore, we selected this combination for our final version of the model.

4. Hyperparameter Tuning

The GridsearchCV was used to perform an automatic hyperparameter turning. To get a reliable set of hyperparameters, we initially applied GridsearchCV with a smaller range of potential parameters during the training stage. The reason was that we were continually changing and improving the training. After we finalized the training stage, a comprehensive set of potential hyperparameters were used to perform a complete hyperparameter selection. This process took half an hour to run but it ensures that we have the most reliable parameters and significantly improved the performance of the model.

5. Data Balancing

One of the biggest challenges we met during the training is that the classes are not balanced. After categorizing the games into "popular" and "unpopular", 82% of the records belong to the "popular" category and only 18% classified as "unpopular". The unbalance of the class labels led to a wrongly high accuracy score as even if the model failed to predict all the “unpopular” games, the model would still achieve an accuracy of 82% by correctly identifying all the "popular" games. In order to overcome this challenge, we calculated the proportions of the class labels, and provided them to the machine learning algorithms. For the scoring, the f1-macro was used instead of accuracy to provide a more reliable representation of the model’s performance. As a result, our approach successfully improved our f1-macro score for approximately 10%, which is also the most significant performance increase throughout this project.

6. Conclusion

In this machine learning model training section, we underwent various iterations and improvements in our approach as we keep going back to refine our approach after each result. We started with the hypothesis that we could predict a game’s rating in categories, and our initial attempt to produce a reliable multi-class model was unsuccessful. However, we then focused our effort on predicting a binary outcome, which simplified our problem without sacrificing any core information, and also proves the importance of the iterative process in machine learning development.

Feature engineering was also a crucial part of this section, we performed comprehensive feature extraction and selection to get the most useful features out of the original raw dataset.

Despite the challenges we faced, in the end, we successfully produced a reliable model on predicting a game's popularity with a respectable f1-macro score of 73%. This shows the potential of using machine learning methods to predict the popularity of games that are still in development with only information such as genre, price, and publication date, among other factors.

10. Limitation & Potential Future Study

As discussed in this report, we encountered many challenges and we managed to overcome most of them. However, there are still many potential improvements we could have made if we had more time and resources to produce better results and a more reliable model.

For example, during model training, we performed comprehensive data processing, which greatly reduced the size of our dataset from 50,000 to around 1,800 records of useful data. This size is further reduced to 1000 when combined with the average play time data, and it led to a decrease in the model score.

Thus, we believe that expanding the dataset could potentially improve the score of our model. Further, exploring other machine learning models or techniques outside of this course, as well as enhancing our model with more useful features, could also potentially improve the results.

Overview

Ali Nikan:

- Created and put into use a data cleaning pipeline using Python and pandas for transforming an unstructured dataset of more than 40,000 Steam games.
- For a better user experience and more user engagement, a customized game recommendation system was implemented using the Nearest Neighbours model from Sklearn and fuzzy string matching.
- used statistical techniques, matplotlib, and seaborn libraries to do comprehensive data analysis, resulting in helpful visualizations and findings.
- Assisted with the writing, editing, and review of a thorough project report that included the presentation of the project's goals, procedures, findings, and conclusions.

Dingshuo Yang:

- Responsible for the whole Model Training part, built a game popularity prediction model and achieved a 73% score by performing thorough data cleaning, comprehensive feature engineering, tuning and training on a combination of multiple datasets.
- Performed general feature selection and robust outlier removal on both datasets by using both the automati Z-score approach and manual investigation and cleaning.
- Structured the report, and participated in the writing and reviewing of all sections, and the whole part of Model Training.
- Acquired and managed the play time dataset, including all stages of its data preprocessing, analysis with plotting, and its integration with the main dataset through fuzzywuzzy string match to produce more features to improve our model performance.
- Responsible for the Repo Manager role by creating and maintaining the repo with professional PR reviews and branch management.

Mohammad Parsaei:

- Using Pandas, performed a data cleaning on a Steam game datasets to eradicate game entries in dataset with non valid data to prepare the dataset for the data analysis process. After cleaning, the new dataset contains around 1800 games instead of 40000.
- Used Pandas and Multilabel Binarizer from Sklearn to transform columns with nominal data to columns with binary ordinal data in order to prepare dataframe for ML model.

- With the help of Sklearn, made a ML model to predict game popularities based on genre, game play time by using a voting classifier of KNN classifier, Gaussian Naive Bayes and Decision tree classifier.