Exploration of Steam User Behavior with Fine Tuned TFIDF-Vectorizer-Regression Model for Categorization of Reviews and Deep Classifier for Prediction of Playtime Frequency

Abstract

In real-world scenarios, many websites aim to maximize their revenues by analyzing customer targets in multiple aspects. In this report, we seek to understand how playtime varies among game players and design a model that predicts the playtime category users belong. By collecting various datasets from the steam platform, we want to explore the features that impact these factors and build predictive machine-learning models upon them. At the same time, we analyze the review data and expect to train a model that predicts whether the user recommends the game or not. We constructed multiple machine learning models like logistics regression, and linear SVC to analyze the patterns. We end up with over 80% F1 score on the former task and around 49% accuracy on the second task.

I. DATASET

As a group of four, we split our tasks into two parts:

- Game-Playtime Classification Task: Given user and game, predict the playtime category(threshold we will explore later) of the user.
- Review-Recommend Classification Task: Given the user's review text, predict the category {1: Recommend, 0: Not Recommend} of the review text.

1. Dataset Description:

Datasets are collected secondly from https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data. In preparation of future analysis, we store all the entries of the selected json files into a list of dictionaries, including user_reviews.json.gz, user_items.json.gz, and etc.

users_review_data: stores information from user_reviews.json.gz in a list of dictionaries, where each dictionary contains a unique user_id and their review history. There are 59305 distinct review texts, 25485 distinct users, and 3682 distinct items.

items_data: a list of dictionaries, each dictionary pair contains related information of the game like category, genre, year published etc.

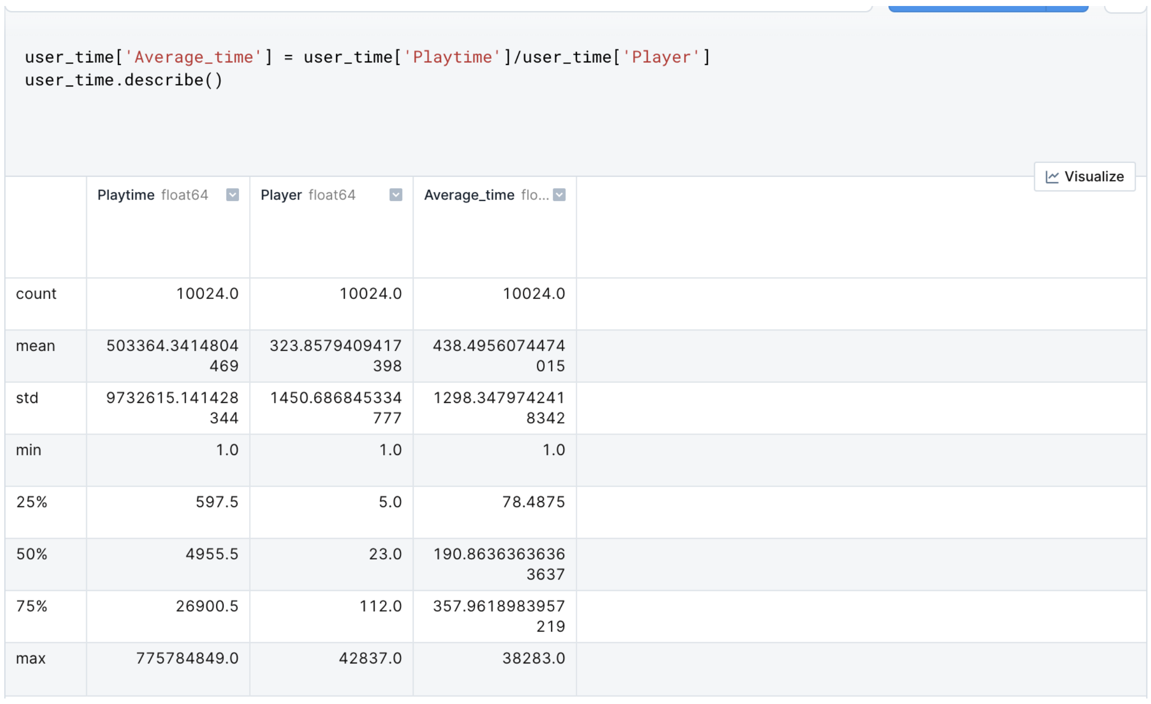
users_items_data: a list of dictionaries, each dictionary pair contains a unique user_id and their purchase game list.

One big issue we encountered when doing the Game-Playtime Classification Task was that the items_data set did not completely match with items_data, and we could eventually run into trouble with having to fill feature vectors with nans or equivalent when doing user-item prediction pairs which would hurt our model. We collected a matching set by using apis from steamspy.com to get info about a game's and loaded our own json file. This new dataset is a dictionary containing app_id, name,developer, publisher, numbers of positive and negative reviews, tags, category, languages for a given steam app.

2 EXPLORATORY DATA ANALYSIS

Game-Playtime Classification Task:

From the dataset above, we first plot out the graph that demonstrates the distribution pattern of all users' playtime of the game to find the best categorizing threshold of the game playtime:



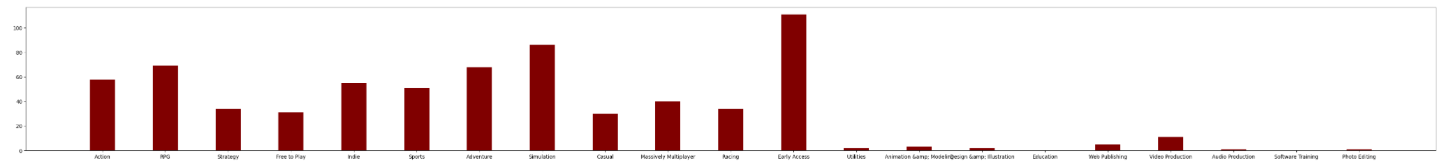
```
user_time['Average_time'] = user_time['Playtime']/user_time['Player']
user_time.describe()
```

	Playtime float64	Player float64	Average_time flo...
count	10024.0	10024.0	10024.0
mean	503364.3414804469	323.8579409417398	438.4956074474015
std	9732615.141428344	1450.686845334777	1298.3479742418342
min	1.0	1.0	1.0
25%	597.5	5.0	78.4875
50%	4955.5	23.0	190.86363636363637
75%	26900.5	112.0	357.9618983957219
max	775784849.0	42837.0	38283.0

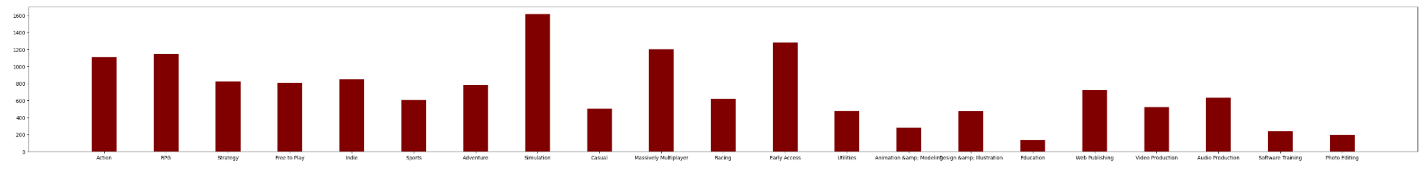
Graph 1: Statistics table about the distribution pattern of the users' playtime of the game

From the graph above, we notice that 26.2% of players tend to have at least one 0 playtimes in their game list, and 39.3% of players tend to play games under 45 minutes, which means we need to adjust the threshold lower than what we expected to categorize the playtime data more accurately. From the dataset above, we were able to obtain the average playtime per user for each game. We can notice that the maximum average playtime is 775784849 mins and the minimum is only 1 min. The statistics above demonstrated how players' time varies among different games.

Also, we plot out the graph that demonstrates the distribution pattern of user's mean/median playtime with the genre of the game:



Graph 2: X-axis: Genre of the game; Y-axis: Mean playtime



Graph 3: X-axis: Genre of the game; Y-axis: Median playtime

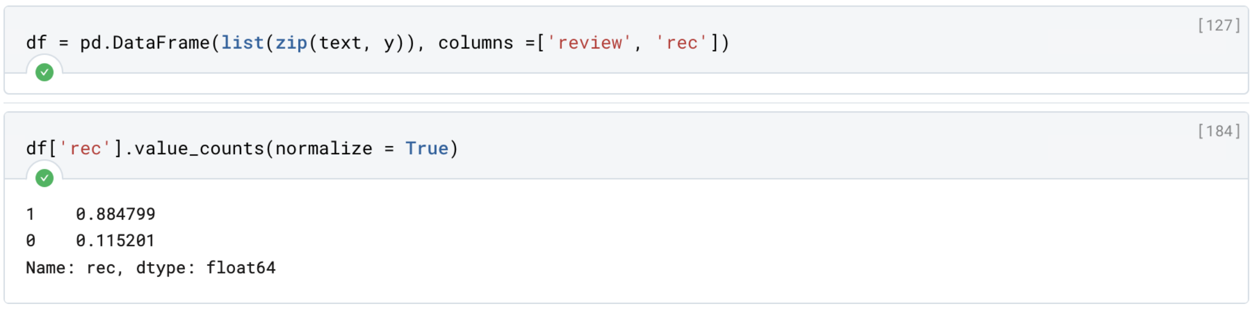
From the two graphs above, we notice there are distinctions between each genre of game's playtime, which indicates that the genre of the game may be an important factor that has influence on the user's playtime of the game. The first graph shows user's mean playtime as the y-axis and game's genre as the x-axis. The second graph shows the user's median playtime as the y-axis and game's genre as the x-axis.

Inspired by varied playtime with varied factors discovered in the dataset, we would like to explore how the game's playtime can be predicted by analyzing the users' gaming behavior, like the mean/median playtime of other users of the game, the mean/median playtime of the user, or the genre of the game.

Review-Recommend Classification Task:

From the picture below, we observe a moderate class imbalance in the **rec** dataset where recommend makes up the majority 88% of the dataset while not recommend makes up the rest 12%. This highly imbalanced data can lead to several problems in training and evaluating models including but not limited to overfitting to the majority class, ignorance of the minority class, and accuracy paradox.

We also contemplate that this highly skewed distribution is indeed a reflection of the real world true distribution in the sense that users are more likely to spend time on writing reviews for games they enjoyed and not wasting time on games they don't like. Thus we would create models with and without balanced weight to testify whether this statement holds by comparing their generalization performance.

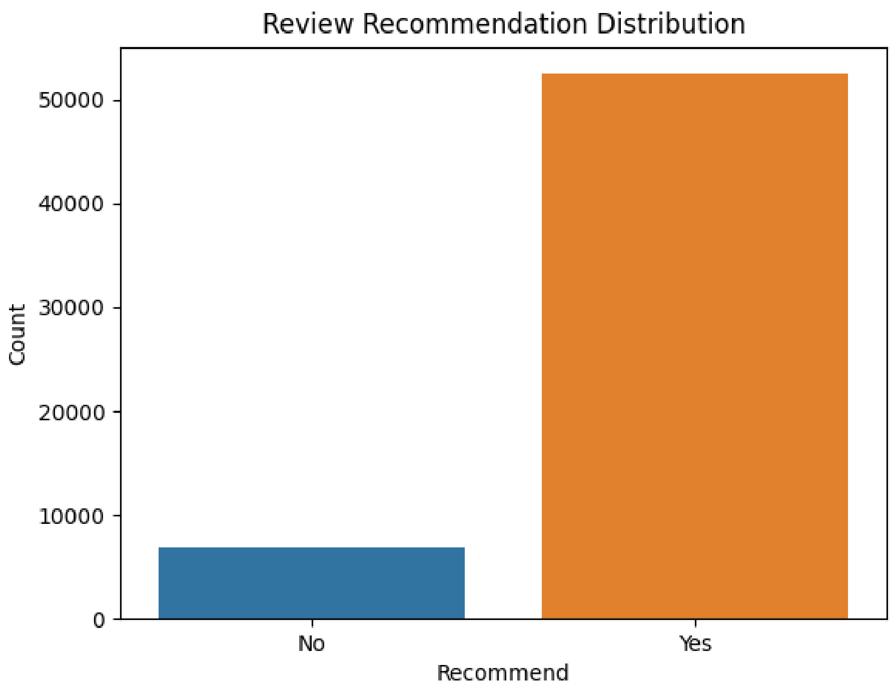


```
df = pd.DataFrame(list(zip(text, y)), columns =['review', 'rec'])
```

```
df['rec'].value_counts(normalize = True)
```

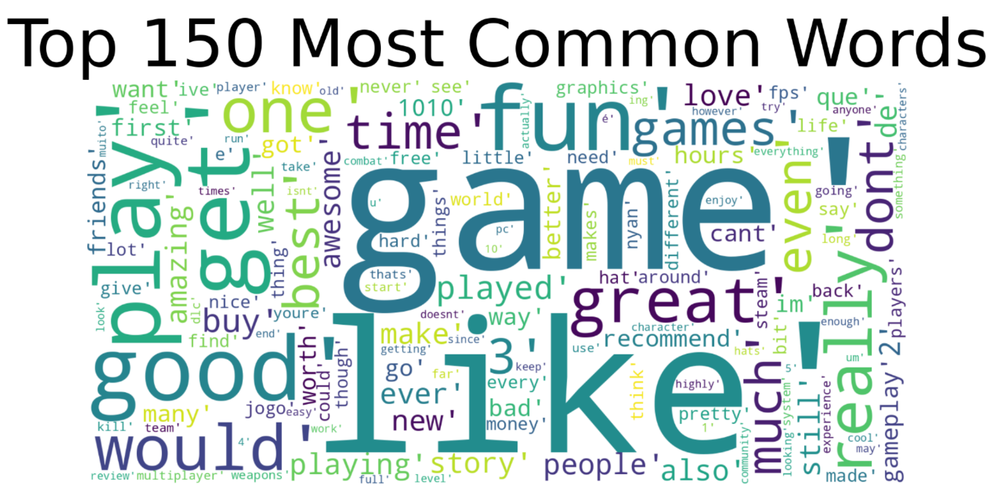
```
1    0.884799
0    0.115201
Name: rec, dtype: float64
```

Graph 4: Distribution of recommended and unrecommended reviews

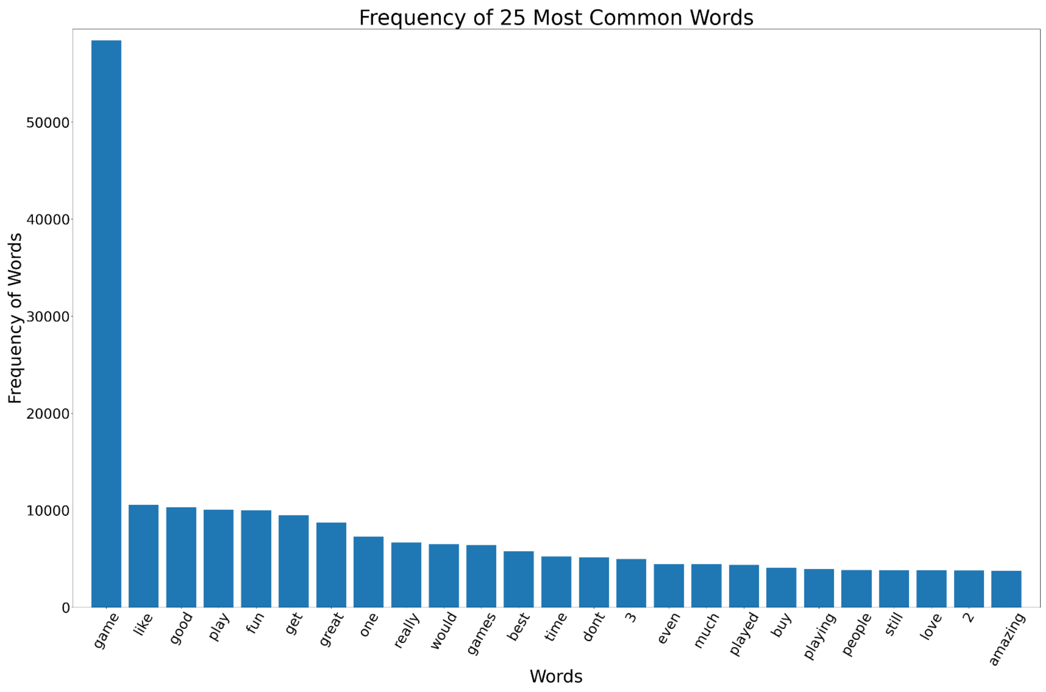


Graph 5: Distribution of recommended and unrecommended reviews

In order to develop a general idea of our review text data, we generate a word cloud of the top 150 most common words and plot the distribution of 25 most frequent words as shown below. Based on the above observation that the rec data is moderately skewed toward recommendation, the term frequency graph certainly supports the overall positive feedback (i.e. likes or recommends the game) from the users. For instance, terms such as "like", "good", "fun", "great", "best" point to a positive game experience and imply that the writer of the review is very likely to enjoy and recommend a game.



Graph 6: Top 150 Most Common Words



Graph 7: Frequency of 25 Most Common Words

Based on the above observations and our prior knowledge about steam reviews as a user, review text seems to be a close reflection of the recommended status. However, by taking a close look at the review text data along with its corresponding recommended status, we notice that some of the text is highly correlated to a negative feedback by human interpretation yet it is a recommended review. Triggered by this counter intuitive sample, we are motivated to explore "to what extent is the review text an accurate indicator of the review's recommended status".

II. Predictive Task

1. Relevant Baseline Selection

Game-Playtime Classification Task: Trivial Heuristic (Baseline): predicting all playtime as 0. accuracy = 0.256. Since we have relatively even distribution, this number should be close to 0.25%, as expected.

Review-Recommend Classification Task: Baseline Model: Trivially predict all reviews are written to recommend the corresponding game.

2. Data Selection

Game-Playtime Classification Task:

We create standard data structures such as userPerItem and itemPerUser dictionaries for the training set. We also create playTimeAverageByGame to store a dictionary of lists storing minutes playtime for any player that has launched the game. We also grouped by categories as well as tag and calculated their average play time. All of these data structure will come in handy for one-hot-encoding and feature extraction,

Based on our initial stage of data exploration, we hypothesize that the following features could potentially help us correlate a user item pair with playtime.

User related	Item Related
Median playtime of user for all other games (original value)	Game category (one-hot encoded)
User preference vectors (z-score normalized, see below for detail)	Global median playtime of game
User avg playtime per category(one-hot encoded index)	Percent of positive reviews of game

Table 1: Feature table

In the attempt to make our models more personalized, we attempted to also create a personal preference vector, in which we sum up all of the tag values out of all the games the user has played, and one hot encode into 439 dimension vectors. If we were to rank a user's preference tag value in order, we can see which tag the user likes the most. We hypothesize that adding this feature to our deep learning model alongside categorical information about a game will help our model pickup any correlation between a user's favorite types of game to play vs playtime.

In our game dataset, we have access to 439 different tags assigned to a game with value corresponding to how much that tag is relevant to the game. Take the game CSGO, for instance, will have the following value:

{'Action': 1339, 'FPS': 979, 'Shooter': 723, 'Multiplayer': 582, 'First-Person': 435, 'Classic': 379,

'Singleplayer': 360, 'Tactical': 355, 'Team-Based': 303, 'Competitive': 288,'Strategy': 173, 'Online Co-Op': 162, 'Military': 151, 'Masterpiece': 87, 'Adventure': 83, 'Survival': 73, 'Atmospheric': 56, 'Open World': 51, 'Simulation': 38, 'Dark': 38}

We one-hot encoded the category using our previous data structure so this could be fed into any model directly as a feature, or be used to compare users' similarity with each other.

Review-Recommend Classification Task: Within users_review_data, we extract the review text and its corresponding recommended status into text and rec respectively by looping through each user and dive into their review history. However, neither of these variables would be utilized in the baseline model since it depends on nothing but the trivial heuristic of simply predicting every review as a recommended review.

In order to perform a more advanced classification technique, some feature engineering is required to be done on text. Our experiment ends up taking two approaches to develop our features, 1) Encoding raw text string into **[bias_term, text_length, exclaimation_count]** and forward this custom feature into Logistic Regression model; 2)Transforming raw text into TF-IDF Vectorizer with maximum features 30000 by deploying the Scikit Learn package, where the input text would be converted into lowercase and filtered out English stop words before getting tokenized into unigram word.

3. Validity Assessment

Game-Playtime Classification Task: We divide our dataset into three parts: Train set(60%) and Test set(40%). In our baseline model, since we only return the trivial heuristic value 0 as predicted playtime y_pred, we get actual playtime y by extracting the 'playtime_forever'part in each of the data in the Testset and outputting the category the data falls into. Then with y and y_pred, we can calculate the accuracy and thus assess the validity of our baseline model predictions.

The original data is very skewed towards lower playtime frequency. By analyzing total playtime distribution on the whole dataset as well using some common sense, we split our data labels into four roughly even distributed categories based on play time minutes[0:<=45(not interested), 1:<=220(tried the game), 2:<=840(likes the game), 3:>840(loves the game)] for the validation and testing set, ensuring that each class will be no more than ~1.5% away from 25%. This way, we can simply evaluate all of our models using loss and accuracy and not have to worry about class imbalance. We also ensure that we took the top 60% of user-items pairs from each user to ensure that an even representation of the portion of the user's steam library is present in the training set. This will help simulate as closely as possible to a real life scenario, when we know some preferences about the user and try to see how long the user will spend on a game.

Review-Recommend Classification Task: Given the fact that the rec dataset is highly imbalanced, a general train_test split would lead to accuracy paradox. Instead, after splitting rec and text into a training set and testing set using 80-20 ratio, we shrink the testing set into a balanced dataset by randomly choosing not the recommended amount of recommended data. For instance, if the testing set has a length of 10000, where 9000 are labeled as recommend and 1000 as not recommend, we would randomly choose 1000 recommend data and combine with the not recommend data to form a new testing set. Our balanced testing set has length 2074.

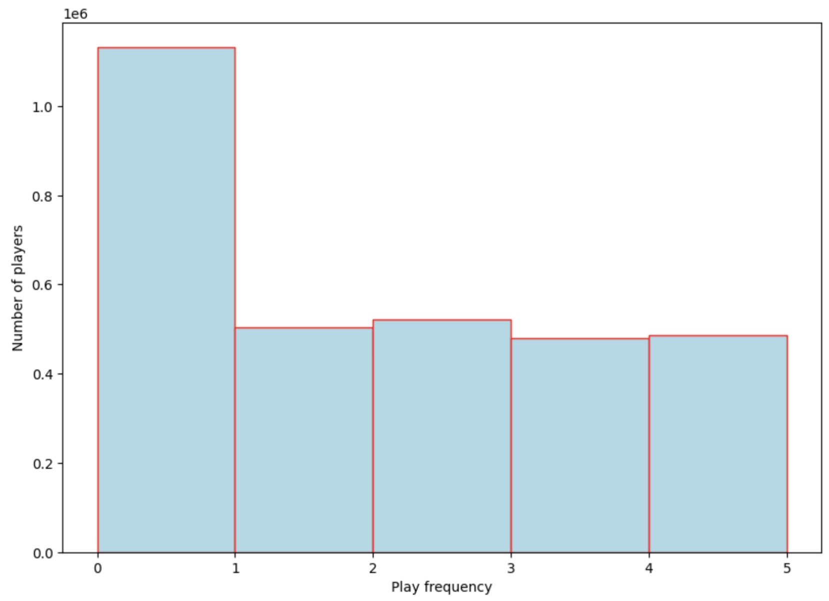
Moreover, in order to avoid misleading evaluation given a highly skewed class distribution, we compute the confusion matrix and select F1 Score and ROC-AUC Score in addition to accuracy as the evaluation metrics for every model. In this situation, both classes are important since we don't want the new users to overlook the negative feedback of the game which sometime could be a more important factor of their purchase decision. Thus a high F1 score on both classes is the essential determinator in best model selection.

Beside creating a balanced test set, we perform a 10-Folds validation on each model and use the average validation metric scores as a reference in hyperparameter tuning and best model selection.

III. Model Description

Game-Playtime Classification Task:

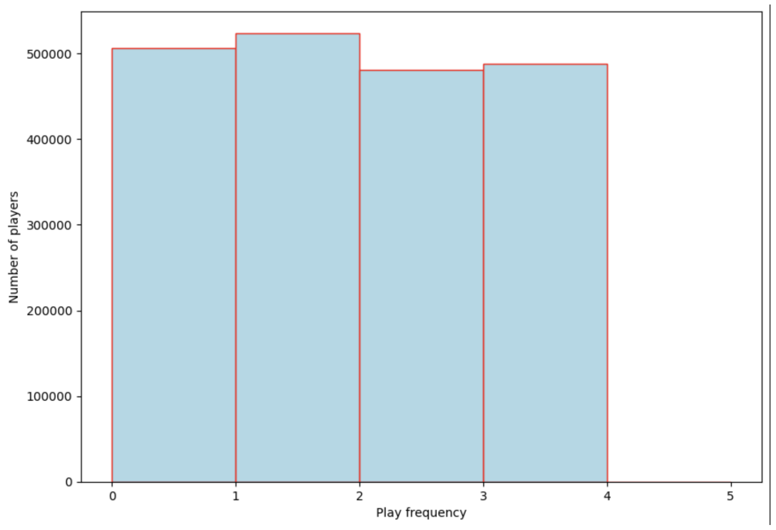
1) Popularity Predictor with Difference: predicting playtime based on other users' mean/median playtime of this game. Splitting the playtime into four categories with thresholds as [0:==0, 1:<=45, 2:<=220, 3:<=840, 4:>840] and output the playtime category that is closest to the threshold list. For example, if the predicted playtime of the game equals 230, then the Popularity Predictor with Difference will output category 2.



Graph 8: Distribution of players with play frequency: [0:==0, 1:<=45, 2:<=220, 3:<=840, 4:>840]

The model has a poor performance of 35.4% when choosing to predict playtime by the global mean playtime of the game without excluding the zero playtime category. It improves up to 41.8% when choosing the global median playtime to return. Since many players have zero playtime in the game, excluding numerous amounts of zero playtime data can help exclude the bias that drags down the mean playtime of the game. Thus outputting median playtime helps reach better accuracy.

Then we try to set the threshold for categorizing playtime in a relatively even distribution by excluding zero playtime from the category. Therefore we plot out the graph with the x-axis as the playtime threshold [0:<=45, 1:<=220, 2:<=840, 3:>840] and the y-axis as the number of players that reaches the playtime threshold, which we can see the distribution of playtime categories 1, 2, 3, 4 are quite even. This is the distribution we use when we prepare our training and testing dataset.



Graph 9: Distribution of players with play frequency: [0:<=45, 1:<=220, 2:<=840, 3:>840]

The model has a poor performance of 39.4% when choosing to predict playtime by the global median playtime of the game, excluding the zero playtime category. It improves up to 41.2% when choosing the global mean playtime to return. In this case, since 39.3% of all players only play games under 45 minutes as shown in the EDA, it is biased to choose returning median playtime that will make the predicted playtime much lower than it actually is. Therefore, choosing to return mean playtime of the game will get better accuracy when we choose to exclude zero out of the playtime category.

2) Popularity Predictor with Threshold(mean & median): predicting playtime based on other users' mean/median playtime of this game. Splitting the playtime into four categories with thresholds as [0:==0, 1:<=45, 2:<=220, 3:<=840, 4:>840] and output the playtime category that is within the threshold list. For example, if the predicted playtime of the game equals 230, then the Popularity Predictor with Threshold will output category 3. Then we try the threshold that excludes the zero playtime: [0:<=45, 1:<=220, 2:<=840, 3:>840].

This model behaves a little better than Popularity Predictor with Difference when choosing to output the global median playtime of the game to the user, regardless of excluding zero playtime out of the category. See the results section for a complete list of accuracy.

3) Genre and Player Predictor:

These are more nuanced heuristics that take a player's information into account when predicting their play frequency. We tried to simply take the player's median and mean playtime across all other games played, and ended with ~30% accuracy only. For the categorical model, we improve this by selectioning the mean of sum of all the categories of game played by the user. In both cases we use the calculated average/median and our preset threshold to decide between the 4 categories of play frequency. This improves the previous models to roughly ~34%. These improvements from baseline signaled that they are useful features but not the most important.

4) Logistic regression classifier

LogisticRegression / SGD / SVC classifier : combine genre of item(total 21 genres of game), global mean playtime of the game, mean playtime of the user, and bias term, then feed it into regression models. After trying a combination of features (see results section), we notice the accuracy only hovers around 43%, regardless of models. Adding new features would only slightly improve this and training would also be difficult due to our large features (in the biggest case, we have about 3 million user training set with over 300 features), so we decided to try a more complicated model to better fine tune and observe improvements.

5) **Deep neural network classifier (Best Model)**

Since our accuracy consistently hovers around 43% with linear regression and some SVC classifier with a variety of different features, we decided to try using some multilayer neural network, to capture non-linear relationships but also allow us to fine tune and analyze which features are more important to our model. We start with a pretty standard network with two linear layers and 100 hidden units with RELU as the activation function. This is further fine tuned by adjusting the number of hidden layers, units, activations, hyperparameters for optimizer, and adding regularization such as dropout layers.
With the best configuration of our models from logistic regression and fine tuning with Adam optimizer, we were able to improve our test accuracy to around 47.3%. Which is a bit of improvement but still not high. One of the issues we run into is that our model seemingly fails to recognize 'patterns' that can help us get through 45% accuracy, since changing several models did not help improve our accuracy. Adding the user preference vector improved our training loss by roughly 10%, and we were able to achieve above 51% training accuracy, yet it drops the validation acc, and overall performed slightly worse as a model since it is overfitting too much to users in the training set. Adding regularization such as the dropout layer only slightly improved the situation.

After trying a combination of features, we found out that the most important feature is the global median play time for the game. Adding the mean play time also increases the accuracy decently as well, as shown in experiment 7. After fine-tuning our model, including changing learning rates, adding layers and other hyperparameters, we were able to achieve close to 60% training accuracy and close to 49% accuracy on the testing set.

Review-Recommend Classification Task:

- Model Comparison:
    - Custom Feature + Logistic Regression
    - TF-IDF + Selected Classification Model:
        - Logistic Without Balanced Weight
        - **Logistic With Balanced Weight (Best Model)**
        - Linear SVC Without Balanced Weight

        - Linear SVC With Balanced Weight
    - Downsample Training + Logistic Regression

- Trivial Heuristic: combine review text length and the number of exclamation marks with a bias term to form a n-darray feature, then feed it into a logistic regression for prediction.

  Based on common sense and experience as a user, we propose that longer review text length and larger amount of exclamation marks would be associated with recommendation and vice versa. One way to optimize this model is to create thresholds for both parameters and manually tune them rather than directly feed the feature into the logistic regression model as sometimes a simple heuristic supported by descent background knowledge would outperform a complex model. This model would run into an overfitting issue since the training set is imbalanced and thus overfitting to the recommended class. This model is considered as an unsuccessful attempt since there is minimal or even neglectable improvement compared to the baseline model.

- TF-IDF + Selected Classification Model:

  For the rest of our experiment, we extract keywords for a text and train the model with it such that it can learn the internal patterns between text and the target output/labels. We choose TF-IDF because it is relatively simple and inexpensive to compute, plus it is an intuitive approach to weighting words, allowing us to retrieve the most relevant words of a given review text. Since we are not combining any other features, there is no scalability issue. In terms of overfitting, we propose and observe that models without balanced weight run into such issues whereas models with balanced weight don't.

  Models without balanced weight are considered unsuccessful attempts, although they outperform the baseline model in the testing set. The major drawback of these unsuccessful models are their poor performance in predicting not recommended labels due to an imbalanced training set. The strength of these unsuccessful models is that it would outperform models with balanced weight if the real world true distribution of recommended reviews might actually be highly skewed.

  Models with balanced weight are considered as successful since they outperform both the baseline model and all other non-baseline models in the testing set. Since we penalize mistakes on the minority class by an amount proportional to how under-represented it is, these models end up in similar performance in predicting not recommended as recommended. Within these models, the Logistic model slightly outperforms the Linear SVC model in all evaluation metrics on both balanced and unbalanced test sets.

- Downsample Training + Logistic Regression:

Since we need sufficient representation of both classes to learn well for the model, we decided to downsample the pairs with recommendations. By resampling the data to retain the balance, the model won't place a higher weight on the majority class. However, it can discard potential useful information which could be important for building the classifiers.

The advantages of the logistic classifier are that it makes no assumptions on the distribution of classes in feature space. On the other hand, this classifier takes less time to train and classify unknown records.

IV. Literature Studying Description

We acquired our dataset from the resource pages shared by the professor and we decided to only use two of the dataset among all of the steam bundle data. By working closely with similar datasets that contain review and user information in the past, we can gain a deeper comprehension of the recommender system. For instance, the book review and product review data we used in the previous assignment have similar features to our task of Review-Recommend Classifier. The datasets include attributes like user rating, review content, book genre, and other information that can help to reveal the customers' preferences over the products. Since a dataset like this contains textual information, we usually use natural language processing methods to analyze the text and build predictive models. TF-IDF vectorizer and Count vectorizer are the commonly used transformers for unstructured textual data. Preprocessing techniques like tokenization, stemming and lemmatization, and removal of stopwords and punctuations are helper tools to prepare readily available input for the scikit -learn models.

The paper "Predicting the Popularity of Games on Steam" focuses on predicting the popularity of games on Steam. By using a normal model and a folded normal model, they were able to obtain a conclusion from the data analysis that the measurement of popularity is the number of players in a period of time for each game. Our metric for the Game-Playtime Classification Task is similar since we look through the feature of mean/median playtime of the game and their effects on users' game playtime, which we kind of use the measurement of popularity to predict the game's playtime: the longer the game's global mean/median playtime is, the more playtime users will spend on the game. The paper also found out that games released at the beginning of the month and games of certain genres correlate with the game's popularity. In our analysis, we consider similar features like the genre of the game, the global mean/median playtime of the game, or the mean/median game playtime of the user. We notice that players are inclined to spend more time in games that are popular (higher global mean playtime is an excellent indicator). However, the genre factor in our model can bring some improvement from our baseline model but brings little or even a negative effect on the accuracy of our final predictions, which is demonstrated in detail in our Results and Conclusions section.

V. Results and Conclusions

Review-Recommend Classification Task Model Results:

- Model Comparisons:
  - Baseline Model:
    - Test Accuracy: 0.5000
    - Test F1 Score: 0.6667
    - Confusion Matrix: TN = 0, FP = 1037, FN = 0, TP = 1037
  - Custom Feature + Logistic Regression
    - Test Accuracy: 0.5005
    - Test F1 Score: 0.6668
    - Confusion Matrix: TN = 1, FP = 1036, FN = 0, TP = 1037
  - TF-IDF + Selected Classification Model:
    - Logistic Without Balanced Weight:
      - Imbalanced Validation Accuracy: 0.9058
      - Imbalanced Validation F1 Score: 0.9483
      - Imbalanced Validation ROC_AUC Score: 0.8912
      - Balanced Test Accuracy: 0.6692
      - Balanced Test F1 Score: 0.7483
      - Balanced Test ROC_AUC Score: 0.6692
      - Confusion Matrix: TN = 368, FP = 669, FN = 17, TP = 1020
    - **Logistic With Balanced Weight (Best Model)**:
      - Imbalanced Validation Accuracy: 0.8598
      - Imbalanced Validation F1 Score: 0.9168
      - Imbalanced Validation ROC_AUC Score: 0.8879
      - **Balanced Test Accuracy: 0.8100**
      - **Balanced Test F1 Score: 0.8285**
      - **Balanced Test ROC_AUC Score: 0.8100**
      - Confusion Matrix: TN = 728, FP = 309, FN = 85, TP = 952
    - Linear SVC Without Balanced Weight:
      - **Imbalanced Validation Accuracy: 0.9060**
      - **Imbalanced Validation F1 Score: 0.9484**
      - Imbalanced Validation ROC_AUC Score: 0.8863
      - Balanced Test Accuracy: 0.6760
      - Balanced Test F1 Score: 0.7522
      - Balanced Test ROC_AUC Score: 0.6760
      - Confusion Matrix: TN = 382, FP = 655, FN = 17, TP = 1020
    - Linear SVC With Balanced Weight:
      - Imbalanced Validation Accuracy: 0.8602
      - Imbalanced Validation F1 Score: 0.9174
      - Imbalanced Validation ROC_AUC Score: 0.8814
      - Balanced Test Accuracy: 0.7994
      - Balanced Test F1 Score: 0.8204
      - Balanced Test ROC_AUC Score: 0.7994
      - Confusion Matrix: TN = 708, FP = 329, FN = 87, TP = 950

- Downsample Training + Logistic Regression
    - Balanced Test Accuracy: 0.8006
    - Balanced Test F1 Score: 0.8030
    - Balanced Test ROC_AUC Score: 0.8005

Review-Recommend Classification Task Conclusion:

Overall, TF-IDF vectorized features with fine-tuned balanced logistic regression outperforms both the baseline and non-baseline models in testing datasets, achieving **0.8285 F1 score.** A high F1 score indicates the lack of problems with false positives and false negatives, meaning that the model did not ignore the input and always predicted the majority class.

TF-IDF encoded features defeat the custom features of text length and exclamation marks count which is practical in the sense that the actual word contains more latent information than the numerical properties does.

The model parameter C represents the inverse of regularization strength. Regularization will penalize the extreme parameters, the extreme values in the training data leads to overfitting. By setting **C = 2.0** in our best model, we place slightly more weight on the balanced training dataset.

The proposed best model succeeds not only because it outperforms other models in the balanced test set, but also its capability to maintain relatively high metric scores in the imbalanced validation set. Widely speaking, our best model can preserve its high performance in both settings: one with imbalanced review recommend ratio which is similar to the true distribution of the real world drawn from our best knowledge and background research, and one with an ideal balanced review ratio which is constructed to testify the model's ability in predicting minority class. Other models would not be considered as failing but inconsistent in their performance within different settings.

Game-Playtime Classification Task Model Results:

`Popularity Predictor Difference:`

Threshold	Mean	Median
`[0:==0, 1:<=45, 2:<=220, 3:<=840, 4:>840]`	0.354	0.418
`[0:<=45, 1:<=220, 2:<=840, 3:>840]`	0.412	0.394

Table 2: Test accuracy results of the Popularity Predictor Difference Model

Popularity Predictor Threshold:

Threshold	Mean	Median
`[0:==0, 1:<=45, 2:<=220, 3:<=840, 4:>840]`	0.260	0.427
`[0:<=45, 1:<=220, 2:<=840, 3:>840]`	0.383	0.426

Table 3: Test accuracy results of the Popularity Predictor Threshold Model

genre of item(total 21 genres of game), global mean playtime of the game, mean playtime of the user, and bias term
0.386
average playtime of item(total 21 genres of game), global mean playtime of the game, mean playtime of the user, and bias term
0.4024
genre of game + game review positive rate, global mean playtime of the game, median playtime of the user, and bias term
0.4293575885001257

Table 4: Linear Regression Model with different features

Key experiment	Features	Model	Train Acc	Train Loss	Test Acc
1	genre of game + game review positive rate + global mean playtime of the game + median playtime of the user + player item count	Two Linear + 100 unit hidden layers with RELU	0.48	1.152	0.473

2	Same as 1 but replacing category with user preference vector	same as 1	0.495	1.13	0.455
3	Same as 1 but replacing category with encoding of playtime per category	same as 1	0.505	1.10	0.435
4	same as 3 with regularization	same as 1 with dropout layer of 0.6	0.5221	1.10	0.459
6	All features in Table 1	same as 1 with double hidden units	0.563	1.05	0.482
7	**All features in Table 1 + mean play time**	**same as 6**	**0.5889**	**0.98**	**0.4884**

Table 5: Test accuracy results of the Deep Learning Model with different features

Game-Playtime Classification Task Model Conclusions:

Overall, simple heuristics like the median playtime performed relatively well compared to the trivial predictor, while heuristics that try to analyze the user's playtime performed poorly. Vectorizing various users and items features and feeding them into classifier models yielded a low accuracy like we had hoped for, which points to a limitation of the methods we had chosen. This could also be a limitation of the task we had chosen, since this type of classification relies heavily on human behavior and the variance from one person to another may be too high for us to generalize well with a machine learning model, as opposed to image classification or NLP. However, an accuracy close to 50% and a cross-entropy loss of around 1 would still mean we could reasonably estimate how much a person would spend time on a game they haven't played just by looking at their steam library.

Alternatively, we could have gone for more complex heuristics instead of supervised learning, such as Jaccard Similarity and other personalized recommender algorithms, which would require us to compare and consider user similarities and other interactions instead of focusing on the user and item pair. It is possible that exploration of these techniques could end up with a better result than the deepest network, which we, unfortunately, do not have the time to get to. Nonetheless, we hope our model could set the foundation for further improvements in our tested techniques and give some insights into the behavior of video game players.

VI. REFERENCES

[1] Wang-Cheng Kang, Julian McAuley. Self-attentive sequential recommendation *ICDM*, 2018

[2] Mengting Wan, Julian McAuley. Item recommendation on monotonic behavior chains *RecSys*, 2018

[3] Apurva Pathak, Kshitiz Gupta, Julian McAuley. Generating and personalizing bundle recommendations on Steam *SIGIR*, 2017

[4] Andraz De Luisa, Jan Hartman, David Nabergoj, Samo Pahor, Marko Rus, Bozhidar Stevanoski, Jure Demsar, Erik Strumbelj. Predicting the Popularity of Games on Steam, 2021