Video Game Review Sentiment to Popularity

Advisor: Amarnath Gupta
Robert Coughlin

Abstract

When a video game is reviewed positively as a great form of entertainment, one can make an educated guess that said game will sell well. If the hypothesis can be quantified into a machine learning problem and tested, it provides the potential for use by developers and marketers to interpret user feedback and determine if their game product is on track to meet sales milestones. To evaluate the hypothesis, the problem is simplified into a classification problem to determine if a review can indicate if the corresponding game would meet a maximum ownership threshold on the Steam game distribution platform of greater than, but not equal to, 500,000 users (such games henceforth referred to as bestsellers for conciseness). Snippets of review documents are then fitted and fed into a Gensim Doc2Vec model to convert them into document vectors to serve as inputs the classification learners can interpret.

The project uses web APIs to extract review data from OpenCritic, game lists from Steam, and sales approximations from SteamSpy to form a dataset with 15% of reviews belonging to bestseller games. Using this data, a logistic regression classifier has been constructed which has achieved a test accuracy of 0.86 and a test F1 of 0.58 using Gensim Doc2Vec to feed document vectors into a Random Forest Classifier. This pipeline served as part of a backend Python Flask server, which takes in user input from a web-based dashboard that sends either text or file contents containing review documents, and processes them into a prediction if the review(s) indicate the game will meet the bestseller threshold, followed by a percentage probability generated by the model to indicate confidence in the prediction. With this, customers can feed reviews into the model to generate predictions.

However, the model’s weakness is its F1, which represents a mediocre predictive power for positive cases. Presentation wise, it also does not provide explanation power as to why it arrived at its prediction.

Introduction and Question Formulation

Video games are part of a multi-billion-dollar economy with a large network of game developers, business investors and publishers who work to conceptualize, build, and market games to a community composed of millions of people. Included are the thousands of journalists, amateur and professional, who publish reviews which indicate to the community if the game is worth playing, and this information can both affect and be reflective of the sentiments of a game.

Focusing on the reflective aspect, the question then becomes: could the text of those reviews be reflective of how well a game sells in a measurable way, instead of just how good it is? This is not a trivial distinction as the development and publication of a video game is extraordinarily expensive and costs millions of dollars for a high-end studio to create one. In one outstanding case, the cost ballooned to as high as $316 million (Sinclair). If the game's sales cannot match the costs that go into creating it, it may not matter whether it is considered a high-quality game. Without the money to make up for the costs, developers risk losing their career, their business employer risks failing, and the organization may not be able to produce more games. Thus, selling well becomes a matter of making a profit or a return on investment and ensuring the business can continue operating, and being able to predict if their game can meet a certain sales threshold, it will help to make a more informed business decision.

For now, it is only a question of if such a sentiment exists and to what extent it could make an accurate prediction. The choice was made to simplify the problem as a binary classification problem, defining a class called "bestseller," which are games with more than 500,000 owners on Steam. This choice loses the opportunity to discover if reviews can give sales predictions within a certain margin but provides scoring metrics such as accuracy or F1 which are more easily interpretable for the purpose of answering if a sales sentiment is present and works in hand with NLP Python libraries that operate on class labels.

Related Work

"What makes the difference between popular games and unpopular games? Analysis of online game reviews from steam platform using word2vec and bass model." works with game popularity as well using a binary classification for popular and unpopular games, though the work uses Word2Vec to vectorize text keywords, and investigates an unsupervised learning problem to try and identify what makes a game popular based upon already classified data instead of predicting popularity.

"On the Robustness of Text Vectorizers." is related in that while it does not involve video games, it is a study of text vectorizers, a tool which was used extensively for analysis and as a part of the solution architecture. The conclusions it drew were that text vectorizers, among which includes Gensim Doc2Vec, are robust to the introduction of word replacements, with the error of a fixed number of replacements declining as the text in question becomes longer. This is an important consideration in the event of noisy and misspelled reviews and is thus a consideration which is included in analysis for verification.

Team Roles and Responsibilities

Robert Coughlin serves as the single member of the project, responsible for data acquisition, preparation, analysis, solution architecture implementation, and reporting.

However, when Group 6B was Group 6, components of the project were developed by Polina Hyracha, who provided the question and suggested the solution architecture, and implemented the data collection notebooks for extracting information from Steam and SteamSpy which have since only been lightly modified.

Data Acquisition

Data Sources

The data sources used for the project are Steam, SteamSpy, and OpenCritic.

Steam is a PC-based video game distribution software developed by Valve Corporation and is one of the most popular methods of purchasing games and game-related software applications through non-physical copies. Using Steam as a data source, it provides the scope of the project by narrowing down the games analyzed to a subset of games that are available on PC on the Steam platform and contains both application name and application ID data to connect the data sources. Through its API, about 18,000 games can be found a second, with over 100,000 Steam-based applications on the platform, and approximately 6,000 of them released in the year 2022 alone.

SteamSpy is a third-party online service that provides statistical data relating to 63,000-64,000 games on Steam which is not available directly from Steam's API. For the project, the statistic of most interest is the number of owners, which will serve as a measurement of the number of sales the game has achieved, serving as the output for supervised learning. Using the collection technique outlined in Data Collection, about 5,000 games' sales data can be retrieved every second.

Last is OpenCritic, a review aggregation website which collects reviews for video games across multiple platforms including the PC via a web crawler. According to the OpenCritic website FAQ, this process happens every 10-25 minutes. It provides data on review authorship, review scores, if the review is a recommendation, and a snippet of the review. The latter is of primary interest, as the snippets will be the inputs used for supervised learning. It is the slowest of the data sources which can be retrieved, requiring parallelization to try and speed up the process. It can perform approximately 20 searches a second when needing to search OpenCritic game IDs to match Steam IDs using the game title, and then with that data, the number of reviews retrieved hovers around 70 a second using the search data.

All data sources are collected as semi-structured JSON objects with known schemas. OpenCritic's data however includes nested object structures within them.

These are dynamic data sources, and as such may provide different data if the same data collection is performed at different points in time. The last data retrieval was performed between May 25th, 2023.

Data Collection

The data sources identified all have web APIs from which the data is collected. To make use of them, they are each called through a separate Jupyter notebook to access and process the data in Python.

For Steam, the requisition of games' Steam application IDs (referred interchangeably as Steam game IDs) is done through a URL-based request in the Steamworks Web API to retrieve a list of JSONs of all apps in Steam. This is an exhaustive list and includes Steam applications that do not qualify as video games, something that must be accounted for in data preparation.

Of the three, only the OpenCritic API required an API key to access. Due to the narrow throughput and the volume of data requests needed to extract the desired information, a premium subscription was required to attain the necessary throughput to gather the desired reviews. OpenCritic reviews cannot be accessed using the Steam app IDs, so instead the API's search interface is used to find the closest match via the inverse trigram distance between the title of the game returned from the Steam API. It is expected that a game in OpenCritic will be hit multiple times, and thus the Steam game with the shortest distance to a given OpenCritic game is selected for the match. Then the OpenCritic ID retrieved from the match is used to request the corresponding game's reviews. Each search and each game's reviews must be requested one at a time. To accelerate the process, this step is parallelized so multiple requests are made at once.

SteamSpy is the last data collection stage, making use of the OpenCritic data collection stage to identify Steam IDs which have found a match. Like with the Steamworks Web API, a list can be generated, but this is divided into pages of 1000 entries each. To resolve this, data collection is performed within a function that handles one page at a time, recursively calling itself to handle the next page until a desired number of random games are selected, or the function reaches a page with a number other than a thousand games (indicated it is the last)

In all cases the information is then converted from JSON or a collection of JSONs into a Pandas dataframe and then written to a local .csv file as a cache for further pre-processing, computations, and exploration.

As part of data preparation, the data is cleaned up and then uploaded into the SQL server. As of May 28th, the data currently cached and the data residing on the database server is represented by the following. Complications with data extraction and variable dates in which they were performed may have caused results that will be inconsistent with future runs in addition to the fact the data source is dynamic.

Source	Cache Size	Cache Entries	Database Size	Database Entries
Steam Games (Cache Only)	3.2 MB	105,051	N/A	N/A
SteamSpy Games & Sales	0.15 MB	10,250	0.68 MB	9,958
OpenCritic Reviews	122 MB	105,916	40 MB	85,620

Data Pipeline

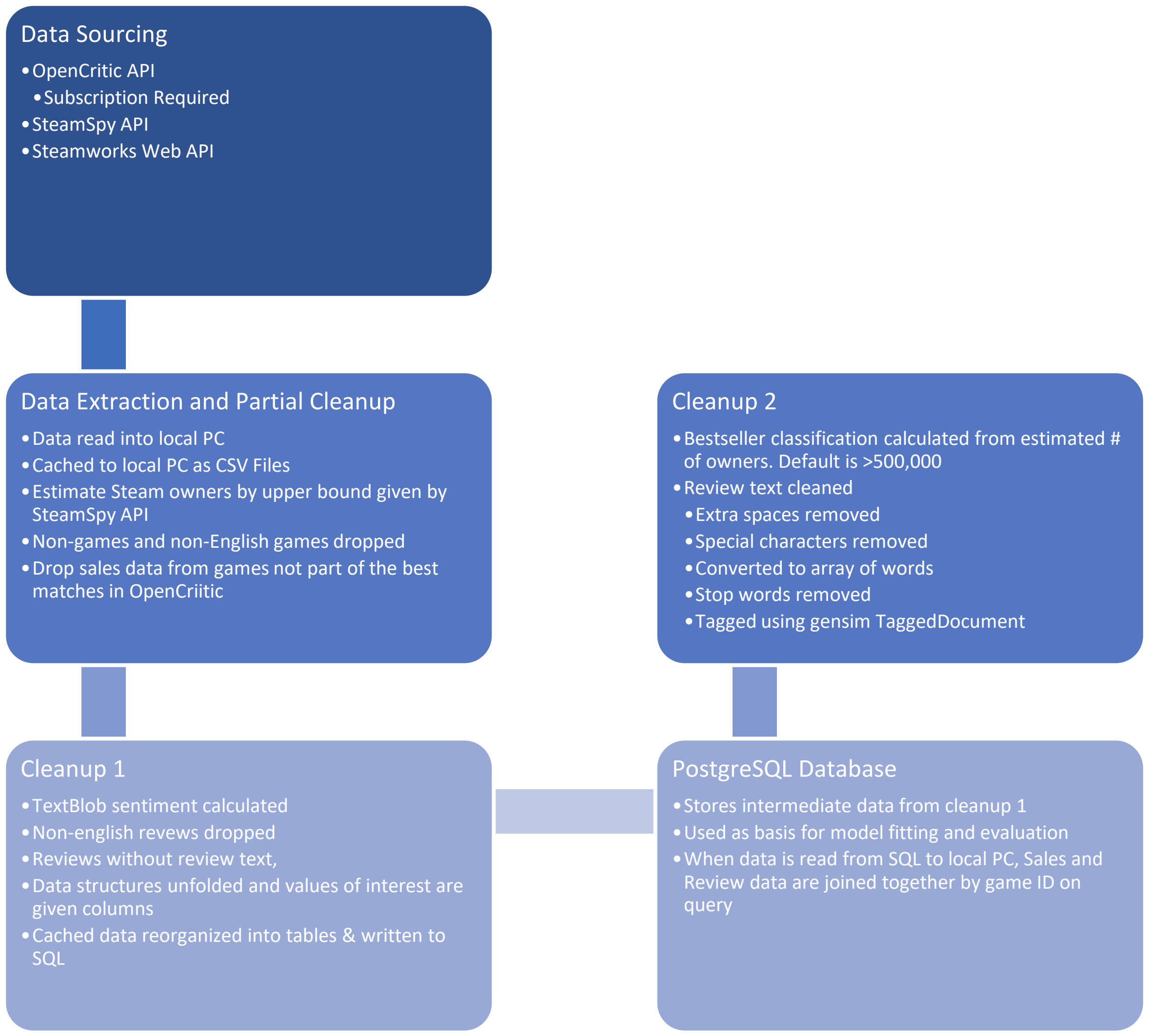

Data Sourcing
• OpenCritic API
• Subscription Required
• SteamSpy API
• Steamworks Web API
Data Extraction and Partial Cleanup
• Data read into local PC
• Cached to local PC as CSV Files
• Estimate Steam owners by upper bound given by SteamSpy API
• Non-games and non-English games dropped
• Drop sales data from games not part of the best matches in OpenCriitic
Cleanup 1
• TextBlob sentiment calculated
• Non-english revews dropped
• Reviews without review text,
• Data structures unfolded and values of interest are given columns
• Cached data reorganized into tables & written to SQL
PostgreSQL Database
• Stores intermediate data from cleanup 1
• Used as basis for model fitting and evaluation
• When data is read from SQL to local PC, Sales and Review data are joined together by game ID on query
Cleanup 2
• Bestseller classification calculated from estimated # of owners. Default is >500,000
• Review text cleaned
• Extra spaces removed
• Special characters removed
• Converted to array of words
• Stop words removed
• Tagged using gensim TaggedDocument

Figure 1. The full data pipeline. This includes steps from data collection down to the last step of cleanup during and before classification.

Data Environment

The environment is a hybrid of cloud databases and local flat files. The raw data extracted through the APIs are cached to the local file system within the project's Git repository with light data cleanup as outlined in the Data Pipeline section. This allows for quick refactoring of code for cleaning and engineering data without needing to repeatedly extract new data, which may not be the same as last time.

Initial cleanup is performed on the cached data, downsizing it, and extracting 1-dimensional features before uploading the information to a PostgreSQL database, from which the data is used by the learner to build the classification model. This architecture is a legacy of Group 6 with the goal of a common picture of data for the individuals working on the project. Its retention lies in the ease with which data can be combined between tables using SQL during queries, and its utility to act independently of the working computer, which can be useful if needing to deploy the dashboard server to new servers.

Data Preparation

Data preparation is done in three steps: during collection, during database data uploading, and during analysis. The process is done mostly within Pandas dataframes, as this allows for the easy management of features, dropping unwanted entries, and appending engineered features.

Data collection converts the JSON data from the APIs into .CSV files for the next stage of cleanup to read from. For OpenCritic, the raw data from the web API, including the nested objects, are kept as is inside the .CSV files, one file for the search results, and the other for review data.

Quality wise, the review texts from OpenCritic are mostly well formed and well spelled sentences. However, some reviews are missing, and NLP word analysis may not perform as well if fed just the raw strings.

To resolve this, text cleanup is performed before text is ever fed into NLP processors. Text cleanup will refer to the following steps: newlines, extra whitespace, and special characters are removed, and the text is set to lowercase. Then the string is split into a word list with special stop words dropped and the remaining words stemmatized.

In both data collection and data uploading, the cached OpenCritic search results are used, and the Steam game with the lowest inverse trigram distance to a given OpenCritic game ID is used. These will be referred to as the best search results.

During collection, the sales data from SteamSpy is not collected as a single integer. This is due to SteamSpy having imprecise data on Steam sales data. Instead, a string range is provided with a maximum and minimum delineated using two dots as well as comma separators in the numbers (i.e., “500,000 .. 1,000,000”). For analysis in the project, the higher number is selected and converted to an integer. As previously stated in Data Collection, the game ID for SteamSpy sales which are not part of the best search results are dropped before the sales data is cached to the local PC. Of the data collected, only the columns for Steam app ID and the sales estimate are retained.

The data collector also drops Steam apps which are either not games or are in a foreign language. Both are determined based upon keywords in the app name. Only the app name and the app ID are retained during caching.

In the second stage of cleanup, before the data is uploaded to the PSQL database, the OpenCritic data is dropped from reviews with a missing score, missing description, missing NP score, or if it is in a non-English language. Any duplicate reviews are also removed and is non-deterministic on which duplicate is removed.

The nested objects from the cache are not easily transferable to SQL, and so they are unwrapped and the OpenCritic game ID within is mapped to the best search results to associate a Steam app ID with each review. A temporary text cleanup of each review snippet is performed and then fed into TextBlob's sentiment scorer to output the polarity and subjectivity of the data (Shah). Lastly, the data columns of interest are selected, renamed, and uploaded to the database while others are dropped. The columns related to the analysis in use are the review IDs, the summary/review snippets, and the Steam app ID.

The Steam sales data is uploaded as is from the cache, with any duplicate information dropped if present.

The final data preparation takes place during modeling. Using the SQL query language, the reviews and sales data are joined along the Steam app ID to associate the games' sales data to each of their reviews. The bestseller classification is then computed, generating a boolean on if the game has more than 500,000 Steam owners.

Because we are only concerned with the contents of the text, it and any sentiment values derived are the only features selected from review beyond review ID and game ID for identification purposes. However, classifiers cannot accept text directly as an input, and exploration has determined the sentiment features created during the second stage are insufficient. The last engineered feature, created dynamically during analysis, is a document vector transformer called Gensim Doc2Vec. Document vectors are n-dimensional numeric vectors that serve as compressed representations of a document, which in this context means the full snippet text after it is fully pre-processed. The vectors add the necessary dimensionality to the review text for the classification learners to try and determine if the class labels can be discerned.

This step is not included in previous cleanup phases as the primary document to the vector class provided by Gensim will be modeled alongside the classifier. This is done by performing text cleanup, then wrapping the cleaned-up text into a Gensim tagged document with the tag in question true/false for if the review is for a bestseller. The inclusion of appropriate tags will influence the document vector to attain values closer to those with the same tag.

This training will be done using only documents from the training set whenever data is split using train-test split. For all other documents (validation, test, dashboard input), the trained Doc2Vec model will infer the vector without referring to the document tag, as the tag would not be known for unlabeled text. Selection of the number of vector features for the Doc2Vec will be performed during Hyperparameter Tuning Analysis.

Analysis

Preliminary Analysis

For analysis, the classification pipeline will refer to a scikit-learn Pipeline object composed of the following: a custom wrapper class which allows Gensim Doc2Vec to function within an Scikit-learn pipeline and expose some of its hyperparameters to the Pipeline class, a min-max scaler which normalizes the data to the range [0,1] using training data to set the scale (generally from [-1,1] to [0,1]), and a classifier which is interchangeable depending on the test case. The use of a Doc2Vec instead of a Word2Vec is due to the need to analyze the document rather than keywords during classification, using document tagging to attach a label to the documents during fitting (Li).

Preliminary analysis was performed using accuracy and F1 to measure the pipeline's performance by splitting the data into test and training sets, then fitting the pipeline using logistic regression as the classifier, modifying only the C-value. It was found that the model was extremely accurate but was unable to predict any positive values with an F1 near zero. This led to counting the values for bestsellers directly, and it was found that the number of positive cases is heavily outweighed by the number of negative cases (8% to 15% depending on the collection date).

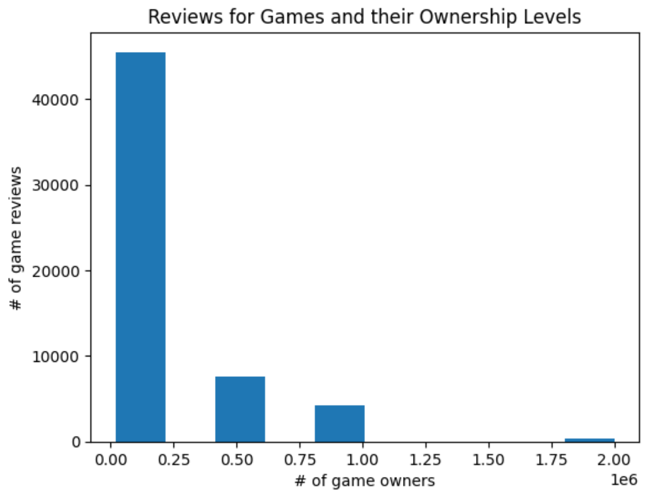


Figure 2. A histogram of game reviews based upon the number of owners for each review's game. This graph was generated with an older extract for steam games, but nonetheless is the visualization for which it was decided 500,000 represented an ideal cutoff point to indicate games that sold exceptionally well while balanced against the need for sufficient positive cases.

This led to performing the same test again, this time by downsampling the number of negative cases to match the number of positive cases, leading to an accuracy and F1 of both around 0.75. From there it became the baseline starting point for the three analyses performed below.

Later, during the analysis, however, it was determined that downsampling was the incorrect choice as it distorts the expected proportion of reviews which are expected to be received in the real world, especially as they are expected to be unlabeled. Thus, downsampling was reversed, and the process has been reworked to account for the full imbalance dataset. In addition, Doc2Vec was found to be non-deterministic in its outputs, resulting in changing scores in every re-run. This led to the choice to obtain averages for most of the data by obtaining the test score more than once.

Because of how high the unweighted accuracy of the model was for the imbalanced dataset, the primary question was no longer if it is accurate, but if it can predict positive cases well enough.

Analysis Techniques

The analysis techniques used for the project were simple comparisons of different parameters and preparation techniques for the data before and after it is processed by the classifier to try and improve the test accuracy, precision, recall, and F1 scores of the outputs for the classification pipeline shown below. This forms the reasoning for choosing to convert the numeric regression problem of prediction the number of owners to prediction classification along an ownership threshold as an easier means to compute accuracy alongside the rationale of simplifying the prediction problem.

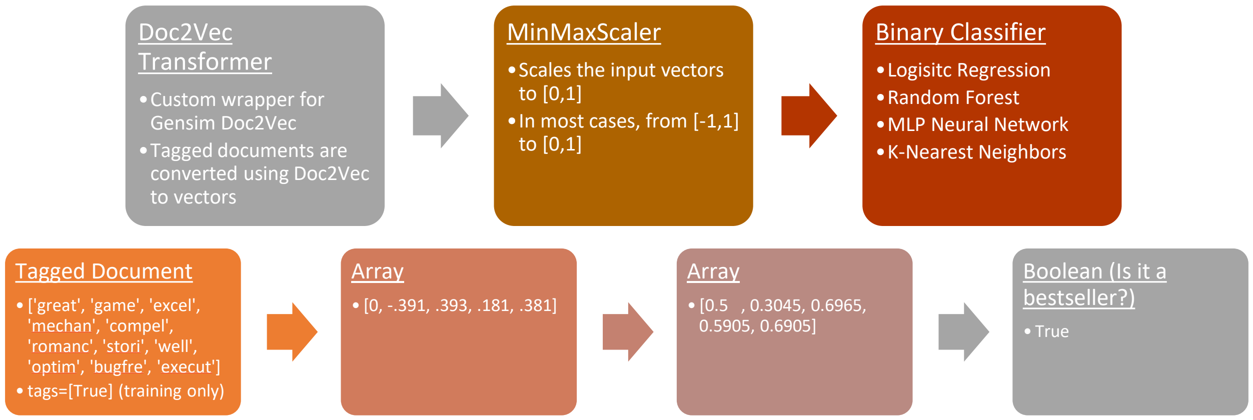

Doc2Vec Transformer
Custom wrapper for Gensim Doc2Vec
Tagged documents are converted using Doc2Vec to vectors
MinMaxScaler
Scales the input vectors to [0,1]
In most cases, from [-1,1] to [0,1]
Binary Classifier
Logisitc Regression
Random Forest
MLP Neural Network
K-Nearest Neighbors
Tagged Document
['great', 'game', 'excel', 'mechan', 'compel', 'romanc', 'stori', 'well', 'optim', 'bugfre', 'execut']
tags=[True] (training only)
Array
[0, -.391, .393, .181, .381]
Array
[0.5 , 0.3045, 0.6965, 0.5905, 0.6905]
Boolean (Is it a bestseller?)
True

Figure 3. A flowchart of the classification pipeline (top), which includes feature engineering as a part of the process. The classifier learner is interchangeable with any of the four learners listed inside the textbox. Below is a state of the data at each stage of the Pipeline.

The scores are obtained by fitting the classification pipeline to training data, then predicting the labels on the test data and comparing them to the real labels. Because the scores are used for model selection through both automated and manual selection, they form the endpoint of all the following analysis tests.

Four analytical workflows were created during the project, each dealing with a different facet of the process workflow: hyperparameter tuning, data preparation alternatives, data scaling and data aggregation. In aggregate, the analyses seek to determine which of the four classifiers selected for analysis–logistic regression, random forest, multilayer perceptron, and k-nearest neighbors–in combination with different preparation techniques, will perform best, with an additional analysis evaluating how well the pipeline performs under scaling and robustness tests.

Applying the techniques involved changing the parameters and processes of the pipeline, how data is prepared, how data is sampled or split, and what hyperparameters are used. Then the training data is fit to the classification pipeline to learn a new classifier, which is then evaluated via the test data that is outputted to determine the changes in performance. Training and testing data is split 60% to 40% (60-40) unless otherwise specified.

Processing Environment

The execution environment is within an Anaconda distribution on Windows. Extraction, cleanup, analysis, and model learning is performed within Jupyter Notebooks executed either through Jupyter Lab or the Visual Studio Code application on Anaconda. VS Code is also the application whose terminal and debug tools are used to build and execute the dashboard solution. Each of the below subsections are run on one independent Jupyter notebook each and can be executed from start to finish. Hyperparameter Tuning must be performed before Scaling and Robustness and Data Aggregation workflows.

Hyperparameter Tuning

Hyperparameter tuning seeks to alter the hyperparameters of Gensim Doc2Vec and the classifier in use to improve performance without changing the inputs or the output format beyond the standard data pipeline.

Analysis of optimal parameters was performed using Scikit-learn GridSearchCVs. This allows for the automatic and exhaustive testing of various combinations of hyperparameters on the various classifiers being evaluated. Each grid search performs a 3-fold cross-validation with F1 being the score used to automatically evaluate the hyperparameters using the same training and test data to generate scores. The GridSearchCVs will then record which are the best hyperparameters and its validation F1, then the test data is fed into the best model to generate predictions to get the best F1, which will be the only validation score used in this workflow.

There are also two additional tests to evaluate feature selection and balance. The first introduces a linear support vector classifier (SVC) to scikit-learn’s SelectFromModel feature selector, which is then introduced between the min-max scaler and the primary classifier to try and narrow the vector elements down to those which are significant, reduce overfitting, and increase test performance. The SVC and SelectFromModel use default hyperparameters, and the main classifier uses the best hyperparameters found from tuning the logistic regression pipeline.

The last test is if giving disproportionate weight to positive cases on the linear classification pipeline instead of a balanced weight will improve the F1 of the pipeline.

Once all the pipelines are fitted to their best hyperparameters, each generates a single set of test scores at the end of the workflow. Only the cross-validation, and the corresponding validation F1, will display a mean finding.

Learner Tuned	**Hyperparameters**	**Additional Notes**
Doc2Vec	'dm': [0, 1], 'dbow_words': [0, 1], 'vector_size': [50, 200], 'min_count': [2, 5], 'negative' : [0, 5, 10], 'hs': [0, 1]	Uses logistic regression with default hyperparameters except class_weights=‘balanced’ Best results used for Doc2Vec pipeline are added to the parameters for remaining grid searches

Logistic Regression	{
	'clf__C': [1e-3, 1e-2, 1e-1,1, 1e2, 1e3, 1e4, 1e5],
	'clf__penalty': ['l2'],
	'clf__solver': ['lbfgs']
	},
	{'clf__C': [1e-3, 1e-2, 1e-1, 1, 1e2, 1e3, 1e4, 1e5],
	'clf__penalty': ['l1', 'l2'],
	'clf__solver': ['liblinear']
	}
		Limited-memory BFGS (LBFGS) cannot be used with an L1 penalty, and thus uses its own set of hyperparameters to distinguish penalty sets.
		
		The classifier also has the following hyperparameter included: class_weights=‘balanced’

Random Forest	'clf__n_estimators': [50,100,150,200], 'clf__criterion': ['gini','entropy', 'log_loss'], 'clf__class_weight': ["balanced","balanced_subsample"], 'clf__max_depth': [None, 5, 10, 15, 20]	
Multi-Layer Perceptron	'clf__hidden_layer_sizes': [(10,30,10),(20,),(100,300,100),(50,100,100,50)], 'clf__activation': ['tanh', 'relu'], 'clf__solver': ['sgd', 'adam'], 'clf__alpha': [0.0001, 0.05], 'clf__learning_rate': ['constant', 'adaptive']	Parameters derived from “scikit learn hyperparameter optimization for MLPClassifier.”” (Panjeh)
K-Nearest Neighbors	'doc2vec__vector_size': [50, 100, 200], 'clf__n_neighbors': [2, 4, 6, 8, 10], 'clf__weights': ['uniform', 'distance']	To increase the number of cases investigated, vector size for Doc2Vec was modified outside of the choice from the first test to tune to different vector sizes
Linear Regression w/ SelectFromModel LinearSVC feature selection	None	Uses the best hyperparameters from linear regression case

Linear Regression w/ varying weights	'clf__class_weight': [None, "balanced", weight1, weight2, weight3]
	
	weight1 = {1: 100 ,0: 1}
	weight2 = {1: 50 ,0: 1}
	weight3 = {1: 200 ,0: 1}
		Uses the best hyperparameters from linear regression case
		
		0 = False
		1 = True

The outputs from this workflow specifically are used to generate hyperparameters for the solution architecture.

Data Preparation Alternatives

This workflow sought to test various techniques of data preparation beyond what was outlined in the data pipeline to improve the inputs to the data pipeline without having to modify hyperparameters. Improvements can then be added to hyperparameter tuning (if any), creating a better solution for the end-user.

First is by simply re-splitting the same data five times to get new training and test data for each test, to assess if scores are an artifact of the manner data was split at the time of test. This will also be the baseline of the workflow, as the same technique will be applied to other data prep analyses.

Second is appending the previously retrieved sentiment and polarity values during data preparation to the document vectors, to assess if the two dimensions contain independently varying information that can inform the learner with information not in the vectors. The third way excludes the two sentiment values and instead aggregates the input vectors along the game ID, to assess if the man of the input vectors will provide more accurate information and control for the fact that reviews for individual games can express differing opinions.

The last data prep analysis is from substituting Gensim Doc2Vec with spaCy's en_core_web_sm model for generating vectors, first by combining sentence vectors into a mean, then by generating a doc vector using default settings for spaCy's class constructor.

Each data preparation technique will be fed into the four classifiers chosen for analysis. The exception will be the spaCy data prep method, which will only use the Random Forest and Logistic Regression classifiers. Each case will generate average scores from the 5 re-splits used to generate the training and test data to control variation in the scores, and those same scores will be compared within each learner type using Welch's t-test to establish statistical significance against the first resampling with no modifications.

Scaling and Robustness Analysis

The scaling and robustness workflow seeks to determine how well the model could be scaled up to receive new data, and handle minor to moderate errors in the review text. The classification pipeline for this workflow is the logistic regression pipeline with the best hyperparameters in findings.

It should be noted that this workflow was performed using downsampling of the negative cases using an older review dataset with an 8% positive label composition. The findings from this section were used to update the workflow in other sections. Thus, the hyperparameters used for the classification pipeline and the results for the baseline are expected to be very different from the other workflows.

Because the data was smaller, the training and test data split was 80-20 to allow enough training data to be part of the fitting process. The test data is then transformed into predictions and evaluated 5 times (as Doc2Vec is non-deterministic) to attain an average. The classification pipeline is tuned to the best hyperparameters for logistic regression as found in hyperparameter tuning.

Analyzing the model's scalability was done in multiple ways. First, by reducing the classification threshold from > 500,000 owners to >= 500,000 owners, which changes the proportion of positive cases in the dataset. A second control group is formed where the training data and test data are still the same size as before.

With the same balanced subset, the data is split so that the training data is the same absolute size as the control groups to verify if the model performance does not drop off as more unlabeled data of similar quality as before is added. Next, the train-test data is 80-20 to determine if adding more training data will improve the scores with the new classification threshold.

A final scalability test is to use the balanced training data from the first control to fit the model and use an imbalance test set with the same proportion as the entire dataset by appending negative data points.

Because a control group was used for this analysis unlike the previous workflows, the multiple score samplings taken for each test undergo a Welch's t-test to validate if the changes between cases and the control groups are statistically significant.

Robustness was tested by scrambling 4 characters in each review document, swapping two sets of adjacent words per document, scrambling 20 characters in every 5 documents, and finally a combination of all the above. Character scrambling is applied before text cleanup to verify that cleanup can handle the introduction of this type of noise, and word swap will occur after text cleanup and before Doc2Vec conversion, as the documents would already be organized into word lists at that point and would be no different if the change was introduced earlier.

These four cases will have outputs predicted on two different types of pipelines: the control pipeline, and an individual pipeline fitted with the training data with the corresponding type of noise introduced to them. All test data will receive the noise corresponding to their test case.

Because the workflow was concerned with precisely where among precision and recall the models are affected and not the overall performance, F1 was omitted.

Data Aggregation

Aggregating data outputs seeks to achieve a similar aim as the third data prep case from the above section, but training data is fitted to the pipeline as they normally are. Instead, test data is aggregated in one of two ways: by concatenating the review texts together to generate one prediction per game, or by averaging the prediction probabilities for the game from every review, then vote for the class label with the highest probability.

These aggregation analyses are designed to determine if combining review data for each game and considering how each review can express different potential outcomes can create an aggregate picture that more accurately reflects the bestseller state of the game than if reviews were looked at individually.

The classification pipeline used will include a logistic regression classifier using the best hyperparameters in the Hyperparameter Tuning findings and balanced class weights. Twenty re-evaluations of the test set will be done for each case to account for non-deterministic outputs, then they will run against the non-aggregated test output using Welch's t-test to determine if there is a statistically significant difference.

Findings and Reporting

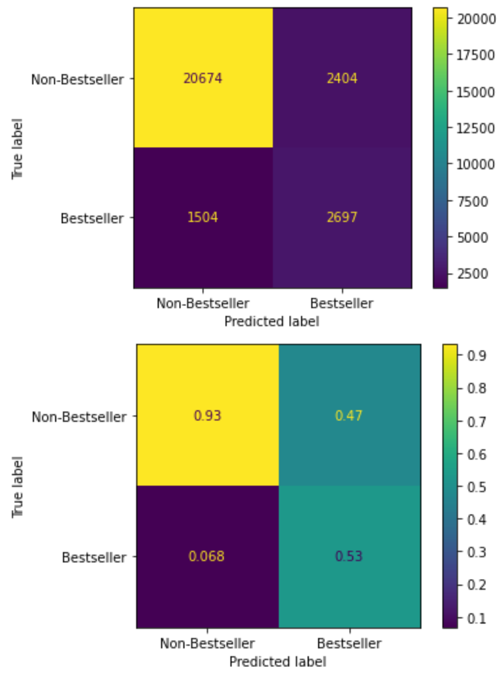
The scores represent the average of multiple scores evaluated for each model.

Hyperparameter Tuning

The test F1 indicates Gensim Doc2Vec, combined with a Random Forest classifier, seems to be the best performing model during hyperparameter tuning. Using linear SVM for feature selection and changing weights manually does not seem to provide any improved results.

Learner Tuned	**Best Hyperparameters**	**Valid. F1**	**Test Acc.**	**Test Prec.**	**Test Recall**	**Test F1**	**Additional Notes**
Doc2Vec	'dm': [0], 'dbow_words': [0], 'vector_size': [50], 'min_count': [2], 'negative' : [10], 'hs': [0]	0.546	0.826	0.452	0.720	0.556	Uses logistic regression with default hyperparameters except class_weights=‘balanced’
Logistic Regression	'clf__C': [0.001], 'clf__penalty': ['l2'], 'clf__solver': ['lbfgs']	0.547	0.835	0.475	0.670	0.556	Uses logistic regression with default hyperparameters except class_weights=‘balanced’
Random Forest	'clf__class_weight': [‘balanced_subsample’], 'clf__n_estimators': [150], 'clf__criterion': ['entropy"], 'clf__max_depth': [20]	0.572	0.858	0.531	0.644	0.582	
Multi-Layer Perceptron	'clf__hidden_layer_sizes': [(100,300,100)], 'clf__activation': ['relu'], 'clf__solver': ['adam'], 'clf__alpha': [0.05], 'clf__learning_rate': ['constant']	0.568	0.882	0.667	0.474	0.554	
K-Nearest Neighbors	'doc2vec__vector_size': [200], 'clf__n_neighbors': [8], 'clf__weights': ['distance']	0.549	0.869	0.581	0.557	0.568	

Linear Regression w/ SelectFrom Model LinearSVC feature selection	None	0.548	0.835	0.475	0.672	0.557	
Linear Regression w/ varying weights	'clf__class_weight': ["balanced"]	0.544	0.830	0.667	0.548	0.548	



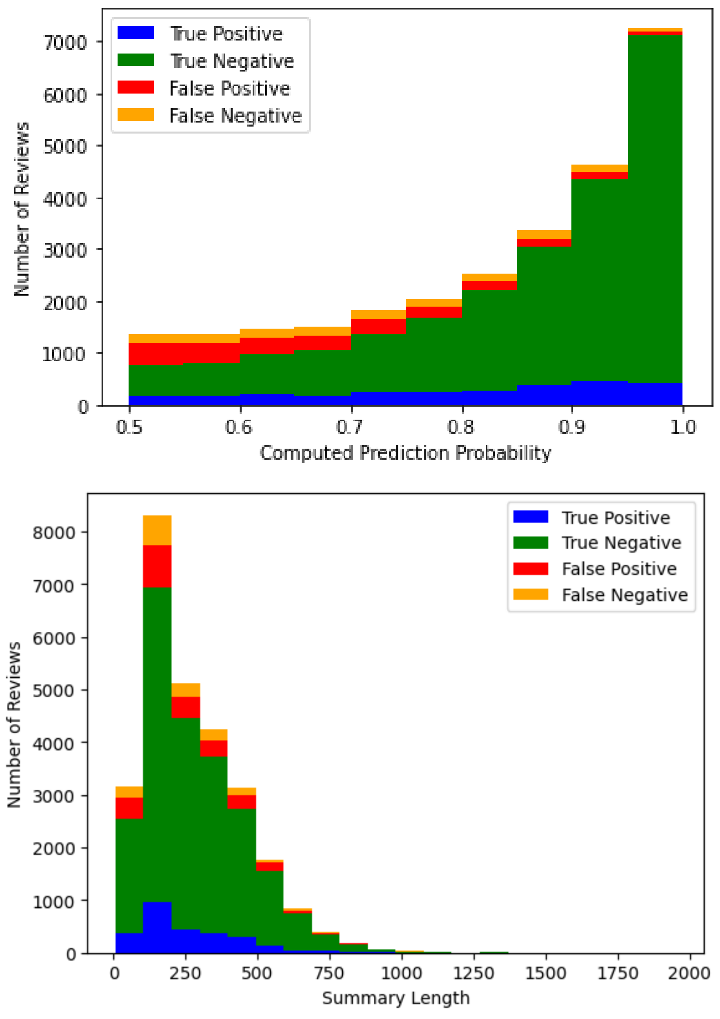


Figure 4.1. Histogram of the predicted and actual classes (right) when organized by prediction probability (top) and review character length (bottom)–as well as a confusion matrix of the predictions by absolute value (top left) and normalized by prediction (bottom left)–of a random forest classifier, which has the best F1. Note that as prediction probability increases, the number of false classifications tends to decrease.

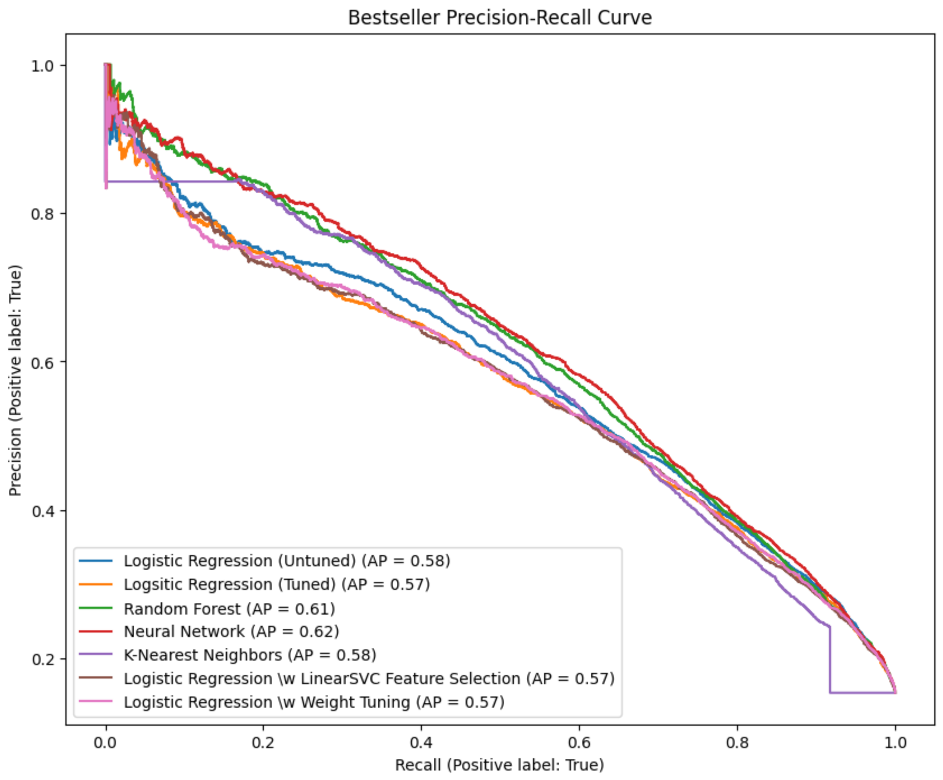


Figure 4.2. Precision-recall curve of the best estimators for each of the cases tested. Note that Neural Network (MLP), Random Forest, and K-Nearest neighbors performed better than the logistic regression implementations.

Data Preparation Alternatives

None of the alternative data preparation methods evaluated in this workflow appear to provide a consistent improvement in model performance to justify their addition to the data pipeline, with spaCy's model in particular eliminating almost any predictive power for the classification pipeline. p values are very high in almost any case, making any improvement questionable at best.

Data Prep	Model	Test Accuracy	Test Precision	Test Recall	Test F1	Additional Notes
Baseline Resampling	Logistic Regression	0.791	0.402	0.731	0.519	Uses logistic regression for classifier
	RF	0.847	0.894	0.0104	0.0206	
	MLP	0.872	0.618	0.479	0.535	
	KNN	0.838	0.384	0.0798	0.132	
Sentiment Score	Logistic Regression	0.79 p = 0.9	0.401 p = 0.85	0.728 p = 0.462	0.518 p = 0.633	

	RF	0.847 $p = 0.826$	0.884 $p = 0.64$	0.0124 $p = 0.107$	0.0244 $p = 0.107$	
	MLP	0.878 $p = 0.153$	0.667 $p = 0.131$	0.427 $p = 0.154$	0.519 $p = 0.28$	
	KNN	0.838 $p = 0.692$	0.379 $p = 0.561$	0.0801 $p = 0.917$	0.132 $p = 0.983$	
Aggregate	Logistic Regression	0.790 $p = 0.752$	0.401 $p = 0.776$	0.73 $p = 0.728$	0.518 $p = 0.583$	
	RF	0.847 $p = 0.914$	0.901 $p = 0.796$	0.011 $p = 0.485$	0.022 $p = 0.486$	
	MLP	0.874 $p = 0.718$	0.633 $p = 0.679$	0.466 $p = 0.751$	0.532 $p = 0.82$	
	KNN	0.838 $p = 0.784$	0.388 $p = 0.664$	0.0805 $p = 0.837$	0.133 $p = 0.812$	
spaCy Aggregate Sentence	Logistic Regression	0.589 $p < 0.01$	0.191 $p < 0.01$	0.514 $p < 0.01$	0.278 $p < 0.01$	
	RF	0.846 $p = 0.206$	0.513 $p < 0.01$	0.00228 $p < 0.01$	0.00454 $p < 0.01$	
spaCy Doc	Logistic Regression	0.602 $p < 0.01$	0.195 $p < 0.01$	0.507 $p < 0.01$	0.282 $p < 0.01$	
	RF	0.846 $p = 0.204$	0.508 $p < 0.01$	0.00233 $p < 0.01$	0.00464 $p < 0.01$	

Scaling and Robustness Analysis

The scaling analysis indicated that proportionally increasing data did not cause a significant drop-off in performance, with precision being statistically negligible. However, the use of a balanced subset to train the model caused a significant drop-off in precision, making the continued practice unviable when confronted with real-world proportions of unlabeled data. The scaling findings informed the revision of the other workflows to consider imbalance data reflective of the real world.

The following is data previously collected from Report #8. The p-values for the t-test against the second control will be bolded.

Scaling Case	**Test Accuracy**	**Test Precision**	**Test Recall**
Control 1 (> 500,000 owners)	0.761	0.749	0.762
Control 2 (>= 500,000 owners)	0.692 ($p < 0.01$)	0.666 ($p < 0.01$)	0.725 ($p < 0.01$)

>= 500,000, balanced, small training set, big test set	0.694 (p < 0.01) **(p < 0.01)**	0.694 (p < 0.01) **(p < 0.75)**	0.694 (p < 0.01) **(p < 0.01)**
>= 500,000, balanced, 80% training set, 20% test set	0.756 (p < 0.05) **(p < 0.01)**	0.760 (p < 0.01) **(p < 0.01)**	0.747 (p < 0.1) **(p < 0.01)**
Real Imbalanced (> 500,000 owners)	0.755 (p = 0.021)	0.203 (p < 0.01)	0.765 (p = 0.183)

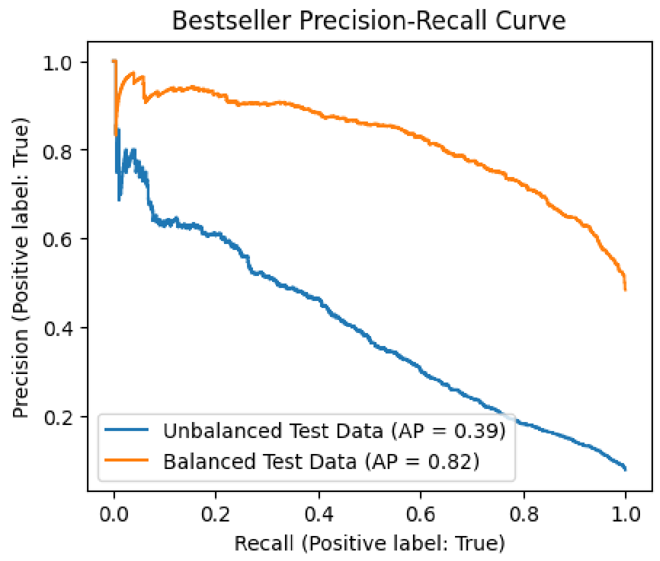


Figure 5. The precision-recall curve of the control pipeline when given an imbalanced and a balanced test set. The curves indicate a massive drop in performance when classifying the bestseller class.

The model pipeline was found to be robust in the event of minor misspellings or rare cases of many misspellings and is unaffected by a small number of word swaps in a statistically significant manner. Even if trained on the noisy data, the models retain some of their performance. The case when there is a major drop-off is if all three cases used were combined and included in the training data.

The following is data previously collected from Report #8.

	Predictions made on control model			**Predictions made on model fit to noisy data from the same case**		
Robustness Case	**Test Accuracy**	**Test Precision**	**Test Recall**	**Test Accuracy**	**Test Precision**	**Test Recall**
Control 1	0.762	0.744	0.753	0.762	0.744	0.753
4-character scrambles in all reviews	0.725 (p > 0.01)	0.718 (p > 0.01)	0.713 (p > 0.01)	0.738 (p > 0.01)	0.722 (p > 0.01)	0.746 (p > 0.01)
20-character scrambles in 1 out of 5 reviews	0.747 (p > 0.01)	0.732 (p > 0.01)	0.751 (p = 0.61)	0.740 (p > 0.01)	0.725 (p > 0.01)	0.748 (p = 0.50)

2-word swaps in all reviews	0.756 (p = 0.89)	0.745 (p = 0.87)	0.754 (p = 0.96)	0.764 (p = 0.68)	0.744 (p = 0.76)	0.780 (p = 0.70)
All cases combined	0.723 (p > 0.01)	0.712 (p > 0.01)	0.720 (p > 0.01)	0.69 (p > 0.01)	0.67 (p > 0.01)	0.722 (p > 0.01)

Data Aggregation

The two methods of aggregation evaluated in this workflow appear to have marginally increased the F1, though concatenating reviews caused a drop in precision while voting has increased Precision, Recall, and F1 at the same time, making it the better choice of aggregation and an improvement over looking at singular reviews. The following data was collected from Report #9.

	Single Review	Concatenated Review		Vote by Probability Sum	
Statistic	Scores	Scores	P-value from single review	Scores	P-value from single review
Accuracy	0.844	0.831	> 0.01	0.886	> 0.01
Precision	0.495	0.417	> 0.01	0.543	> 0.01
Recall	0.632	0.797	> 0.01	0.706	> 0.01
F1	0.555	0.547	> 0.01	0.614	> 0.01

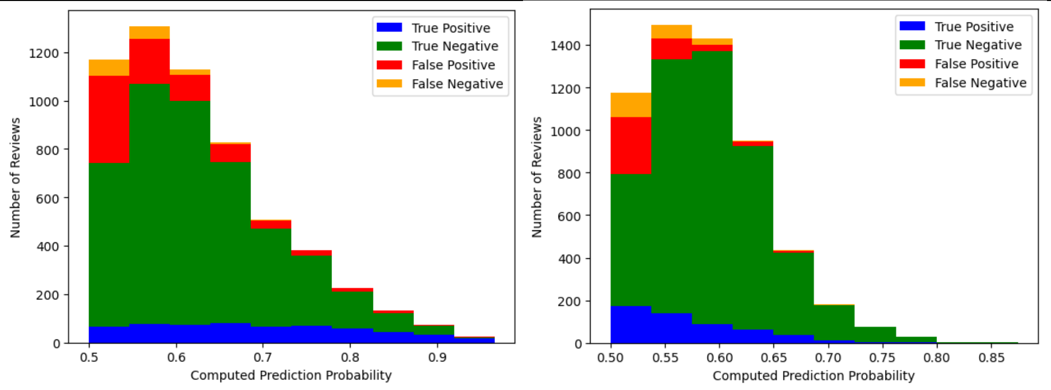


Figure 6. Histogram of the predicted and actual classes of a logistic regression classifier when predicting class labels from combining the review text (left) or the prediction probabilities of reviews (right) for each game. Note that the prediction count skews between 0.55 and 0.60 instead of towards 0.90 as in **Figure 4**. Again, as prediction probability increases, the number of false classifications tends to decrease.

Reporting

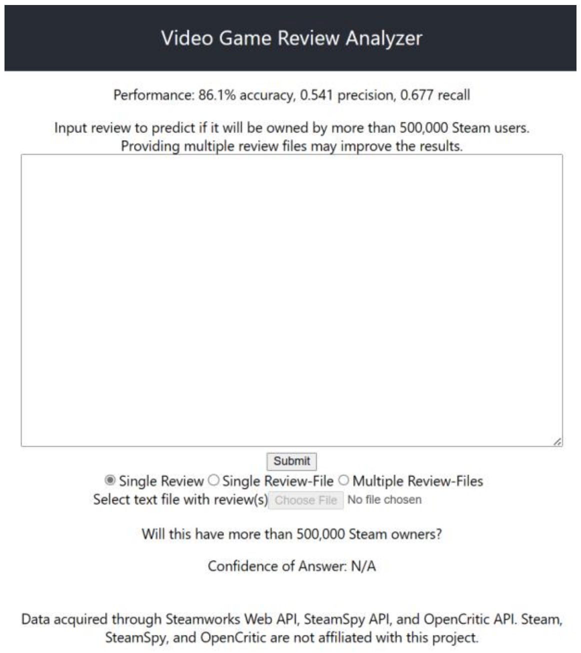
Few of the findings are presented directly to the customer, and instead inform the project for evaluating the models and refining the analysis. These findings were reported in the paper as a matter of performance evaluation, model tuning, and development of the solution architecture.

Reporting the accuracy allows for measuring the overall performance of each case for modifying the model, its inputs, or outputs, while the F1, precision, and recall helped to indicate which performed best at predicting the positive cases.

What is reported to a customer directly depends upon the customer in question. For IT technicians, whose job is to maintain the model as part of the server hosting the solution architecture, they are presented with the various scores given in the findings if they run the notebooks, and the best hyperparameters are each exported to a JSON file as part of the notebook workflow for Hyperparameter Tuning. The JSON reports are of greatest interest to the technicians, as these hyperparameters can be selected in a config file and then used in generating a new classification pipeline whenever the reporting dashboard's server is enabled. Using the score findings, particularly for F1, helps inform them which model to choose from to have the best chance at identifying bestsellers.

For business customers, such as game developers, marketers, and publishing organizations, most of the previous findings are not presented to these customers directly. Their priority is not how the product functions in the background nor how it was developed, but instead on what it produces. In this case: a true/false classification label if text describing a game with unknown ownership levels is likely to have more than 500,000 owners on steam.

To present this reporting, a dashboard is created to service predictions for new inputs. The dashboard is created using React.js to construct the user interface and is hosted on a Python Flask server to take advantage of the experience developed from the analysis to construct a new model used exclusively for the dashboard (Reine). The model in question is selected within a configuration file, and its hyperparameters are selected from a JSON file generated from Hyperparameter Tuning. The server and the database could be serviced by the business's IT department, or by a separate party who services multiple businesses.


Video Game Review Analyzer

Performance: 86.1% accuracy, 0.541 precision, 0.677 recall

Input review to predict if it will be owned by more than 500,000 Steam users.
Providing multiple review files may improve the results.

[✓ Single Review] Single Review-File Multiple Review-Files

Select text file with review(s) No file chosen

Will this have more than 500,000 Steam owners?

Confidence of Answer: N/A

Data acquired through Steamworks Web API, SteamSpy API, and OpenCritic API. Steam, SteamSpy, and OpenCritic are not affiliated with this project.

Figure 7. The frontend of the Video Game Review Sentiment to Popularity Dashboard, shortened to "Video Game Review Analyzer." Note the presence of options to either submit text within the given textbox or submit multiple text files for a game to compute the prediction.

The dashboard can take either text or a .txt file and use it to generate a prediction. Based upon findings from Data Aggregation, it was decided to include the option to import multiple review files, which will then have their predictions aggregated by vote using the technique described Data Aggregation Analysis Methods. At the bottom of the page, the prediction will be displayed, and the prediction's probability as described by the classification pipeline will be displayed.

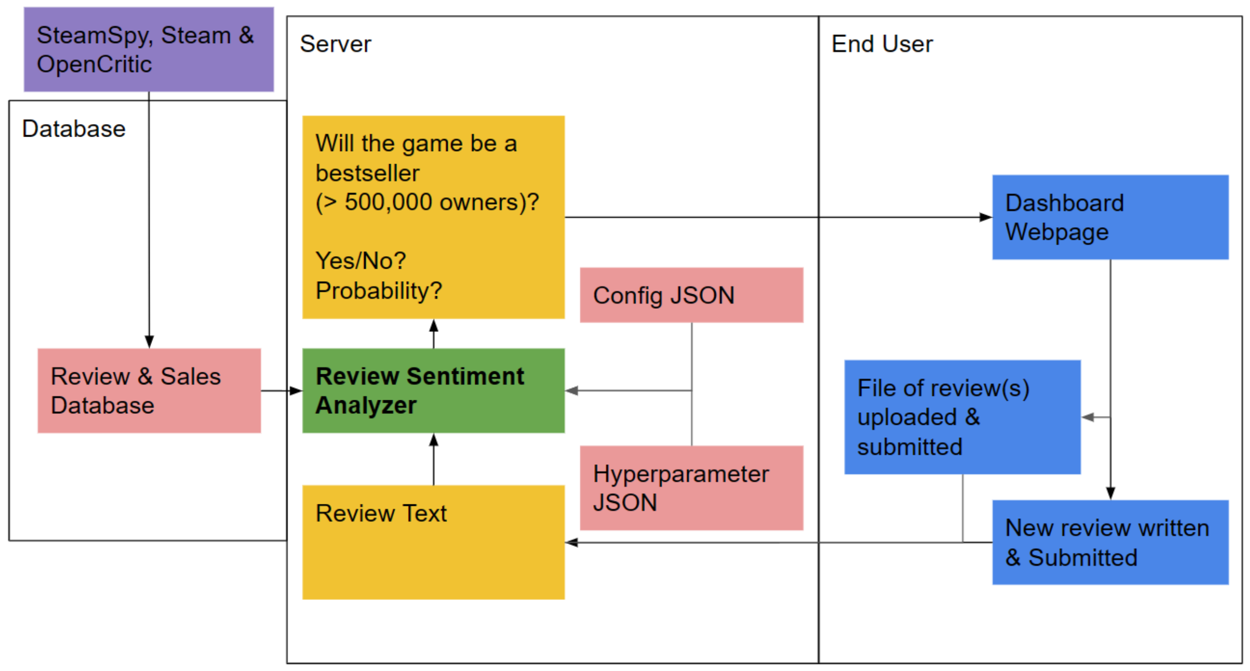
The exceptions to presenting performance findings are the findings for test scores for the model chosen by the technician for use in their business, for which the test evaluation scores are reported at the top of the dashboard which includes accuracy, recall, and precision scores. This is included as a necessary piece of information, in combination with the confidence score given for a prediction, for businesses to assess how trustworthy certain predictions may be. As the other types of classifiers remain unused, and thus of little immediate interest to the dashboard user, their information will be omitted on the dashboard.

Performance and Evaluation

Performance in all cases was obtained primarily through the F1 scores are included as well to evaluate the performance for positive cases, as it is expected that end-users are most interested in ensuring their product becomes a bestseller and will want to identify positive cases as accurately as possible. In Hyperparameter Tuning, the GridSearchCV is tuned using just the F1, and thus the best score is measured only in F1. Precision and recall, of which F1 is a function of, are kept in findings to further breakdown how positive cases are treated. The models are then evaluated by comparing all the test cases in each workflow, then selecting the cases which have the best F1.

Based on findings during Hyperparameter Tuning, it can be concluded that the text of a review has predictive power for a game's ability to meet a fixed ownership threshold of more than 500,000 Steam owners, with the best performing model used during analysis by F1 being a Random Forest classifier with an overall accuracy of 0.858 and an F1 of 0.582, and a precision and recall of 0.531 and 0.644 respectively. The MLP classifier meanwhile has a lower recall of 474, but a higher accuracy and precision of 0.822 and 0.667, making it the second best. The F1 is improved even further by aggregating reviews together by probability vote and deriving a prediction from it.

Solution Architecture, Scaling, & Budgeting



SteamSpy, Steam & OpenCritic
Database
Server
End User
Will the game be a bestseller (> 500,000 owners)? Yes/No? Probability?
Config JSON
Review & Sales Database
Review Sentiment Analyzer
Hyperparameter JSON
Review Text
Dashboard Webpage
File of review(s) uploaded & submitted
New review written & Submitted

Figure 8. A simplified flowchart of the solution architecture. The three white box sections refer to discrete parts of the solution, which can be present on one computer or multiple ones. Omitted from the flowchart are the score findings of the selected model, which is provided to the webpage on loading.

The solution architecture of the Review Sentiment Analyzer Dashboard takes the hyperparameter data retrieved from the findings and uses them as the settings for the classification pipeline as previously mentioned in Reporting as being exported to a JSON file. In figure X this pipeline is referred to as the Review Sentiment Analyzer. Instead of using the models trained during analysis, each time the server is started, the analyzer trains directly from the same database used during analysis, which in-turn can be updated to include new data from SteamSpy, Steam, or OpenCritic. It then receives text from dashboard requests and returns the prediction and probability to the dashboard.

In matters of scaling the models for deployment, the solution is expected to receive a small throughput at a given time each hour, much smaller than the dataset used to train and test them. Scaling analysis findings showed that new inputs of similarly clean review data do not result in a large drop in performance either. Instead, the focus was on streamlining their deployment on the dashboard server to reduce latency between requests for predictions and the predictions themselves.

The initial design had the model be fit during a request, but given the very high latency of such a design, the design was further revised to fit on server startup. To further improve its scalability in the event that the model is fitted with an especially large dataset, the Skops library is used to dump the model, while the test scores gained after fitting are dumped to a JSON, allowing for them to be quickly loaded again on rebooting the server.

For budgeting, the only resources which required direct payment from the start of the project were operation costs for the PSQL server, which was hosted using Amazon Web Services, and the Subscription for the OpenCritic API to overcome the daily query limits of the API.

To budget for the PSQL, the payment came out of an account provided by UCSD. To ensure it can stay active at any time, the smallest pre-defined set on AWS is used so that only around $0.50 a day is expended whenever it is active. This allowed for a cost-efficient 24/7 operation.

For OpenCritic, only the highest subscription rate allowed for the number of searches and reviews needed to be accomplished for the data collection in a timely manner ($50/month). This was paid out of pocket and will be unsubscribed to when it is deemed no longer needed after any repeat extraction operations are complete.

Instead of hosting the dashboard on the remote server, for the purposes of this project it was hosted locally for simplicity, which also avoids expenses for hosting the Flask server using a web service.

Conclusion

The texts of reviews provide a rough but non-random ability to successfully predict games' ability to meet an ownership threshold on Steam above 500,000 owners. These findings can then be used within a web server data loop to give users feedback from the input text to indicate the sales outlook for a given game product. If the solution architecture can be applied within a business development environment, using textual feedback data from focus groups, test engineers, and customer preview feedback, it can better inform where to improve a product based on who to target, what reviews indicate who or why someone would or would not buy it that is not explicitly stated in text. Being able to aggregate the data provides the potential to provide an overall picture of the health of the project from a marketing and sales point.

Future work into the topic should investigate identifying key factors in the documents which cause predictions to weigh towards certain labels, i.e., what keywords do each include, what categories of information do the reviews provide, to give end-users more granular information to provide better feedback. Using a fixed threshold of 500,000 is also non-indicative for the exact number of sales achieved for a game, and it may be useful to investigate converting the problem from a classification problem to a regression problem to better indicate *where* within each label a game sells.

References

Ahn, Sang ho & Kang, Juyoung & Park, Sang-Un. "What makes the difference between popular games and unpopular games? Analysis of online game reviews from steam platform using word2vec and bass model." *ICIC Express Letters.* 11. 1729-1737. 10.24507/icicel.11.12.1729.

Catellier, Rémi & Vaiter, Samuel & Garreau, Damien. 2023. "On the Robustness of Text Vectorizers."

Li, Susan. “Multi-Class Text Classification with Doc2Vec & Logistic Regression.” *Medium*, 4 Dec. 2018. towardsdatascience.com/multi-class-text-classification-with-doc2vec-logistic-regression-9da9947b43f4#:~:text=Doc2vec%20is%20an%20NLP%20tool.

Panjeh. “scikit learn hyperparameter optimization for MLPClassifier.” *Medium.* 29 June 2020. https://panjeh.medium.com/scikit-learn-hyperparameter-optimization-for-mlpclassifier-4d670413042b

Reine, R. “How to build & deploy a react + flask app.” *Towards Data Science.* 29 June 2022. https://towardsdatascience.com/build-deploy-a-react-flask-app-47a89a5d17d9

Sinclair, B. “CD Projekt refunded around 30,000 Cyberpunk 2077 copies. GamesIndustry.Biz.” 22 April 2021. https://www.gamesindustry.biz/cd-projekt-refunded-around-30-000-cyberpunk-2077-copies

Shah, P. “Sentiment Analysis using TextBlob.” *Towards Data Science.* 6 November 2020. https://towardsdatascience.com/my-absolute-go-to-for-sentiment-analysis-textblob-3ac3a11d524

Appendices

A. DSE MAS Knowledge Applied to the Project
   1. Supervised learning, classification
      i. Logistic Regression
      ii. Random Forest Classifier
      iii. K-Nearest Neighbors
      iv. Neural Networks
   2. Model Evaluation
      i. Confusion Matrix
      ii. Accuracy
      iii. F1
      iv. Precision
      v. Recall
   3. Model Selection
      i. Validation score
      ii. Hyperparameter Tuning, Grid Search
   4. Min-Max Normalization
   5. Natural Language Processing
      i. Text Vectorization
      ii. Text Cleanup
         1. Drop special characters
         2. Drop stop words
      iii. Text Distance Measurement (behind API)

B. Link to the Library Archive for Reproducibility

   1. https://doi.org/10.6075/J06D5T5H

Credit to Polina Haryacha for their help in developing the intro and question formulation, and for developing the Steam and SteamSpy extraction notebooks.