# Predicting Steam Sale Dates using Machine Learning

Ryan Carnes
*Computer Science 2024*
*University of Tennessee*
Knoxville, USA
rcarnes@vols.utk.edu

Dean Longmire
*Computer Science 2024*
*University of Tennessee*
Knoxville, USA
jlongmi9@vols.utk.edu

***Abstract*—Our objective for this project was to create a machine learning model that can predict, with some accuracy, the date that a video game sold on the Steam online storefront will go on sale. We did this by utilizing a dataset of steam store application data from Mendeley, which included features such as release date, developer, publisher, player count, and price history. We had to do a decent amount of pre-processing on our data to prepare it for our model, and to throw out the bad or irrelevant data (such as free-to-play games since they cannot go on sale). Our initial efforts appeared to be fruitful but after further investigation proved meaningless due to the imbalanced nature of our dataset. After stepping back, reorganizing and resampling our data, then changing our approach, we were able to come up with a model that is able to predict the initial sale date with good accuracy, but struggles with categorizing non-sale dates.**

## I. Introduction and Motivation

Steam is an online storefront that distributes computer applications, mainly videogames. Steam also regularly hosts sales for their applications, where they are discounted at a percentage set by the game's developer/publisher. These sales typically run for a range of days and can be influenced by a variety of factors including time of year, player count, and even the historical sale data of a publisher/developer. Major Steam sales typically happen seasonally, with the Winter and Summer sales being the biggest by far. It's fairly easy to predict when a Steam sale will happen due to this pattern, but knowing which individual games will go on sale is difficult without taking into account the multiple factors listed above. Sometimes games go on sale outside of these Steam sanctioned store-wide sales, and being able to predict that information would be useful for consumers who are trying to take advantage of these sales. For uninformed consumers it can be hard to predict whether a game will go on sale soon, which can affect purchasing decisions and can easily lead to buyers' remorse when a game bought for $60 suddenly goes on sale for $20 a week later.

We decided to apply machine learning to this problem to better understand why games go on sale at certain times, and to assist consumers who want to make more informed purchasing decisions.

## II. Dataset

Our dataset was sourced from Mendeley [1]. It consisted of 7 main csv files and 3 zip files with information on the top 2000 most played steam games as of December 11th, 2017. A good portion of our time spent on this project was related to pre-processing our data since it was set up in an unfriendly way for our purposes. We did not use all the data present since most of it was irrelevant to our objective, but we did merge the applicationDevelopers.csv, applicationPublishers.csv, and applicationInformation.csv files to create one merged csv with our relevant information. We also dropped our unusable or irrelevant entries (such as free-to-play games) and ended up with 1248 usable applications.

Unlike the above csv's, which have one line per application, the Price History and Player Count History zip files contained 1 csv for every application, keyed on the appid, each of which contained that application's price and player count history. The top 1000 games had player count history for 5-minute intervals and the bottom 1000 had the player count history in daily intervals. The price history was also in daily intervals.

We took the top 1000 game's player count history and normalized it to daily intervals using the average player count so that we could merge all our price and player count histories into one csv. At this point in pre-processing, we had our merged application information csv, with one entry per application, and then one csv for every application with that applications price and player count history.

We combined all of these into one giant ~590,000-line csv, which contained all our data used in our model. In this file we broke up our current date and release date into 3 integer fields and enumerated our non-number fields to make them compatible with our model. We also changed the "discount" field to be a binary classifier instead of the percentage of discount so that we could focus on classifying the sale date instead of the discount amount.

The first thing we did to explore relationships in our dataset was to print out a graph of each game's player count graphed against its sales to see if there is any obvious visual correlation. Our initial thought was that player count would likely have the

highest impact on sale dates, but based on the examples below, it's obvious that this is not the case.

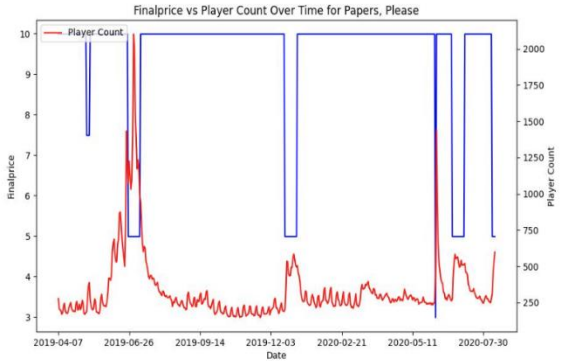


Fig. I. Player Count vs Final Price for "Papers, Please"

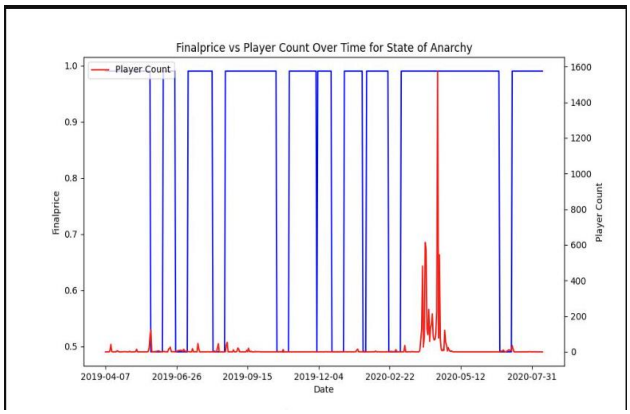


Fig II. Player Count vs Final Price for "State of Anarchy"

After making this observation we decided to create a correlation matrix to get a sense of what labels have the highest impact on our feature of interest, "discount".

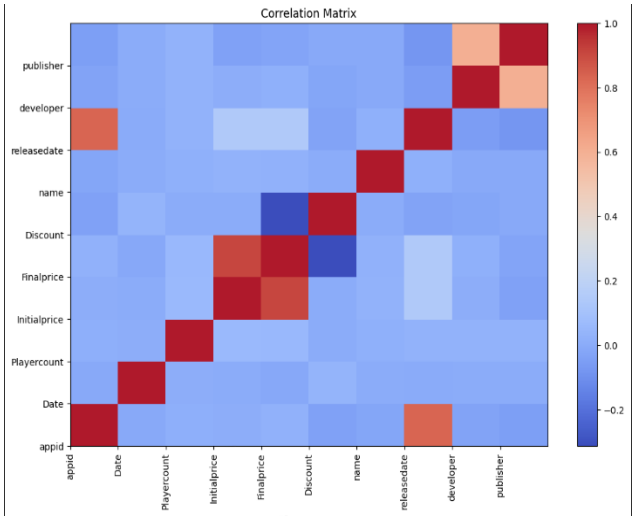


Fig. III. Correlation matrix of data features

The only label that had a clear strong relationship with "discount" was "final price", which had an inverse correlation. This makes logical sense when considering the effect that a discount has on the final price of a game. We chose to omit the "final price" label from our model, since it essentially spoils whether a sale is happening on that day.

One thing worth noting at this point about our dataset is that it is highly imbalanced for our use case. Our feature of interest is the binary classifier "discount". Roughly 86% of our raw data fell under the "not on sale" category while the remaining 14% fell under the "on sale" category. This is an issue that we did not initially anticipate or catch, which led to some failed models which will be discussed in the next section. After discovering our problem, we were able to oversample the "on sale" training data to have the same number of samples as our "not on sale" training data, which helped our model make actual predictions.

## III. Machine Learning Approaches and Methodology

The first thing we did to get a sense of how to approach our problem was to run our data through a variety of different types of models to decide which to use. Our initial run of these models yielded the following results.

TABLE I. Training and testing accuracies for different learning models

| Classifier | Training Accuracy | Testing Accuracy |
|---|---|---|
| Nearest Neighbors | 0.855 | 0.822 |
| Decision Tree | 0.836 | 0.861 |
| Random Forest | 0.830 | 0.861 |
| Neural Net | 0.830 | 0.861 |
| AdaBoost | 0.831 | 0.861 |
| Naïve Bayes | 0.829 | 0.860 |

Since we got the same testing results for a few different models, we decided to go with a decision tree since it is simple, easy to interpret, and has few hyperparameters to tweak. These identical testing results should have been an early red flag of a problem that we encountered shortly after this. Our accuracy was much higher than expected, especially for our initial model, so we started tweaking hyperparameters to see how they changed the results. To our surprise, tweaking our hyperparameters made no changes to our testing accuracies, which we found very bizarre. To investigate this anomaly, we decided to print out a confusion matrix of our results, which confirmed our growing suspicions of our model's behavior.

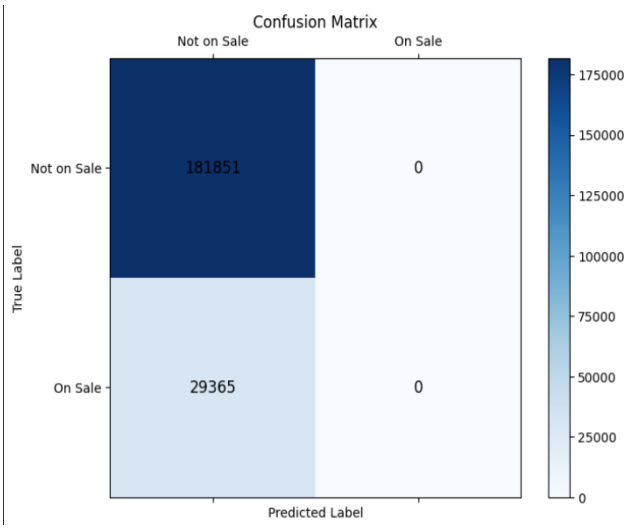

Fig. IV. Confusion matrix for first initial model selections

What we discovered was that our model was not actually making any form of prediction at all. It was simply defaulting to the "not on sale" classification for every single sample. This was due to the imbalanced nature of our dataset. Our model realized that it could get higher accuracy through this brute-force method than by mistaking actual predictions, since most of our samples fell under that category. Although our accuracy was high, our model was essentially worthless, and we had to take a different approach to our problem. This led to us re-sampling our training data, which involved oversampling our minority class up("on sale") to our majority class("not on sale").

Another potential problem we noticed with how the data was being presented was that all sales start on a certain date and then may run for multiple days after that date. This may make it too easy for the model to predict sales on the testing set because it may pick up on this pattern. The model can then find consecutive dates and predict a sale on that day since there was one before or after it. Since the sale start date is something we want the model to be able to predict and will not be available when guessing on new data, we decided to roll the sales into one day. This means that when a sale started that day was kept as the "on sale" day and all consecutive days the game was on sale afterwards were marked as "not on sale". This should force the model to make informed predictions only on the features we want it to.

After these changes in approach, we re-ran our data through the various models and received the following, much more realistic results.

TABLE II.
Training and testing accuracies on sanitized data for different models

| Classifier | Training Accuracy | Testing Accuracy |
|---|---|---|
| Nearest Neighbors | 0.950 | 0.953 |
| Decision Tree | 0.769 | 0.681 |
| Random Forest | 0.653 | 0.653 |
| Neural Net | 0.504 | 0.032 |
| AdaBoost | 0.738 | 0.671 |
| Naïve Bayes | 0.511 | 0.060 |

To showcase whether this made an effective difference, we decided to briefly focus on our decision tree model again, even though the nearest neighbor's model had a higher initial accuracy when using basic parameters. We created a confusion matrix for our new decision tree model, which proved that our model, although less accurate, was making actual predictions on the testing samples based on our training samples.

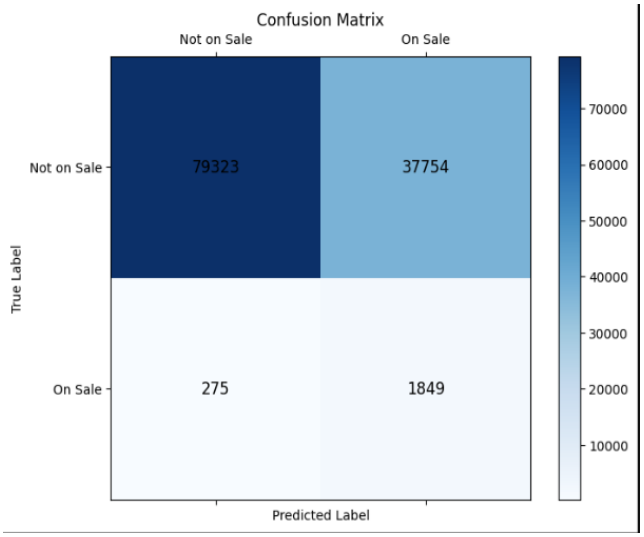

Fig. V. Confusion matrix for a decision tree using the newly balanced dataset

This model had an easy time identifying the "on sale" dates with relatively high accuracy (~87%) but struggled to fully identify the "not on sale" dates and misclassified many of them as "on sale". This model's overall accuracy is much lower than our initial model, but this is a result of it making actual varied predictions and not brute forcing a high accuracy through our imbalanced training data.

## IV. Results

Looking back on the results found when running all the classifiers with our newly balanced dataset, we noticed that the K Nearest Neighbors Classifier produced the highest values with accuracies at ~95%. This led us to further investigate using this model with different hyperparameter settings. The hyperparameter that we decided to test first was the number of neighbors that each data point will evaluate when assigning it label. Running passes with values at 3, 4, 5, 6, 7, 8, 9, 10, and 15 we gathered the following accuracies seen in *Table I*.

TABLE III.
Training and Testing Accuracies for K-Nearest Neighbors

| Number of Neighbors | Training Accuracy | Testing Accuracy |
|---|---|---|
| 3 | 0.950 | 0.953 |

| | | |
|---|---|---|
| 4 | 0.949 | 0.954 |
| 5 | 0.964 | 0.937 |
| 6 | 0.963 | 0.938 |
| 7 | 0.973 | 0.923 |
| 8 | 0.973 | 0.923 |
| 9 | 0.969 | 0.910 |
| 10 | 0.970 | 0.910 |
| 15 | 0.949 | 0.870 |

The data gathered indicates that setting the hyperparameter of number of neighbors to four will provide the most accurate guesses on an unlabeled dataset. However, after looking at the confusion matrix shown in *Figure VI*, we can observe that the model is still predicting many "on sale" dates incorrectly.

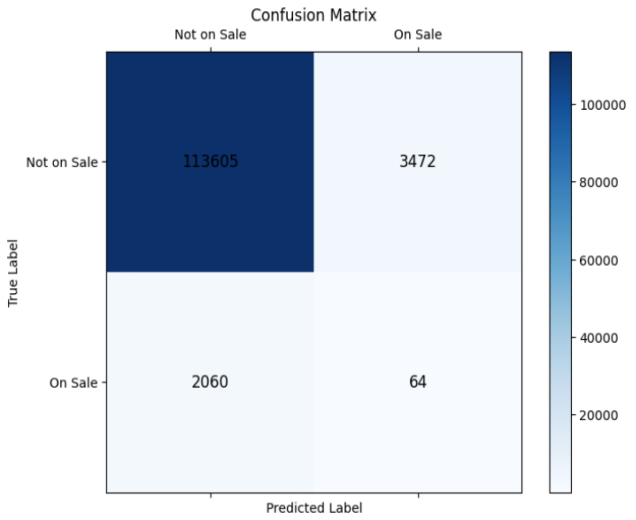


Fig. VI. Confusion matrix for 4 neighbors

While the model seemed to still guess poorly on choosing actual dates that were on sale, this was still by far the best model that we were able to train. It should be noted that it did guess a significantly less number of wrong dates that would be on sale leading to the increase in accuracies.

## V. Discussion, Conclusion, and Future Work

As it turns out, predicting the starting date of sales is quite difficult, due to the variety of factors that influence it. There are many behind-the-scenes business decisions being made to decide these sales that we as consumers cannot possibly know, and many economic factors that are unaccounted for in our model, such as the developer/publisher income and employee layoffs. These factors can also vary greatly between different companies, making it increasingly harder to find any patterns in the data.

In addition to that, our dataset is quite small and does not span a long period of time. It only spans from April 7th, 2019 – August 12th, 2020, about a year and a half, which fails to fully capture the long-term market trends that heavily affect sales, such as seasonal sale dates. Our dataset also only looks at 1248 games, which represents only 1.26% of the 99011 [2] games on Steam. This is not a good representation of the Steam Market as a whole and is another factor that contributes to our model failing to capture long-term trends.

Increasing the scope of our model across multiple facets would absolutely help it capture the true nature of the market. But there will always be anomalous influences and unexpected outcomes that a machine learning model will have a hard time capturing, at least without a much higher complexity compared to what we accomplished here.

Were we to continue the exploration of predicting steam games sales, it would likely be best if we trained individual models for different games rather than combining all games into one dataset. This would likely solve two problems. The first being that we had to greatly reduce the range of our data due to some games not having entries for days that other games did have. This would give our model more data to make predictions off of and could potentially lead to more accurate results. The second, and more impactful factor, is that it would pick up on patterns related specifically to that game. When all the games are lumped into one dataset, it can become difficult for a model to find the patterns for individual games or developers. If we were to train a model off a much larger, single game's dataset, it could more rigidly define patterns and make smarter predictions.

## References

[1] Wannigamage, Dulakshi; Barlow, Michael; Lakshika, Erandi; Kasmarik, Kathryn (2020), "Steam Games Dataset : Player count history, Price history and data about games", Mendeley Data, V1, doi: 10.17632/ycy3sy3vj2.1 https://data.mendeley.com/datasets/ycy3sy3vj2/1

[2] SteamDB, https://steamdb.info/instantsearch/?refinementList%5BappType%5D%5B0%5D=Game