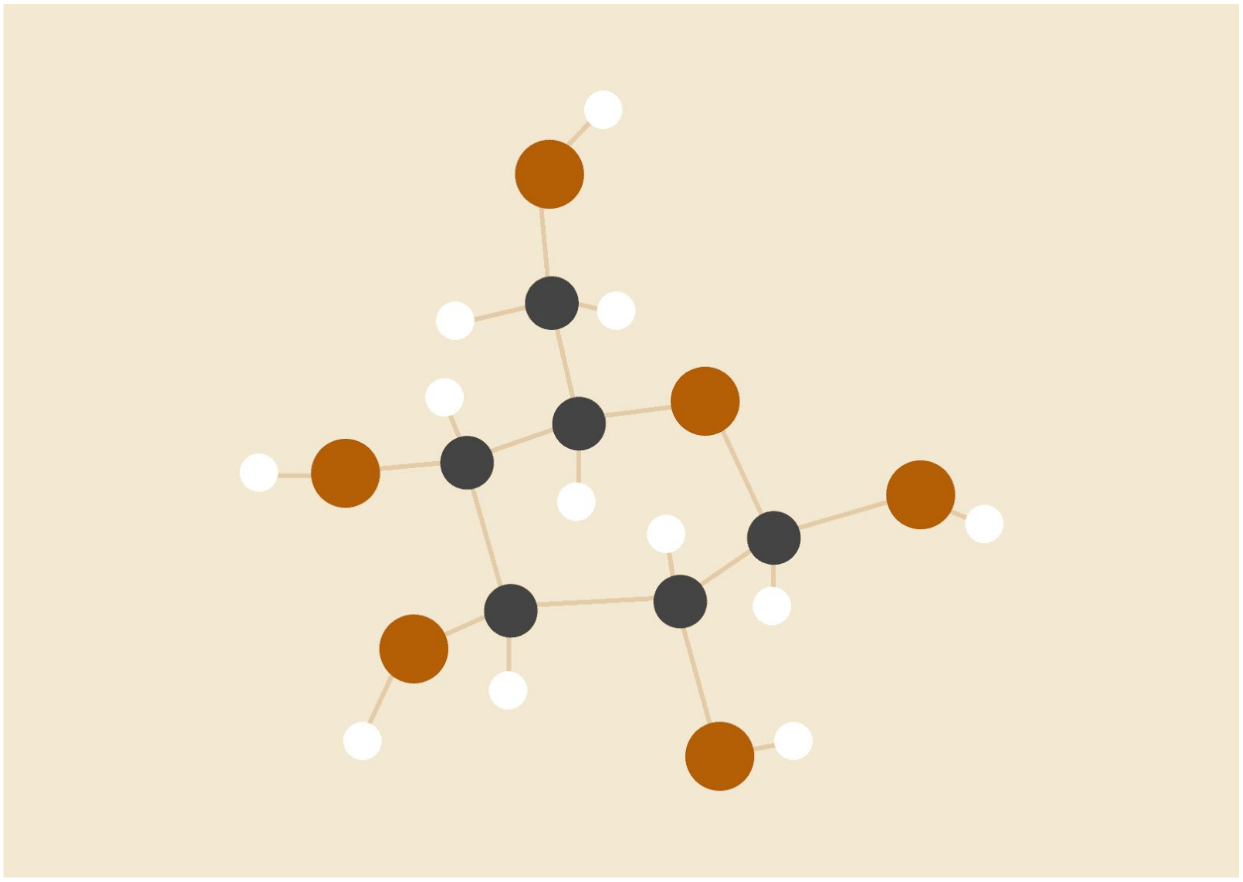
Steam Market Analysis

A data mining process working on game data on Steam Platform



Samuel Hobson

Xiang Ao

11.07.2017

CSCI 420

ABSTRACT

This paper describes a market analysis experiment on the Steam online PC game store using modern data mining techniques. This paper will look at two aspects. The first is examining how a game's genre and publisher will impact its popularity using different classification techniques, including decision trees, Naïve Bayes, and Adaptive Boosting, and comparing the results. The second is using clustering in an attempt to discover any underlying features that may not be reflected in the classification, or that could improve the classification. The clusters studied are Otsu's method for one-dimensional clustering, k-Means, and Hierarchical Agglomerative Clustering.

INTRODUCTION

Data mining is already prevalent in many aspects and has been shown to be extremely valuable to making business and market analysis, trend predictions and potential user group minings. This document outlines our group's plan to apply data mining statistics and predictive analysis about the games hosted on Steam, an online platform and store for PC games. Steam provides an extensive framework for commerce, online multiplayer gaming, score leaderboards, and social networking, and is currently the largest vendor of PC games.

The database website, steamdb.info[1], tracks an extensive amount of information of each game hosted. By the data gathered from website, we have captured a portion of data set within a certain time period (a week) to work on. During the data mining process, we employed several different algorithms to do the clustering and classifications. Based on the result came out from the data mining process, we have made assumptions and predictions about relationships between sale status and game attributes on the Steam platform.

[1] https://steamdb.info/

BACKGROUND

The founder of the website Steam Spy[2], Sergey Galyonkin, has been researching and analyzing the data from Steam database. The research article, *Some things you should know about Steam*, illustrates what have been found during the data mining process. It divides the research into three main aspects: *Geography*, *Sales and genres*, *Common misconceptions* and *Doom and gloom and a way to succeed*. By the Geography, the research exposes a lot of stereotypes about what people from different countries are playing and what genres they do prefer. At the beginning it focus on six representative countries: United States, Russia, Germany, United Kingdom, China and Japan, with their local game-play status data and background analysis. Then the research poses the relationship between game genres and their sale status, which has drawn the slope of decreasing trend for the "success rate" among the whole game market on Steam platform. Ending with analysis and suggestions for game producers and publishers, the research concludes that Steam is an awesome platform to sell your games and despite many shortcomings, it is still the most game developer friendly environment to work with.

In general, there is no ethical concerns related to this research. While it does talks about the common misconceptions in game field, the research claims that games aimed exclusively at a female audience are less likely to succeed on Steam than on other more inclusive platforms like for example Facebook or iOS. It also mentions aspects about game production and selling strategies, like Early Access or Cloning the currently hottest game. Based on it's stable research and findings, we believe that Steam Spy is a great data mining and user research source for game producers and publishers to achieve the business success.

DATA

[2] http://steamspy.com/

Our data set are made up of various attributes for 305 PC games sold through Steam. The set exists in two forms differing in how they store genre information. The attributes across both sets are as follows:

- Application ID - the unique identification number for the game in Steam's catalogue.
- Title - the name of the game.
- Developer - the name of the studio which developed the game.
- Publisher - the name of the house which published the game.
- Price - the price of the game at the time of mining.
- Rating (Percentage) - the percentage of user reviews which recommend the game.
- High Rating - A binary attribute which is true if the game has a rating of at least 75%.
- Multiplayer - A binary attribute which is true if the game supports more than one player at a time.
- Number of Owners - The total number of players who own the game
- Playtime (2 weeks) - Numeric values representing the mean and median number of players online over a two week period
- Playtime (All time) - Numeric values representing the mean and median number of players online over the time the game has been on the market.

From here, the data sets differ. The first data set, referred to henceforth as the “broad genre” set, contains an additional categorical attribute called “Genre”, which is a categorical attribute specifying the overall type of game with respect to gameplay. Legal values are Adventure, Strategy, Casual, Simulator, Platformer, FPS, Arcade, Action, Fighting, and Sports.

The second data set, referred to henceforth as the “genre matrix” set, contains a binary attribute for every possible genre and store tag encountered across the games. Store tags are additional fields which describe additional features of a game. For every data point, if a particular genre or tag applies to a game, then the relevant attribute is set to True; otherwise, it’s

False. Because there is a wide array of possible tags and genres, we hope to get a more precise idea of how these particular features of a game can impact its popularity.

Data Retrieval

The data was obtained from https://steamdb.info/, a third party website which tracks an extensive amount of information for each game hosted on Steam. Each entry includes commerce information, activity statistics (such as number of players who own the game, average and median playtimes, and peak number of players playing at the same time), categorical data, and plenty more. This website is unaffiliated with Valve Corporation, the creators of Steam, however, the data hosted there was itself obtained using Steam's own API[3].

The raw data was scraped from the website and organized using a Python script with the requests and lxml modules. The script takes a list of Application ID's (which were themselves scraped) as input and mines the relevant data through a series of Xpath lookups. The data is then stored and written to a Comma Separated Value file for maximum software compatibility.

Data Issues & Cleaning

Because the data hosted on steamdb was itself obtained using Steam's API, it is an accurate and up-to-date copy of the data stored on Steam's servers. Therefore, the data obtained was guaranteed to be largely consistent. That being said, there were some instances where we had to throw out some of the data. The reason for this was typically due to one of the following:

1. The application was not a game: in addition to games, Steam also hosts a number of gaming-related applications. These can include certain OS tweaks, creative tools, and tools for designing games.
2. The application was not sold in the US: for various reasons, certain games are not permitted to be sold in the United States, or are a special version of an existing game for a

[3] https://steamcommunity.com/dev, https://wiki.teamfortress.com/wiki/User:Rjackson/StorefrontAPI, https://github.com/SteamRE/SteamKit

region other than the United States.

3. The application had no rating

These cases were removed from the base data set during the scraping process. The entire scraping function is surrounded with a try-catch block, and any uncaught variety of exception results in the entry being skipped.

The next issue with the data came with the "genres" and "store tags" fields, which often overlapped and could contain any number of tags which describe various aspects of the game. Such aspects can include the genre, the style of gameplay, the setting of the game, the gender of the protagonist, and so on. Furthermore, the store tags field was extracted as one long string which needed to be parsed into usable data. Therefore, in order to clean this data, two methods of cleaning were devised.

The first method resulted in the broad genre set. We simply went through the unioned sets of genres and store tags and looked for any tags which could indicate one of the broad genres described above. This process took place in two parts. The first part was largely done through a script containing a large block of if-else statements. These statements were implemented using a great deal of domain knowledge about the games and possible genres of games. The second part involved looking at the resulting genres and correcting any miscategorizations by hand, again using domain knowledge.

The second method, resulting in the genre matrix set, involved creating an n x m matrix of binary attributes, where n is the number of data objects (in this case, game entries), and m is the number of possible values across the set of genres unioned with the set of store tags. When the matrix is created, all values are initialized to False. For each game entry, if a given tag applies to the entry, then the attribute for that tag in the matrix is set to True.

METHODS

We ran a series of clusterers and classifiers on this data. For classification, we will be attempting to predict if a game will achieve a rating of 75% or higher based on its developer, publisher, features, and genres. With respect to the dataset, we are attempting to predict the “High Rating” attribute. For this experiment, we removed the playtime and number of players attributes, focusing only on publisher information and genre features.

To measure our results, we used three measures. First, we used true positive rate in order to gauge the overall weighted error rate. F-measure was used to get an idea of the ratio of how many true positives were reported to the total number of true positives. Finally, we used the Matthew Correlation Coefficient (MCC) to measure the overall correlation of the classifications. To be more specific, the MCC is a value between 1 and -1, where 1 suggests perfect predictions, 0 suggests totally random predictions, and -1 suggests completely wrong predictions.

We will use clustering to try and find any underlying structure and relationships within the data. To this end, we opted not to remove any features from the data set in hopes of discovering patterns which may lead to improved classifications later on.

All algorithms were run using either Weka 3.8.1 (Eibe et al. 2016) or SciPy 0.19 (van der Walt et al. 2011). For validation, we first performed a 10 fold cross-validation for our classifiers, followed by the use of a test set of 94 additional game entries which weren’t a part of the original data set. The remainder of this section will describe in detail our chosen algorithms and the settings we used, while the next section will detail the results.

Correlation Coefficient

The correlation coefficient is a measure that determines the degree to which two variables' movements are associated. The range of values for the correlation coefficient is -1.0 to

1.0. If a calculated correlation is greater than 1.0 or less than -1.0, a mistake has been made. A correlation of -1.0 indicates a perfect negative correlation, while a correlation of 1.0 indicates a perfect positive correlation. By looking for Correlation Coefficient between different attribute we can find how close relationships they could have with each others.

Correlation Coefficient can be calculated by the function:

$$\rho_{xy} = \frac{Cov(r_x,r_{y)}}{\sigma_x\sigma_y}$$

By looking for Correlation Coefficient between different attribute we can find how close

relationships they could have with each others.

Otsu's Method

Otsu's Method for One Dimensional Clustering:

1. Breaks the data into two clusters
2. Iteratively tries all possible thresholds
3. The goal is to minimize the weighted variance of both clusters
4. The clustering is evaluated using the weighted sum of the two variances:
   - wL = fraction of data <= threshold
   - wR = fraction of data > threshold
   - σ ^2L = variance of data <= threshold
   - σ ^2R = variance of data > threshold
   - weighted_variance = wL* σ ^2L + wR* σ ^2R
   - best_threshold = arg min(wL σ 2L + wR σ 2R)

Hierarchical Agglomerative Clustering

An agglomerative clustering algorithm starts with N cluster prototypes, each containing one unique point from the data set. The most similar clusters are then merged together until there is only one cluster. From there, we can determine our clusters by searching each iteration for the largest difference between prototypes. To measure similarity, we calculated the Euclidean distance between the center of mass for each pair of prototypes. A formal description of our algorithm is as follows:

1. Assign each point to a cluster prototype, and set the center of mass as the point itself.
2. For every pair of prototypes, calculate the Euclidean distance between their centers of mass.
3. Take the pair with the shortest distance and add their points to a new prototype.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size N.

k-Means

The k-means algorithm is first applied to an N-dimensional population for clustering them into k sets on the basis of a sample (MacQueen 1967). The algorithm is based on the input parameter k. First of all, k centroid point is selected randomly. These k centroids are the means of k clusters. Then, each item in the dataset is assigned to a cluster which is nearest to them. Then, means of all clusters are calculated again with new points added to them, until values of means do not change. Alpaydin (2014) symbolizes this algorithm like below where m is sequence of means, x^t is sequence of samples, and b is sequence of estimated labels.

```
Initialize $m_i$ , $i = 1,\ldots,k$, for example, to k random $x^t$
Repeat
    For all $x^t \in X$
        $b_i^t \leftarrow 1$ if $|| x^t - m_i || = \min_j || x^t - m_j ||$
        $b_i^t \leftarrow 0$ otherwise
    For all $m_i$ , $i = 1,\ldots,k$
        $m_i \leftarrow \Sigma_t b_i^t x^t / \Sigma_t b_i^t$
```

```
Until $m_i$ converge
```

Naïve Bayes

The Naïve Bayes classifier is based on Bayes' theorem with independence assumptions between predictors. A Naïve Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naïve Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods.

Bayes' theorem provides a way of calculating the posterior probability, P(h|d), from P(h), P(d), and P(d|h). Naïve Bayes classifier assume that the effect of the value of a predictor (d) on a given class (h) is independent of the values of other predictors. This assumption is called class conditional independence, the equation for which is formally defined as

$$P(h|d) = (P(d|h) * P(h)) / P(d)$$

Where

- P(h|d) is the probability of hypothesis h given the data d. This is called the posterior probability.
- P(d|h) is the probability of data d given that the hypothesis h was true.
- P(h) is the probability of hypothesis h being true (regardless of the data). This is called the prior probability of h.
- P(d) is the probability of the data (regardless of the hypothesis).

For Naïve Bayes, we used the Weka implementation (John, Langley 1995) with default settings. For both the broad genre and genre matrix sets, we removed the ID and Title fields. Additionally, we removed the Rating Percentage from which we derived our target High Rating field to prevent overfitting.

Decision Tree

For classification with decision trees, we used J48, a Java implementation of the C4.5 training algorithm in Weka (Quinlan 1993). C4.5 builds decision trees in a simple depth first fashion, expanding an entire path until a satisfactory leaf node is reached and then backtracking to expand all remaining paths. In order to gauge the purity of splits in the data, Gain Ratio is used to compare how much new information is learned from a given split.

Adaptive Boosting

The goal behind adaptive boosting is to build multiple models from the base data set, using previous models to focus its attention on data points where were misclassified. The models are typically built using weak or lazy classifiers. We used the Weka implementation, AdaBoostM1 (Freund and Schapire 1996), with two different classifiers. We initially planned to use Decision Stumps as our classifier, but we determined that tree based algorithms aren’t ideal for this data set. Instead, we opted for the K* algorithm. K* classifies points by observing the classes nearby points, using a distance metric based on entropy to measure similarity (Cleary and Trigg 1995).

RESULTS

Correlation Coefficient

For the data we gathered from the website, we first tried to figure out relationships between each attributes of the games. Correlation Coefficient was used as the sample to present how each attribute has a close relationship to one another. Using adjacency matrix to go through all these attributes, we picked the top 5 pairs of attributes with greatest correlation coefficients.

Correlation Coefficient	Attribute One	Attribute Two
0.572	#_OF_OWNERS	ONLINE_PEAK_(ALL_TIME)
0.559	DEVELOPER	PUBLISHER

0.458	PLAY_TIME_2WKS_(MEDIAN)	PLAY_TIME_2WKS_(MEAN)
0.424	PRICE	PLAY_TIME_2WKS_(MEDIAN)
0.407	PRICE	PLAY_TIME_2WKS_(MEAN)

Table CC-1: top 5 pairs of attributes with greatest correlation coefficients

By the first pair, "#_OF_OWNERS" and "ONLINE_PEAK_(ALL_TIME)", it simply gives us the information the the number of owners of a game would have a closest relationship with the number of online peaks from all time, which means that the more people have bought the game, the more people would have a look on it, or in reversed order.

The second pair of attributes is meaningless. Since the developer and the publisher of a game are very likely to be the same, the correlation coefficient between these two attributes do have reason to be high. But it does not contribute any help to our data mining process while this relationship is nothing about the digital data that we have found for games. Hence we mark this pair as meaningless one.

The rest three pairs generally shows relationships between the game price and the playtime by its owners. It illustrates the relationship that as the price of game rises, the playtime of users on this game will also increases. It's reasonable since we know that a cost of one game means how much content the developer have developed into it, and which makes users spend time on it.

Otsu's Method

According to the result we found in Correlation Coefficient part, we decided to find the threshold that minimizes the weighted within-class variance between two pairs of attributes, Number of Owners & Online peaks, and Price & Playtime for 2 Weeks. In this case, Otsu's Method is employed as the One-Dimensional Clustering algorithm to figure out the threshold.

The first pair of attributes, Number of Owners & Online peaks, has the result of threshold as number of owners of the game that is 1372007 being the one with the lowest variance (Figure OM-1). It means that at round this threshold we could split the data into 2 main clusters.

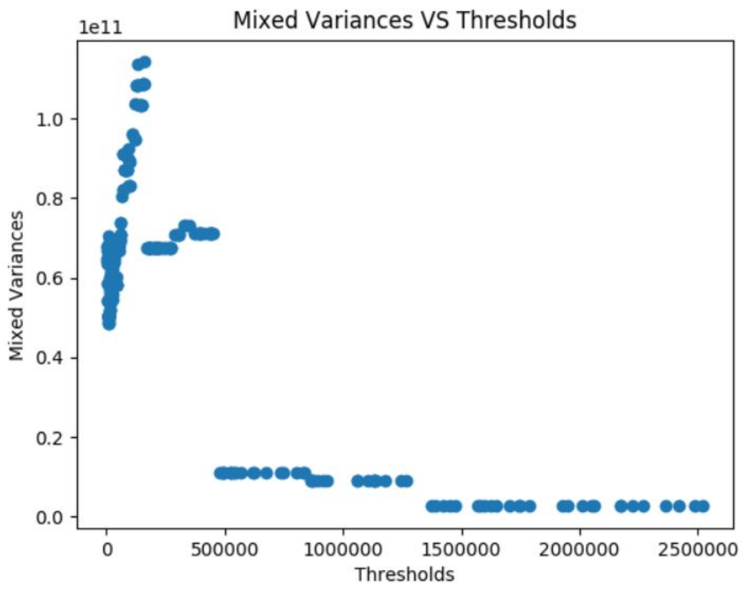


Figure OM-1: Mixed Variances v.s Thresholds for Number of Owners & Online peaks

And also for the second pair, Price & Playtime for 2 Weeks, has the result of threshold as price of the game that is 49.99 being the one with the lowest variance (Figure OM-1).

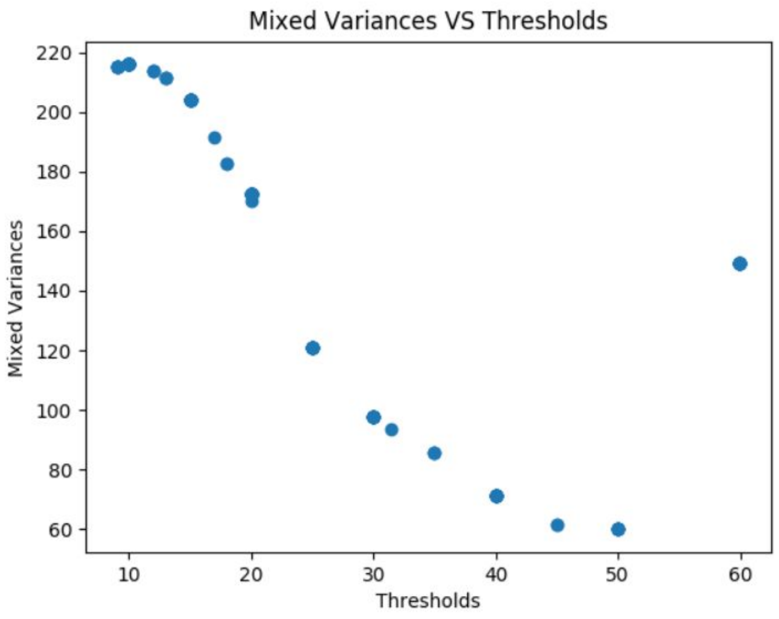


Figure OM-2: Mixed Variances v.s Thresholds for Number of Price & Playtime for 2 Weeks

Both of the clustering processes will tell us about the divisions for the groups. For example, by the second pair of attributes we can have the idea which divide games into two types different by prices, which could be given as a reference to developers to make decisions between cost and payback for the future-developing games.

Hierarchical Agglomerative Clustering

For the Hierarchical Agglomerative Clustering, we used Java to build the algorithm structure, analyze gathered data and do the clustering. The clustering result shows 3 main clusters divided from the game lists. Based on the data attributes of these games, we named each cluster as: Massive Production Games, Low-cost Independent Games and Some Sport Games.

Cluster Name	Number of Games Included
Massive Production Games	56
Low-cost Independent Games	241
Some Sport Games	8

Table HAC-1: Optimal clusters resulting from HAC

k-Means

Another way we do clustering for the dataset it to use k-Means Algorithm. The k-Means Algorithm let us choose the best K values which is the number of final clusters after clustering process is done.

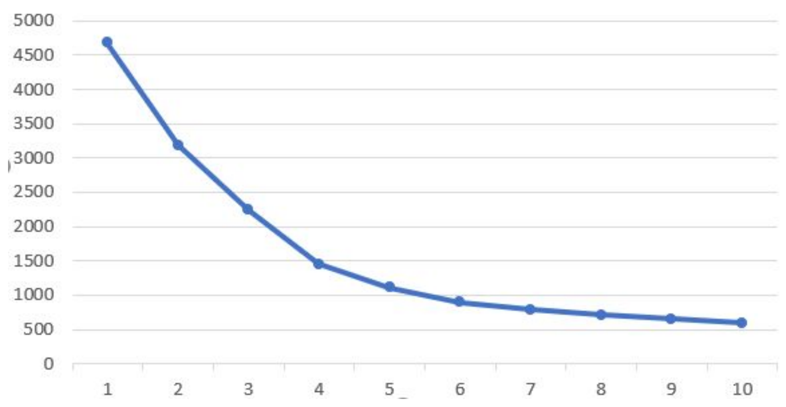


Figure KM-1: SSE v.s K

According to the graph presenting relationships between SSE and K, we decided to use the value of 5 as K since it is lay down at the knee point. After making decision of K, we can simply run the clustering process again to divide the data set into 5 clusters.

Cluster Number	Number of Games Included
1	118
2	34
3	74
4	22
5	57

Table KM-2: Optimal clusters resulting from k-Means

The clustering result comparing to the one from HAC does show some difference. It may caused by some errors, or just by the essences of different algorithms.

Naïve Bayes

For the dataset with the broadly categorized genres, Naïve Bayes successfully classified the data in a 10 fold cross-validation with a True Positive rate of 0.6, a precision rate of 0.521, and an F-measure of 0.565 (Table NB-1a). In the validation with a test set, the True Positive rate was

0.796, the precision rate 0.788, and the F-measure 0.756 (Table NB-1b).

	Classified As: True	Classified As: False
True	164	39
False	83	19

(a) Cross Validation

	Classified As: True	Classified As: False
True	68	2
False	17	6

(b) Test Set

Table NB-1: Confusion matrices for Naïve Bayes with broad genres

For the dataset with the genre matrix, Naïve Bayes performed better overall. When testing using 10-fold cross-validation, the classifier returned a True Positive rate of 0.623, a precision rate of 0.687, and an F-measure of 0.633 (Table NB-2a). With the test set, the classifier returned a True Positive rate of 0.774, a precision rate of 0.778, and an F-measure of 0.776 (Table NB-2b).

	Classified As: True	Classified As: False
True	117	86
False	29	73

(a) Cross Validation

	Classified As: True	Classified As: False
True	59	11
False	10	13

(b) Test Set

Table NB-2: Confusion matrices for Naïve Bayes with the genre matrix

Decision Trees

We initially built our model allowing tree pruning with a confidence factor of 0.25. For both data sets, it proved largely ineffective. The resulting tree from the broad genre set contained only a single node which defaults to True. Both the cross validation and the test set classified all points as True. The genre matrix set produced similar results. The resulting tree contained nodes which tested two genre attributes before making a decision (Figure DT-1). The cross validation classified only 6 points as False, two of which were correct. For the test set, all but 3 points in the set were classified as True, and of the 3, only one was correctly classified (Table DT-1).

		TP	F-Measure	MCC
Broad Genre	CV	0.666	0.532	0
	Test Set	0.753	0.646	0
Genre Matrix	CV	0.659	0.540	0
	Test Set	0.742	0.659	0.036

Table DT-1: Scores for pruned C4.5 trees

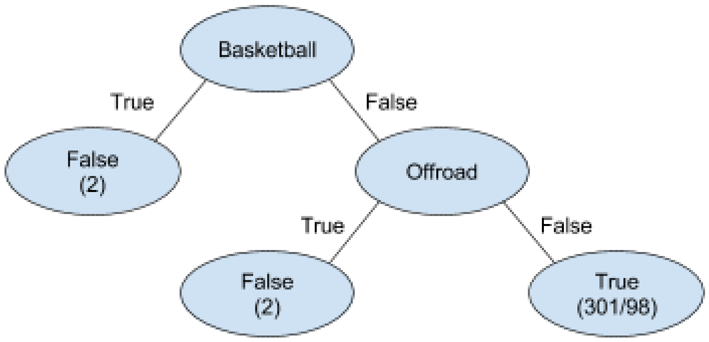


Figure DT-1: Pruned decision tree for the genre matrix set

With pruned trees, the results varied. For the broad genre set, the scores didn't improve (Table DT-2), however we were able to see why the tree was being pruned as it was. The first node was checking the Developer tag, which resulted in only a few developers leading to a classification of False (Table DT-3). Removing the developer fields hardly improved the score, and we've decided that decision trees aren't good for the broad genre task.

	TP	F-Measure	MCC
CV	0.656	0.538	-0.016
Test Set	0.763	0.687	0.117

Table DT-2: Scores for unpruned C4.5 trees with broad genre

	Classified As	
	True	False

	Classified As	
	True	False

True	59	11
False	10	13

(a) Cross Validation

True	69	1
False	21	2

(b) Test Set

Table DT-3: Confusion Matrices for unpruned C4.5 trees with broad genre

Pruning the trees initially didn't improve the scores, but like with the broad data set, we could see that the tree was being cut off at the developer attribute, leading to the same issue. When we removed the developer attribute, we saw an immense improvement. The confusion matrix for both the cross validation and the test set had a much more even distribution, and the test set showed overall higher scores than the cross validation. This would suggest that the risk of overfitting is minimal despite this being an unpruned tree.

	TP	F-Measure	MCC
CV	0.597	0.601	0.118
Test Set	0.699	0.712	0.282

Table DT-4: Scores for unpruned C4.5 trees with genre matrix

	Classified As	
	True	False
True	136	67
False	56	46

(a) Cross Validation

	Classified As	
	True	False
True	52	18
False	10	13

(b) Test Set

Table DT-5: Confusion Matrices for unpruned C4.5 trees with genre matrix

Adaptive Boosting

The results of Adaptive Boosting differed greatly between the broad genre and the genre matrix sets. For the broad genre set, we saw somewhat favorable results (Table AB-1). The classifications were both skewed towards True for both the test set and the cross validations (Table AB-2). The similar true positive rates between the cross validation and test sets suggest a low risk of overfitting.

	TP	F-Measure	MCC
CV	0.610	0.594	0.068
Test Set	0.634	0.650	0.123

Table AB-1: Scores for AdaBoost with K* with broad genre

Classified As

	True	False
True	156	47
False	72	30

(a) Cross Validation

Classified As

	True	False
True	49	21
False	13	10

(b) Test Set

Table AB-2: Confusion matrices for AdaBoost with K* with broad genre

For the genre matrix set, however, we found that boosted K* actually overfits the data. The cross validation showed better scores overall than the broad genre set (Table AB-3), but the test set resulted in results similar to the decision trees, in that all points were classified as True (Table AB-4b).

	TP	F-Measure	MCC
CV	0.603	0.613	0.180
Test Set	0.753	0.646	0

Table AB-3: Scores for AdaBoost with K* with genre matrix

Classified As

	True	False
True	126	77
False	44	58

(a) Cross Validation

Classified As

	True	False
True	70	0
False	23	0

(b) Test Set

Table AB-4: Confusion matrices for AdaBoost with K* with genre matrix

CONCLUSION

By applying several methods to do the data mining process on the dataset we gathered from Steam Database, we could make conclusions depending on what we have found. The result from Correlation Coefficient part would tell us about the the closest relationship patterns over all data attributes, and by analyzing and understanding these relationships we may make predictions about them. For example, it may tell us about the possible number of customers who will purchase the developer's on sale games by comparing it with the number of current viewers on the game. Furthermore, clustering processes and other data mining techniques will be able to give us a picture about how to more specifically divide all games into different group and to do further analyzes. With the sense of these clusters it will get easier for the developer to give their games a accurate market position assumption, or to formulate better plans on future games based on the attribute clustering. Also by sorting on these clusters the customer will be able to more likely find out the game that is been looking for or closest to what he have owned.

REFERENCES

1. Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.
2. Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. **The NumPy Array: A Structure for Efficient Numerical Computation**, Computing in Science & Engineering, **13**, 22-30 (2011), DOI:10.1109/MCSE.2011.37
3. George H. John, Pat Langley: Estimating Continuous Distributions in Bayesian Classifiers. In: Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo, 338-345, 1995.
4. Alpaydin, E. (2014). *Introduction to Machine Learning* (3rd ed.). Cambridge, MA: MIT Press.

5. Zhao, Y., and Karypis, G., “Evaluation of hierarchical clustering algorithms for document datasets”, International Conference on Information and Knowledge Management, McLean, Virginia, United States, pp.515-524, 2002.
6. MacQueen, J. B., "Some Methods for classification and Analysis of Multivariate Observations”, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", Berkeley, University of California Press, 1:281-297, 1967.
7. Sergey Galyonkin, “Some things you should know about Steam”, Jun 19, 2015 https://galyonk.in/some-things-you-should-know-about-steam-5eaffcf33218
8. Ross Quinlan (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.
9. Yoav Freund, Robert E. Schapire: Experiments with a new boosting algorithm. In: Thirteenth International Conference on Machine Learning, San Francisco, 148-156, 1996.
10. John G. Cleary, Leonard E. Trigg: K*: An Instance-based Learner Using an Entropic Distance Measure. In: 12th International Conference on Machine Learning, 108-114, 1995.