Andrew Riggs

CS376B, Spring 2017

5/1/17

**Predicting Videogame Playtime Through Classification**

**Introduction**

Videogames are becoming an increasingly substantial portion of the entertainment industry, with numerous companies and organizations dealing in the funding, creation, publishing, and marketing of these games. In addition, many of these companies work solely within the realm of videogames, making their main source of income the profits made by videogame sales.

With the rise of more advanced computer hardware, more ambitious games could be created and viably sold to a large consumer base. By steadily increasing the scope and quality of games, expectations for the average game have been rising steadily. A game that may have sold at the top of the market in 2000 will likely now be considered heavily flawed, by the standards of 2017.

The increased expectations of the consumer base presents a new challenge, however. To meet this new quality metric, the companies and studios involved must often drastically increase their monetary investment in a game to allow for the inclusion of new technologies and features expected from a modern product.

In addition to the increase in monetary investment, a greater amount of time must also be allotted to the development of these products to allow for the completion of these new features.

Industry common technologies such as fully acted motion capture, user modification support, fully supported multiplayer interactions, and immersive high-definition graphics are all incredibly consuming in terms of both money and time.

As a result, each of these games becomes a rather substantial investment. The amount of money involved increases the sales figures required for profit, and the amount of time involved drastically limits the number of games that can be put out by a single studio. This combines to make every game a quite significant risk.

One way that videogame creators and publishers have begun to counteract this risk is by creating games that are specifically designed to have a long lifespan. By creating games that will be played for numerous years, and inserting additional repeatable transactions into those games, the risk can be somewhat decreased. In addition, the game is no longer just a single burst of revenue at release, but instead provides a more consistent income.

With this in mind, this project will attempt to predict what kind of attributes are important to creating a game that maintains a high average playtime. Numerous classification models will be created, evaluated, and used to this end.

**Methodology**

This project followed the CRISP-DM project framework. Business understanding was largely covered in the introduction section. As such, there were five main phases.

1. Data Acquisition
2. Data Understanding
3. Data Preparation
4. Data Modelling
5. Model Evaluation

***Data Acquisition***

The data used in this project is from Steam, an online games distribution platform that is essentially the de facto game marketplace of the PC market. To obtain this data, a website called SteamSpy (steamspy.com) was used. This website takes in the raw sales figures and derives from them a range of features, such as average playtime and game tags.

When retrieved from the website, the data was initially in JSON format. After the intervention of Professor Read, the data was converted into a CSV file as well as an Rda file.

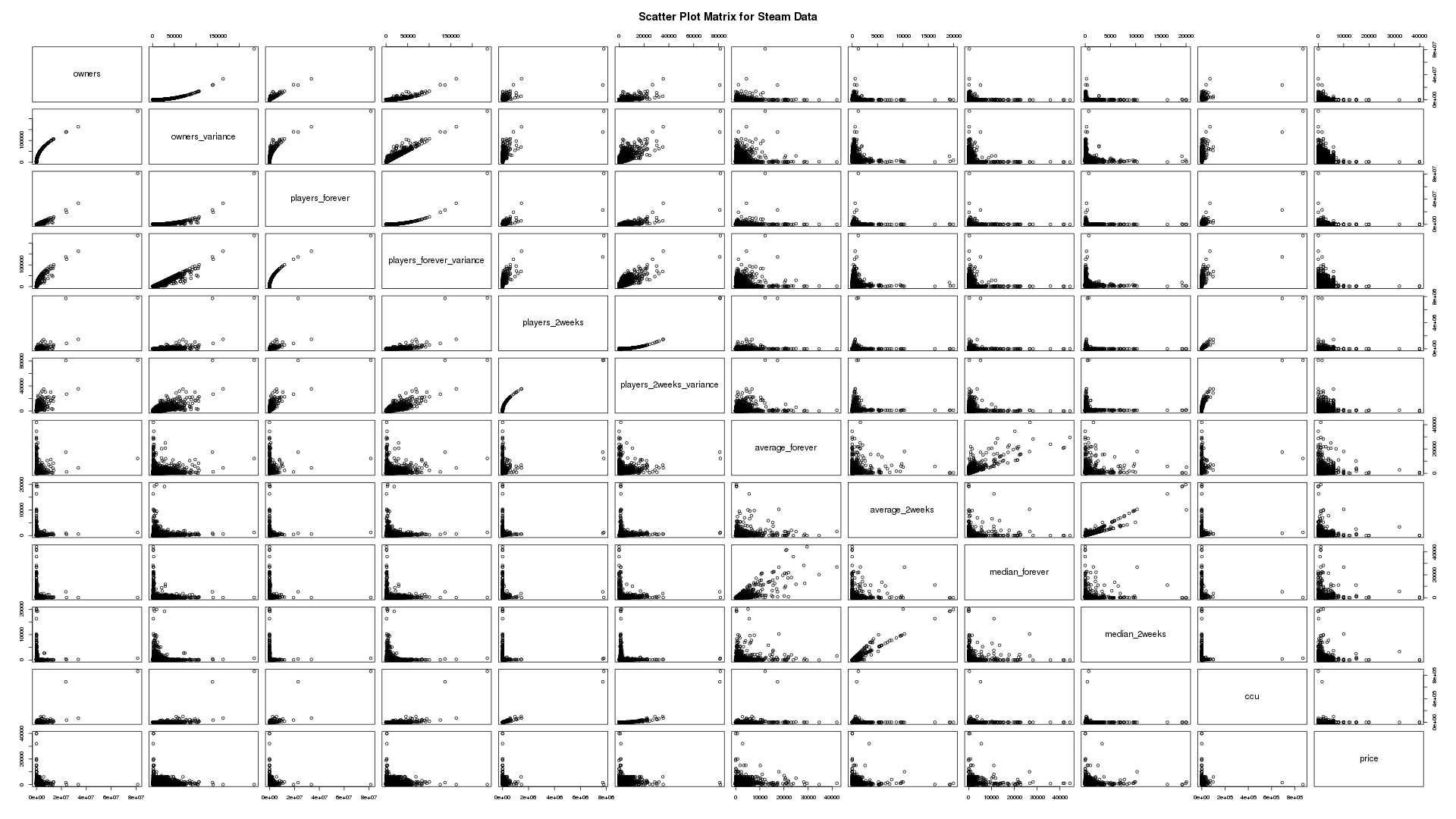


Figure - Initial Scatter Plot Matrix of the Dataset

***Data Understanding***

Data exploration occurred via R scripts, using R Studio.

The initial dataset was made up of 11956 instances, with 347 variables each. It was immediately apparent that certain features of the dataset were not going to be particularly relevant, such as the name of each game or the steam-assigned application ID number.

It also quickly became apparent that the target feature (average\_forever) was heavily skewed towards 0. This made a decent amount of sense and likely did not represent an error in the data itself, as steam is a store that anyone can easily publish a product to. There are likely many products on the store that have never been played for any significant amount of time.

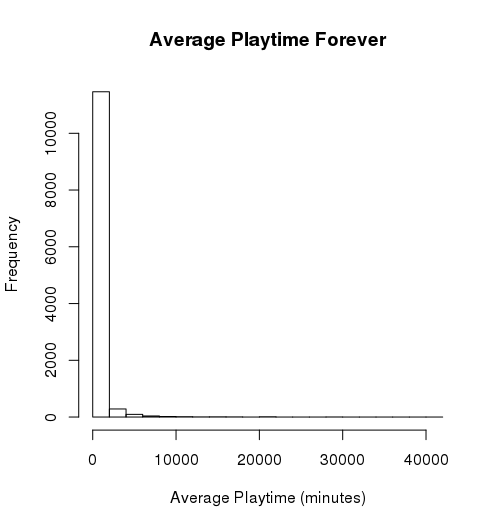


Figure - Distribution of Values in average\_forever

In addition to this, there were quite a few descriptive features that were highly correlated with each other. These features seemed to correlate so highly primarily because they were all derived features that described similar aspects of the data.

For example, there was players\_forever and players\_2weeks, describing total number of players for all time and the total number of players for the last two weeks.

These features seemed to be largely irrelevant to the target feature, yet gave a great deal of correlation. These pieces of data were effectively different metrics of how many people were playing the game. As such, they would likely have created an overfit model that was too reliant upon data that was not useful for real world predictions.

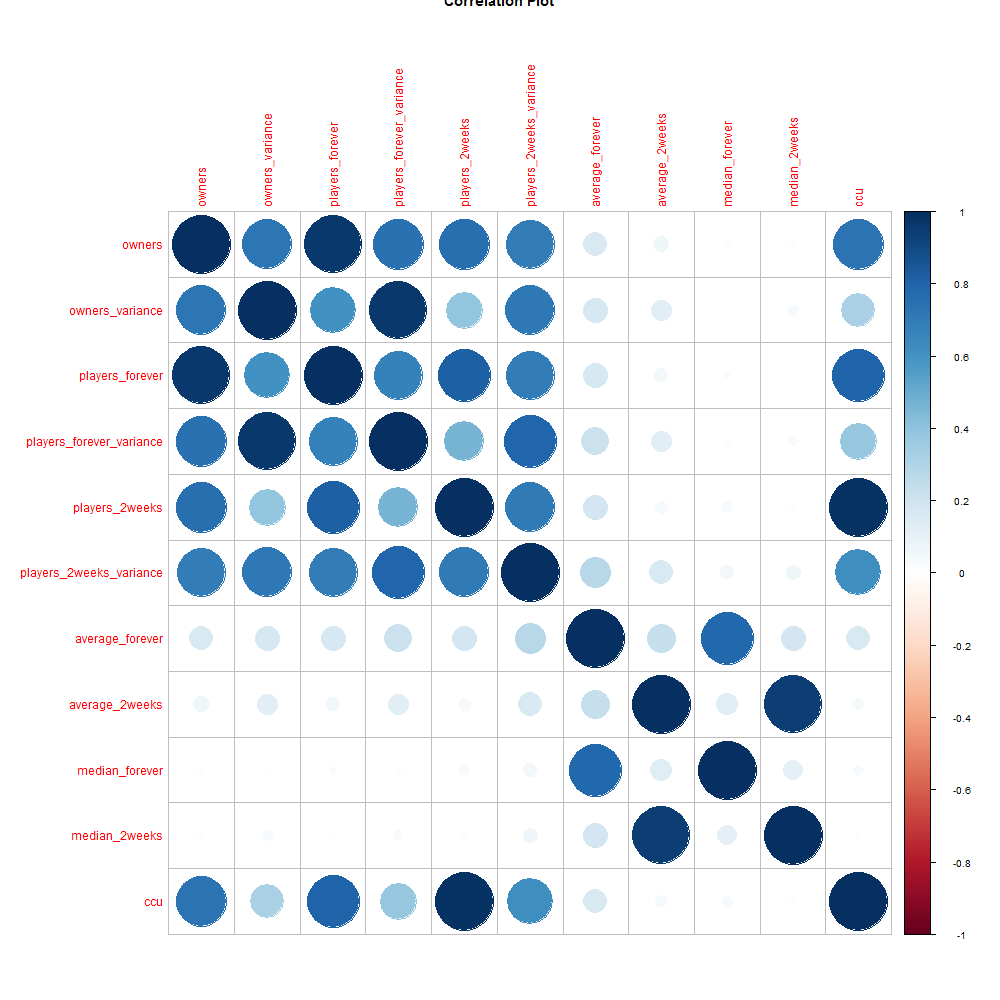


Figure - Correlation Plot for the Numeric Non-Tag Features

The majority of features in the dataset represent the tags a game can be tagged with to denote certain gameplay features or genres. Each instance in the feature was either a numeric value denoting the number of times the game was tagged with that label, or NA value.

This format was somewhat problematic, as it brought unnecessary weight to the tag values and left many NA values to contend with.

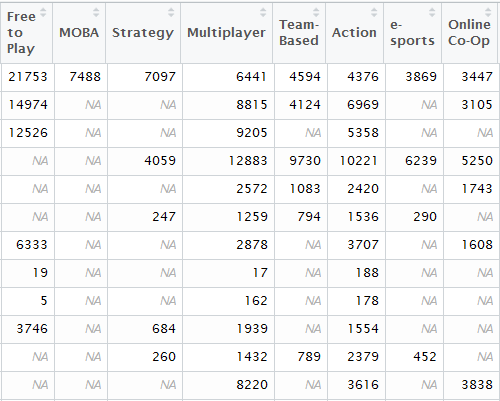


Figure 4- Initial State of the Dataset's Tag Features

As a result of the tag formatting, there was also an excessive number of features, some of which hardly ever actually occurred.

***Data Preparation***

Preparation of the data was also done via R scripts using R Studio.

**Tag Feature Format**

The first change that was made to the data was done to rectify the aforementioned issue with the tag feature format. Had the original format been used, classifiers would likely have been reading into the individual values to too great an extent, leading to deceptive results.

In addition, the presence of NA values would have caused problems for many classifiers later on.

The formatting of the tag features was changed to be a binary value of 1 or 0. The instance would be assigned a 0 for that feature if it previously had a NA value. The instance would be assigned a 1 for that feature if it previously had a positive numeric value.

This produced a feature format that would work much more effectively with the classifiers that would be used later in the project, and avoided the use of missing values.

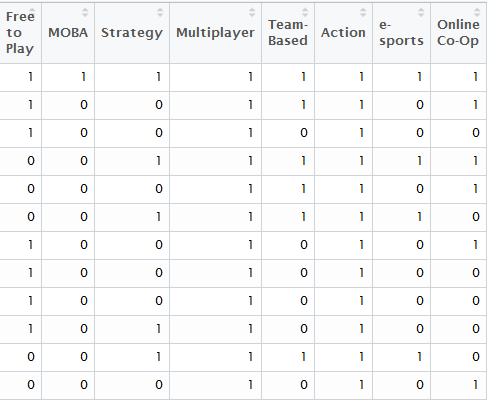


Figure 5- Final State of the Dataset's Tag Features

**Removal of Irrelevant and Highly Correlated Descriptive Features**

This change was necessary to avoid creating a model that was overfitting based on highly correlated descriptive features that held no value in relation to the target feature.

First, irrelevant features that held no bearing on the target feature were removed to avoid misleading classifiers. This included the Steam application ID, and the name of the game itself. Both of these features held values that never repeated, and were only used for game identification, making them completely irrelevant to the target feature.

Next, a single tag feature simply called ‘tag’ was removed from the dataset. This feature held no values, and was simply left over from the conversion process from JSON to CSV.

Following this, the aforementioned highly correlated descriptive features needed to be addressed. As previously stated, these features were all different metrics of how often the game was being bought/played. As a result of this feature similarity, their data was often very highly correlated, as they functioned based on the same fundamental calculations.

To avoid classifiers relying on and overvaluing these irrelevant features, they had to be removed. A total of 10 features were removed from the dataset for this reason. All of these features can be seen in Figure 2, along with the target feature (average\_forever).

**Further Removing of NA values**

Some of the values in the price and score\_rank features of the dataset had the value of NA, denoting that there was no pricing or score information available. This did not, however, denote that the game had a price or score 0, as there were far more values in the features which had a value of 0 than NA.

Upon further inspection of these NA values, it became clear that instances that had a price or score of NA represented products that were not standalone products. These instances were products that had been bundled with other products, and never sold on their own, making pricing and scoring information unavailable. Often these were simply tools, patches, or addons for the actual product itself.

Due to this, instances with values of NA in the price or score\_rank features have been removed from the dataset.

**Type Conversion**

Despite representing numeric values, the price and score\_rank features were nominal initially. To allow for proper comparison and classification, these features were converted from nominal to numeric features.

**Further Pruning of the Dataset**

At this point, the dataset still has 11690 instances with 334 distinct features, with some of these features only occurring a few times. To avoid leading the classifiers astray, the size of the dataset must be reduced.

To do so, each tag feature was tested for correlation with the target feature. The result of these correlation tests was stored in a new dataframe made specifically for this purpose. The 1st and 3rd quartile values of this new dataset were then found.

Any features whose correlation values were less than the 1st quartile or greater than the 3rd quartile values were kept in the dataset. All others were removed. The 1st and 3rd quartile values of the new dataset were then found, and the process was repeated.

This process was done a total of 3 times.

After removing these features, each instance now had to be checked to make sure they were not now devoid of all tags.

To do so, the number of tags applied to each instance was recorded. If this sum was less than 1, the entire instance was removed from the dataset, as it had exclusively been tagged with the now removed features.

As a result of these removals, the dataset now had a total of 10430 instances with 19 distinct features. This size is much more manageable.

**Discretizing the Target Feature and Skewed Data**

The target feature of the dataset is initially numeric. While this may work with some models, many classifiers cannot use numeric targets.

To attempt to rectify this, the target feature was discretized. This was done using equal frequency binning.

Equal frequency binning was used due to the aforementioned skew towards 0 in the dataset’s target feature. 10 bins were used.

During primary modelling, however, this method of discretization created models that were entirely useless, with accuracies ranging between 10%-20% and kappa statistics ranging between 0.0-0.2.

To address this issue, the dataset was converted into a two-class problem, described below.

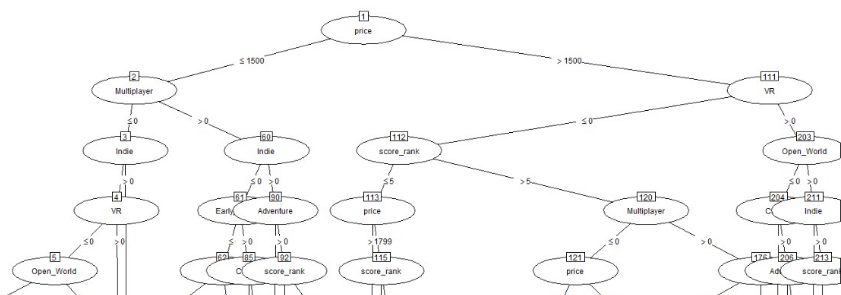


Figure 6- Upper Levels of C4.5 Tree

**Conversion to Two-Class Problem**

In an attempt to produce more useful models than those produced using binning, the dataset was converted to a two-class problem.

To do so, the target feature average\_forever was removed, and replaced with a binary feature called likely\_to\_succeed.

For each instance, if the value of average\_forever was greater than or equal to the mean of all average\_forever values, likely\_to\_succeed was set to “Yes.” If it was below the mean, likely\_to\_succeed was set to “No.”

***Data Modelling***

With the modifications to the dataset complete, numerous classifiers could now be run on the dataset. For these classifiers, the dataset was split into training and test sets. Two thirds of the dataset went to making the training set, while one third of the dataset went to making the testing set.

Modelling was done with a selection of five classifiers, with the implementation of each described below.

**Naïve Bayes**

The two-class dataset was used for the implementation of Naïve Bayes. This model was created using the e1071 package for R.

**C4.5**

The two-class dataset was used for the implementation of C4.5. This model was created using the RWeka package for R.

Portion of tree can be seen in Figure 6.

**RIPPER**

The two-class dataset was used for the implementation of RIPPER. This model was created using the RWeka package for R.

JRIP rules:

===========

(Multiplayer >= 1) and (price >= 2999) => likely\_to\_succeed=Yes (144.0/13.0)

(Strategy >= 1) and (price >= 1899) and (score\_rank >= 39) => likely\_to\_succeed=Yes (134.0/22.0)

(price >= 1399) and (Open\_World >= 1) and (score\_rank >= 41) and (Indie <= 0) => likely\_to\_succeed=Yes (55.0/9.0)

(price >= 1399) and (Indie <= 0) and (Multiplayer >= 1) and (score\_rank >= 73) => likely\_to\_succeed=Yes (22.0/6.0)

(price >= 1399) and (Open\_World >= 1) and (score\_rank >= 90) => likely\_to\_succeed=Yes (10.0/1.0)

(price >= 1399) and (Open\_World >= 1) and (Indie <= 0) and (score\_rank >= 19) => likely\_to\_succeed=Yes (27.0/7.0)

(price >= 959) and (Strategy >= 1) and (Indie <= 0) and (score\_rank >= 18) => likely\_to\_succeed=Yes (165.0/63.0)

(price >= 2499) and (VR <= 0) => likely\_to\_succeed=Yes (112.0/41.0)

(Multiplayer >= 1) and (Indie <= 0) and (Casual >= 1) => likely\_to\_succeed=Yes (42.0/17.0)

(Multiplayer >= 1) and (Indie <= 0) and (score\_rank >= 18) and (COOP >= 1) and (price <= 749) => likely\_to\_succeed=Yes (38.0/11.0)

(Multiplayer >= 1) and (Adventure >= 1) and (score\_rank >= 47) and (COOP >= 1) => likely\_to\_succeed=Yes (76.0/28.0)

(Indie <= 0) and (score\_rank >= 41) and (VR <= 0) and (Adventure <= 0) and (Action <= 0) and (price >= 349) and (Strategy <= 0) => likely\_to\_succeed=Yes (68.0/26.0)

=> likely\_to\_succeed=No (5365.0/698.0)

Number of Rules : 13

The model ended up having a total of 13 rules, and favored using features such as price, multiplayer, indie, strategy, and score rank.

**Random Tree**

The two-class dataset was used for this implementation of Random Tree. This model was created using the Weka KnowledgeFlow environment.

The final size of tree totaled to 4255.

**PART**

The binned dataset was used for this implementation of PART. This model was created using the Weka KnowledgeFlow environment. The total number of rules created by this model was 216, relying mainly on the same features as RIPPER.

**SVM**

The two-class dataset was used for this implementation of SVM. This model was created using the e1071 package in R.

***Model Evaluation***

After each model was created with the training set, they were then tested against the testing set. All testing of the models occurred in the environments they were created and trained in.

**C4.5**

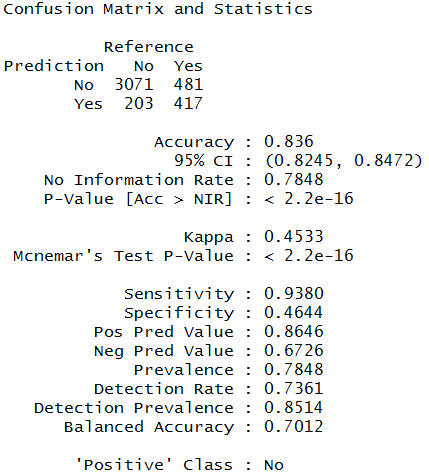


Figure 7- Evaluation Result for C4.5

The overall accuracy for the C4.5 implementation was 83.6%. While this is rather high, the kappa statistic is rather low, at 0.4533. This would imply that the model is only slightly better than guessing.

A reason for this may be found in the specificity and sensitivity values. The sensitivity is rather high, implying that the model can reliably classify a “No” value. However, the specificity is rather low, implying that the model cannot accurately predict a “Yes” value.

This may be due to the general skew towards “No” in the data, in which the majority of instances have a value of “No.”

Overall, this model is one of the higher scoring attempts at classifying the data. There are, however, still many problems with it, such as the kappa and specificity scores. Due to these issues, this model likely cannot be trusted to accurately determine whether or not a game will have a high average playtime.

**RIPPER**

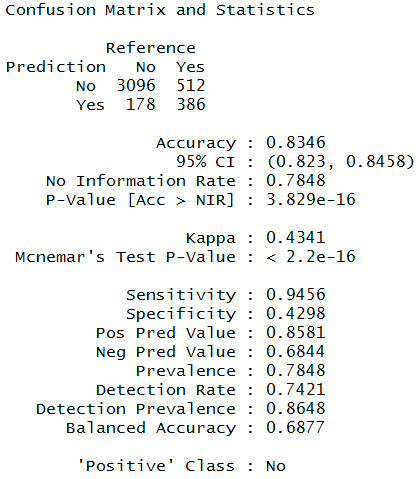


Figure 8- Evaluation Result for Ripper

The overall accuracy for the RIPPER implementation is 83.43%. The kappa statistic is 0.4341.

The skew between the sensitivity and specificity continues to be a problem, further implying that this is caused by either an underlying issue in the dataset or in the two-class splitting.

**Naïve Bayes**

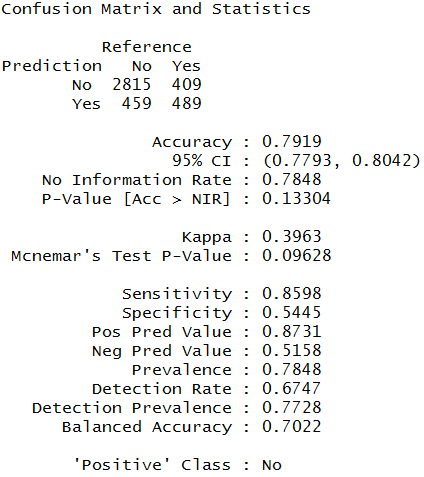


Figure 9- Evaluation Result for Naive Bayes

The overall accuracy for the Naïve Bayes implementation is 79.19%. The kappa statistic is 0.3963.

The skew between sensitivity and specificity values has been lessened somewhat, with the specificity value rising from the previous implementations.

The model is still largely unreliable, however, as the change in specificity is not enough to counteract the decreased kappa statistic.

**RandomTree**

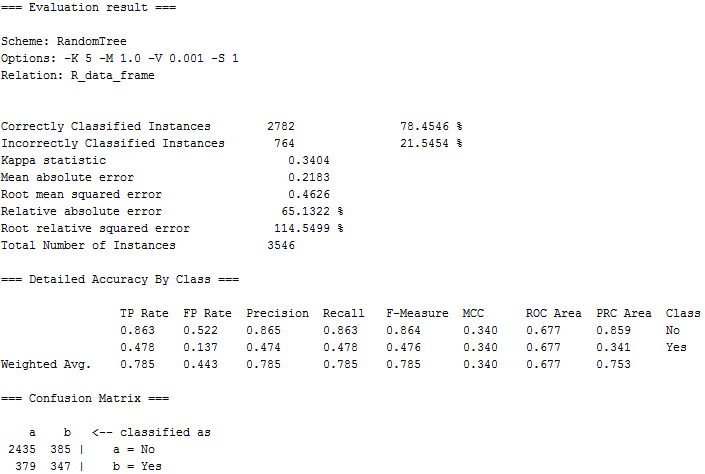


Figure 10- Evaluation Result for RandomTree

The overall accuracy for the RandomTree implementation is 78.4546%. The kappa statistic is 0.3404.

Both relative absolute error and root relative squared error are rather high, and the ROC area is too low to consider the model useful.

The skew between sensitivity and specificity is still present (here represented as the TP and FP rates of both classes), but is higher than some of the other models.

Overall, this is the worst of the models used, and seems to hold no particular value.

**PART**

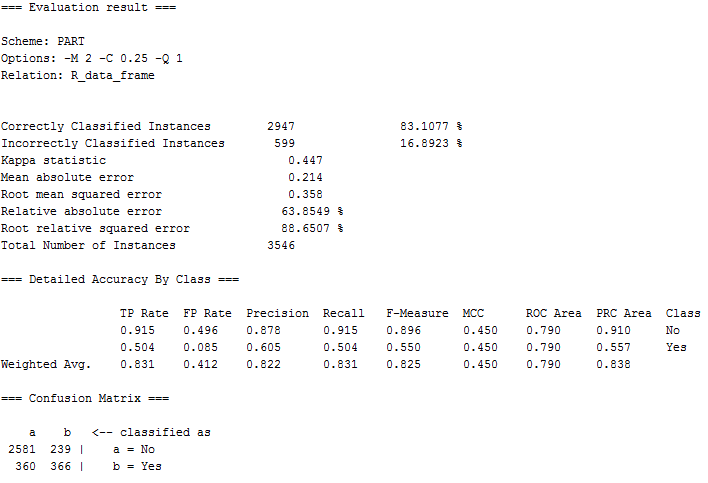


Figure 11- Evaluation Result for PART

The overall accuracy for the PART implementation is 83.1077%. The kappa statistic is 0.447.

The relative absolute error and root relative absolute error have come down a bit from the RandomTree vaues, but are still rather high.

The ROC area has increased a great deal, and appears to be decent, but the difference between the TP Rates is still largely present.

Overall, this model is one of the higher performing ones, but still has the same issues with correctly classifying instances of “Yes.”

This further points toward an issue in the underlying data, likely the way the data was split during the two-class conversion. An abundance of “No” values is likely skewing the models to place values in “No,” as it has a decent chance of being correct.

This is reflected in the low kappa statistics, as random guessing would also likely get a great deal right simply due to the abundance of “No” values.

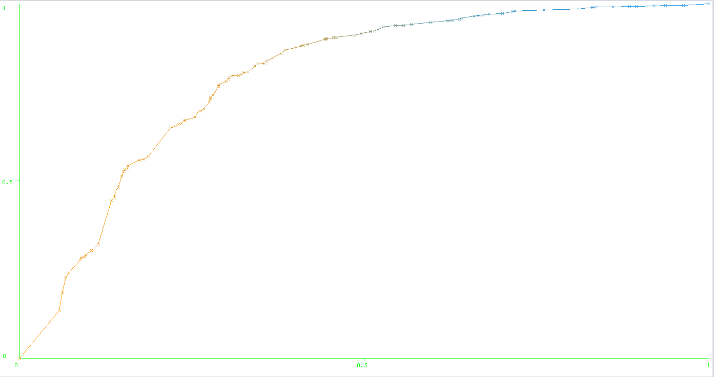


Figure 12- PART ROC

**SVM**

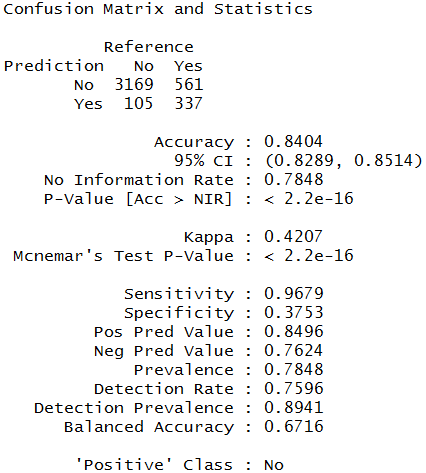


Figure 13- Evaluation Result for SVM

The overall accuracy for the PART implementation is 84.04%. The kappa statistic is 0.4207.

The skew between sensitivity and specificity is rather prominent in this model, even more so than other models. This means that this issue was present in every single model built, furthering evidence of an underlying error or complication in the two-class conversion process.

**Conclusions**

The original goal of this project was to find a way to predict how long a game would be played after purchase, based upon certain aspects of the game itself.

Determining the average playtime per owner of a game would assist in determining what types of games would have the most longevity, and therefore potentially create the most consistent revenue.

To that end, data was extracted from Steam using the SteamSpy API, converted into a usable format, and submitted to the CRISP-DM process to attempt to create viable models.

Numerous issues with the data were found in the data understanding phase, and these issues were addressed in the data preparation phase.

From here, the completed data was used to create, train, and test models in an attempt to predict the longevity of a game product.

Unfortunately, the project seems to have failed to create a reliable method of doing so at this time. As can be seen from the evaluation of the created models, none of the produced classifiers seem to be useful in predicting the playtime of a game.

The first issue that plagues all the models is that of low kappa values. While these values are not as low as those produced by the original binned data, these values never rose above the point where they might be considered much better than guessing.

The second issue, and likely the largest one, is the skew between sensitivity and specificity. While most classifiers had a high true positive rate for “No” instances, this was coupled with high false positive rates for “No,” and low true positive rates for “Yes.”

This, combined with the low kappa values, implies that the created models are simply guessing “No” because doing so will still produce decently high accuracy rates.

This is likely due to an error made when converting the dataset into a two-class problem. The final split in the target feature was roughly 20% “Yes”, and 80% “No.”

Having said this, those models that performed the highest made classification choices that seemed to make sense.

Among C4.5, RIPPER, and PART, all three relied on generally the same features to make a determination regarding the game’s playtime. These features being price, multiplayer, strategy, score rank, coop, and indie.

If the price tended to be higher, the model tended to classify the instance as a success. This is likely due to the fact that larger games are often sold at a higher price point than shorter, independently made games. In games where the price is higher, there may simply be more content for the player to play through.

If the game had been tagged as multiplayer, the game tended to be classified as a success. This makes sense as well, as having a multiplayer component can extend the life of a game beyond the crafted singleplayer content, with players often spending a great deal of time experiencing the same online content, over and over. This is also likely the reasoning behind the use of the coop tag as well, as cooperative play is another form of multiplayer gameplay.

If the game had been tagged as strategy, the game tended to be classified as a success. Many strategy games are also multiplayer games, so this may explain the use of this feature. In addition, strategy may imply a type of game that simply lasts longer, due to the nature of the genre.

If score rank was higher, the game tended to be classified as a success. Score rank is a direct measurement of a game’s quality, determined by the players of that game. It makes a fair bit of sense that higher quality games would receive more playtime than low quality ones.

If the game had been tagged as indie, the game tended not to be classified as a success. This could be due to either the size of these games, or their quality. Indie (independently made and published) games are often the work of a small studio, team, or even a single person. This often leads to these games having fewer features and less to do. In addition, it can often lead to games that are simply not very well made. Both of these possibilities could explain why they are used in the classifiers.

Overall, however, none of the classifiers can be reasonably trusted.

To attempt to rectify this, more testing with the two-class conversion would be necessary. Choosing a better split point that leads to a more even distribution could be a good start.

In addition to tuning the two-class conversion, other methods of dealing with the heavy skew in the target feature could be explored. Perhaps one method may lead to models that accurately represent some form of relationship in the data.