Course Project: League of Legends

Preston Robertson

Department of Industrial and Systems Engineering, Mississippi State University, Mississippi State, MS, 39762 USA

Keywords: Long-Short Term Memory, Squish, Neural Network, Activation Function

1. Dataset Description

1.1. Introduction to the Dataset

This dataset is on an online competitive game called League of Legends. I chose this dataset to challenge myself; the dataset's unique nature will require me to apply techniques I have learned in classes this semester, while teaching myself and applying new techniques. The objective of the dataset is to find the features that will most impact the win, so that it is easier to balance the game. "Balancing" refers to updating the game, such as downgrading strategies that are too strong and upgrading strategies that feel too weak. This document will serve as the preliminary to champion balance by correlating specific stats to a win, so that in the future someone may correlate these stats to champions in the game. This goal may seem convoluted; however, each game will have 5 winners and 5 losers meaning that a single champion's impact on the game is roughly 10%. If a champion is unbalanced, being double the strength of another champion, that only raises the 10% to 20%. Due to the law of averages, each champion will still have around 50% win rate despite being too weak or too strong. Therefore to properly balance a champion, one must look at the correlation between stats of the champion to the win-rate of each of those stats. This effectively takes out the problem of bias since every game has at least one winner and one loser. This dataset has 23,752 data points and 24 features (or columns). Features refer to a measurable piece of data, such as champion name, damage done, etc. These features do not refer to game features but rather the name of the data being measured. It is a complicated dataset, with several variables requiring several stages of feature modification to run the code. The code is also large enough to have significance. This dataset was specifically chosen due to my prior familiarity with the data, allowing me to focus on the machine learning techniques.

Email address: pgr41@msstate.edu (Preston Robertson)

1.2. Brief Description of League of Legends

The online competitive multiplayer game, League of Legends is a part of the MOBA genre of games and is considered the most widely popular game of all time. Its most recent tournaments have made more money than the Superbowl; in 2020 League of Legends made 1.7 billion dollars **??**. The premise of the game is two teams fighting to destroy the enemy Nexus. Below is the map of the game so it is easier to reference the variables given.

Figure 1: Map of League of Legends ([1])

1.3. Description of the Map

The map of League of Legends contains 3 paths, each a lane with a corresponding name. Top Lane, Mid Lane (short for Middle), and Bot Lane (short for Bottom Lane). Each lane spawns "minions" to help push lanes. These minions are very easy monsters to defeat and provide gold if a player lands the killing blow. Each lane has 3 towers and an inhibitor. All three of one lane's tower + inhibitor need to be destroyed before a player can reach the nexus. The towers provide protection to players by hurting enemy champions when they are in range. The inhibitor provides no uses to ally team; however, if a player destroys

the enemy inhibitor then "Super Minions" will spawn in that lane. These buffed minions help push to finish the game. There are also forests called the Jungle in the middle of these lanes. In the jungle there are several monsters' worth gold and that grow stronger as the game goes on. Some monsters, if killed, can even provide special bonuses. All these monsters can be killed by one player. The blue section seen in 1 that splits the map into two sides is known as the river. This is the equivalent of the half-way line in soccer. In this part of the map, Large Monsters spawn that require a group effort to takedown but give huge bonuses.

1.4. Description of the Gameplay

The game is known for its complexity and if you want a comprehensive guide, I have provided a link that I think does an excellent job (([2]) and ([3]) provide great comprehensive guides). This paper will explain only the minimal necessities to follow the data. There are 5 players on each team and each player plays a champion. This champion is a character with unique gameplay, stats, and abilities. These 5 players will each fill a specific role: Top Lane goes to the Top Lane, Mid Lane goes to the Mid Lane, the Jungler goes to the Jungle, and the Attack Damage Carry (ADC) and Support go to the Bot Lane. In each of their respective locations, each role attempts to make gold and level up. The gold is used to buy items (each with unique effects, and every champion picks from the same pool of items), so more gold means a player can buy more items. Player get gold by killing different things on the map (for example: minions, monsters, large monsters, and enemy players) and can also be obtained by everyone at a constant passive rate. The more items a player has the stronger and more capable they are, and therefore the more likely their team is to win the game. Levels occur naturally when the player gains experience by being around combat or by killing enemy units (same as gold) this; however, does not passively increase and must be earned. With each level up the basic stats become stronger, which means the player is stronger and can level up current abilities or unlock new abilities.

2. Data Analyzation, Visualization, and Pre-processing

2.1. Further Description of Dataset

The dataset went through preliminary data analysis, which is discussed in this section. First the feature selected to study is "Win" which is a Boolean variable; this means that the machine learning models used in this experiment will be classification models. Basic analysis was done on the dataset, which can be found in the appendix as Figure 2: Basic Data Analysis of Numerical Features. Looking at this information, dropping the feature "Damage to Turrets" could increase the accuracy of our model since the standard deviation is higher than the mean. It can also be gathered that this data could cut the farthest data points to see an increase in model accuracy since the maximum and minimum are so far from the mean in most features. However, I will not take out these variables since the objective of this experiment is to change as little as possible to not interfere with results.

After the data analysis was completed, the next step was running checks on each feature to ensure that each was capable for this experiment. A cardinality check and a missing percentage were taken to ensure these qualities. The cardinality was taken since if a feature has far too many unique variables, then the feature will only harm the machine learning algorithms. This missing percentage was taken to ensure no feature had too many variables missing where data analysis on that feature would be impossible. Refer to the appendix for Table 5: Cardinality of Each Feature and Table 6: Percentage Missing of Each Feature for specifics. This analysis has shown that no features were missing variables and that only one feature has too high cardinality to be included. This feature was the anonymized player name that basically became a second index. This feature will be dropped since it serves no value in this dataset for either machine learning algorithms or game balance.

2.2. Data Visualization

The next preliminary data analysis that were conducted on the dataset were a heat map (correlation matrix) and a scatter plot. A heat map (shown in Figure 2) is basically a correlation matrix that shows how much any two features interact with one another. The heat map improves upon this idea by also adding the temperature value so it can easily be seen how much two variables correlate. This temperature value can be any value between -1 and 1, where -1 is a sign of perfect negative correlation and +1 is a sign of perfect positive correlation. The darker the color on the heat map, the "colder" the temperature and vice versa for the brighter colors. Looking at the Figure 2, some values have very high correlation; however, looking at most of these variables, most features will increase in value as the time moves forward. This gets rid of most of the correlation since these values are interconnected through the feature "duration". Even with this information, there are some values of note. For example, there is a high correlation between "Win" and "Win Rate of Champion" (the highest correlation to win) while "Win" and "Champion" do not correlate. This shows the problem discussed earlier in this experiment where the natural bias of the game does not allow for just a normal correlation to be sufficient. Other information to gather from this heat map, is that values previously thought important for a winning team are not very highly correlated to a win. This shows the necessity of such data analysis to the game. Not much else should be deciphered from this heat map since it is helpful when visualizing data, but the most precise with results.

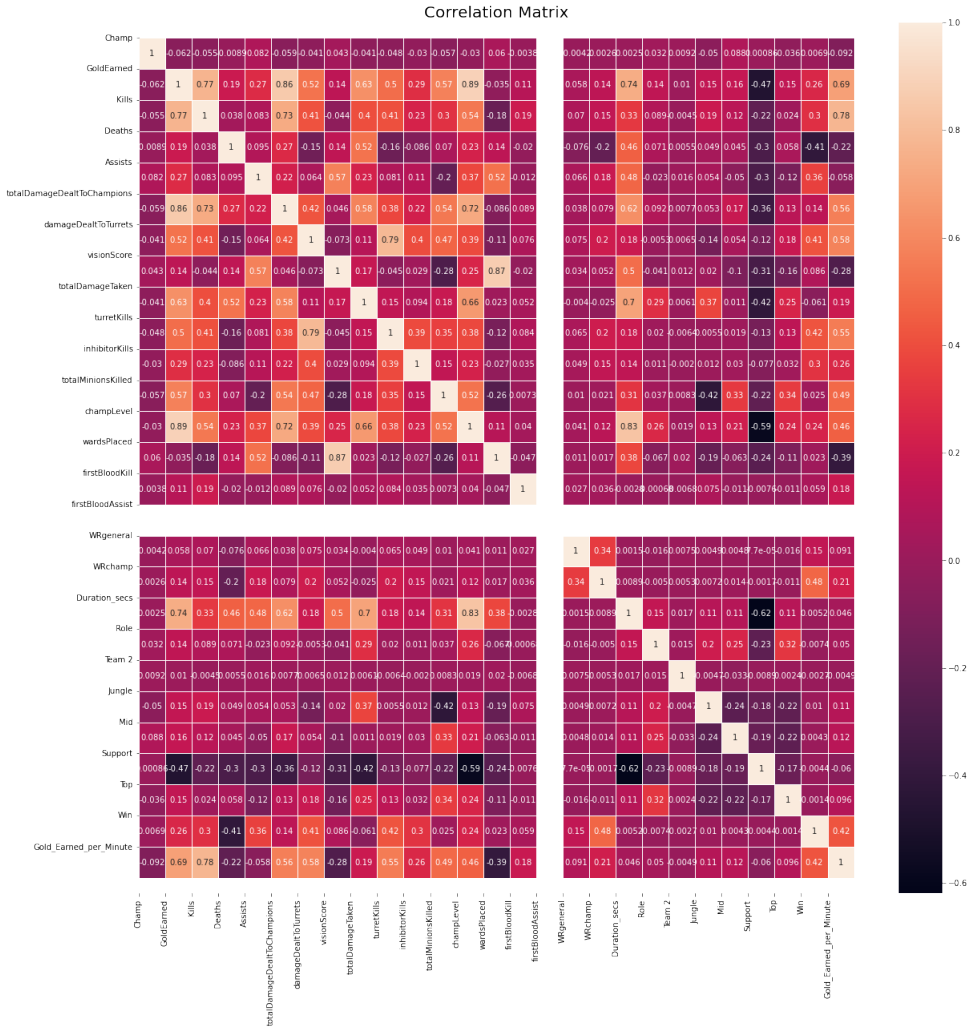

Figure 2: Heat Map of the Dataset

The next data visualization done to the dataset was a scatter plot. A scatter plot graphs each feature with respect to each other. This can be very useful for seeing trends visually before any data analysis is done. Figure 3 shows the specifics of each section of the scatter plot. Not much can be gathered from the champion section of this scatter plot since each champion is given their own x value and there is no way to see the x values with the size of this scatter plot. Comparing "Kills" and "Deaths", it is possible to see the problem discussed earlier with the heat map. Looking at both stats, it is easy to see that the amount of gold earned increases no matter what due to each variable being connected through the duration of the game. However, it is possible to see that each variable has its own effect in the scatter plot still, since there is a difference between "Kills" and "Deaths".

The main take away from the scatter plot is the difference being consolidated data and more spread-out data points for all features. That is best information to gather from this scatter plot since the scatter plot is used to visualize the data and not supposed to be used for thorough data analysis.

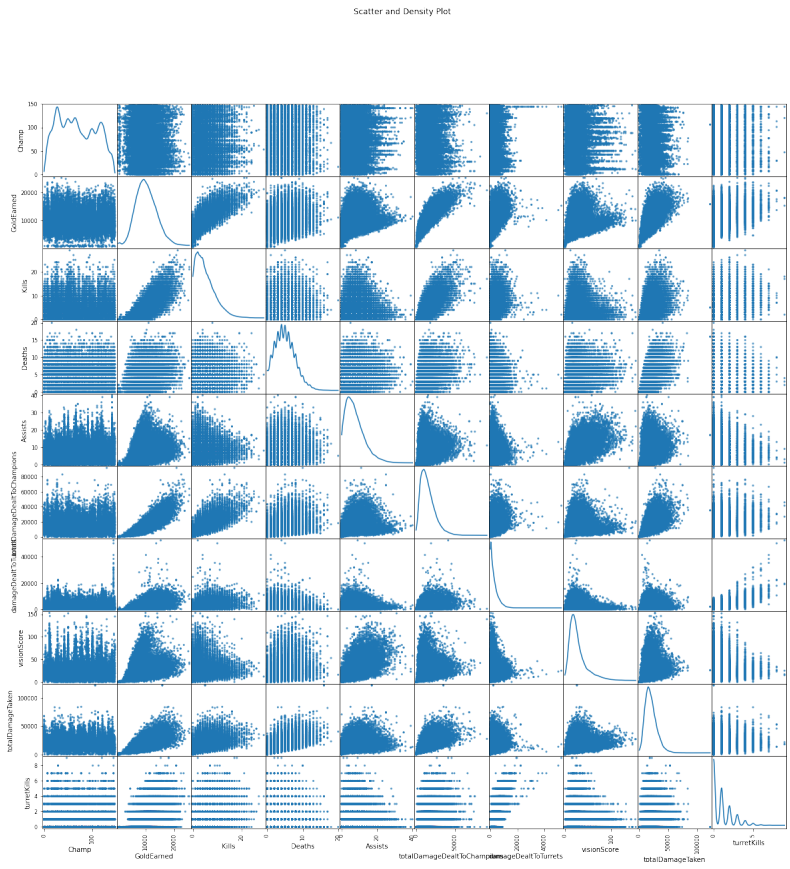

Figure 3: Scatter Plot of the Dataset

2.3. Data Pre-Processing

After the dataset was thoroughly analyzed, the data was processed to prepare it to be put through machine learning algorithms. This set-up is rather short since the point of analysis is to be thorough and not exclude many features or samples. Since time is not an issue with this dataset it is not a problem to perform feature reduction on the dataset. This is important to keep track and ensure the machine algorithm does not lose value of any one feature. Despite the Principal Component Analysis method mentioned in the project proposal, it will not be implemented for those reasons stated above.

First data pre-processing done on the dataset was using a “One Hot Encoder”. This

algorithm splits up string values into their own column and turns into a Boolean value. For example, in this dataset, Position was changed into 4 new features with capability of displaying all 5 unique values in that feature. This was used on the features: Team, Position, and Win. This method can quickly accumulate the number of features a dataset has, so I thought it was important to include the lowest cardinality string variables to allow more the most accurate predictions without having 100+ features. After this is applied to the dataset, there are now 27 features (including the features removed previously). The rest of the string values were processed through a string indexer which turns all string variables into numbers. This has the benefit of being easily recognizable to the machine but has the downside of the computer thinking the value is a continuous integer; however, this process is necessary to do before running some of the chosen machine learning algorithms. Ideally, One Hot Encoding would be the best option if it were not due to the cardinality of some features in this dataset. The next piece of data preprocessing was adding an interaction term, Gold per Minute. This stat combined gold and duration of the game to see if the rate of the gold gathered had an impact on win. Finally, the data was split into testing (20%) and training data (80%).

3. Methods and Performance Comparison

3.1. Design of Experiment

As discussed earlier in this document, this dataset is specifically a classification problem trying to determine the Boolean value of the feature "Win". Each algorithm was selected to classify a discrete value rather than a continuous one. Each algorithm will have the same testing and training data and will be compared to one another using Training Accuracy, Testing Accuracy, and the R2 score. These metrics were chosen to measure performance since training accuracy and testing accuracy can be accurate measures of performance in a classification problem. The R2 score was selected to compare the model with other models that did not have the same dataset. The final metric taken was time. Even though, it is not very important in this analysis since time is not a heavily impacting factor, it was still taken since it can help decided which model is best for this dataset. Before the discussion of each dataset, it is important to note that each algorithm had its hyper parameters tuned to perform optimally for this experiment. The results of the parameter turning are listed below with the discussion of each machine learning method. The appendix includes resources and references for more details such as Figure 4.

3.2. Random Forest

The Random Forest algorithm is a non-parametric model that performs the best in classification. Random Forest is a combination of several decision trees and makes decisions based on a subset of the data. Since most of the features in the dataset are binary, it was assumed this model would perform the best. The parameters tuned were the max depth, the max features, and the number of variables in each subset. The max depth is farthest

the random forest can construct a decision tree. For example, if the max depth is 3, then the maximum number of decisions in each decision tree is 3. This keeps the model from overfitting and improving the run time. The max feature is selecting if any feature reduction should be done to the dataset before each decision tree. The number of variables per subset is self-explanatory. The results of the Random Forest Classifier grid search were: 'max˙depth': 40, 'max˙features': 'log2', 'n˙estimators': 100 This shows that the Random Forest Classifier attempts to overfit the model to achieve the optimal values. This shows this model might need to be further tuned if this model is selected. The results of the model are: Test Score: 90.107, Training Score: 99.994, R2: 0.604 This model performed very well, and might be the candidate for future analysis if tuned even further.

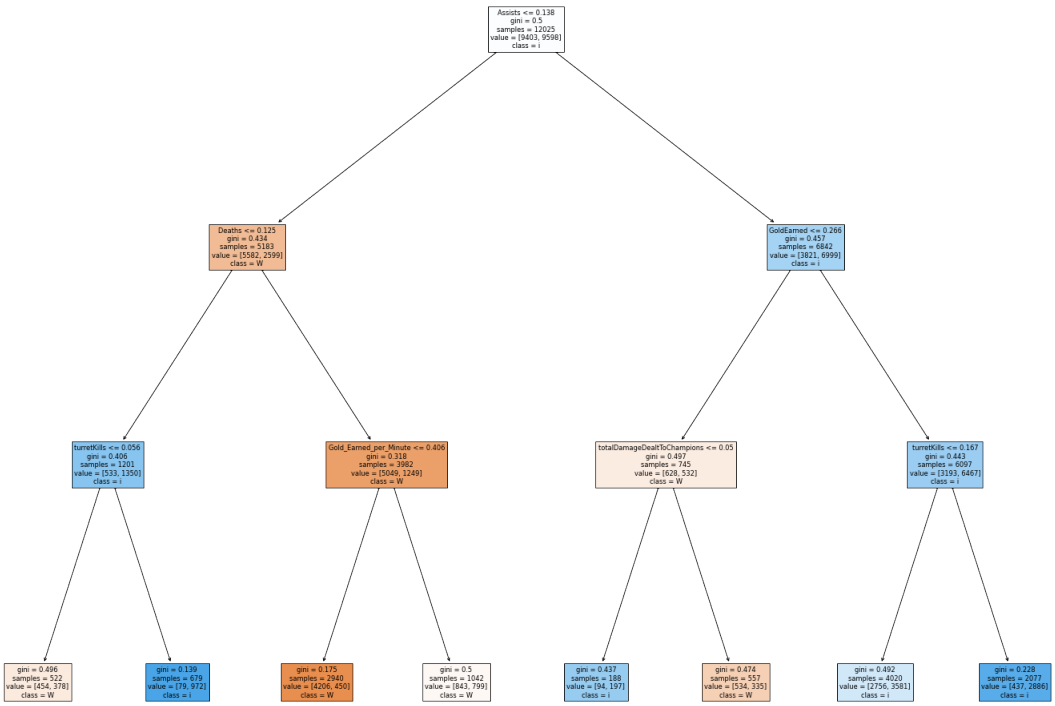

Figure 4: Example of one layer of the Random Forest

3.3. Naïve Bayes Classifier

Naïve Bayes Classifier model is the simplest model selected in the group of 5. The model was expected to perform the worst despite the nature of the dataset being a great

fit for this algorithm. Naïve Bayes does not have any hyper parameters to tune so once implemented the results are the best they can get. The results of the Naïve Bayes Classifier are: Test Score: 79.183, Training Score: 77.785, R2 Score: 0.167 This model performed better than expected, and under fit the training data. The results are not as high as needed, so this model will not be selected.

3.4. Support Vector Machine

The support vector machine is a supervised learning algorithm that can perform both regression and classification analysis. The support vector machine works by projecting the data to a higher dimension this allows the use of a hyperplane to split the data into each classification. The parameters tuned with this model were the kernel and initial C value. The c value is the regularization parameter and is a non-negative value. The higher the c value, the lower the initial regularization of the model. The kernel is the specific type of SVM model used in the experiment. Among several options, the ones picked to be tuned were rbf, linear, poly, and sigmoid. The results of the grid search were: 'C': 2, 'kernel': 'rbf'. This shows the model performs best with lower regularization and uses the standard rbf kernel. The support vector machine performed as followed: Training Score: 90.342, Testing Score: 89.854, R2 Score: 0.594. This model performed very well, and its run time was not as high as expected. This model could be a great candidate for research with this dataset.

3.5. K-Nearest Neighbors

The next model tested with this dataset was K-Nearest Neighbors (KNN). This model finds the nearest datapoints to the tested datapoint to determine the classification. The model was chosen since KNN performs very well under 2 parts classifications. The parameters tuned with this model were the number of neighbors considered, the weight of each neighbor, and the leaf size. The number of neighbors is important because if the value is too large or too small, then it will be difficult to accurately compute what classification the datapoint should be. The next parameter, weights, is the effect each neighbor has on the datapoint. For example, with uniform weighting, all neighbors will have the same weight, but with distance weighting, closer neighbors will have more impact on the classification than farther neighbors. Finally, the leaf size is a value that changes how the values are stored and how many. This mostly affects the speed of the model. The results of the grid search were as follows: 'leaf size': 2, 'n˙neighbors': 10, 'weights': 'distance'. This shows the leaf size should be much smaller than normal for this dataset, only 10 neighbors should be considered, and each of those neighbors should be weighted on how close they are to the datapoint. This model still overfit the data with scores of: Training Score: 100.0, Testing Score: 86.213, R2 Score: .448. Looking at the difference in testing and training accuracy, it is easy to see this model is still overfitting the data despite the parameters being tuned. The R2 score also reflects how model was overly complicated towards the training data. This model could possibly be further improved on but will not be used for this experiment.

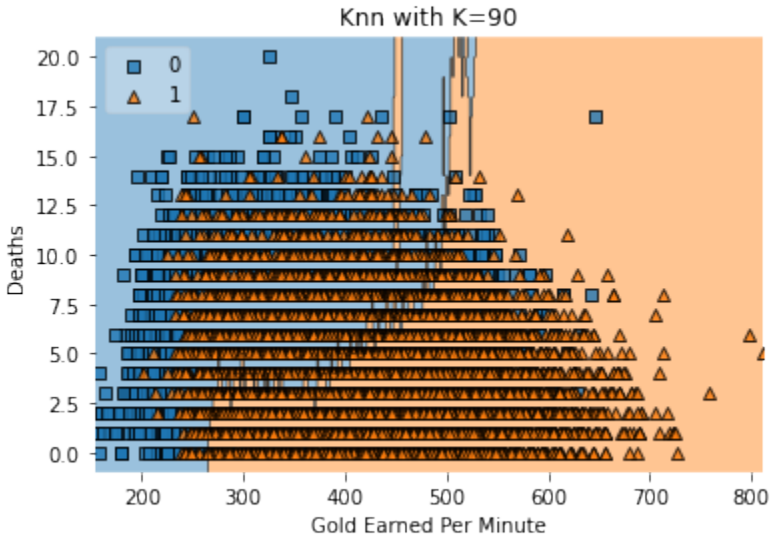

Figure 5: Example of an individual KNN mapping the "Win" feature

3.6. Adaboost

The Adaboost method was the final algorithm selected. This model was not used in this class; however, I thought it would perform very well in this data set. The model attempts to fit a classification map to the dataset and will continue to add more and more classification maps changing the weights to describe the dataset more accurately. The parameters tuned with this model were the number of estimators and the learning rate. The number of estimators is how many times the weights can be changed before the model stops. Ideally, it is important to find the perfect number of estimators to allow for an accurate model but does not have overfitting. The learning rate is how much the classification map is allowed to change with each iteration. The same as gradient descent, finding the optimal learning rate is important to not skip global minimum or to find the global minimum sooner. The results of the grid search were: 'learning rate': 0.7, 'n˙estimators': 80. This learning rate is very close to optimal, and the number of estimators was higher than anticipated for the parameter tuning. There could be more testing done to optimize the number of estimators; however, this value of 80 estimators kept the model from overfitting the training set. The results of the Adaboost were: Training Score: 88.926, Testing Score: 88.718, R2 Score: 0.548. These results were not as good as expected; however, that could be attributed to user error, since I am very new to Adaboost models. This model should be considered for this dataset.

3.7. Comparing Each Model

This section will not contain much text other than show the results of each model used in this experiment such as 2. Since model selection is not as important since all models

have a test accuracy of around 88+% (Other than Naïve Bayes) as shown in Figure 7. Figure 8 depicts the training scores of each model. This allows for the selection of any of the models and to do a deeper dive in to parameter tuning.

Table 1: Table caption

Algorithm	R2 Scores	Testing Scores	Training Scores	Run Time (s)
Random Forest	0.604173	90.107	99.994	262.217
Naïve Bayes Classifier	0.167079	79.183	77.785	0.058
Support Vector Machine	0.594067	89.854	90.342	213.238
K-Nearest Neighbors	0.448369	86.213	100.000	229.297
Adaboost	0.548589	88.718	88.926	101. 849

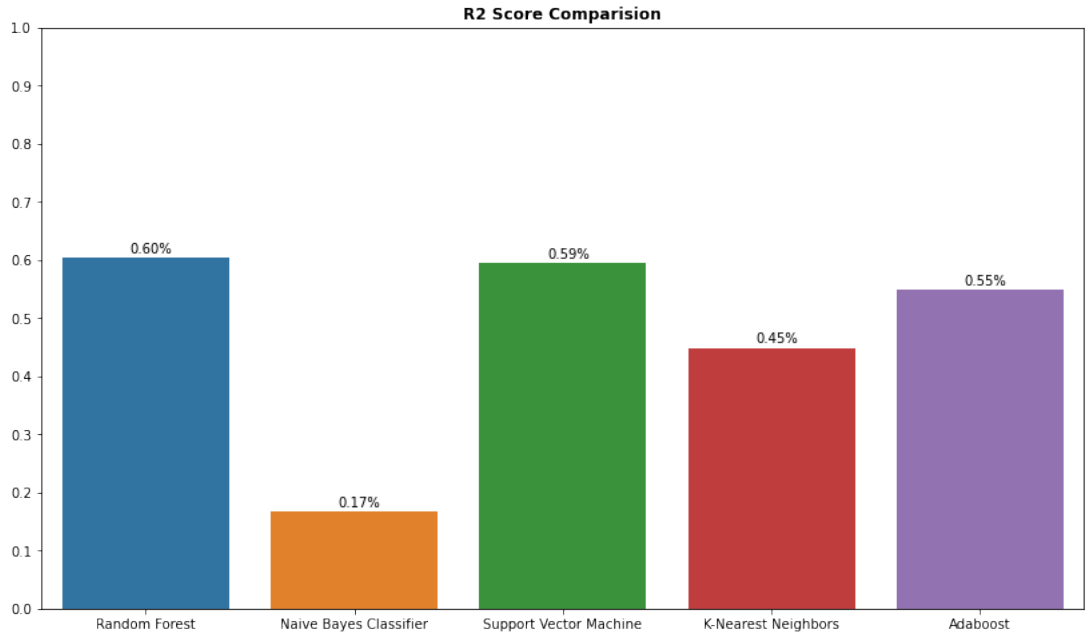

Figure 6: Visual Comparison of R2 Scores

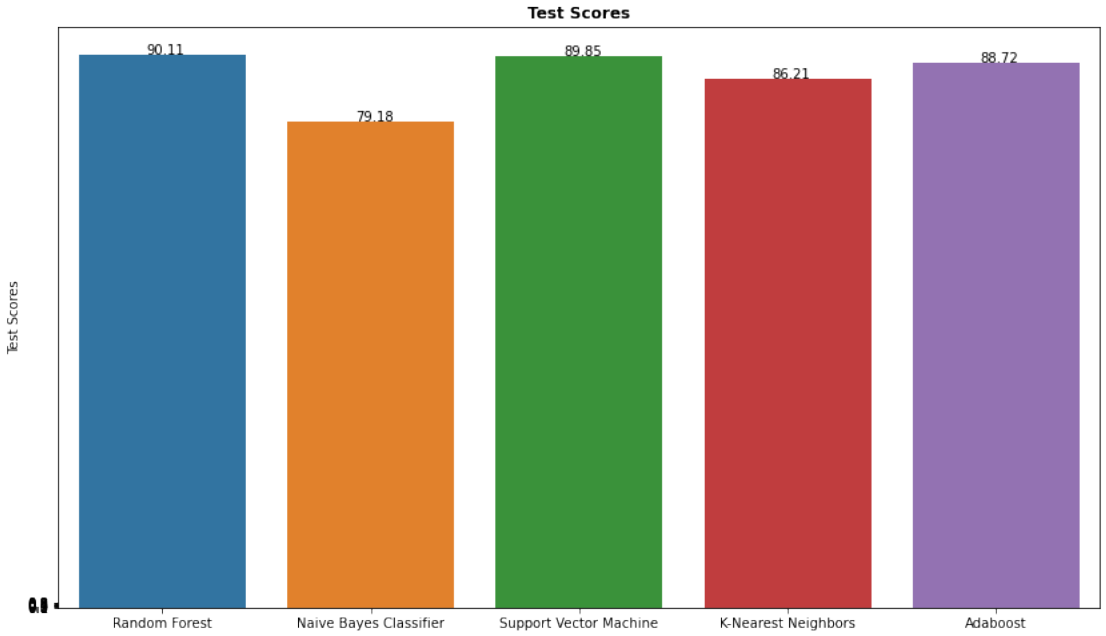

Figure 7: Visual Comparison of Testing Scores

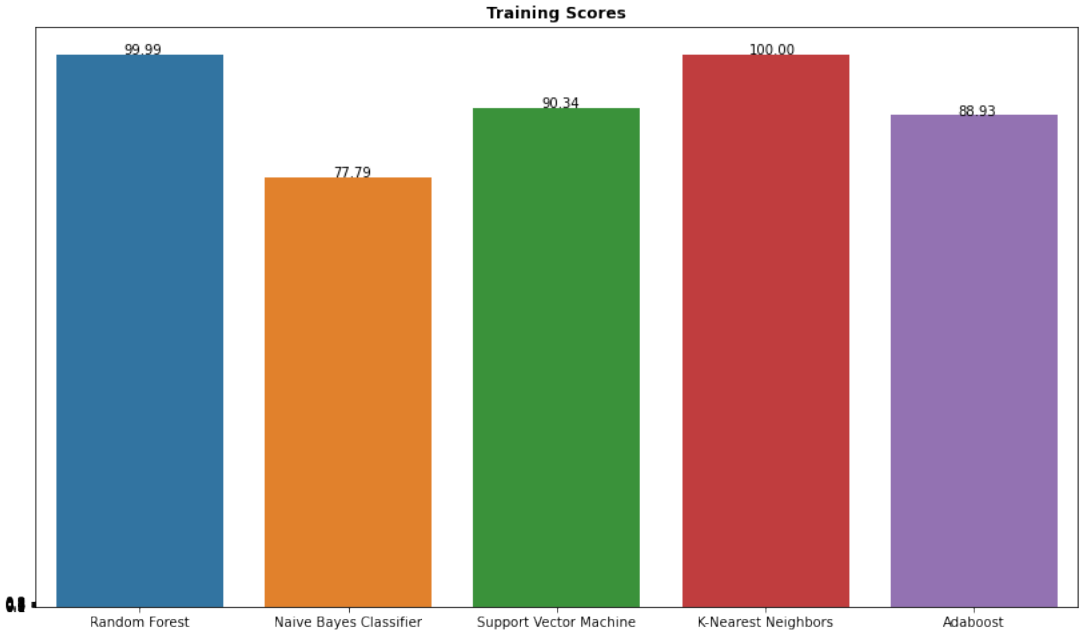

Figure 8: Visual Comparison of Training Scores

4. Results and Discussions

4.1. Possible Improvements

Before the conclusion and results of the paper, it is important to discuss possible improvements that could be made to this experiment. These improvements were not included due to either limited time when running the experiment or the lack of knowledge on how to implement these improvements. The first improvement is properly scaling values at during data preprocessing. After the experiment, I realized that I could min max scale the string indexed value allowing for faster computing time of all machine learning techniques. The second improvement would be implementing a neural network. These can be very taxing to create, taking a great deal of time to even begin having a working model. This neural network would show great improvements to the model since the gradient descent and back propagation of neural network models can very easily be implemented into this dataset due to the nature of all datapoints rising over time making a global minimum easier to find than it in random locations. Another improvement can be made the machine learning model's tuning in this experiment. If more time is dedicated to properly tuning each model, one could increase the accuracy even further. Since I did not have time to tune each model accordingly, I did not want to tune any of the models in-depth. Another improvement could be changing the K-Nearest Neighbors to Radius Neighbor Regressor or possibly K-Nearest Centroids. These models were not selected due to time constraints, and would not qualify for the project; however, these models could show an improvement in testing accuracy. The final improvement would be to add more interaction terms like Gold per Minute. Adding this term increased the accuracy of all 5 models, suggesting that adding more interaction items would increase the accuracy of testing results.

4.2. Conclusion

Onto the results of this experiment, the main conclusion is that it is possible to balance the game using machine learning. Machine learning algorithms can show the relationship between each stat and the win. With earlier preliminary data analysis also proving that the game needs other methods than just looking a win rate to determine if something is too strong or too weak, machine Learning techniques have proven through this experiment to work very well with this dataset, having upwards of 90% testing accuracy, which could possibly be further improved upon. This shows that there are strong correlations in the dataset, meaning there are reasons to do an algorithm by hand. Showing the math and work for one of these, preferably the most optimal, model will show the correlation each stat has to winning the game and the data points that help and/or disrupt. Since that is outside the scope of this project, This was not conducted; however, the algorithms prove that it is indeed possible and justifiable to investigate machine learning techniques when it comes to balancing games. Having this web of champion to stats and stat to win may seem convoluted; however, as shown here it gets around the bias present in multiplayer games.

5. Appendix

Table 2: Table caption

Feature Name:	Kills	Deaths	Assists	Gold Earned
Count	23752	23752	23752	23752
Mean	5.011	4.965	7.434	10223.871
STD	3.977	2.859	5.375	3479.974
MIN	0.00	0.00	0.00	666.00
MAX	29.0	20.00	40.00	25133.00

Table 3: Basic Data Analysis of Numerical Features

Feature Name:	Damage to Champions	Damage to Turrets	Damage Received	Vision Score
Count	23752	23752	23752	23752
Mean	15166.688	2685.702	20085.445	27.511
STD	9393.446	2965.702	10096.737	19.051
MIN	0.00	0.00	0.00	0.00
MAX	91579.00	52378.00	122378.00	153.00

Table 4: Data Type for Each Feature

Feature Name:	Data Types
Unnamed	int64
Champ	object
Team	int64
Gold Earned	int64
Kills	int64
Deaths	int64
Assists	int64
Win	bool
Total Damage Dealt to Champions	int64
Damage Dealt to Turrets	int64
Vision Score	int64
Total Damage Taken	int64
Turret Kills	int64
Inhibitor Kills	int64
Total Minions Killed	int64
Champ Level	int64
Wards Placed	int64
First Blood Assist	bool
WR general	float64
WR champ	float64
Duration (in Seconds)	int64
Position	object
Role	object

Table 5: Cardinality of Each Feature

Feature Name:	Cardinality
Unnamed	23752
Champ	148
Team	2
Gold Earned	10602
Kills	29
Deaths	20
Assists	39
Win	2
Total Damage Dealt to Champions	15129
Damage Dealt to Turrets	7304
Vision Score	137
Total Damage Taken	15768
Turret Kills	10
Inhibitor Kills	5
Total Minions Killed	361
Champ Level	18
Wards Placed	83
First Blood Kill	2
First Blood Assist	1
WR general	79
WR champ	198
Duration (in Seconds)	1863
Position	5
Role	5

Table 6: Missing Percentage of Each Feature

Feature Name:	Missing Percentage
Unnamed	0.0000%
Champ	0.0000%
Team	0.0000%
Gold Earned	0.0000%
Kills	0.0000%
Deaths	0.0000%
Assists	0.0000%
Win	0.0000%
Total Damage Dealt to Champions	0.0000%
Damage Dealt to Turrets	0.0000%
Vision Score	0.0000%
Total Damage Taken	0.0000%
Turret Kills	0.0000%
Inhibitor Kills	0.0000%
Total Minions Killed	0.0000%
Champ Level	0.0000%
Wards Placed	0.0000%
First Blood Kill	0.0000%
First Blood Assist	0.0000%
WR general	0.0000%
WR champ	0.0000%
Duration (in Seconds)	0.0000%
Position	0.0000%
Role	0.0000%

Table 7: Figure 13: Machine Learning Algorithm Supplementary Information

Machine Learning Algorithm	Citation 1	Citation 2
Random Forest	([4])	([5])
Naive Bayes Classifier	([6])	—
K-Nearest Neighbors	([7])	([8])
Support Vector Machine	([9])	([10])
Adaboost	(Schapire, 2013)	([11])

References

[1] G. Gao, A. Min, and P. C. Shih, "Gendered design bias: gender differences of in-game character choice and playing style in league of legends," in *Proceedings of the 29th Australian Conference on Computer-Human Interaction*, 2017, pp. 307–317.

[2] J. Wolf, "League 101: A league of legends beginner's guide," 2020.

[3] A. Macabasco, "Absolute beginner's guide to league of legends," *Mobalytics*, 2021.

[4] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, no. 2, pp. 197–227, 2016.

[5] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How many trees in a random forest?" in *International workshop on machine learning and data mining in pattern recognition*. Springer, 2012, pp. 154–168.

[6] S. Mukherjee and N. Sharma, "Intrusion detection using naive bayes classifier with feature reduction," *Procedia Technology*, vol. 4, pp. 119–128, 2012.

[7] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.

[8] H. Samet, "K-nearest neighbor finding using maxnearestdist," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 243–252, 2007.

[9] W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[10] L. Zhang, W. Zhou, and L. Jiao, "Wavelet support vector machine," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 34, no. 1, pp. 34–39, 2004.

[11] X. Li, L. Wang, and E. Sung, "Adaboost with svm-based component classifiers," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 5, pp. 785–795, 2008.