Predicting League of Legends Wins using 10 Minutes of Gameplay

Author: Nicholas Evans

Email: nbe20420@uga.edu

Dataset:
https://www.kaggle.com/bobbyscience/league-of-legends-diamond-ranked-games-10-min

Abstract

The purpose of this project is to use the data taken after 10 minutes of solo ranked gameplay and predict a win. There are 40 attributes recorded for each game: gameID, a blue team win/loss, and the other 38 are split between each team for different goals achieved. These goals include kills, deaths, towers destroyed, etc. This dataset has approximately 10,000 games saved. After doing proper preprocessing and experimenting with this data, three models were used for prediction: Logistic Regression, K-neighbors, and Support Vector Classification (SVC). All models did exceptionally well but two proved about 5% more accurate. After running the models several times, the following are average scores generated: Log-Reg = 0.73, KN = 0.68, SVC = 0.73.

1. Introduction

Competitive gaming is becoming more and more popular. League of Legends has, on average, 50 million people playing daily. Every person playing has a single goal in mind: winning. There are a lot of factors that go into a single game. You must select a character and normally claim a role such as top, middle, bottom, or jungle. The role indicates which lane you plan on manning for the game and can dictate which character you pick. Now, suppose your team is well balanced and you start your game. An average game takes 30 minutes to complete, which means your team either wins or loses. There are many different strategies to winning a game, but in this prediction model, we are just taking the 38 factors into consideration. 19 factors for each team, blue and red. The following figure shows descriptions of the important factors.

Factor	**Description**
Wards	These are objects that can be placed anywhere on the map. There are different types of wards. For example, some reveal an enemy if they enter the area.
Elite Monsters	When these are killed by a team, they give huge bonuses to the entire team. (Gold, exp, stats)
Dragons	A type of elite monster that gives a bonus to the team. The 5th dragon killed is called the Elder Dragon and offers huge bonuses.
Heralds	A type of elite monster that gives stats bonuses and helps to push a lane by destroying structures.
Towers	Structures a team must destroy in order to reach the Nexus (Home Tower).
Champion Level	Increases based on experience and unlocks attributes for the champion. Starts at 1 and maxes at 18.
Experience	You can gain experience from killing enemy champions, minions, and monsters.

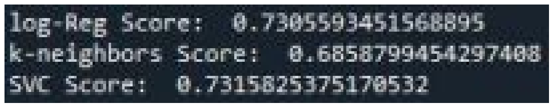
Minions	Spawn to help teams take down structures and kill enemy champions. They are on each lane and grant gold and exp on death.
Jungle Minions	These are on neither team but still give exp and gold when killed.
Gold	Gold is used to buy items to enhance the player’s champion.

During a ranked game, you want to use everything to your advantage to increase your chances of winning. The problem occurs when you need to figure out what exactly you need to do during a game to have the highest chance of winning. Imagine pausing a game after 10 minutes and recording everything both teams did. This dataset does this and includes whether the blue team wins or not. Using this information, we trained a model to predict a win for the blue team. There is a lot that goes into a single game, so there is a lot of preprocessing that must be done.

There are a handful of factors recorded that have different names yet are directly correlated. Blue kills and red deaths are both factors for each team but are the exact same value. Because of relationships like this, we can remove factors to reduce the amount of time it takes to train each model. There are also factors that have little to no correlation to the target value. Each game that is recorded has an Id and, since this has nothing to do with the actual game, it can be removed.

There are numerous factors to sort through and the ranges are very different. The number of dragons killed is minute compared to the total experience an entire team has gained. To pull these closer together and make it more manageable, the data needs to be scaled in some way. The proportions are kept the same, so if you wanted to throw each column onto a graph, you would be able to see the distribution together instead of separated by large numbers.

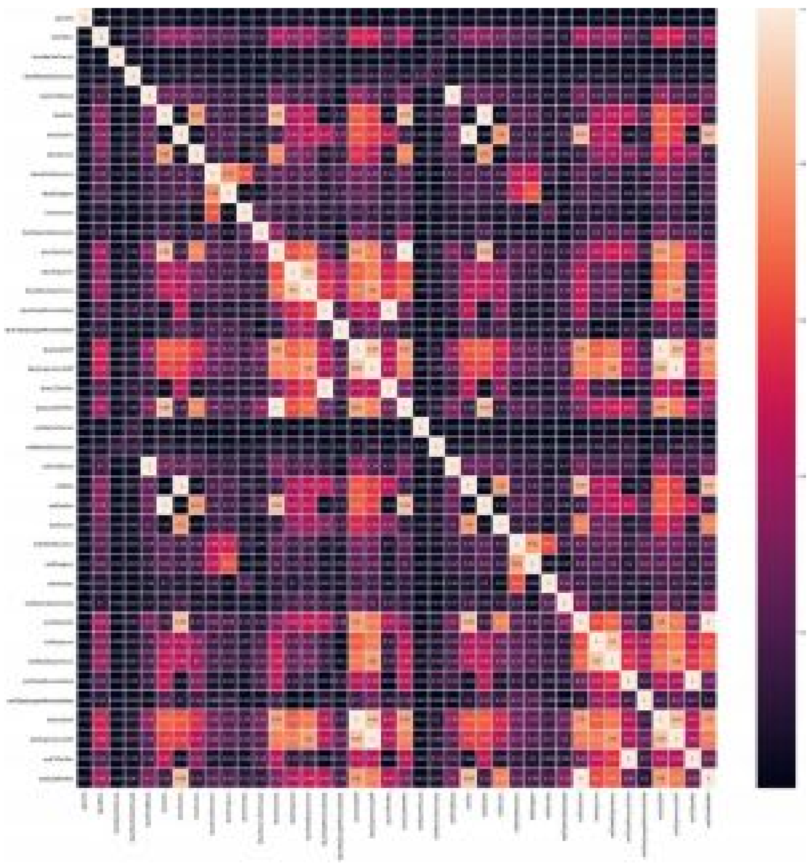
After sorting through the factors of the game and picking the best to keep, we train it and see how the models do. For this dataset, the models used are: Logistic Regression, K-neighbors, and Support Vector Classification (SVC). They each do very well on predicting a win just after 10 minutes of gameplay. Using the Score function for each model, here is how each model did: Log-Reg = 0.73, KN = 0.68, SVC = 0.73. This means the Logistic Regression Model was 73% accurate at predicting a win. After tuning different parts of the preprocessing and the actual training models, this was the best score achieved by all models. The following is an actual prediction score for each model:



```
log-Reg Score:  0.7305593451568895
k-neighbors Score:  0.6858799454297408
SVC Score:  0.7315825375170532
```

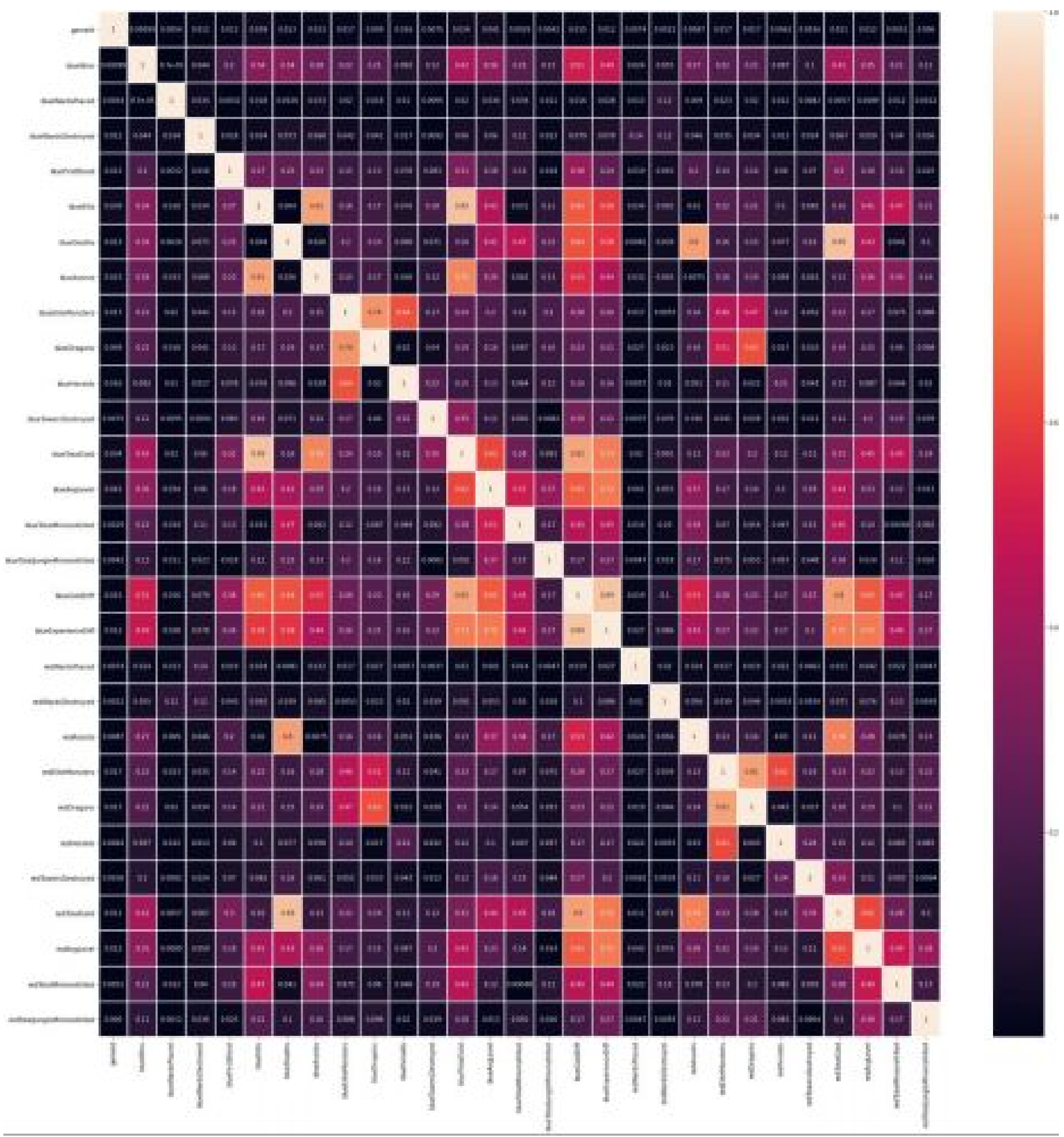
2. Data – Preprocessing

40 columns is a lot to deal with for training a model. The number had to decrease somehow so a few steps were taken to reduce the amount. First, a heat map was created to see the correlation between every attribute. See map below:

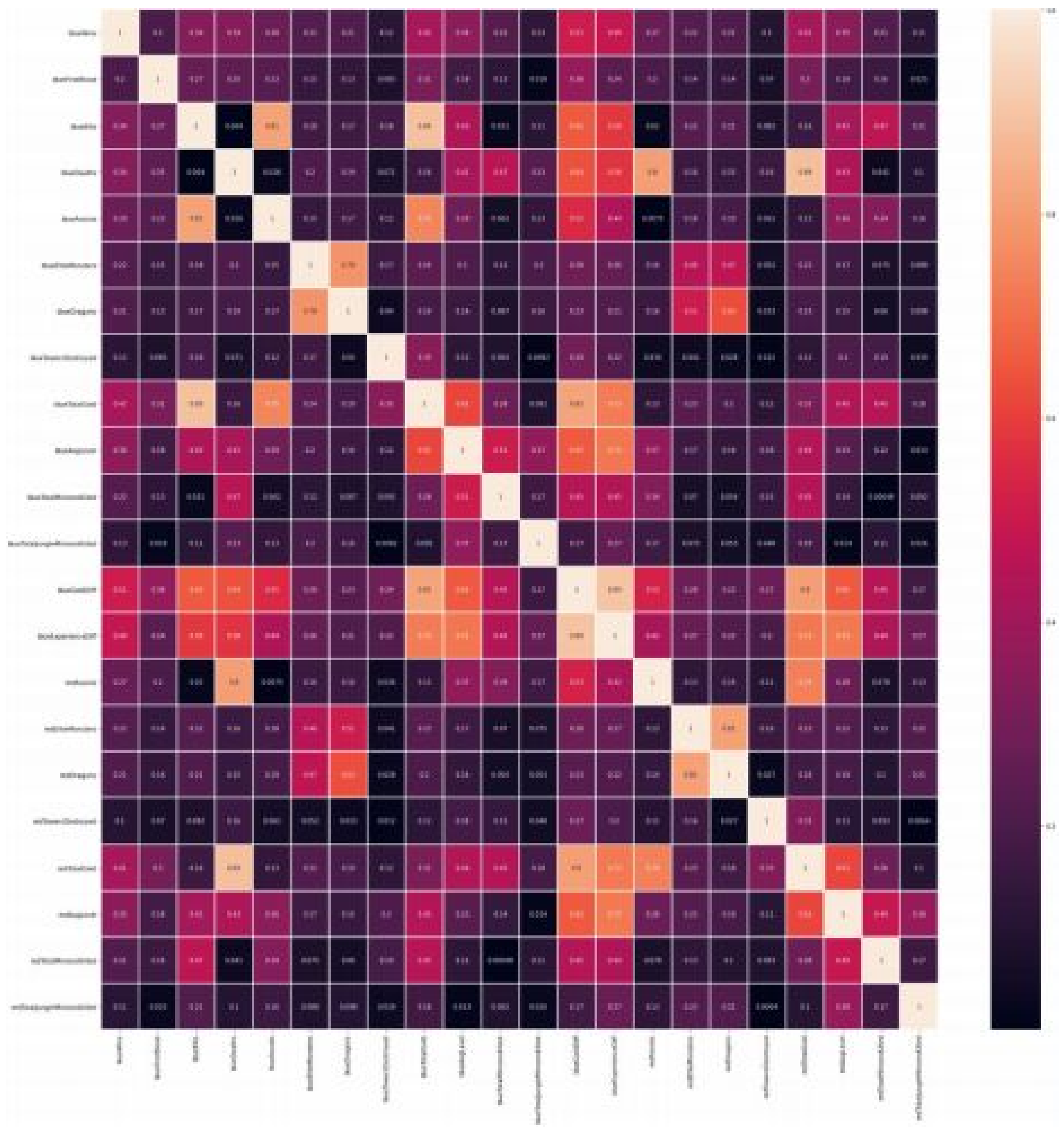


The columns and rows are small and obscure, but the colors are what is worth seeing. The lighter the color, the more correlated the row is to the column. That is why the diagonal is the

lightest color. The correlation is equal to 1. The first step in removing attributes is to find those with high correlation. Any correlation higher than 0.9 was removed from the data set. This is what remained:

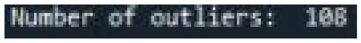


This removed 11 columns leaving us with 29. Next, I looked at the target value and its correlation to every attribute. If the attribute had a correlation less than 0.1, it was removed. The data was left with this:



You will notice the change from the first heat map to the second as most of the brighter colors were removed. The change from the second heat map to the third removed most of the darker colors. This gives us a nice middle portion of the data that will be most useful to train. I now have 22 columns of data to train and test.

There are 10,000 rows that have 22 columns now. The easiest way to make sure there is no data that is corrupted is to remove any outliers in any of the rows. There might be a value that was inputted wrong. If there is a number in a row that stretches too far from that column’s mean, then it is removed. This still gives a nice chunk of data to work with while ensuring that there are no ridiculous errors that will throw off our training model.



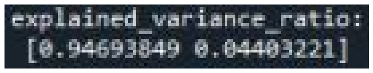
```
Number of outliers:  108
```

After finding our middle ground in both rows and columns, we separate the target from the attributes and scale the attributes. For this dataset, I used preprocessing.normalize(). What this does is shrink all the values for every column and places them between 0 and 1. The proportions are kept the same even though all values are shrunk.

3. Experiments

It is always important to use most of your dataset to train and use the smaller amount to test. First, I tried to use 60% of the data to train and it did okay. Then I brought it to 70% and it did worlds better. After playing with it a bit more, I decided it was best to leave it at 70%. One thing I noticed about my split function is that the shuffle was set to True. I also noticed every other time I ran my model, my accuracy would drop 20%-30%. The only moving factor I had every run was the shuffle. I knew I had a bad piece of data that was causing my testing data to tank the accuracy. That is why I decided to throw out the outliers of the data. Once I did that, it stopped dropping the accuracy.

Even though the data is down to 22 columns, there is still a way to reduce the columns. Principal component analysis is a way to keep all your data but reduce the number of columns the model must read. The only question is how many columns I need to keep to retain a majority of our information. We can calculate that using explained_variance_ratio. This returns a list of percentages starting highest to lowest. Of course, the sum of all these percentages equal 100%, meaning 100% of the data is kept. For this dataset, the top two highest ratio columns contain approximately 98% of the data shown below:



```
explained_variance_ratio:
 [0.94693849 0.04403221]
```

This means I can reduce 22 of the columns down to 2, which will drastically reduce the time to train. So ultimately, we started out with 38 attributes or columns and reduced them down to 2 columns. Now, these two columns are a mixture of those 22 attributes/columns. This is possible by calculating the eigenvectors and eigenvalues which is all done when you call pca.fit_transform().

4. Analysis

Throughout my experimenting, I watched the progress of my accuracy go up and down. I believe I ended up with the best representation of the data presented. I cut out the attributes not needed, and even cut out rows that might have possibly been wrong. Using PCA, I narrowed down the columns to two for a quick training and testing process. I believe the accuracy I reached is the peak of what the data can manage. There are many other reports on people who have predicted wins and they are all around the 73% I reached.

73% accuracy leads me to believe there is more information that is not presented in the dataset that is needed. There are many more factors than even 40 that lead to the product of a game. A teammate might have gotten kicked, someone might have had internet trouble during a match, or even someone got cocky with a lead and died several times giving the enemy team an advantage. 73% is good when you weigh all the factors you may be missing that may be impossible to record.

5. Conclusion

A large portion of training this data was cleaning it. Once you obtained the best of the data to train, it was simple to experiment with and find the best model. 73% accuracy is a good start to finding the best way to go about winning. There are more attributes that could have been saved and maybe there is a data set that would save all this information at the very end of the game when one team wins or loses.

If I had to do it again, I would try a couple of different things. I would try eliminating columns based on my knowledge of the game instead of based on the correlation alone. For example, I know it is not essential to place wards to win a game. However, killing the Elder Dragon gives you a big boost and helps you a great deal in trying to win a game. I also might try different models instead of the ones I used to see if any others would do better. There are always different directions to go.