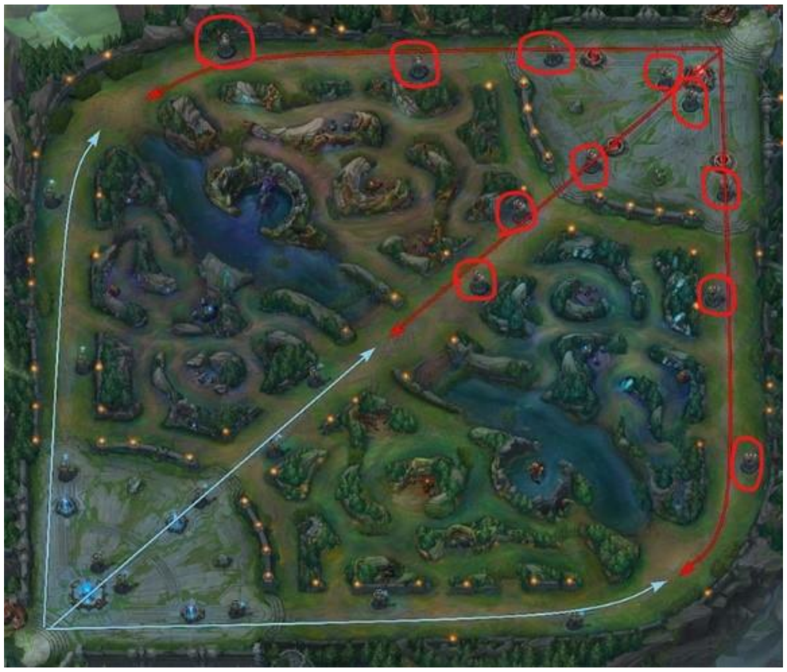
James Colton Behannon
October 19, 2020
Report Version 250

Introduction
League of Legends is a multiplayer online battle arena (MOBA) that has become one of the most popular games of all time. It boasts huge revenue (around $1.5 billion annually) and the most widely viewed esport's championships of any game.

How it's Played
The objective of League of Legends, LoL for short, is to destroy the enemy team's base. Both teams spawn in their respective corner of the map and fight their way through turrets, enemy players, and waves of minions (small NPCs) in order to reach the other teams base and claim victory. All players begin at level 1, meaning they are fairly weak, and continue to level up from dispatching enemy minions or players. Games last 35 minutes on average but the length can vary.

The Map
The entirety of League of Legends is played on "Summoner's Rift". The blue team starts in the bottom left while the red team begins in the top right. Both teams have five players and there is no inherent advantage to being on either team. Both teams push through the three "lanes" visible to destroy the turrets and minions in their way until finally reaching the enemy base. The base can be reached by progressing through any of the three lanes, so it does not matter if only one or all of them are successful at reaching the enemy team's base. The red team's turrets are circled on the map below.



The Shop
The point within the corner of each team's base is "the shop". This is where players start out at the beginning of the game as well as upon each respawn when they are killed by an enemy player, turret, or

minion. Here is where they can spend gold acquired from destroying turrets and defeating enemy players or minions. The gold spent here purchases items for each character that makes them more powerful and more likely to succeed in their ultimate goal.

The Data Set

This data set contains the stats of the first 10 minutes of approximately 10k ranked games from a high level of play (DIAMOND I to MASTER). Players are of roughly the same skill.

Each game is unique. The gameId uniquely identifies each game and could potentially be used to fetch more attributes from the Riot API.

There are 19 features per team (38 in total) collected after 10min in-game.

The numeric features count the kills, deaths, assists, total gold, total experience, average level, wards placed, wards destroyed, towers(turrets) destroyed, total minions killed, total jungle minions killed, the gold difference, the experience difference, the CS (amount of minions killed) per minute, gold per minute, # of dragons taken, # of heralds taken, and # of elite monsters taken, for each team.

The only categorical features provides information on which team got first blood.

The column blueWins is the target value. A value of 1 means the blue team won while 0 means the blue team lost. The split between blueWins and blue losses is roughly 50%.

Glossary

- Warding totem: An item that a player can put on the map to reveal the nearby area. Very useful for map/objectives control.
- Minions: NPC that belong to both teams. They give gold when killed by players.
- Jungle minions: NPC that belong to NO TEAM. They give gold and buffs when killed by players.
- Elite monsters: Monsters with high hp/damage that give a massive bonus (gold/XP/stats) when killed by a team.
- Dragons: Elite monster which gives team bonus when killed. The 4th dragon killed by a team gives a massive stats bonus. The 5th dragon (Elder Dragon) offers a huge advantage to the team.
- Herald: Elite monster which gives stats bonus when killed by the player. It helps to push a lane and destroys structures.
- Towers (turrets): Structures you have to destroy to reach the enemy Nexus. They give gold.
- Level: Champion level. Start at 1. Max is 18.
- Experience: Obtained from taking objectives and getting kills on enemy players and minions. Also slowly earned passively. This is what determines your level.
- CS: “Creep Score”. The number of minions killed.
- NPC: “Non-playable character”. Someone that is controlled by the game’s AI.
- Champion: Unique playable character that each player selects. There are over 140 options.

Source

Link to the dataset and its user provided description on Kaggle:
https://www.kaggle.com/bobbyscience/league-of-legends-diamond-ranked-games-10-min

Step 2:

1.1 The classification task at hand is to predict the winner of any ranked League of Legends match at a high level of play based on the statistics of the first 10 minutes of gameplay.

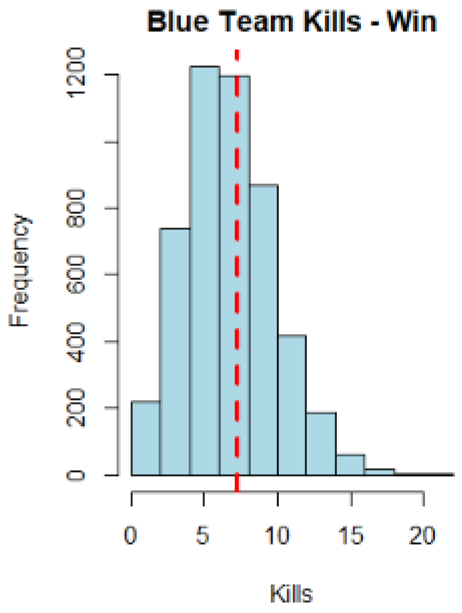
1.2 The data set contains the stats recorded within the first 10 minutes of approximately 10k ranked League of Legends games from a high level of play (DIAMOND I to MASTER). The players are of roughly the same skill level as recognized by their rank. There are 19 features per team (38 in total) as well as the identifier and target variables for a total of 40 columns

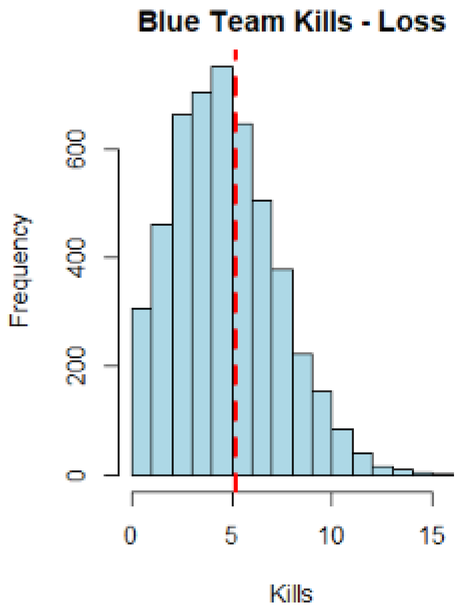
- Each game is unique and is identified by gameId.
- The target variable is blueWins. A value of 1 means the blue team won while 0 means the blue team lost. The split between the number of blue wins and blue losses is roughly 50%.

The following features are for the blue team however it should be noted that each one has a corresponding red team feature. If uncertain of a term, reference the glossary on pages 2-3.

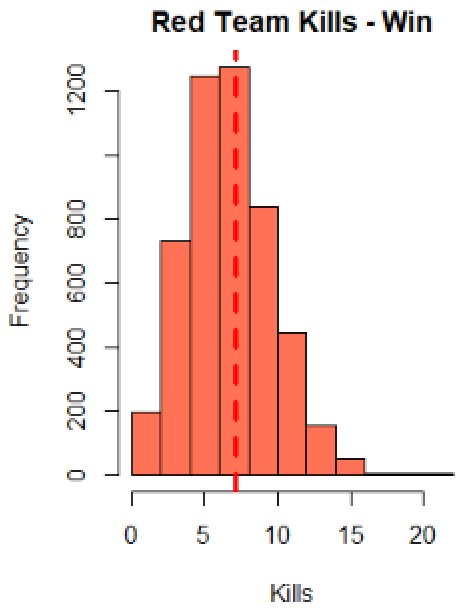
blueWardsPlaced: Number of warding totems placed by the blue team.
blueWardsDestroyed: Number of red team warding totems that blue team has destroyed.
blueFirstBlood: 1 if blue team got the first kill of the game, 0 if otherwise.
blueKills: Number of enemies that blue team has killed.
blueDeaths: Number of times that blue team has died.
blueAssists: Number of kill assists that blue team has.
blueEliteMonsters: Number of elite monsters (dragons and heralds) that blue team has killed.
blueDragons: Number of dragons that blue team has killed.
blueHeralds: Number of heralds that blue team has killed.
blueTowersDestroyed: Number of towers destroyed by the blue team.
blueTotalGold: Total gold acquired by the blue team.
blueAvgLevel: Average level of champions on the blue team.
blueTotalExperience: Total of experience points acquired by the blue team.
blueTotalMinionsKilled: Total minions killed by the blue team (CS).
blueTotalJungleMinionsKilled: Total jungle minions killed by the blue team.
blueGoldDiff: The difference in total gold compared to the red team.
blueExperienceDiff: The difference in total experience points compared to the red team.
blueCSPerMin: The rate of minions killed per minute of gameplay by the blue team.
blueGoldPerMin: The rate of gold acquired per minute of gameplay by the blue team.

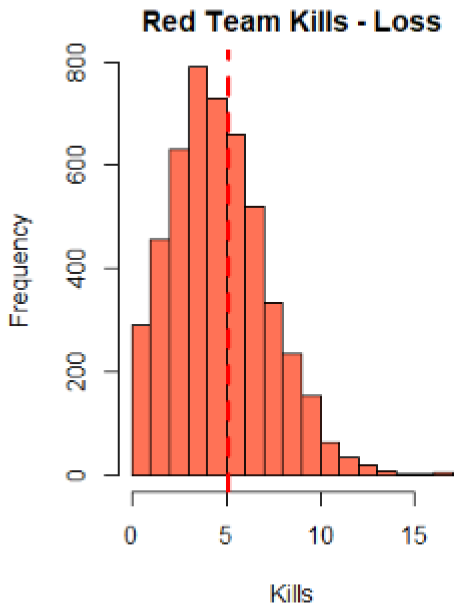
1.3 After using the t-test to compare the mean values of blueKills leading to a win vs. a loss, a p-value of <2.2e-16 is obtained. We can therefore reject the null hypothesis stating that they have equal means. This means blueKills will most likely have discriminatory power in differentiating between wins and losses.



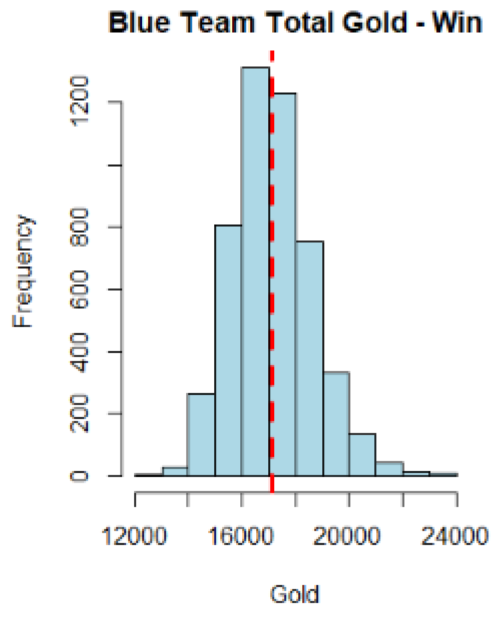


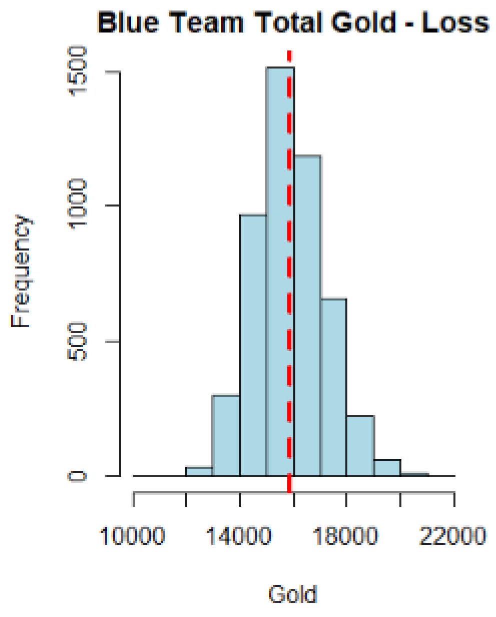
As expected, the variable redKills showed similar results as the teams are arbitrarily chosen. This theme will continue throughout the rest of the research. After using the t-test to compare the mean values of redKills leading to a win vs. a loss, a p-value of <2.2e-16 is obtained. We can therefore reject the null hypothesis stating that they have equal means. This means redKills will most likely have discriminatory power in differentiating between wins and losses.



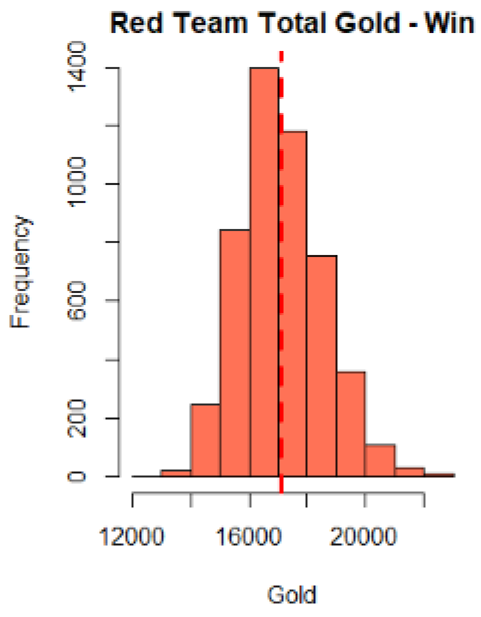


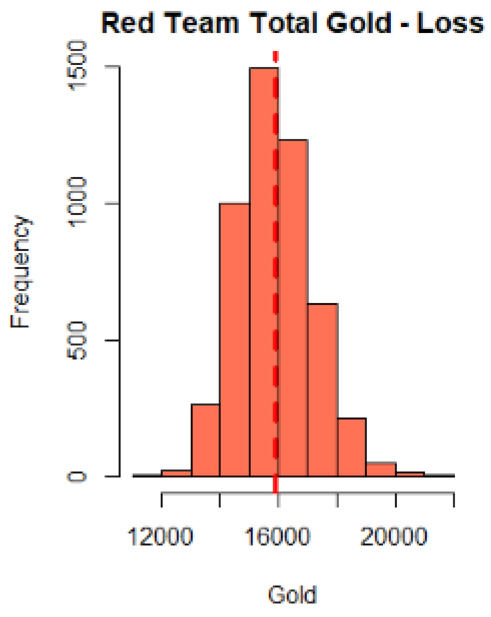
The next featured compared is that of total gold by team in matches that resulted in a win vs a loss. After using the t-test to compare the mean values of blueTotalGold leading to a win vs. a loss, a p-value of <2.2e-16 is obtained. We can therefore reject the null hypothesis stating that they have equal means. This means blueTotalGold will most likely have discriminatory power in differentiating between wins and losses.



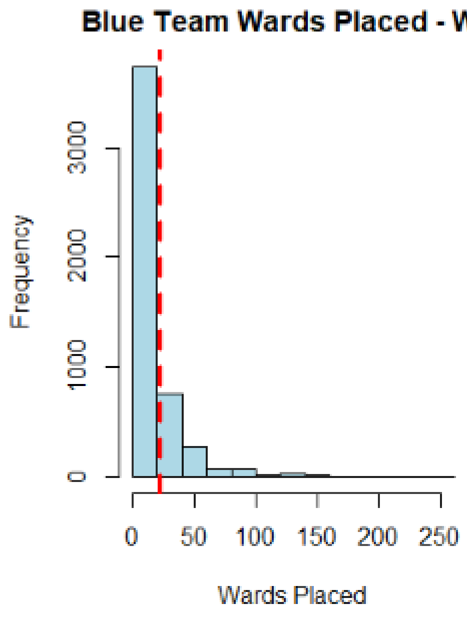


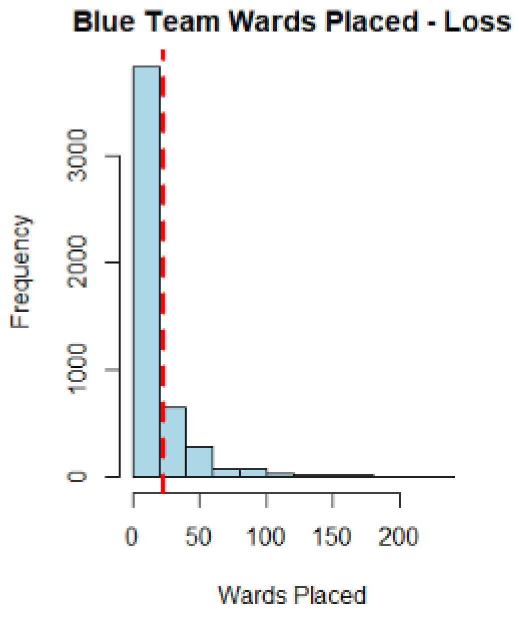
After using the t-test to compare the mean values of redTotalGold leading to a win vs. a loss, a p-value of <2.2e-16 is obtained. We can therefore reject the null hypothesis stating that they have equal means. This means redTotalGold will most likely have discriminatory power in differentiating between wins and losses.



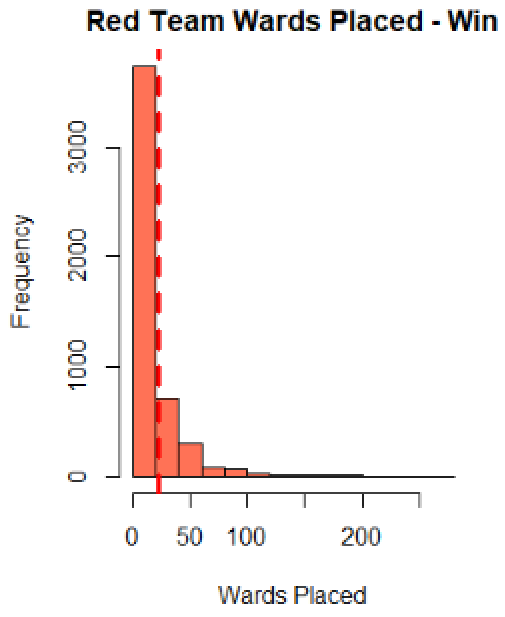


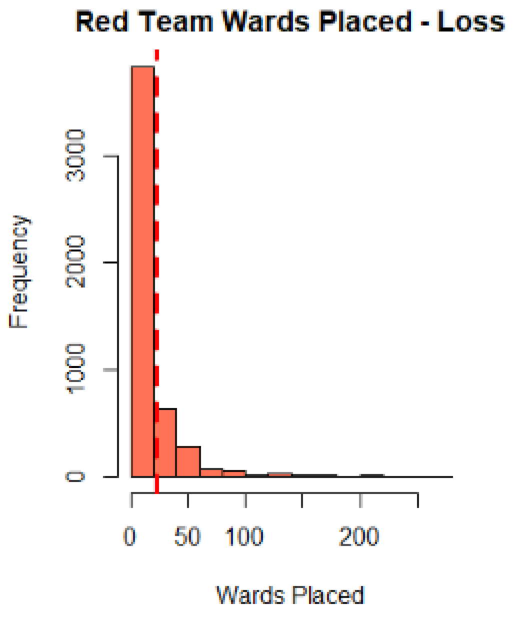
The next variable compared is the number of wards placed in matches that resulted in a win vs a loss.





After using the t-test to compare the mean values of blueWardsPlaced leading to a win vs. a loss, a p-value of 0.9931 is obtained. We therefore cannot reject the null hypothesis, that they have equal means, meaning that the number of wards placed in a win vs a loss do indeed have equal means. This means blueWardsPlaced will most likely have no discriminatory power in differentiating between wins and losses.





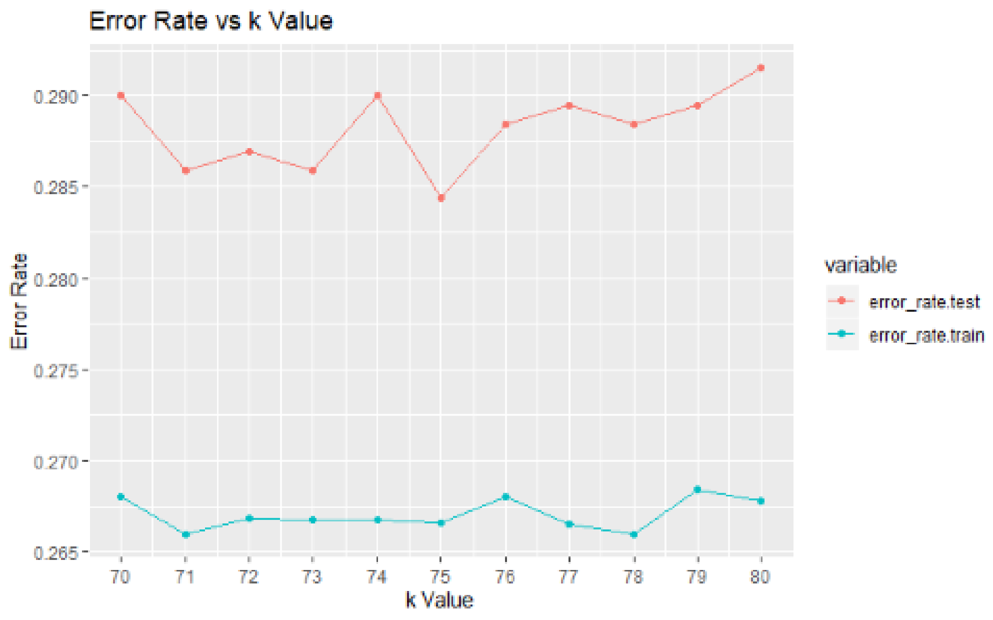
After using the t-test to compare the mean values of redWardsPlaced leading to a win vs. a loss, a p-value of 0.0186 is obtained. We can therefore reject the null hypothesis, that they have equal means, meaning that the number of red wards placed in a win vs a loss could have discriminatory power in predicting the outcome of match. However, the test for blue wards placed just showed that the means were the same for wards placed that lead to a win vs a victory. This could be indicative of red team benefitting more from wards placed within the first 10 minutes of the game. This is unlikely as the sides are arbitrary but there are minor differences on each side of the map so this cannot be completely brushed away.

Step 3:

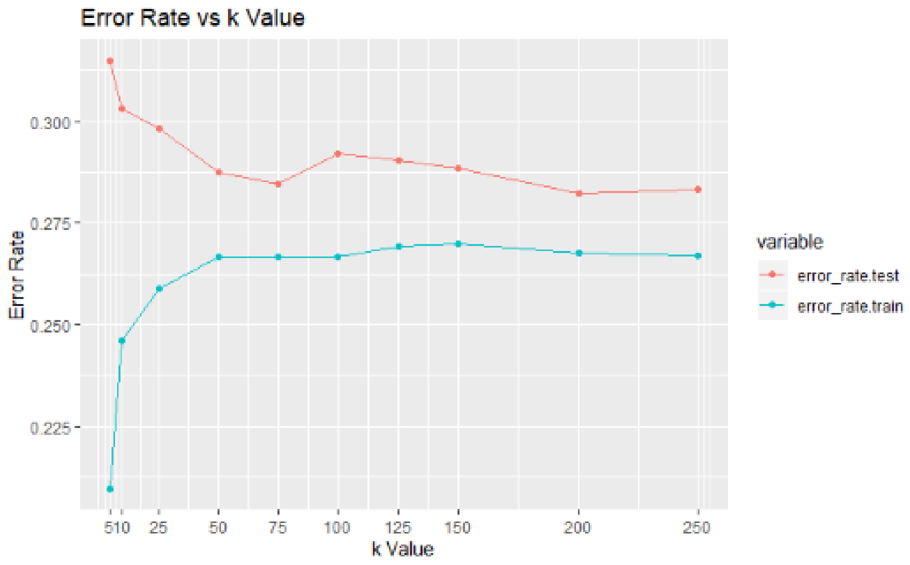
After standardizing the data and splitting 80% into a training set and 20% into the test set, the roughly 50/50 split between blue and red team wins is maintained within both the training and test set. Moving forward, the KNN algorithm will be implemented to predict the winner of the matches based on the data collected from the first 10 minutes of gameplay. But first, what is the KNN algorithm?

KNN is a supervised machine learning algorithm used to classify data under the assumption that similar occurrences exist in close proximity. Each new data point within the test set is compared to the "K" nearest neighbors within the training set with K being defined beforehand. The distance between points can be calculated using various distance functions. Euclidean distance will be used in this case.

To find the best K value, a preliminary set of K values were chosen and then plotted along with their error rates in order to find which area was indicative of the best K. The lowest point on the graph, and therefore lowest error rate, was the K of 75 with no evidence of overfitting.



To make sure there is not a better candidate around 75, another round of tests was run with K values of 70 to 80. The K of 75 yielded the best results from this group as well.



The KNN algorithm with K = 75 has an accuracy of 71.56% on the test set, with a 95% confidence interval of +/- 1.99, and 73.31% on the training set, with a 95% confidence interval of +/- 0.98.

The algorithm correctly predicts a blue loss 73.7% of the time and accurately predicts a blue win 69.9% of the time in the test set. The algorithm also incorrectly predicts a win 26.3% of the time and incorrectly predicts a loss 30.1% of the time in the test set.

	True: blueWins=0 <fctr>	True: blueWins=1 <fctr>
Pred: blueWins=0	73.7%	30.1%
Pred: blueWins=1	26.3%	69.9%

The algorithm correctly predicts a blue loss 73.2% of the time and accurately predicts a blue win 73.4% of the time in the training set. The algorithm also incorrectly predicts a win 26.8% of the time and incorrectly predicts a loss 26.6% of the time in the training set.

	True: blueWins=0 <fctr>	True: blueWins=1 <fctr>
Pred: blueWins=0	73.2%	26.6%
Pred: blueWins=1	26.8%	73.4%

Step 4:

In order to try and improve the performance of the KNN algorithm the following techniques will be tried:

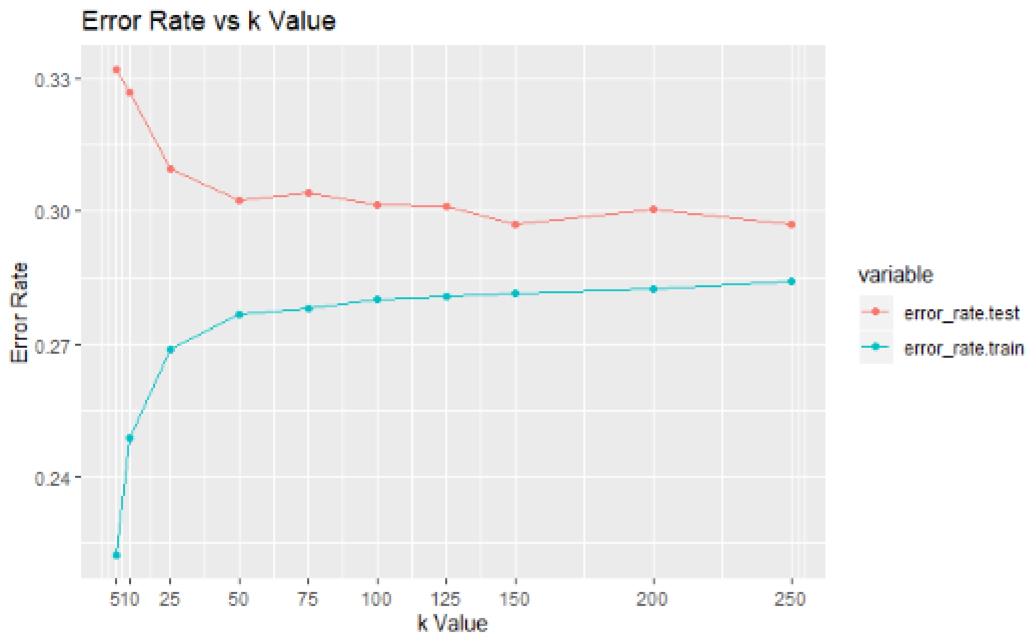
1) Remove a feature from each pair with a correlation over 0.75.
2) Remove a feature from each pair with a perfect correlation of 1.00.
3) Weight the features based on related groups.

1. After checking for multicollinearity, 41 rows came back with a correlation over 0.75. Here are the 10 pairs with the largest correlations.

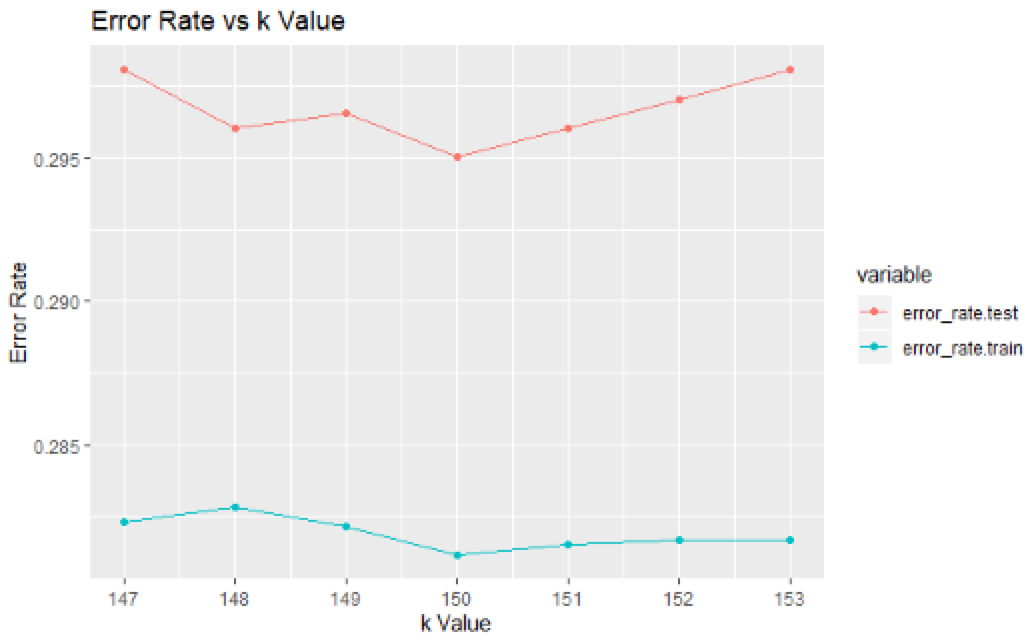
Var1 <fctr>	Var2 <fctr>	Freq <dbl>
blueTotalMinionsKilled	blueCSPerMin	1.00
blueTotalGold	blueGoldPerMin	1.00
blueFirstBlood	redFirstBlood	-1.00
blueDeaths	redKills	1.00
blueKills	redDeaths	1.00
blueGoldDiff	redGoldDiff	-1.00
blueExperienceDiff	redExperienceDiff	-1.00
redTotalMinionsKilled	redCSPerMin	1.00
redTotalGold	redGoldPerMin	1.00
blueAvgLevel	blueTotalExperience	0.90

These values having such high correlations makes sense. Those that are perfectly correlated are either a combination of a total and a rate of that total (ex: blueTotalMinionsKilled and blueCSPerMin) or provide the exact same information but in different terms (ex: blueKills and redDeaths). The high correlations (over 0.75) are values that are largely explained by another (ex: blueKills and blueTotalGold).

After removing the features that created a correlation coefficient over 0.75, we need to find a new best K value. The same method used earlier will be repeated.

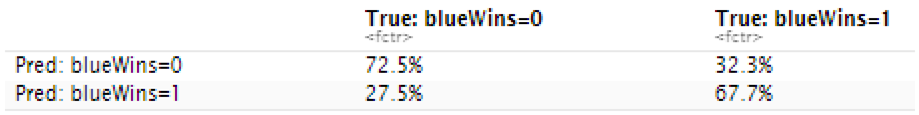


A K of 150 produces the lowest preliminary K value. To make sure this is the best K available, the plot is repeated for K values 147 to 153.



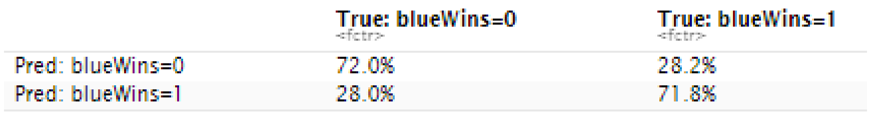
After looking deeper, a K value of 150 is still the best choice with no evidence of overfitting. Using KNN with the K of 150 produces an accuracy of 70.14% on the test set, with a 95% confidence interval of +/- 2.02, and 71.88% on the training set, with a 95% confidence interval of +/- 0.99. This is roughly a little less than a 1% decrease in performance.

The model accurately predicted 72.5% of blue losses and 67.7% of blue wins on the test set. It inaccurately predicted a win 27.5% of the time and a loss 32.3% of the time on the test set.



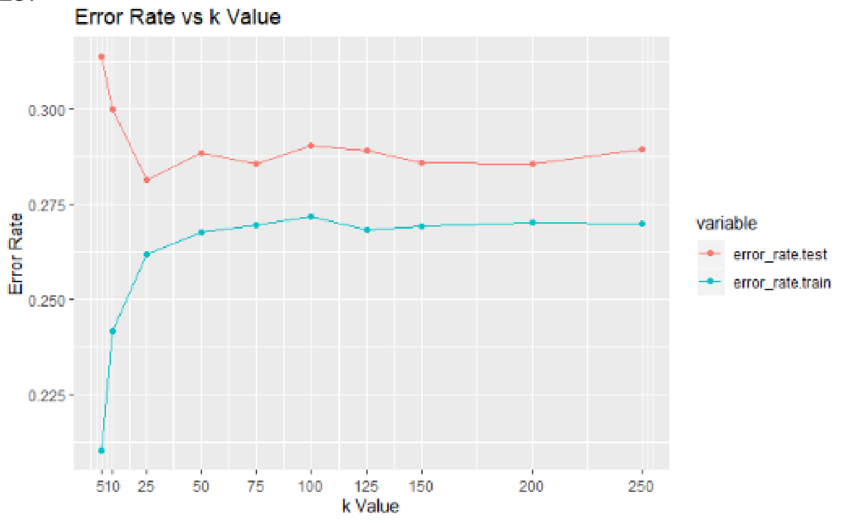
	True: blueWins=0 <fctr>	True: blueWins=1 <fctr>
Pred: blueWins=0	72.5%	32.3%
Pred: blueWins=1	27.5%	67.7%

The model accurately predicted 72.0% of blue losses and 71.8% of blue wins on the training set. It inaccurately predicted a win 28.0% of the time and a loss 28.2% of the time on the training set.

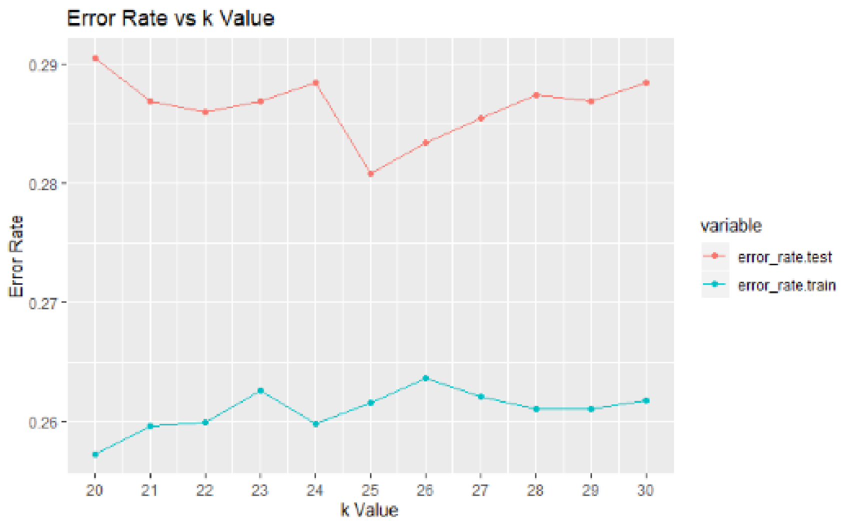


	True: blueWins=0 <fctr>	True: blueWins=1 <fctr>
Pred: blueWins=0	72.0%	28.2%
Pred: blueWins=1	28.0%	71.8%

2. The process used in 1. is repeated here except only the features that are perfectly correlated will be removed. After doing this the best K value is once again searched for. The first search reveals the best value to be 25.

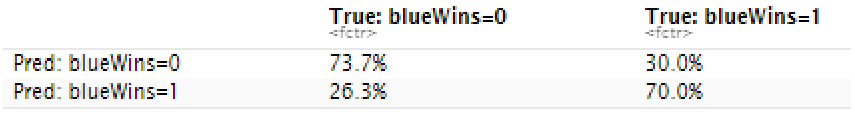


The K values of 20 to 30 are then run to take a closer look around this area. A K of 25 still produces the best results that will be the one selected.



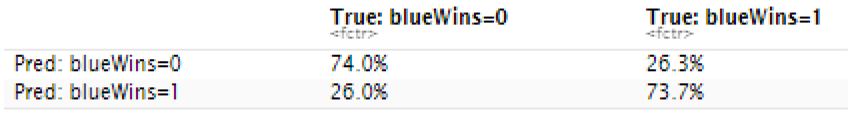
The KNN algorithm without features that were perfectly correlated with other features results in an accuracy of 71.86% on the test set, with a 95% confidence interval of +/- 1.98, and 73.83% on the training set, with a 95% confidence interval of +/- 0.97. This is a 0.30% increase in the previous best of 71.56% which used all of the features. Though there is an increase in accuracy it is extremely minor and possibly negligible.

The algorithm correctly predicts a blue loss 73.7% of the time and accurately predicts a blue win 70.0% of the time. The algorithm also incorrectly predicts a win 26.3% of the time and incorrectly predicts a loss 30.0% of the time.



	True: blueWins=0 <fctr>	True: blueWins=1 <fctr>
Pred: blueWins=0	73.7%	30.0%
Pred: blueWins=1	26.3%	70.0%

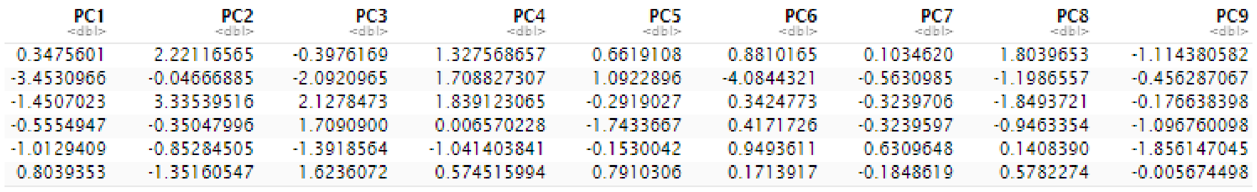
The algorithm correctly predicts 74.0% of blue losses and 73.7% of blue wins on the training set. It inaccurately predicted a win 26.0% of the time and a loss 26.3% of the time on the training set.



	True: blueWins=0 <fctr>	True: blueWins=1 <fctr>
Pred: blueWins=0	74.0%	26.3%
Pred: blueWins=1	26.0%	73.7%

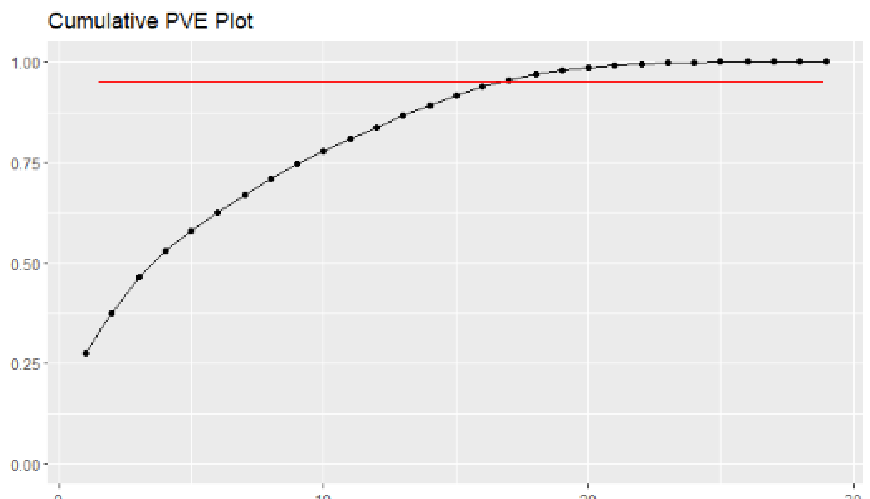
3. In a final attempt to improve the KNN algorithm performance, principal component analysis will be combined with KNN in order to try and obtain more accurate predictions (the features that had perfect correlation will not be used similar to the last attempt).

The first six values of the first nine principal components are below:

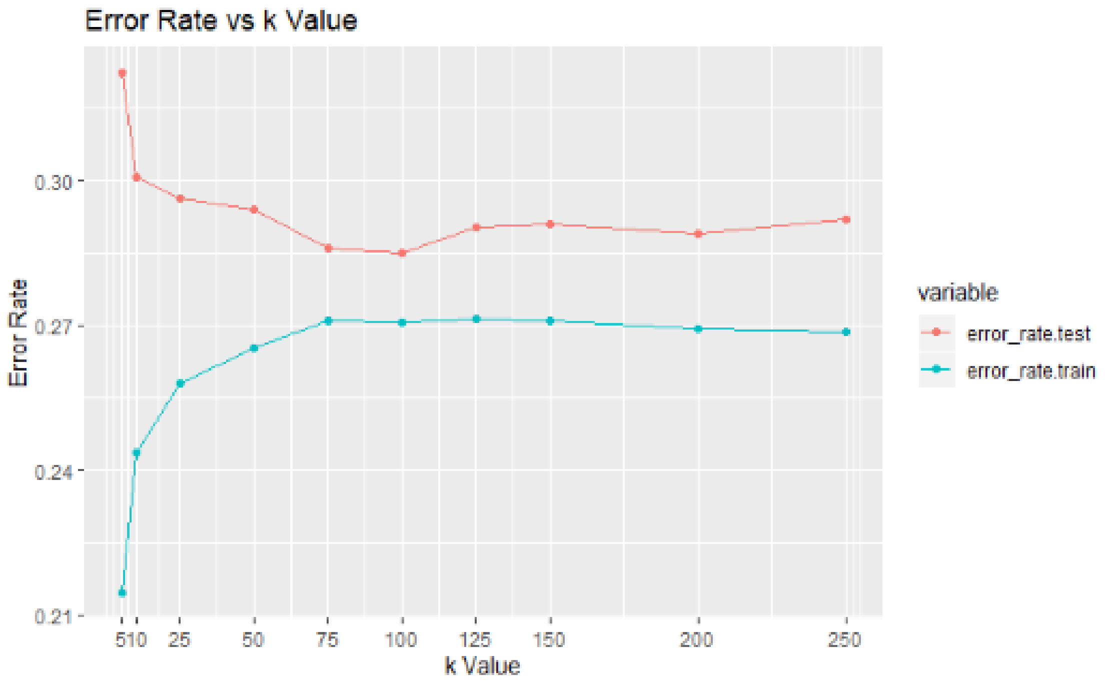


PC1 <dbl>	PC2 <dbl>	PC3 <dbl>	PC4 <dbl>	PC5 <dbl>	PC6 <dbl>	PC7 <dbl>	PC8 <dbl>	PC9 <dbl>
0.3475601	2.22116565	-0.3976169	1.327568657	0.6619108	0.8810165	0.1034620	1.8039653	-1.114380582
-3.4530966	-0.04666885	-2.0920965	1.708827307	1.0922896	-4.0844321	-0.5630985	-1.1986557	-0.456287067
-1.4507023	3.33539516	2.1278473	1.839123065	-0.2919027	0.3424773	-0.3239706	-1.8493721	-0.176638398
-0.5554947	-0.35047996	1.7090900	0.006570228	-1.7433667	0.4171726	-0.3239597	-0.9463354	-1.096760098
-1.0129409	-0.85284505	-1.3918564	-1.041403841	-0.1530042	0.9493611	0.6309648	0.1408390	-1.856147045
0.8039353	-1.35160547	1.6236072	0.574515994	0.7910306	0.1713917	-0.1848619	0.5782274	-0.005674498

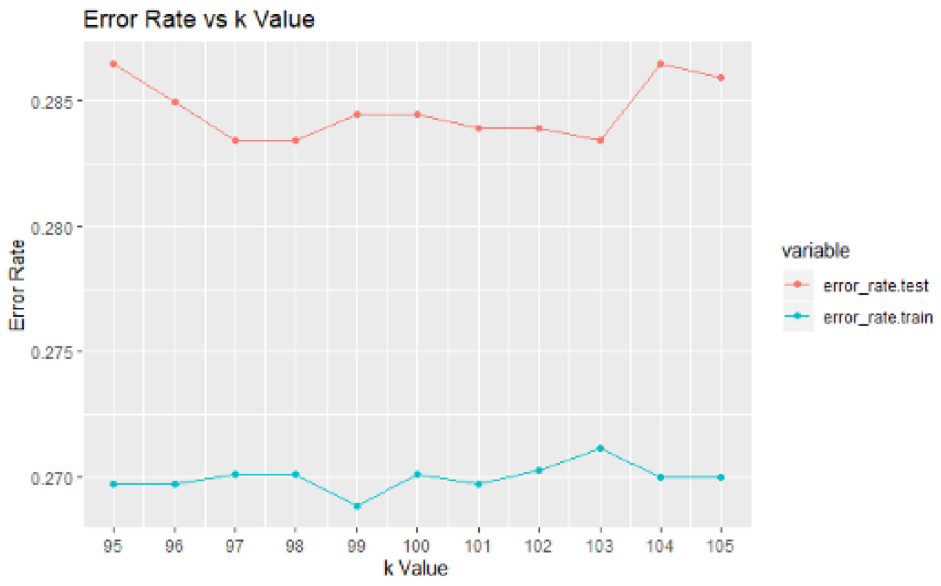
To get a better idea of the variance explained by each component a PVE plot can be used to find at what point the chosen cutoff of 95% variance explained is achieved. This occurs at PC17.



Now KNN can be run using the new features obtained from PCA. Using the previous best K of 25 an accuracy of 70.34% is achieved which is roughly 1.5% worse than the previous best. To try and improve these results a new best K can be found.

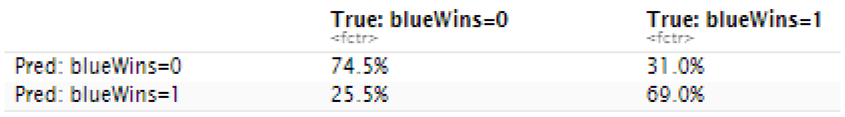


A K of 100 produces the lowest error so another plot can be used to make sure this is the best value.



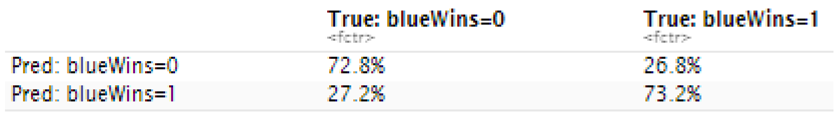
This graph shows that a K of 98 produces the best results. Running KNN with the K of 98 obtains an accuracy of 71.81% on the test set with a 95% confidence interval of +/- 1.98, which is essentially the same result as the previous best of 71.86% +/- 1.98, and 73.07% on the training set with a 95% confidence interval of +/- 0.98.

The algorithm correctly predicts a blue loss 74.5% of the time and accurately predicts a blue win 69.0% of the time in the test set. The algorithm also incorrectly predicts a win 25.5% of the time and incorrectly predicts a loss 31.0% of the time in the test set.



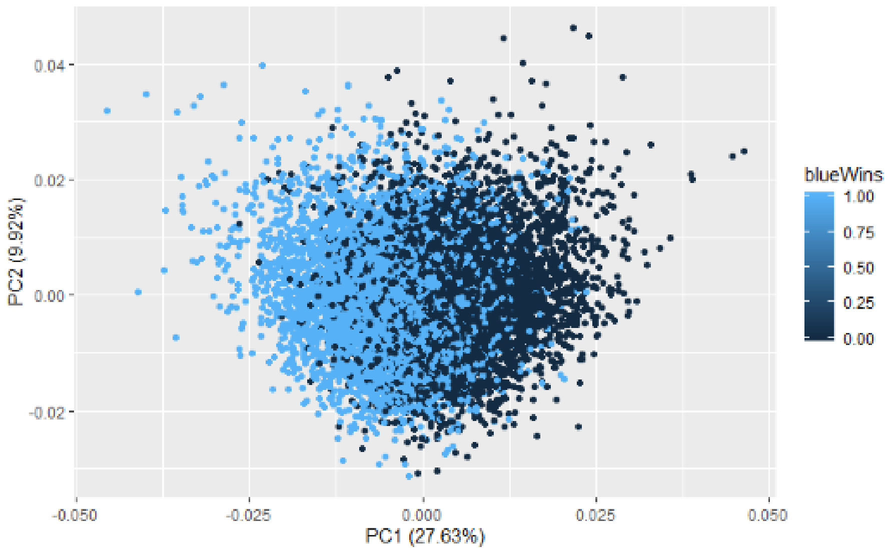
	True: blueWins=0 <fctr>	True: blueWins=1 <fctr>
Pred: blueWins=0	74.5%	31.0%
Pred: blueWins=1	25.5%	69.0%

The algorithm correctly predicts a blue loss 72.8% of the time and accurately predicts a blue win 73.2% of the time in the training set. The algorithm also incorrectly predicts a win 27.2% of the time and incorrectly predicts a loss 26.8% of the time in the training set.



	True: blueWins=0 <fctr>	True: blueWins=1 <fctr>
Pred: blueWins=0	72.8%	26.8%
Pred: blueWins=1	27.2%	73.2%

Upon plotting the two largest principal components it shows that there is not a clear distinction for a large portion of the data. This is most likely what is resulting in the less than phenomenal results. If a clearer distinction could be made between blueWins=1 and blueWins=0 then results would certainly improve.



```
Code:

games <- read.csv("high_diamond_ranked_10min.csv")

# Check for missing data
sum(is.na(games))

## [1] 0

# Compare the sizes of the two classes, blueWins = 0 and blueWins = 1
sum(games$blueWins)/length(games$blueWins)

## [1] 0.4990384

nrow(games)

## [1] 9879
```

There is no missing data and the classes are evenly balanced with blueWins=1 taking up 49.9% (4930) of the 9879 total games. This proportion makes sense as the team that is on blue side is chosen completely arbitrarily and games are matched so that each team is of relatively equal skill level.

Remove the ID and target variable (blueWins), standardize the rest of the data.

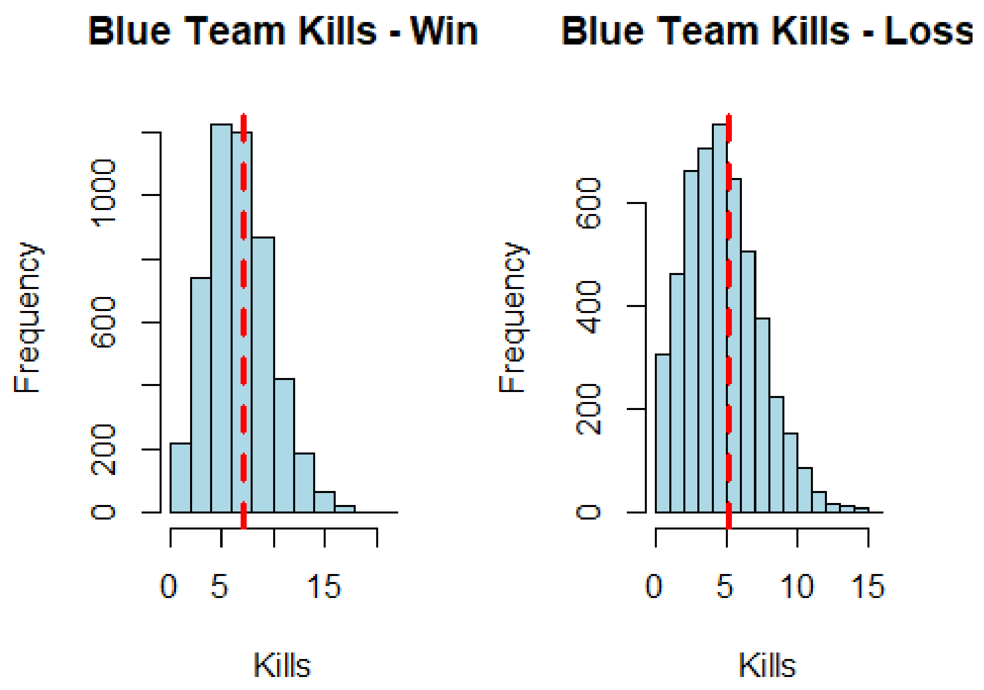
```
ID <- games$gameId
blueWinsVector <- games$blueWins

sgames <- as.data.frame(scale(games[, -c(1,2)]))
```

1.3 Evaluate the discriminating power of a few features : Methods (see HW1) visually compare histograms of feature F across different classes use t-test or KS-test to compare histograms of F between pairs of classes

Kills Comparisons

Comparison of kills in a win vs loss

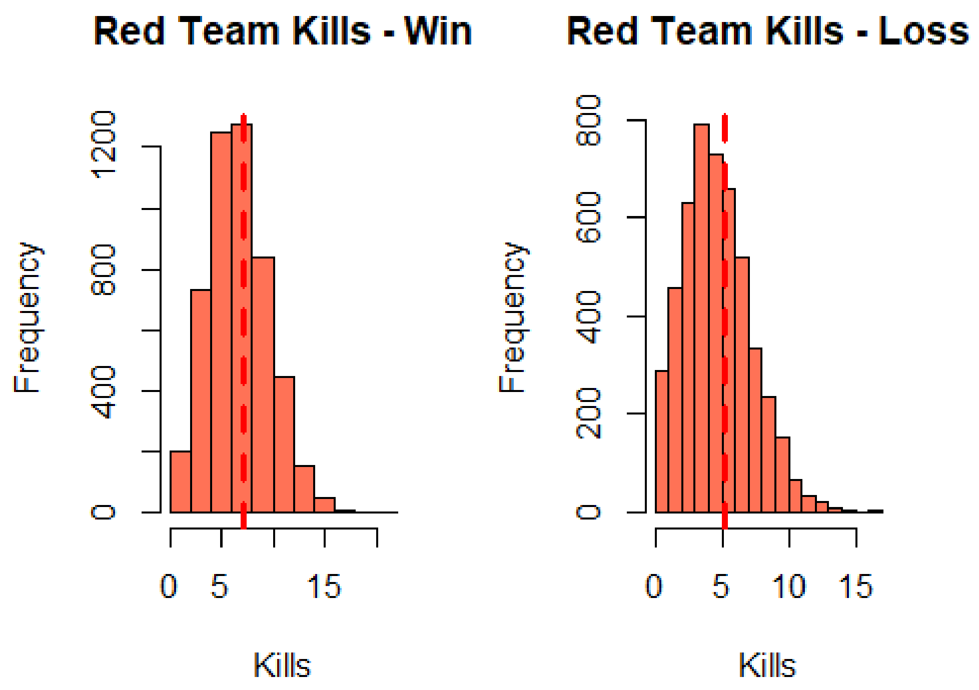
```
par(mfrow=c(1,2))
hist(games$blueKills[games$blueWins==1], col="lightblue", xlab = "Kills", mai
n = "Blue Team Kills - Win")
abline(v=mean(games$blueKills[games$blueWins==1]), col="red", lty=2, lwd=3)
hist(games$blueKills[games$blueWins==0], col="lightblue", xlab = "Kills", mai
n = "Blue Team Kills - Loss")
abline(v=mean(games$blueKills[games$blueWins==0]), col="red", lty=2, lwd=3)
```



```
t.test(games$blueKills[games$blueWins==1], games$blueKills[games$blueWins==0]
)

##
##  Welch Two Sample t-test
##
## data:  games$blueKills[games$blueWins == 1] and games$blueKills[games$blue
Wins == 0]
## t = 35.605, df = 9647.5, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.919646 2.143329
## sample estimates:
## mean of x mean of y
##  7.201623  5.170135
```

Comparison of red team kills in a win vs loss

```
par(mfrow=c(1,2))
hist(games$redKills[games$blueWins==0], col="coral1", xlab = "Kills", main =
"Red Team Kills - Win")
abline(v=mean(games$redKills[games$blueWins==0]), col="red", lty=2, lwd=3)
hist(games$redKills[games$blueWins==1], col="coral1", xlab = "Kills", main =
"Red Team Kills - Loss")
abline(v=mean(games$redKills[games$blueWins==1]), col="red", lty=2, lwd=3)
```



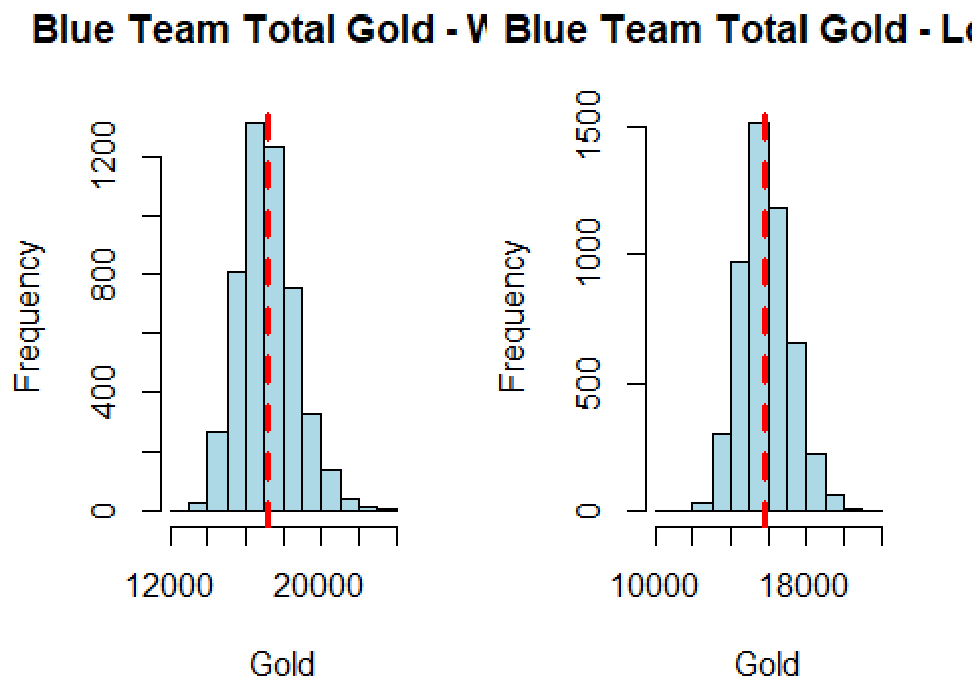
```
t.test(games$redKills[games$blueWins==0], games$redKills[games$blueWins==1])

##
##  Welch Two Sample t-test
##
## data:  games$redKills[games$blueWins == 0] and games$redKills[games$blueWi
ns == 1]
## t = 35.856, df = 9725.4, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.881938 2.099607
## sample estimates:
## mean of x mean of y
##  7.131138  5.140365
```

Gold Comparison

Comparison of blue team gold in a win vs a loss

```
par(mfrow=c(1,2))
hist(games$blueTotalGold[games$blueWins==1], col="lightblue", xlab = "Gold",
main = "Blue Team Total Gold - Win")
abline(v=mean(games$blueTotalGold[games$blueWins==1]), col="red", lty=2, lwd=
3)
hist(games$blueTotalGold[games$blueWins==0], col="lightblue", xlab = "Gold",
main = "Blue Team Total Gold - Loss")
```

```r
abline(v=mean(games$blueTotalGold[games$blueWins==0]), col="red", lty=2, lwd=
3)
```

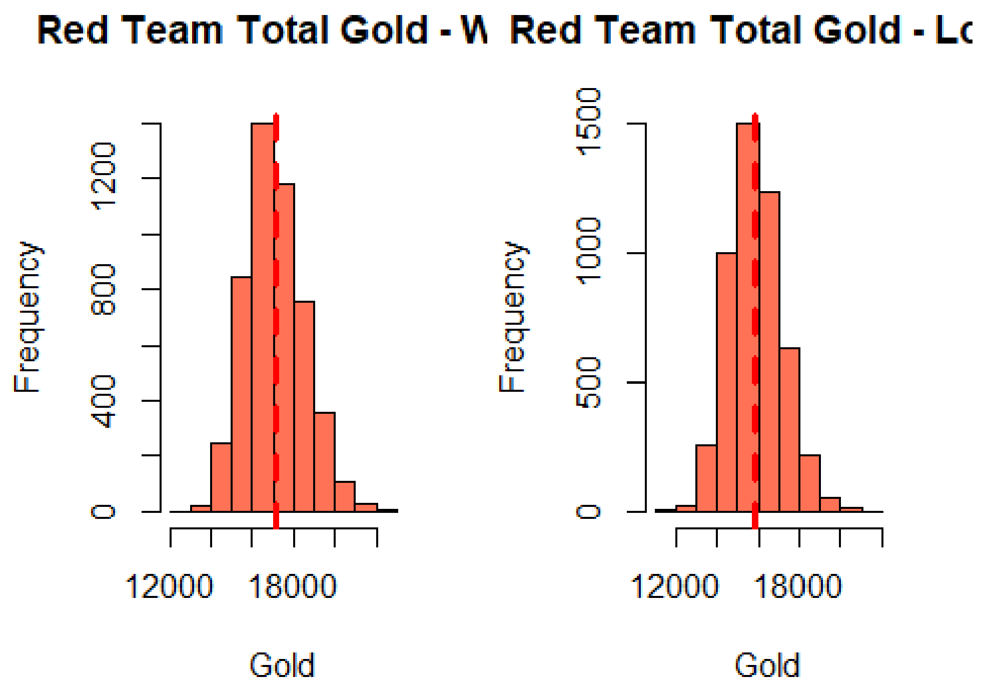


```r
t.test(games$blueTotalGold[games$blueWins==1], games$blueTotalGold[games$blue
Wins==0])
```

```
##
##  Welch Two Sample t-test
##
## data:  games$blueTotalGold[games$blueWins == 1] and games$blueTotalGold[ga
mes$blueWins == 0]
## t = 45.613, df = 9704.4, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1226.096 1336.210
## sample estimates:
## mean of x mean of y
##  17145.26  15864.11
```

Comparison of red team total gold in a win vs loss

```r
par(mfrow=c(1,2))
hist(games$redTotalGold[games$blueWins==0], col="coral1", xlab = "Gold", main
= "Red Team Total Gold - Win")
abline(v=mean(games$redTotalGold[games$blueWins==0]), col="red", lty=2, lwd=3
)
hist(games$redTotalGold[games$blueWins==1], col="coral1", xlab = "Gold", main
= "Red Team Total Gold - Loss")
abline(v=mean(games$redTotalGold[games$blueWins==1]), col="red", lty=2, lwd=3
)
```

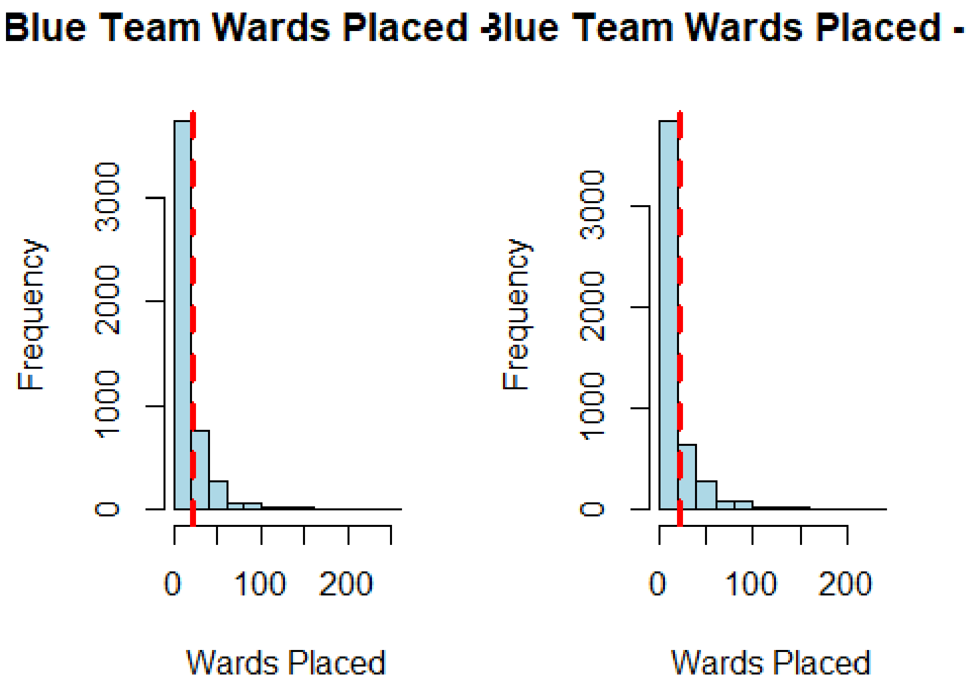


```r
t.test(games$redTotalGold[games$blueWins==0], games$redTotalGold[games$blueWi
ns==1])

##
##  Welch Two Sample t-test
##
## data:  games$redTotalGold[games$blueWins == 0] and games$redTotalGold[game
s$blueWins == 1]
## t = 44.866, df = 9785.8, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1173.041 1280.223
## sample estimates:
## mean of x mean of y
##  17101.18  15874.55
```

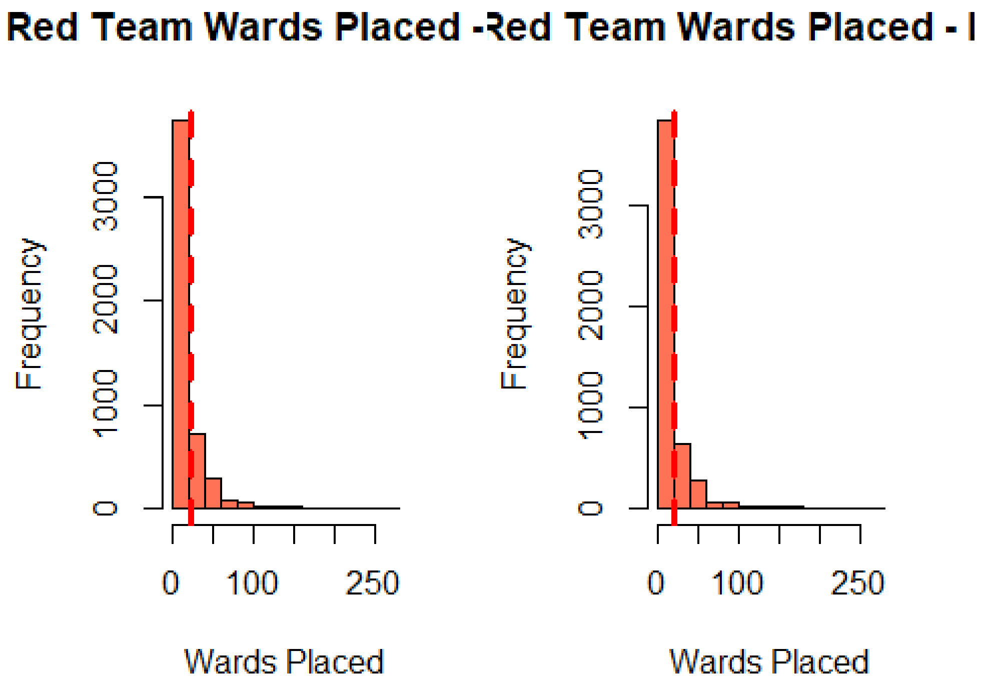
Wards Destroyed Comparison

```r
par(mfrow=c(1,2))
hist(games$blueWardsPlaced[games$blueWins==1], col="lightblue", xlab = "Wards
Placed", main = "Blue Team Wards Placed - Win")
abline(v=mean(games$blueWardsPlaced[games$blueWins==1]), col="red", lty=2, lw
d=3)
```

```r
hist(games$blueWardsPlaced[games$blueWins==0], col="lightblue", xlab = "Wards
Placed", main = "Blue Team Wards Placed - Loss")
abline(v=mean(games$blueWardsPlaced[games$blueWins==0]), col="red", lty=2, lw
d=3)
```



```r
t.test(games$blueWardsPlaced[games$blueWins==1], games$blueWardsPlaced[games$
blueWins==0])

##
##  Welch Two Sample t-test
##
## data:  games$blueWardsPlaced[games$blueWins == 1] and games$blueWardsPlace
d[games$blueWins == 0]
## t = 0.0086422, df = 9859.3, p-value = 0.9931
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.7075791  0.7138460
## sample estimates:
## mean of x mean of y
##  22.28986  22.28672

par(mfrow=c(1,2))
hist(games$redWardsPlaced[games$blueWins==0], col="coral1", xlab = "Wards Pla
ced", main = "Red Team Wards Placed - Win")
abline(v=mean(games$redWardsPlaced[games$blueWins==0]), col="red", lty=2, lwd
=3)
hist(games$redWardsPlaced[games$blueWins==1], col="coral1", xlab = "Wards Pla
ced", main = "Red Team Wards Placed - Loss")
abline(v=mean(games$redWardsPlaced[games$blueWins==1]), col="red", lty=2, lwd
=3)
```



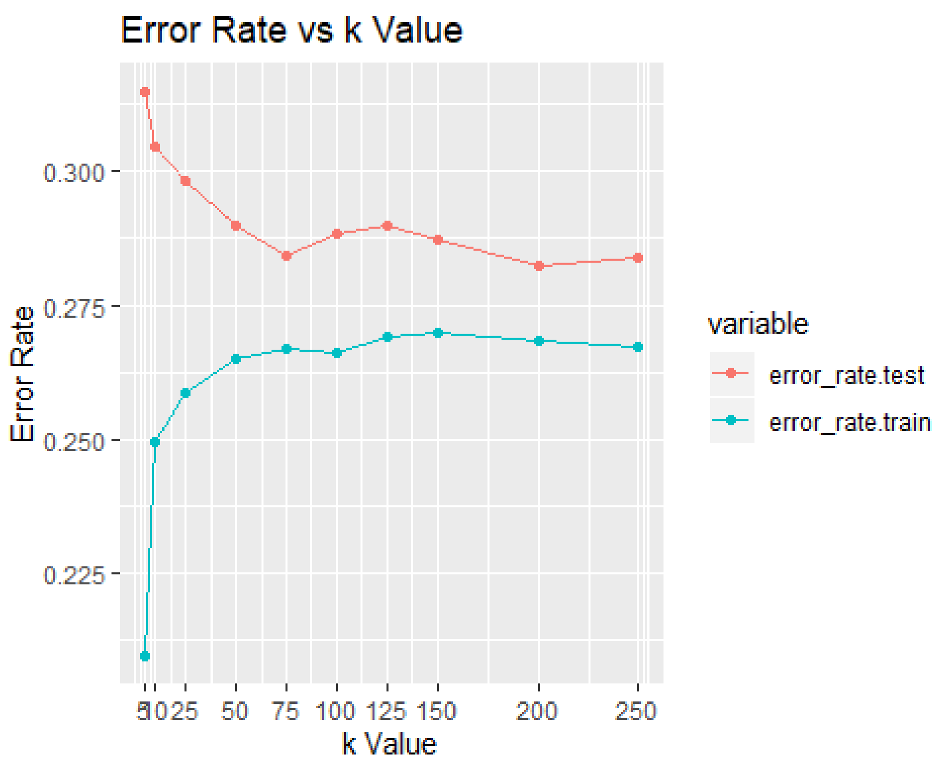
```
t.test(games$redWardsPlaced[games$blueWins==0], games$redWardsPlaced[games$bl
ueWins==1])

##
##  Welch Two Sample t-test
##
## data:  games$redWardsPlaced[games$blueWins == 0] and games$redWardsPlaced[
games$blueWins == 1]
## t = 2.3533, df = 9869.2, p-value = 0.01863
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.1459638 1.6015916
## sample estimates:
## mean of x mean of y
##  22.80400  21.93022
```

Splitting into train/test

```
set.seed(100)
sgames.tr.index <- sample(1:nrow(sgames), nrow(sgames)*.8)
sgames.test.index <- setdiff(1:nrow(sgames), sgames.tr.index)

sgames.tr <- sgames[sgames.tr.index,]
sgames.test <- sgames[sgames.test.index,]

train.labels <- blueWinsVector[sgames.tr.index]
test.labels <- blueWinsVector[sgames.test.index]

sum(games[sgames.tr.index,]$blueWins)/nrow(sgames.tr)

## [1] 0.5010755

sum(games[sgames.test.index,]$blueWins)/nrow(sgames.test)

## [1] 0.4908907
```

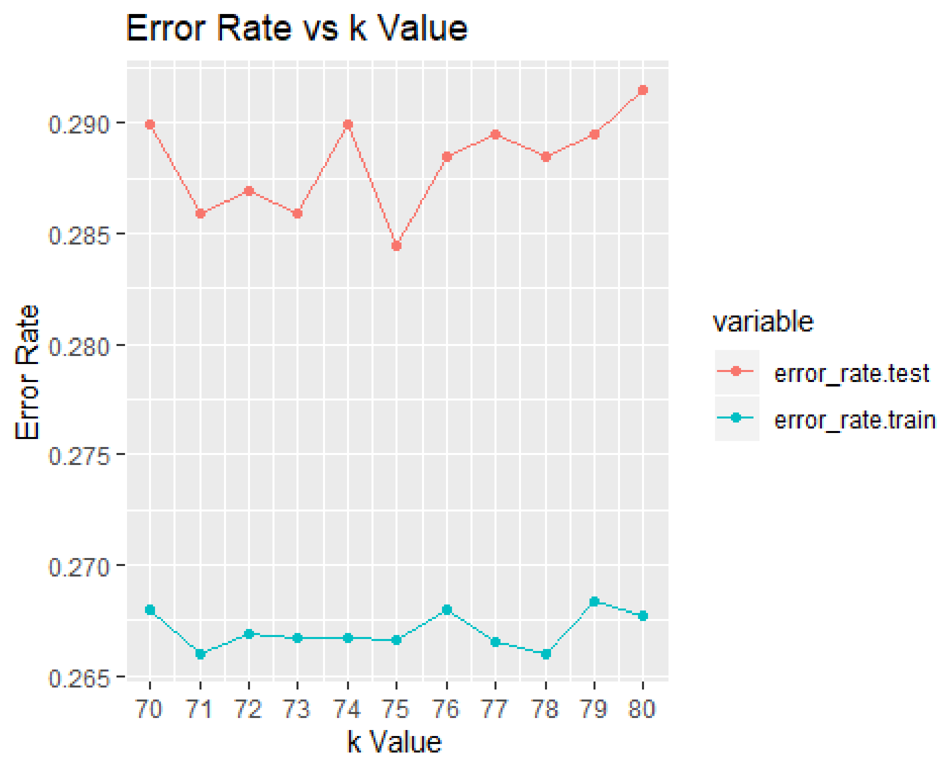
Implement KNN

```
library(class)

## Warning: package 'class' was built under R version 3.6.3

# Create a function to measure the accuracy of the model
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}

error_rate.test = c()
error_rate.train = c()

# loop through k values
k_values <- c(5,10,25,50,75,100,125,150,200,250)
for (i in k_values){
  knn.test <- knn(sgames.tr, sgames.test, train.labels, k=i)
  knn.train <- knn(sgames.tr, sgames.tr, train.labels, k=i)
  conf.mat.test <- table(knn.test, test.labels)
  conf.mat.train <- table(knn.train, train.labels)
  error_rate.test <- c(error_rate.test, 1 - (accuracy(conf.mat.test))/100)
  error_rate.train <- c(error_rate.train, 1 - (accuracy(conf.mat.train))/100)
}


# Plot the error rates of each k
library(ggplot2)
library(reshape2)
error_rate.k.test <- data.frame(k_values, error_rate.test)
error_rate.k.train <- data.frame(k_values, error_rate.train)
error_rate.k <- merge(error_rate.k.test, error_rate.k.train, by="k_values")
error_rate.k_melted <- reshape2::melt(error_rate.k, id.var='k_values')
p <- ggplot(error_rate.k_melted, aes(x=k_values, y=value, col=variable))+
  geom_line()+
  geom_point()+
  scale_x_continuous(breaks = k_values, labels = k_values)+
  labs(title="Error Rate vs k Value", y="Error Rate", x="k Value")

p
```



```
error_rate.test = c()
error_rate.train = c()

# loop through k values
k_values <- seq(70,80)
for (i in k_values){
  knn.test <- knn(sgames.tr, sgames.test, train.labels, k=i)
  knn.train <- knn(sgames.tr, sgames.tr, train.labels, k=i)
  conf.mat.test <- table(knn.test, test.labels)
  conf.mat.train <- table(knn.train, train.labels)
  error_rate.test <- c(error_rate.test, 1 - (accuracy(conf.mat.test))/100)
  error_rate.train <- c(error_rate.train, 1 - (accuracy(conf.mat.train))/100)
}


# Plot the error rates of each k
library(ggplot2)
library(reshape2)
error_rate.k.test <- data.frame(k_values, error_rate.test)
error_rate.k.train <- data.frame(k_values, error_rate.train)
error_rate.k <- merge(error_rate.k.test, error_rate.k.train, by="k_values")
error_rate.k_melted <- reshape2::melt(error_rate.k, id.var='k_values')
p <- ggplot(error_rate.k_melted, aes(x=k_values, y=value, col=variable))+
  geom_line()+
  geom_point()+
  scale_x_continuous(breaks = k_values, labels = k_values)+
  labs(title="Error Rate vs k Value", y="Error Rate", x="k Value")
```

```
p
```



```
# accuracy of test set
knn.test <- knn(sgames.tr, sgames.test, train.labels, k=75)
conf.mat.test <- table(knn.test, test.labels)
accuracy(conf.mat.test)

## [1] 71.5587

# accuracy of train set
knn.train <- knn(sgames.tr, sgames.tr, train.labels, k=75)
conf.mat.train <- table(knn.train, train.labels)
accuracy(conf.mat.train)

## [1] 73.28862

# Confidence interval
interval.test <- 1.96*sqrt((.7161*(1-.7161))/1976)
interval.train <- 1.96*sqrt((.7331*(1-.7331))/7903)

# confusion matrix
library('scales')

blueWinsTrueNeg <- percent(conf.mat.test[1,1]/sum(conf.mat.test[,1]))
blueWinsFalsePos <- percent(conf.mat.test[2,1]/sum(conf.mat.test[,1]))
blueWinsFalseNeg <- percent(conf.mat.test[1,2]/sum(conf.mat.test[,2]))
blueWinsTruePos <- percent(conf.mat.test[2,2]/sum(conf.mat.test[,2]))
```

```
row1 <- c(blueWinsTrueNeg, blueWinsFalseNeg)
row2 <- c(blueWinsFalsePos, blueWinsTruePos)
conf.matrix.percent <- rbind(row1, row2)
rownames(conf.matrix.percent) <- c("Pred: blueWins=0", "Pred: blueWins=1")
colnames(conf.matrix.percent) <- c("True: blueWins=0", "True: blueWins=1")
as.data.frame(conf.matrix.percent)
```

```
##                  True: blueWins=0 True: blueWins=1
## Pred: blueWins=0            73.5%            30.4%
## Pred: blueWins=1            26.5%            69.6%
```

As seen within the confusion matrix, the KNN algorithm with k=75 correctly predicts a blue loss 73.6% of the time and accuractely predicts a blue win 69.6% of the time. The algorithm also incorrectly predicts a win 26.4% of the time and incorrectly predicts a loss 30.4% of the time.

Step 4 to try and improve performance

Correlation of sgames and addressing multicollinearity

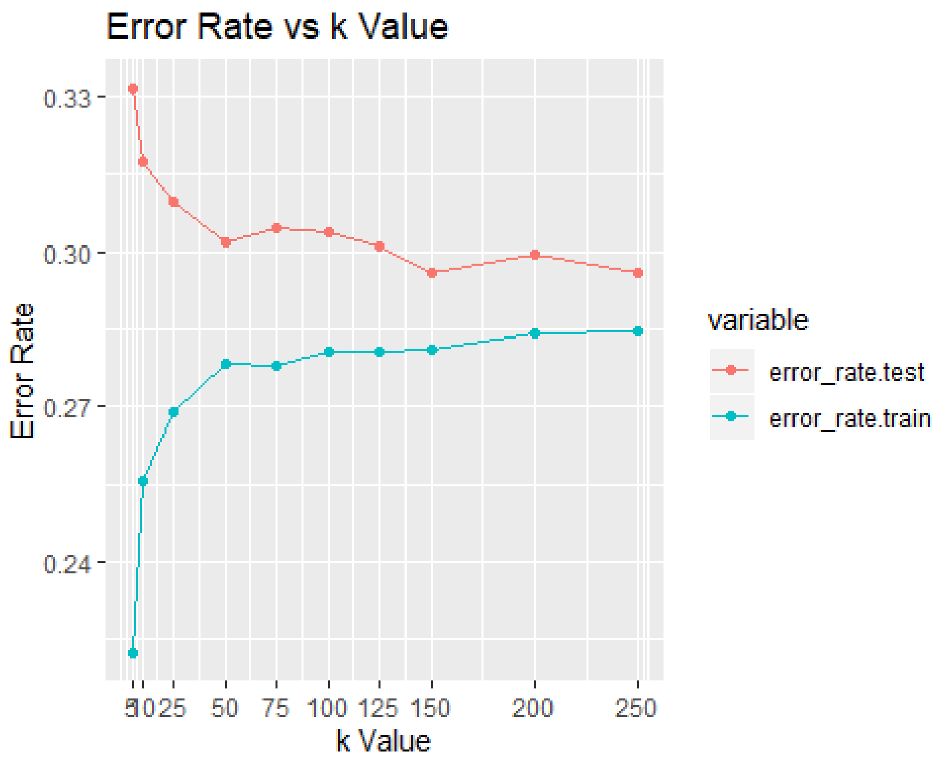
```
corr <- round(cor(sgames), 2)
corr[lower.tri(corr,diag=TRUE)]=NA
corr=as.data.frame(as.table(corr))
corr=na.omit(corr)
corr=corr[order(-abs(corr$Freq)),]
```

Get variable correlations

```
features_high_corr <- corr[abs(corr$Freq) > 0.75,]

features_perf_corr <- corr[abs(corr$Freq) == 1.00,]

features <- colnames(sgames)

features_clean <- features[-c(features_high_corr$Var2)]

features_no_perf <- features[-c(features_perf_corr$Var2)]
```

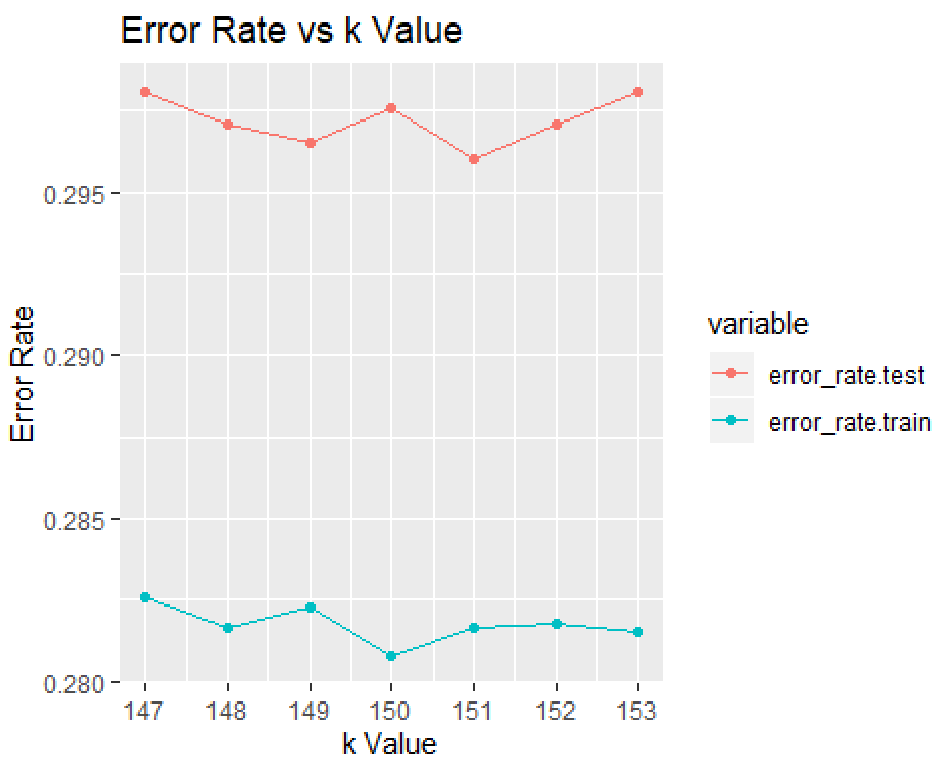
Removed variables with correlation over 0.75 and running knn

```
error_rate.test = c()
error_rate.train = c()

# loop k values
k_values <- c(5,10,25,50,75,100,125,150,200,250)
for (i in k_values){
  knn.test <- knn(sgames.tr[features_clean], sgames.test[features_clean], tra
in.labels, k=i)
  knn.train <- knn(sgames.tr[features_clean], sgames.tr[features_clean], trai
n.labels, k=i)
  conf.mat.test <- table(knn.test, test.labels)
  conf.mat.train <- table(knn.train, train.labels)
  error_rate.test <- c(error_rate.test, 1 - (accuracy(conf.mat.test))/100)
  error_rate.train <- c(error_rate.train, 1 - (accuracy(conf.mat.train))/100)
}


# Plot the error rates of each k
library(ggplot2)
library(reshape2)
error_rate.k.test <- data.frame(k_values, error_rate.test)
error_rate.k.train <- data.frame(k_values, error_rate.train)
error_rate.k <- merge(error_rate.k.test, error_rate.k.train, by="k_values")
error_rate.k_melted <- reshape2::melt(error_rate.k, id.var='k_values')
p <- ggplot(error_rate.k_melted, aes(x=k_values, y=value, col=variable))+
  geom_line()+
  geom_point()+
  scale_x_continuous(breaks = k_values, labels = k_values)+
  labs(title="Error Rate vs k Value", y="Error Rate", x="k Value")

p
```



```
error_rate.test = c()
error_rate.train = c()
```

```r
# further loop k values
k_values <- seq(147,153)
for (i in k_values){
  knn.test <- knn(sgames.tr[features_clean], sgames.test[features_clean], tra
in.labels, k=i)
  knn.train <- knn(sgames.tr[features_clean], sgames.tr[features_clean], trai
n.labels, k=i)
  conf.mat.test <- table(knn.test, test.labels)
  conf.mat.train <- table(knn.train, train.labels)
  error_rate.test <- c(error_rate.test, 1 - (accuracy(conf.mat.test))/100)
  error_rate.train <- c(error_rate.train, 1 - (accuracy(conf.mat.train))/100)
}


# Plot the error rates of each k
library(ggplot2)
library(reshape2)
error_rate.k.test <- data.frame(k_values, error_rate.test)
error_rate.k.train <- data.frame(k_values, error_rate.train)
error_rate.k <- merge(error_rate.k.test, error_rate.k.train, by="k_values")
error_rate.k_melted <- reshape2::melt(error_rate.k, id.var='k_values')
p <- ggplot(error_rate.k_melted, aes(x=k_values, y=value, col=variable))+
  geom_line()+
  geom_point()+
  scale_x_continuous(breaks = k_values, labels = k_values)+
  labs(title="Error Rate vs k Value", y="Error Rate", x="k Value")

p
```



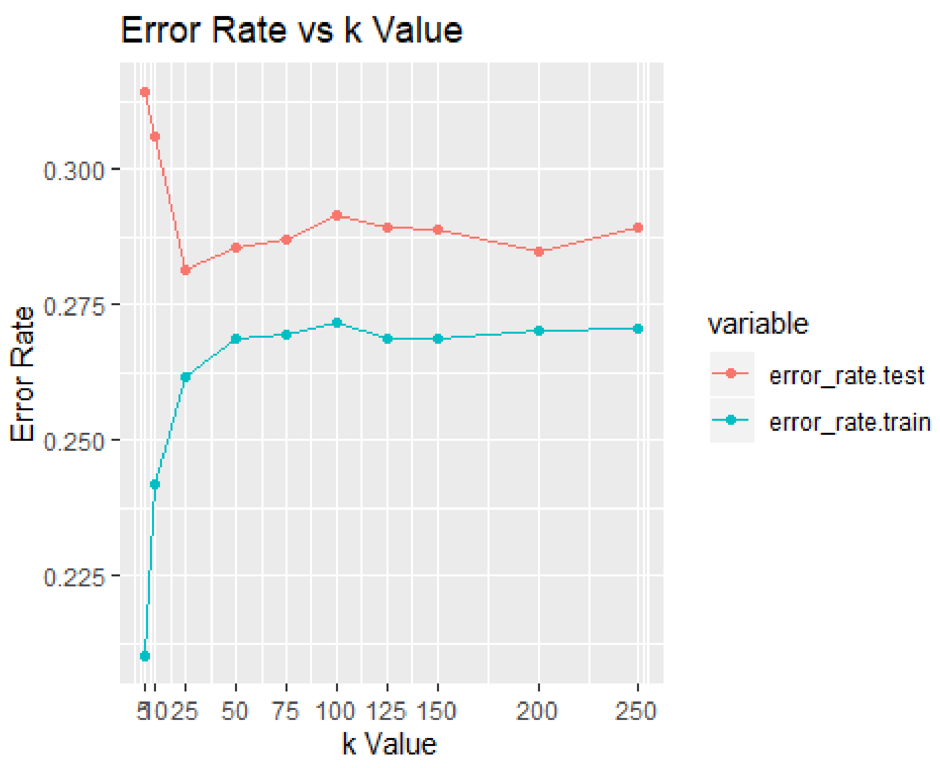
```r
# test set accuracy
knn.test <- knn(sgames.tr[features_clean], sgames.test[features_clean], train
.labels, 150)
conf.mat.test <- table(knn.test, test.labels)
accuracy(conf.mat.test)

## [1] 70.19231

# train set accuracy
knn.train <- knn(sgames.tr[features_clean], sgames.tr[features_clean], train.
labels, k=150)
conf.mat.train <- table(knn.train, train.labels)
accuracy(conf.mat.train)

## [1] 71.87144

# get confidence interval radius
interval.test <- 1.96*sqrt((.7014*(1-.7014))/1976)
interval.train <- 1.96*sqrt((.7188*(1-.7188))/7903)

# confusion matrix
blueWinsTrueNeg <- percent(conf.mat.test[1,1]/sum(conf.mat.test[,1]))
blueWinsFalsePos <- percent(conf.mat.test[2,1]/sum(conf.mat.test[,1]))
blueWinsFalseNeg <- percent(conf.mat.test[1,2]/sum(conf.mat.test[,2]))
blueWinsTruePos <- percent(conf.mat.test[2,2]/sum(conf.mat.test[,2]))

row1 <- c(blueWinsTrueNeg, blueWinsFalseNeg)
row2 <- c(blueWinsFalsePos, blueWinsTruePos)
```

```
conf.matrix.percent <- rbind(row1, row2)
rownames(conf.matrix.percent) <- c("Pred: blueWins=0", "Pred: blueWins=1")
colnames(conf.matrix.percent) <- c("True: blueWins=0", "True: blueWins=1")
as.data.frame(conf.matrix.percent)
```

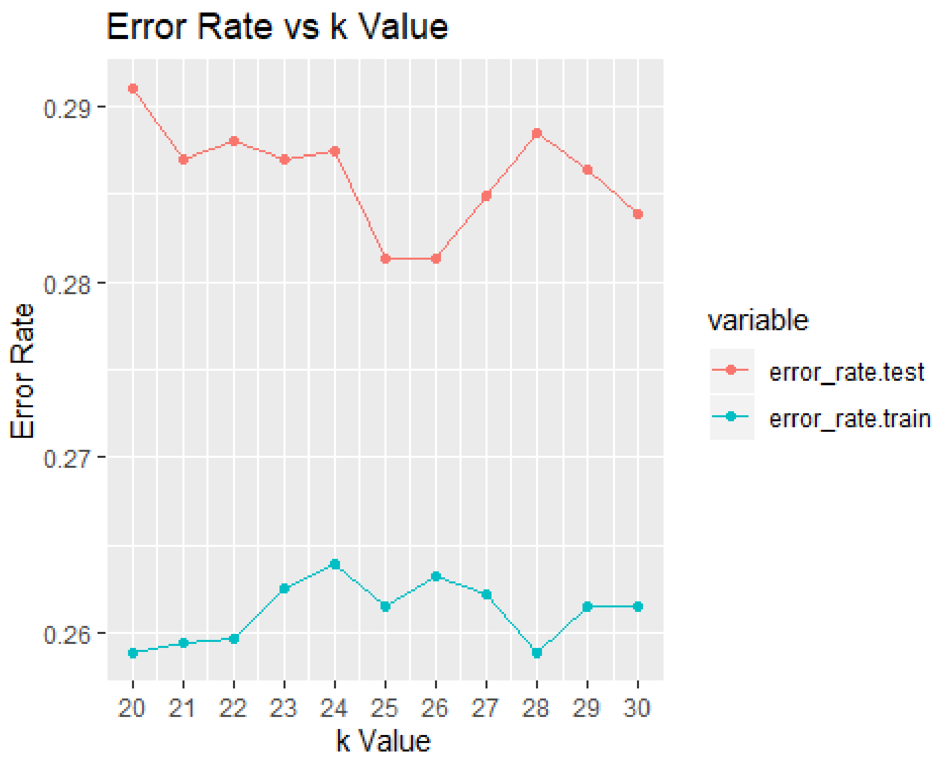
```
##                  True: blueWins=0 True: blueWins=1
## Pred: blueWins=0            72.9%            32.6%
## Pred: blueWins=1            27.1%            67.4%
```

Removed variables with perfect correlation

```
error_rate.test = c()
error_rate.train = c()

# loop through k values
k_values <- c(5,10,25,50,75,100,125,150,200,250)
for (i in k_values){
  knn.test <- knn(sgames.tr[features_no_perf], sgames.test[features_no_perf],
train.labels, k=i)
  knn.train <- knn(sgames.tr[features_no_perf], sgames.tr[features_no_perf],
train.labels, k=i)
  conf.mat.test <- table(knn.test, test.labels)
  conf.mat.train <- table(knn.train, train.labels)
  error_rate.test <- c(error_rate.test, 1 - (accuracy(conf.mat.test))/100)
  error_rate.train <- c(error_rate.train, 1 - (accuracy(conf.mat.train))/100)
}


# Plot the error rates of each k
library(ggplot2)
library(reshape2)
error_rate.k.test <- data.frame(k_values, error_rate.test)
error_rate.k.train <- data.frame(k_values, error_rate.train)
error_rate.k <- merge(error_rate.k.test, error_rate.k.train, by="k_values")
error_rate.k_melted <- reshape2::melt(error_rate.k, id.var='k_values')
p <- ggplot(error_rate.k_melted, aes(x=k_values, y=value, col=variable))+
  geom_line()+
  geom_point()+
  scale_x_continuous(breaks = k_values, labels = k_values)+
  labs(title="Error Rate vs k Value", y="Error Rate", x="k Value")

p
```



```
error_rate.test = c()
error_rate.train = c()

# further loop through k values
k_values <- seq(20,30)
for (i in k_values){
  knn.test <- knn(sgames.tr[features_no_perf], sgames.test[features_no_perf],
train.labels, k=i)
  knn.train <- knn(sgames.tr[features_no_perf], sgames.tr[features_no_perf],
train.labels, k=i)
  conf.mat.test <- table(knn.test, test.labels)
  conf.mat.train <- table(knn.train, train.labels)
  error_rate.test <- c(error_rate.test, 1 - (accuracy(conf.mat.test))/100)
  error_rate.train <- c(error_rate.train, 1 - (accuracy(conf.mat.train))/100)
}


# Plot the error rates of each k
library(ggplot2)
library(reshape2)
error_rate.k.test <- data.frame(k_values, error_rate.test)
error_rate.k.train <- data.frame(k_values, error_rate.train)
error_rate.k <- merge(error_rate.k.test, error_rate.k.train, by="k_values")
error_rate.k_melted <- reshape2::melt(error_rate.k, id.var='k_values')
p <- ggplot(error_rate.k_melted, aes(x=k_values, y=value, col=variable))+
  geom_line()+
  geom_point()+
  scale_x_continuous(breaks = k_values, labels = k_values)+
  labs(title="Error Rate vs k Value", y="Error Rate", x="k Value")

p
```



```
# test set accuracy
knn.test <- knn(sgames.tr[features_no_perf], sgames.test[features_no_perf], t
rain.labels, k=25)
conf.mat.test <- table(knn.test, test.labels)
accuracy(conf.mat.test)

## [1] 71.86235

# train set accuracy
knn.train <- knn(sgames.tr[features_no_perf], sgames.tr[features_no_perf], tr
ain.labels, k=25)
conf.mat.train <- table(knn.train, train.labels)
accuracy(conf.mat.train)

## [1] 73.83272

# confidence interval
interval.test <- 1.96*sqrt((.7186*(1-.7186))/1976)
interval.train <- 1.96*sqrt((.7383*(1-.7383))/7903)

# confusion matrix
blueWinsTrueNeg <- percent(conf.mat.test[1,1]/sum(conf.mat.test[,1]))
blueWinsFalsePos <- percent(conf.mat.test[2,1]/sum(conf.mat.test[,1]))
```

```r
blueWinsFalseNeg <- percent(conf.mat.test[1,2]/sum(conf.mat.test[,2]))
blueWinsTruePos <- percent(conf.mat.test[2,2]/sum(conf.mat.test[,2]))


row1 <- c(blueWinsTrueNeg, blueWinsFalseNeg)
row2 <- c(blueWinsFalsePos, blueWinsTruePos)
conf.matrix.percent <- rbind(row1, row2)
rownames(conf.matrix.percent) <- c("Pred: blueWins=0", "Pred: blueWins=1")
colnames(conf.matrix.percent) <- c("True: blueWins=0", "True: blueWins=1")
as.data.frame(conf.matrix.percent)
```

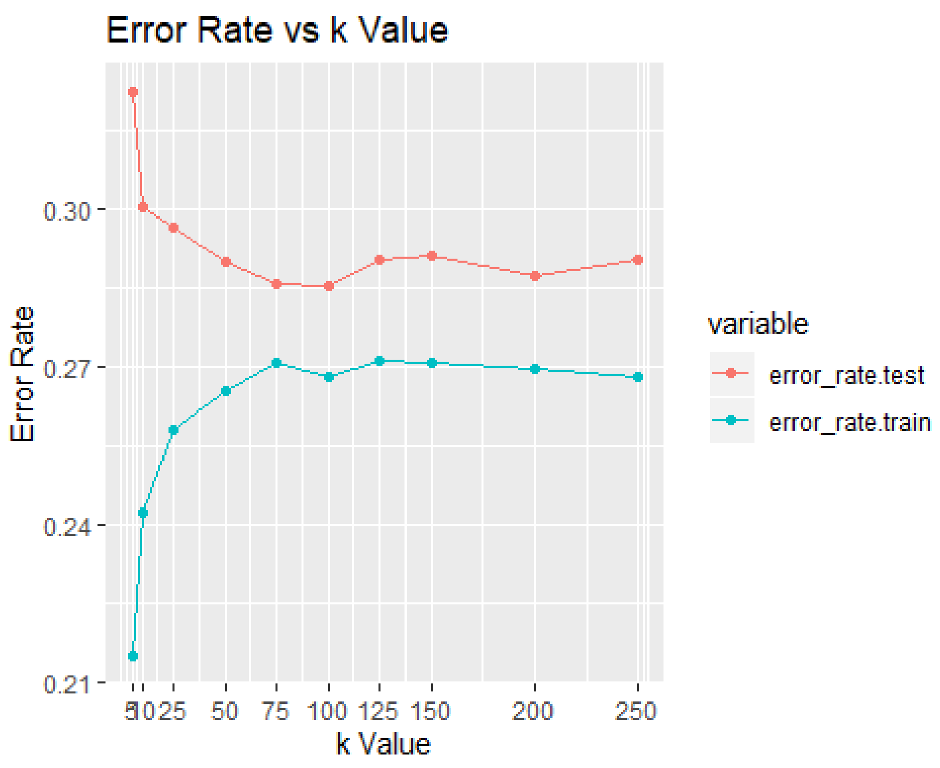
```
##                    True: blueWins=0 True: blueWins=1
## Pred: blueWins=0              73.7%            30.0%
## Pred: blueWins=1              26.3%            70.0%
```

PCA + KNN

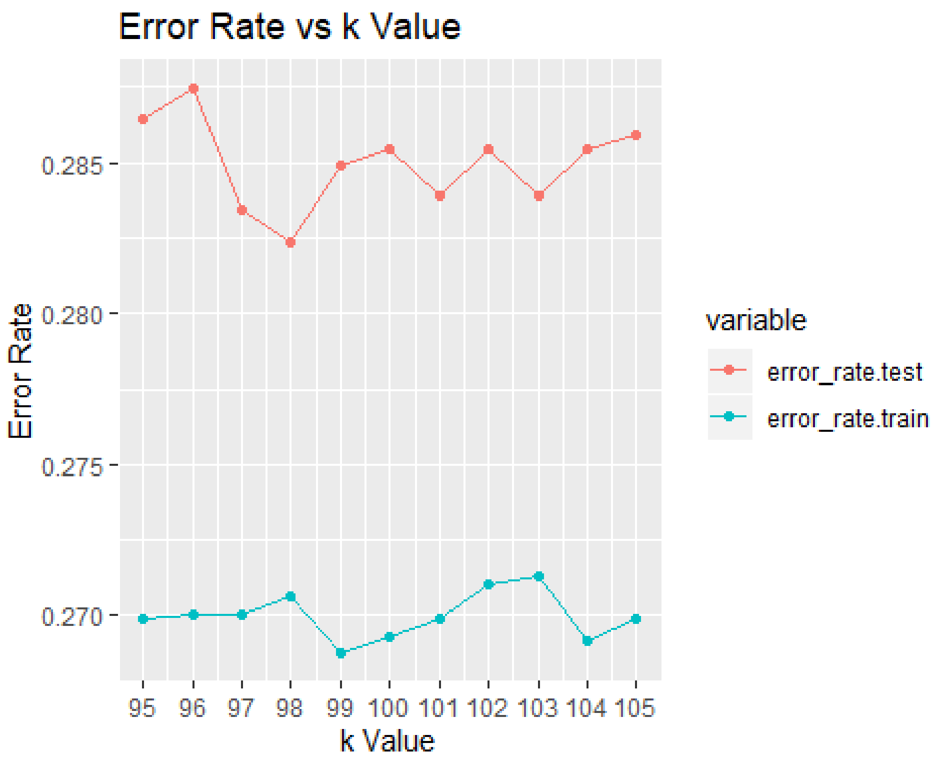
```r
# PCA
sgames.cov <- cov(sgames[features_no_perf])
sgames.eigen <- eigen(sgames.cov)
phi <- sgames.eigen$vectors[,]
phi <- -phi

# Variance explained
PVE <- sgames.eigen$values/sum(sgames.eigen$values)

for (i in 1:length(features_no_perf)){
  assign(paste("PC", i, sep = ""), as.matrix(sgames[features_no_perf]) %*% ph
i[,i])
}

# PC <- data.frame(PC1, PC2, PC3, PC4, PC5, PC6, PC7, PC8, PC9, PC10, PC11, P
C12, PC13, PC14, PC15, PC16, PC17, PC18, PC19, PC20, PC21, PC22, PC23, PC24,
PC25, PC26, PC27, PC28, PC29, PC30, PC31, PC32, PC33, PC34, PC35, PC36, PC37,
PC38)
PC <- data.frame(PC1, PC2, PC3, PC4, PC5, PC6, PC7, PC8, PC9, PC10, PC11, PC1
2, PC13, PC14, PC15, PC16, PC17, PC18, PC19, PC20, PC21, PC22, PC23, PC24, PC
25, PC26, PC27, PC28, PC29)
```

PVE Plot

```r
# it = 0
# PVE.sum <- c()
# for (i in 1:length(PVE)){
#   PVE.sum <- sum(PVE[1:i])
#   it = it + 1
#   if (PVE.sum > .95){
#     break
#   }
# }
#
# # Cumulative PVE plot
# cumPVE <- qplot(c(1:29), cumsum(PVE)) +
#   geom_line() +
#   xlab("Principal Component") +
#   ylab(NULL) +
#   ggtitle("Cumulative PVE Plot") +
#   ylim(0,1)
#
# plot.new()
# cumPVE
# abline(h=.95, col="red")

error_rate.test = c()
error_rate.train = c()

# loop through k values
k_values <- c(5,10,25,50,75,100,125,150,200,250)
for (i in k_values){
  knn.test <- knn(PC[sgames.tr.index,1:17], PC[sgames.test.index,1:17], train
.labels, k=i)
  knn.train <- knn(PC[sgames.tr.index,1:17], PC[sgames.tr.index,1:17], train.
labels, k=i)
  conf.mat.test <- table(knn.test, test.labels)
  conf.mat.train <- table(knn.train, train.labels)
  error_rate.test <- c(error_rate.test, 1 - (accuracy(conf.mat.test))/100)
  error_rate.train <- c(error_rate.train, 1 - (accuracy(conf.mat.train))/100)
}


# Plot the error rates of each k
library(ggplot2)
library(reshape2)
error_rate.k.test <- data.frame(k_values, error_rate.test)
error_rate.k.train <- data.frame(k_values, error_rate.train)
error_rate.k <- merge(error_rate.k.test, error_rate.k.train, by="k_values")
error_rate.k_melted <- reshape2::melt(error_rate.k, id.var='k_values')
p <- ggplot(error_rate.k_melted, aes(x=k_values, y=value, col=variable))+
  geom_line()+
  geom_point()+
  scale_x_continuous(breaks = k_values, labels = k_values)+
  labs(title="Error Rate vs k Value", y="Error Rate", x="k Value")

p
```



```
error_rate.test = c()
error_rate.train = c()

# further loop through k values
k_values <- seq(95,105)
for (i in k_values){
  knn.test <- knn(PC[sgames.tr.index,1:17], PC[sgames.test.index,1:17], train
.labels, k=i)
  knn.train <- knn(PC[sgames.tr.index,1:17], PC[sgames.tr.index,1:17], train.
labels, k=i)
  conf.mat.test <- table(knn.test, test.labels)
  conf.mat.train <- table(knn.train, train.labels)
  error_rate.test <- c(error_rate.test, 1 - (accuracy(conf.mat.test))/100)
  error_rate.train <- c(error_rate.train, 1 - (accuracy(conf.mat.train))/100)
}


# Plot the error rates of each k
library(ggplot2)
library(reshape2)
error_rate.k.test <- data.frame(k_values, error_rate.test)
error_rate.k.train <- data.frame(k_values, error_rate.train)
error_rate.k <- merge(error_rate.k.test, error_rate.k.train, by="k_values")
error_rate.k_melted <- reshape2::melt(error_rate.k, id.var='k_values')
p <- ggplot(error_rate.k_melted, aes(x=k_values, y=value, col=variable))+
  geom_line()+
  geom_point()+
  scale_x_continuous(breaks = k_values, labels = k_values)+
  labs(title="Error Rate vs k Value", y="Error Rate", x="k Value")

p
```



```
# test accuracy
knn.test <- knn(PC[sgames.tr.index,1:17], PC[sgames.test.index,1:17], train.l
abels, k=98)
conf.mat.test <- table(knn.test, test.labels)
accuracy(conf.mat.test)

## [1] 71.40688

# train accuracy
knn.train <- knn(PC[sgames.tr.index,1:17], PC[sgames.tr.index,1:17], train.la
bels, k=98)
conf.mat.train <- table(knn.train, train.labels)
accuracy(conf.mat.train)

## [1] 72.92168

# confidence interval radius
interval.test <- 1.96*sqrt((.7181*(1-.7181))/1976)
interval.train <- 1.96*sqrt((.7307*(1-.7307))/7903)

# confusion matrix
blueWinsTrueNeg <- percent(conf.mat.test[1,1]/sum(conf.mat.test[,1]))
blueWinsFalsePos <- percent(conf.mat.test[2,1]/sum(conf.mat.test[,1]))
```

```
blueWinsFalseNeg <- percent(conf.mat.test[1,2]/sum(conf.mat.test[,2]))
blueWinsTruePos <- percent(conf.mat.test[2,2]/sum(conf.mat.test[,2]))


row1 <- c(blueWinsTrueNeg, blueWinsFalseNeg)
row2 <- c(blueWinsFalsePos, blueWinsTruePos)
conf.matrix.percent <- rbind(row1, row2)
rownames(conf.matrix.percent) <- c("Pred: blueWins=0", "Pred: blueWins=1")
colnames(conf.matrix.percent) <- c("True: blueWins=0", "True: blueWins=1")
as.data.frame(conf.matrix.percent)
```

```
##                  True: blueWins=0 True: blueWins=1
## Pred: blueWins=0            74.0%            31.2%
## Pred: blueWins=1            26.0%            68.8%
```

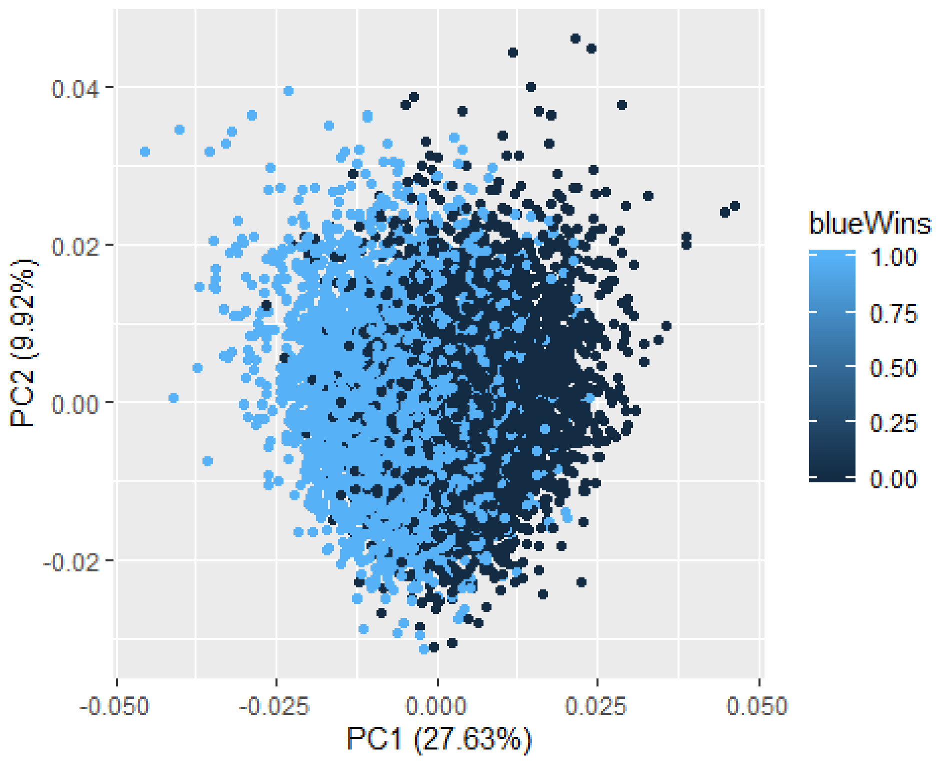
Plot PCA

```
library(ggfortify)
```

```
## Warning: package 'ggfortify' was built under R version 3.6.3
```

```
pca_res <- prcomp(sgames[features_no_perf])

autoplot(pca_res, data = games, colour = 'blueWins')
```

```
## Warning: `select_()` is deprecated as of dplyr 0.7.0.
## Please use `select()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```



0.04
0.02
0.00
-0.02
PC2 (9.92%)
-0.050
-0.025
0.000
0.025
0.050
PC1 (27.63%)
blueWins
1.00
0.75
0.50
0.25
0.00