Code ▾

Predicting the Outcome of a League of Legends game

Weichen Song

2023-03-09

Introduction

League of Legends (LoL) is a popular multiplayer online battle arena (MOBA) video game developed by Riot Games. It is a complex and dynamic game that requires strategic planning, skillful execution, and teamwork. In LoL, two teams of five players each compete against each other to destroy the opposing team's nexus, a structure located in their base. The game is won by the team that destroys the enemy nexus first or forces the enemy team to surrender.

The first ten minutes of a LoL match are crucial, as they set the tone for the rest of the game. During this time, both teams try to gain an advantage over the other by acquiring gold, experience, and objectives on the map. This phase of the game is referred to as the "early game," and it can significantly impact the outcome of the match.

The purpose of this report is to explore the use of machine learning models to predict the outcome of a LoL match based on the performance of both teams in the first ten minutes of the game. Specifically, I will focus on predicting whether the blue side will win or lose based on various factors such as gold, experience, and objective control. I will be using data from Kaggle and implementing multiple methods to yield an excellent model for this classification problem.

Motive

As an enthusiastic League of Legends (LoL) player, I have always been fascinated by the intricacies of the game and the strategies that players use to win. In particular, I have often wondered what I should do in the early game to give my team the best chance of winning. The first ten minutes of a LoL match are crucial, and I have often found myself struggling to make the right decisions during this phase of the game. Sometimes my team had a lead in kills but eventually we lost the game somehow. Therefore, I want to use this project to discover what my team should be focusing on in the early game in order to win. This could also be a valuable insight for other Lol players.

Exploratory Data Analysis

Loading and Exploring our Data

The data was obtained from the Kaggle dataset: League of Legends Diamond Ranked Games(10 min) (https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min?datasetId=600276&searchQuery=classification). Before taking a look at the dataset, let us get familiar with some Lol terms:

- Warding totem: An item that a player can put on the map to reveal the nearby area. Very useful for map/objectives control. Better warding leads to less avoidable deaths.
- 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: Structures you have to destroy to reach the enemy Nexus. They give gold.
- Level: Champion level. Start at 1. Max is 18.
- Gold: Players can buy items with gold. More gold means better items which make character strong in fights.
- Experience: Characters need experience to level up. Experience comes from killing minions, dragons, etc. The higher the level of a character, the stronger it is.

Now let's load our data and packages:

Hide

```
# Load packages
library(workflows)
library(dplyr)
library(tidyverse)
library(dplyr)
library(tidymodels)
library(kknn)
library(ISLR)
library(broom)
library(dials)
library(glmnet)
library(corrplot)
library(ggplot2)
library(rsample)
library(recipes)
library(discrim)
library(vip)

# Load Data
set.seed(39)
lol_data <- read.csv("lol data/high_diamond_ranked_10min.csv")

dim(lol_data)
```

```
## [1] 9879   40
```

Hide

```
colnames(lol_data) # Look at column names
```

```
##  [1] "gameId"                       "blueWins"
##  [3] "blueWardsPlaced"              "blueWardsDestroyed"
##  [5] "blueFirstBlood"               "blueKills"
##  [7] "blueDeaths"                   "blueAssists"
##  [9] "blueEliteMonsters"            "blueDragons"
## [11] "blueHeralds"                  "blueTowersDestroyed"
## [13] "blueTotalGold"                "blueAvgLevel"
## [15] "blueTotalExperience"          "blueTotalMinionsKilled"
## [17] "blueTotalJungleMinionsKilled" "blueGoldDiff"
## [19] "blueExperienceDiff"           "blueCSPerMin"
## [21] "blueGoldPerMin"               "redWardsPlaced"
## [23] "redWardsDestroyed"            "redFirstBlood"
## [25] "redKills"                     "redDeaths"
## [27] "redAssists"                   "redEliteMonsters"
## [29] "redDragons"                   "redHeralds"
## [31] "redTowersDestroyed"           "redTotalGold"
## [33] "redAvgLevel"                  "redTotalExperience"
## [35] "redTotalMinionsKilled"        "redTotalJungleMinionsKilled"
## [37] "redGoldDiff"                  "redExperienceDiff"
## [39] "redCSPerMin"                  "redGoldPerMin"
```

The data set contains 9879 rows and 40 columns. This means our data comes from nearly 10 thousand games! One good thing is that the data set has been cleaned so there is no missing data. Notice that there are 40 predictors. I need to take a look and make some selections.

Tidying the data

Looking at the columns, I notice that the first column is `gameId` which is irrelevant to our prediction so I will remove it. Since we only care about whether the blue team will win, I will remove all the columns about the red team. This is reasonable and will not influence our results because even though the red teams' performance does impact the flow of the game, some of the columns of the red team is 100% correlated with columns of the blue team such as `redGoldDiff` and `blueGoldDiff`.

Besides, the columns are all numeric so we do not need to make any changes on that except for `blueWins`, which I will make another factor column called `Win` with `Win` = "Yes" if `blueWins` = 1 and "No" if `blueWins` = 0.

Hide

```
# Remove the "gameId" column
lol_data <- lol_data[,-1]

# Subset only the columns related to the blue team
lol_blue_data <- select(lol_data, c(1:20))

# Make "blueWins" a factor
lol_blue_data$blueWins <- as.character(lol_blue_data$blueWins)
lol_blue_data$Win <- lol_blue_data$blueWins
lol_blue_data$Win[lol_blue_data$Win == "1"] <- "Yes"
lol_blue_data$Win[lol_blue_data$Win == "0"] <- "No"
lol_blue_data$Win <- as.factor(lol_blue_data$Win)
```

After getting all the columns related to the blue team, I need to further delete some highly correlated columns. For example, `blueCSPerMin` and `blueGoldPerMin`, `blueGoldPerMin` and `blueTotalGold`, `blueAvgLevel` and `blueTotalExperience`, etc.

Hide

```
# Remove unwanted columns
lol_blue_data <- lol_blue_data %>% subset(select=-c(blueEliteMonsters,blueAvgLevel,
blueTotalMinionsKilled,blueTotalJungleMinionsKilled,blueCSPerMin,blueGoldPerMin,blu
eFirstBlood,blueWins,blueAssists,blueWardsDestroyed))
```

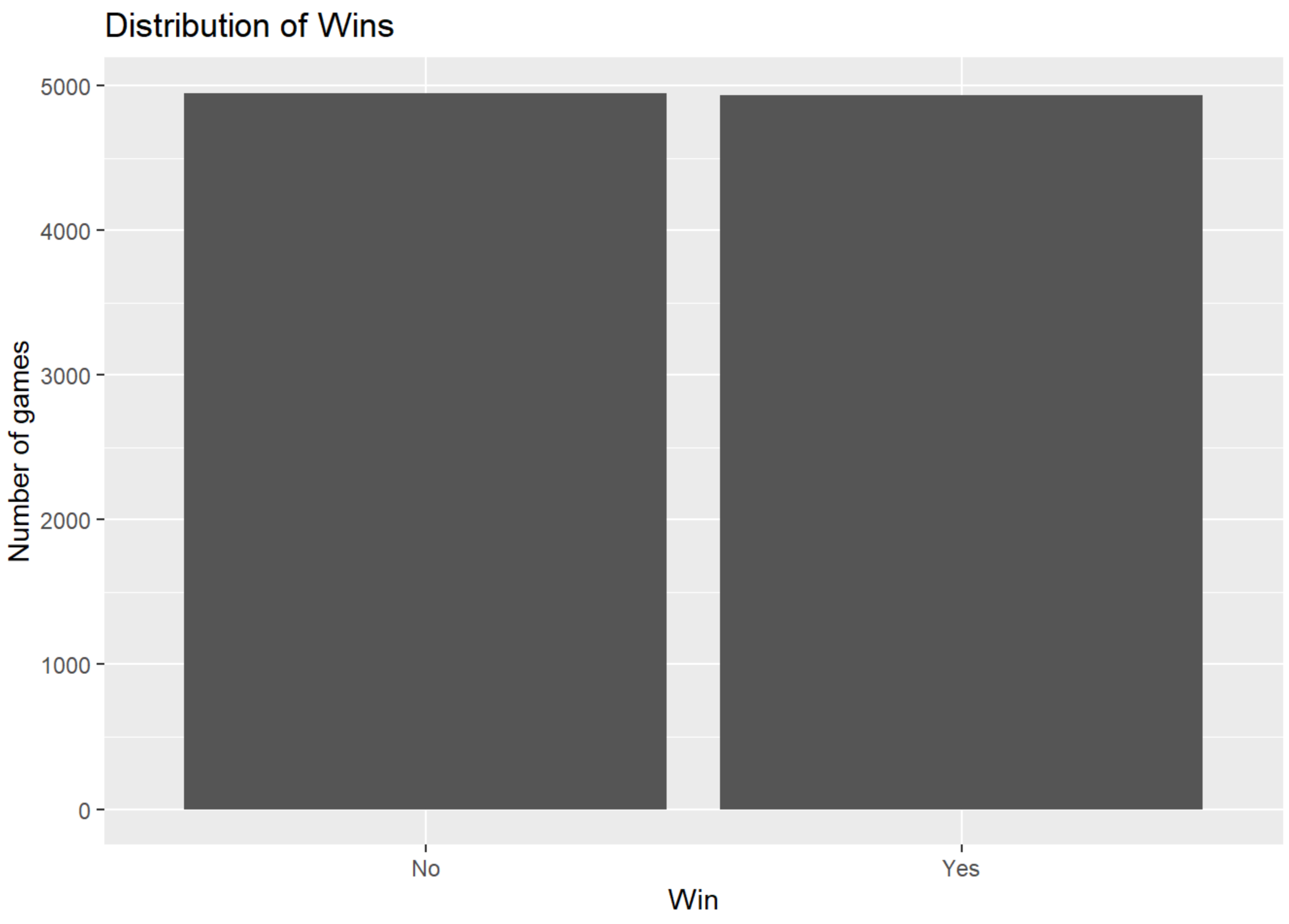
Visual EDA

Now let's get a better idea of the distribution of our response variable and relationships between the predictor variables. Additionally, I'll create visualization plots to see the effect that certain variables of interest have on our response variable.

Distribution of "Win"

Hide

```
# Looking at distributin of "Win"
ggplot(lol_blue_data, aes(x = Win)) +
  geom_bar() +
  labs(x = "Win", y = "Number of games", title = "Distribution of Wins")
```



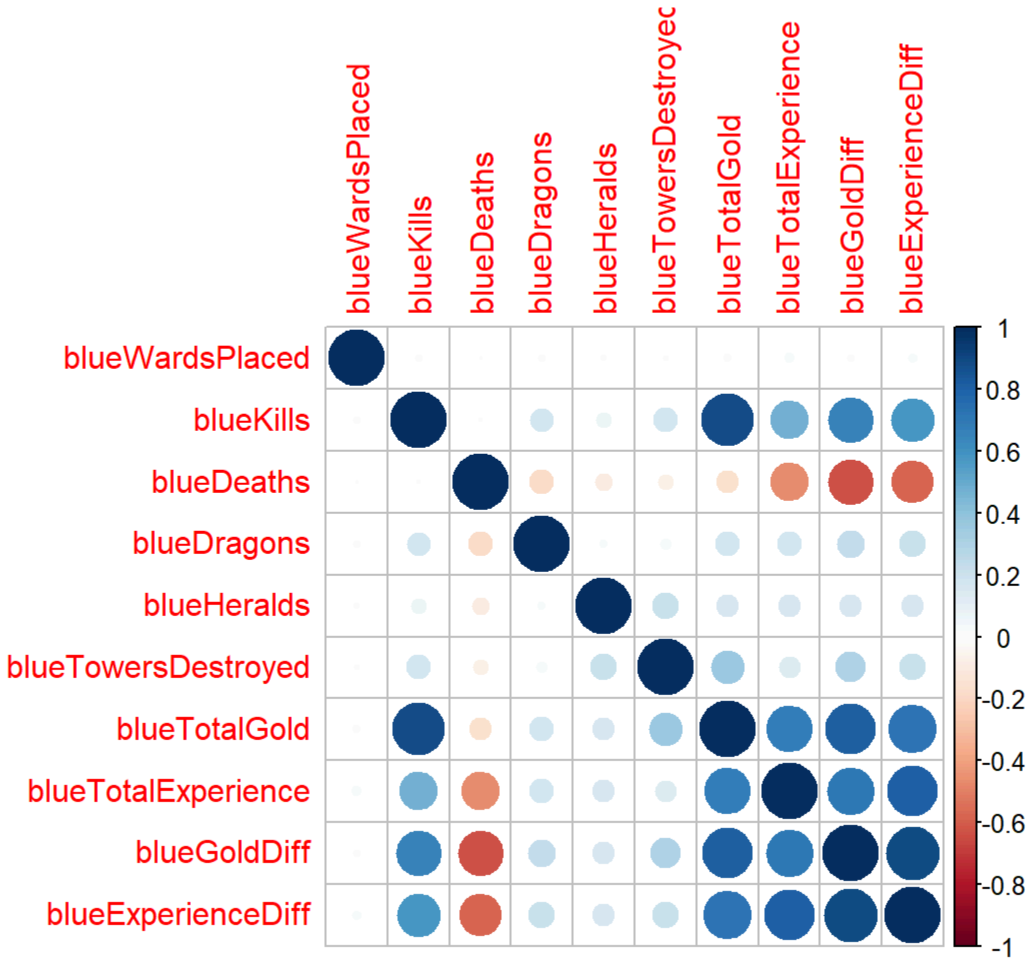
We can see that the percentage of winning and losing for the blue team is approximately equal, which makes sense since Lol's matchmaking system is designed in this way in order to maintain fairness.

Variable Correlation Plot

Now let's make a correlation map of the predictor variables,

Hide

```
numeric_var <- lol_blue_data %>% select_if(is.numeric)
corrplot(cor(numeric_var))
```



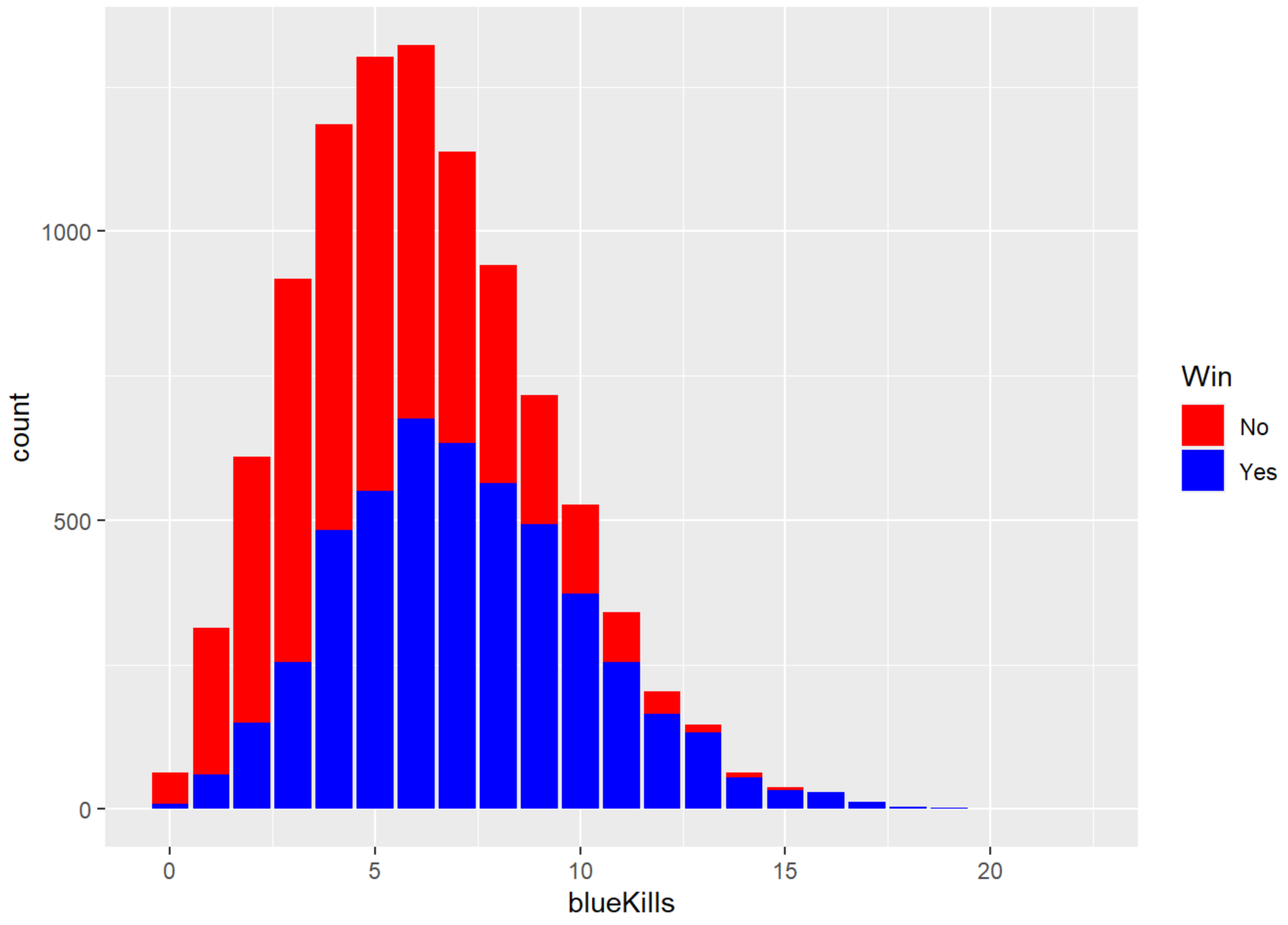
Here we can notice that the variables in the lower right corner of the map are correlated with each other. This is because when the blue team has a high total amount of gold in the first 10 minutes of a game, it is very likely that they have a gold advantage over the red team. In addition, players usually gain gold and experience at the same time (killing minions, elite monsters, enemy champions, etc.) so gold and experience are positively correlated. Moreover, we can see that `blueKills` is positively correlated with variables associated with gold and experience, and `blueDeaths` is negatively correlated with them. This is simply because killing an enemy champion gives a lot of gold and experience.

Kills

Let's look at how number of kills the blue team has in the 1st 10 minute is related to winning:

Hide

```
# Barplot of difference in Wins by blueKills
ggplot(lol_blue_data, aes(blueKills)) +
  geom_bar(aes(fill = Win)) +
  scale_fill_manual(values = c("red", "blue"))
```



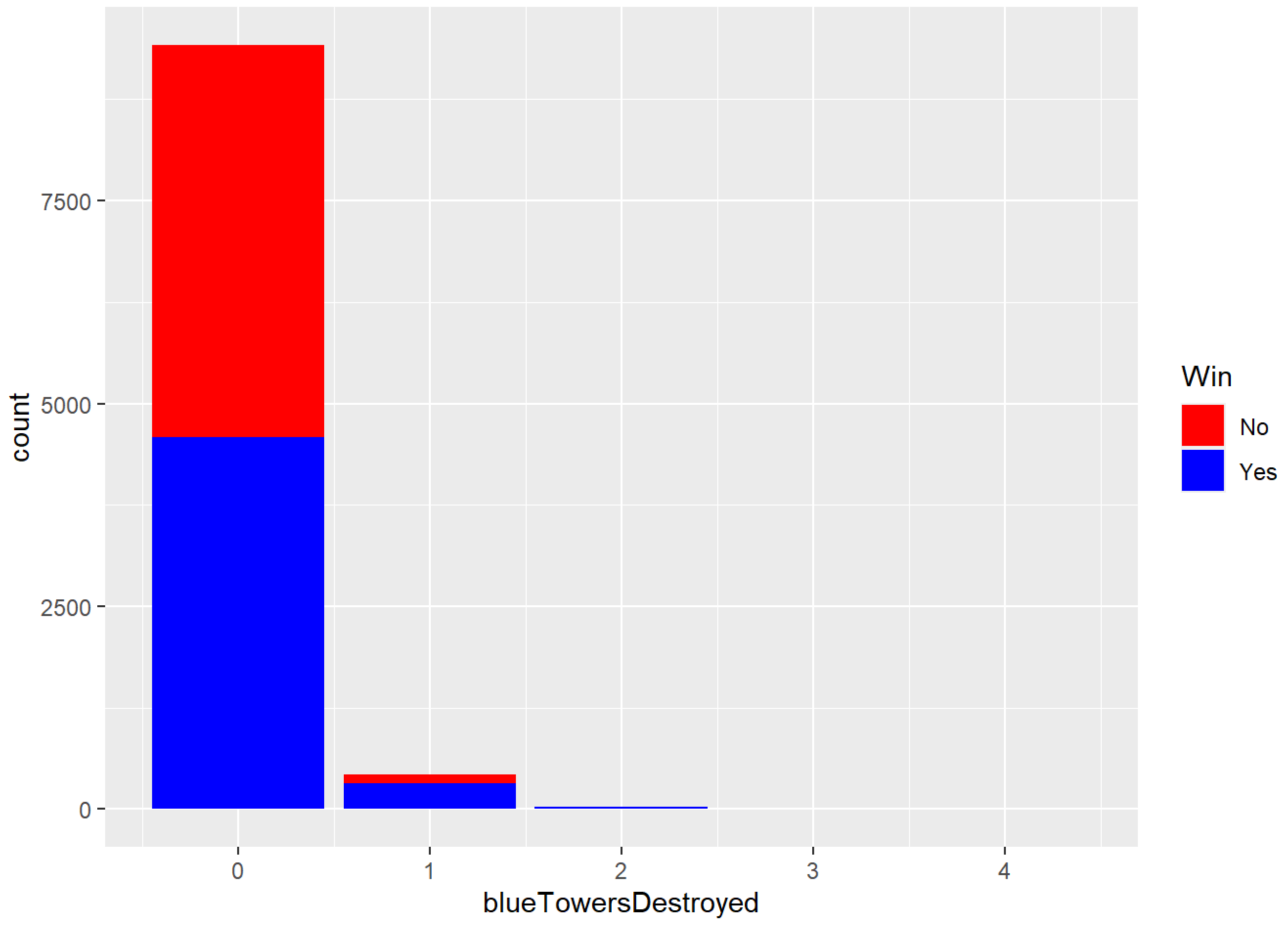
Since lead in kills gives a lot of gold and experience advantage, when the blue team has more than 8 kills in the early game, it obviously has a higher chance of winning.

Towers

While the relationship between kills, gold and experience advantage and winning is somewhat apparent, let’s take a look at if destroying enemy towers has a crucial role.

Hide

```
# Barplot of difference in Wins by blueTowersDestoyed
ggplot(lol_blue_data, aes(blueTowersDestroyed)) +
  geom_bar(aes(fill = Win)) +
  scale_fill_manual(values = c("red", "blue"))
```



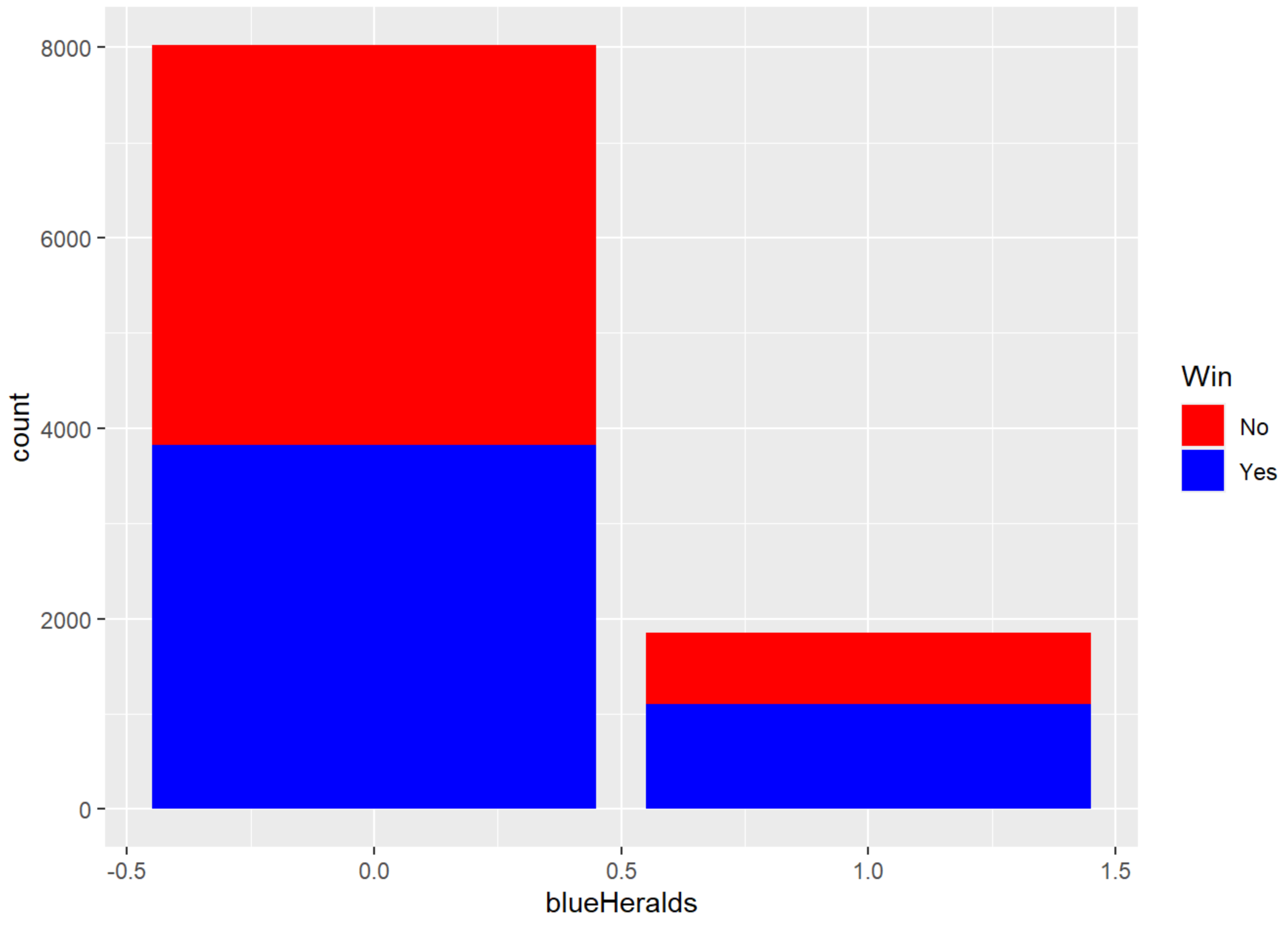
Notice that the maximum number of towers the blue teams destroyed is 1. This is because towers are designed to have high defensive power in the early game. When the blue team destroys zero towers, it is hard to tell how the game will go. On the other hand, if the blue team is able to destroy just one tower in the early game, it will have a much higher chance of winning since destroying the first tower gives a lot of gold for all champions on the blue team.

Heralds

Heralds is a vital object of the game. Only one Heralds will spawn in the first 10 minutes of a game. Once a champion kills it, the champion can summon it and it does heavy damage on enemy powers. Let’s see how it impacts the game:

Hide

```
ggplot(lol_blue_data, aes(blueHeralds)) +
  geom_bar(aes(fill = Win)) +
  scale_fill_manual(values = c("red", "blue"))
```



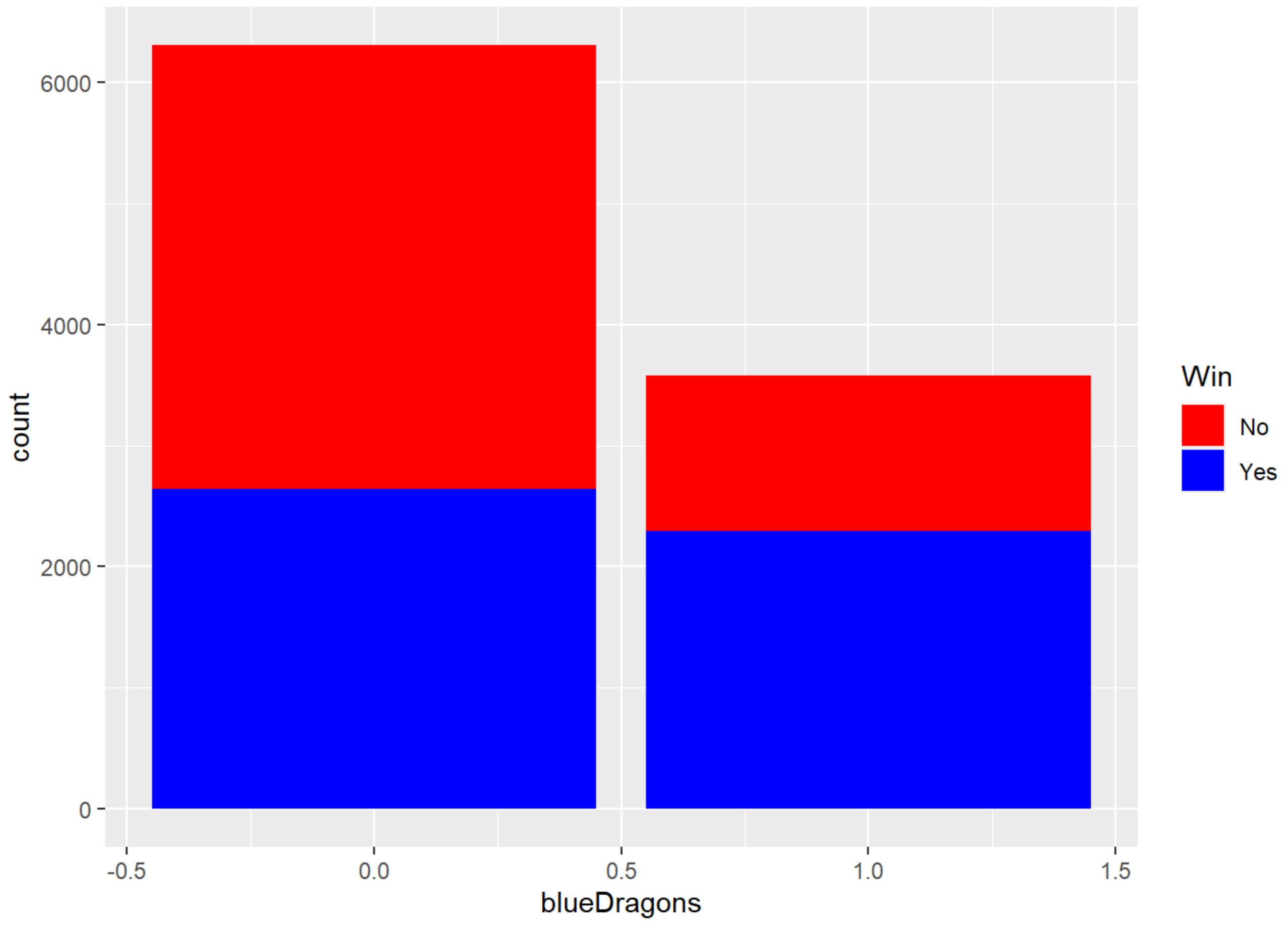
As shown, zero Heralds gives us no useful information. But if the blue team gets the Heralds in the early game, it has a slightly higher change of winning since Heralds is helpful but not a decisive factor.

Dragon

Finally, let's explore how dragons impact the game:

Hide

```
ggplot(lol_blue_data, aes(blueDragons)) +
  geom_bar(aes(fill = Win)) +
  scale_fill_manual(values = c("red", "blue"))
```



Again, just like Heralds, getting the first dragon gives a small lead. Now, combine the previous information, we might guess that getting both the first Heralds and first Dragon gives the blue team a larger advantage.

Setting Up Models

Data Split

Now let's start predicting "Win" using the predictors we have. We will split our data, create recipes and perform cross-validations.

Hide

```
lol_split <- initial_split(lol_blue_data, prop = 0.75,
                              strata = "Win")

lol_train <- training(lol_split)
lol_test <- testing(lol_split)
```

Let's check that our split is correct

Hide

```
# Verify proportion
dim(lol_train)
```

```
## [1] 7408   11
```

Hide

```
dim(lol_test)
```

```
## [1] 2471   11
```

Since 7408/2471 ≈ 2.99, our split is correct.

Building Recipe

After the filtering of variables we discussed earlier, we now will be using 10 predictor variables in our recipe.

Hide

```
# building our recipe
lol_recipe <- recipe(Win ~ blueWardsPlaced + blueKills + blueDeaths + blueDragons +
blueHeralds + blueTowersDestroyed + blueTotalGold + blueTotalExperience + blueGoldD
iff + blueExperienceDiff, data=lol_train) %>%
  step_scale(all_predictors()) %>%
  step_center(all_predictors())
```

K-fold Cross Validdation

We'll stratify our cross validation on `Win` and use 10 folds to perform stratifies cross validation. The advantage of k-fold cross validation is it provides a more accurate estimate of the model's performance than simply using a single train-test split. By repeatedly training and testing the model on different subsets of the data, k-fold cross-validation reduces the risk of overfitting, which occurs when a model is too complex and fits the training data too closely, resulting in poor performance on new, unseen data.

Hide

```
lol_folds <- vfold_cv(lol_train, v = 10, strata = Win)
```

Model Building

Now let's build our models! I chose to construct 4 models: a logistic regression model, a k-nearest neighbor model, an LDA model and an elastic net model. Besides, since we are dealing with a classification problem, I decided to use ROC AUC as my metric of performance. A higher ROC AUC value indicates that the model has a better ability to correctly classify instances. Now let's begin the process!

1. Set up the model by specifying the type of model, engine, the tuning parameters and the model (classification in our case) and the corresponding workflows.

Hide

```r
# logistic regression model
log_mod <- logistic_reg() %>%
  set_mode("classification") %>%
  set_engine('glm')

log_wkflow <- workflow() %>%
  add_model(log_mod) %>%
  add_recipe(lol_recipe)

# LDA model
lda_mod <- discrim_linear() %>%
  set_mode("classification") %>%
  set_engine("MASS")

lda_wkflow <- workflow() %>%
  add_model(lda_mod) %>%
  add_recipe(lol_recipe)

# knn model
knn_mod <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("classification") %>%
  set_engine("kknn")

knn_wkflow <- workflow() %>%
  add_model(knn_mod) %>%
  add_recipe(lol_recipe)

# Elastic net model
elastic_mod <- logistic_reg(mixture = tune(),
                               penalty = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

elastic_wkflow <- workflow() %>%
  add_model(elastic_mod) %>%
  add_recipe(lol_recipe)
```

2. Create tuning grids to decide the ranges of parameters we want to tune and the number of levels.

Hide

```
# Logistic regression
# No grid because we're not tuning any parameters

# LDA
# No grid because we're not tuning any parameters

# Knn
knn_grid <- grid_regular(neighbors(range = c(1, 20)), levels = 10)

# Elastic net
en_grid <- grid_regular(penalty(range = c(-5,5)), mixture(range=c(0,1)), levels = 1
0)
```

3. Tune the models

Hide

```
# Logistic regression
# No tuning

# LDA
# No tuning

# Knn
knn_tune <- tune_grid(knn_wkflow,
                      resamples = lol_folds,
                      grid = knn_grid)

# Elastic net
elastic_tune <- tune_grid(elastic_wkflow,
                          resamples = lol_folds,
                          grid = en_grid)
```

4. Collect the best ROC AUC values of the models using `show_best()` and save them for comparison.

Hide

```
# Logistic regression
log_fit <- fit_resamples(log_wkflow, resamples = lol_folds)
log_roc_auc <- show_best(log_fit, metric = "roc_auc", n = 1)

# LDA
lda_fit <- fit_resamples(lda_wkflow, resamples = lol_folds)
lda_roc_auc <- show_best(lda_fit, metric = "roc_auc", n = 1)

# knn
knn_roc_auc <- show_best(knn_tune, metric = "roc_auc", n = 1)

# Elastic net
elastic_roc_auc <- show_best(elastic_tune, metric = "roc_auc", n = 1)
```

Model Results

Let’s compare the performance of our models using a simple table!

Hide

```
# Creating a tibble of the ROC_AUC values of all models
comparison_table <- tibble(Model = c("logistic Regression", "LDA", "K Nearest Neigh
bors", "Elastic Net"), ROC_AUC = c(log_roc_auc$mean, lda_roc_auc$mean, knn_roc_auc
$mean, elastic_roc_auc$mean))

comparison_table <- comparison_table %>% arrange(-ROC_AUC)

comparison_table
```

```
## # A tibble: 4 × 2
##   Model               ROC_AUC
##   <chr>                 <dbl>
## 1 Elastic Net           0.809
## 2 logistic Regression   0.808
## 3 LDA                   0.808
## 4 K Nearest Neighbors   0.776
```

Model Autoplots

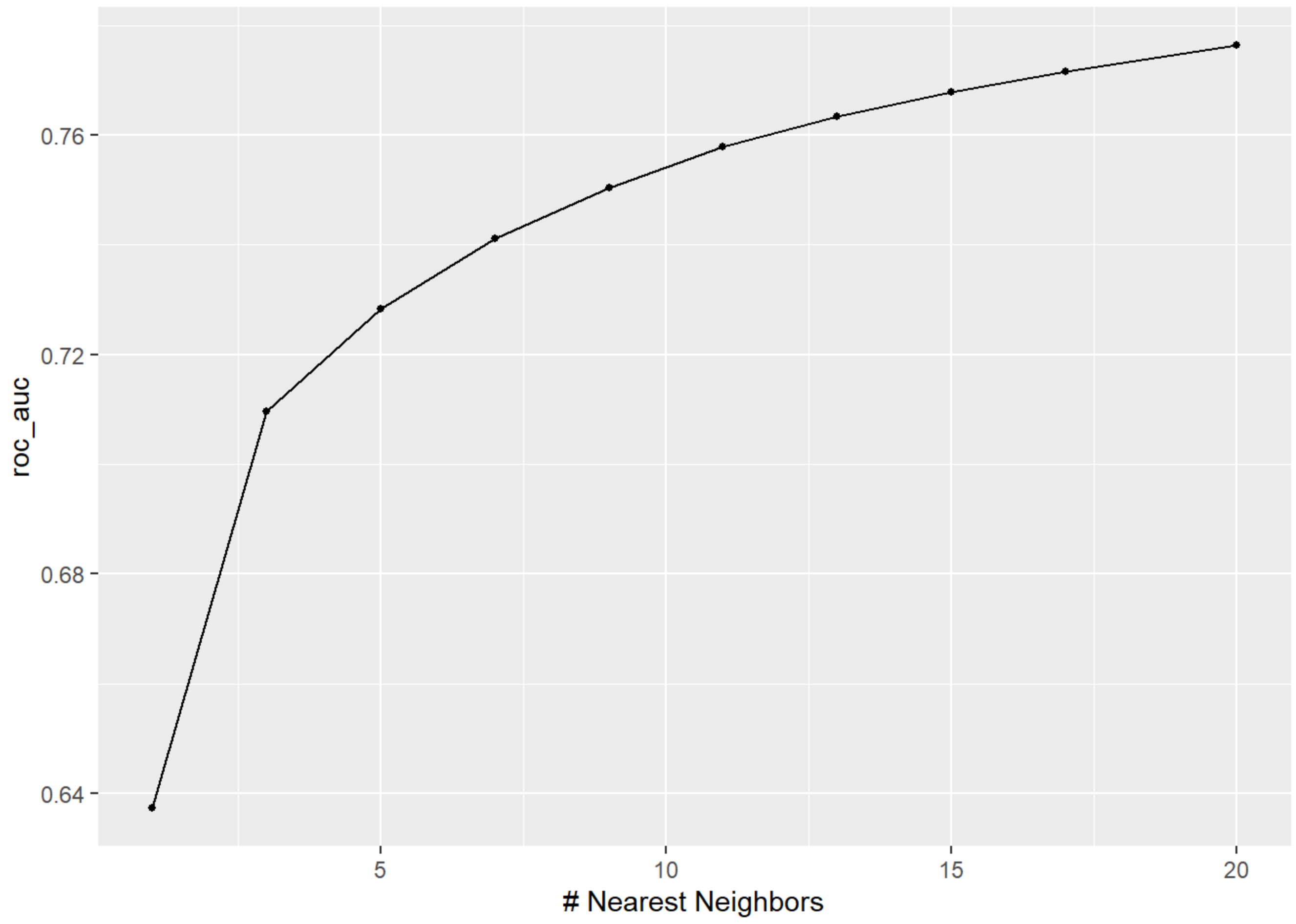
Let’s visualize the impact of the tuning parameters on the performance of each model.

K-Nearest Neighbor

We can notice that the accuracy of model increases with increase in number of neighbors. The highest ROC AUC is near 0.8, occurring at 20 neighbors.

Hide

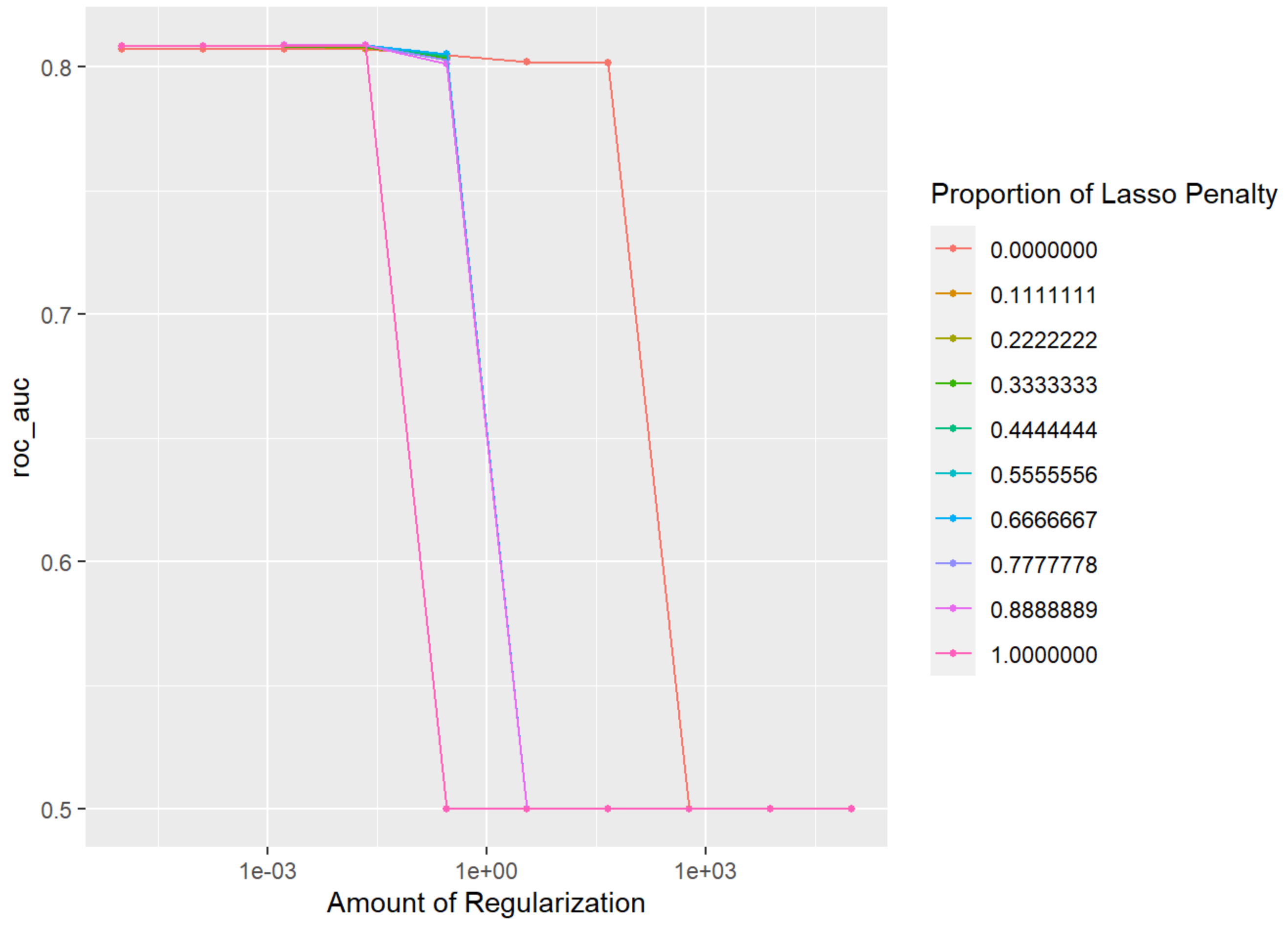
```
autoplot(knn_tune, metric = "roc_auc")
```



Elastic Net

Hide

```
autoplot(elastic_tune, metric = "roc_auc")
```



It seems that our model performs best at low regularization and high proportion of lasso penalty, it suggests that our model is able to achieve good performance by using a sparse set of predictors, where some of the coefficients are set to zero. This means that the model is effectively selecting only a small number of the most important features for the prediction task, and ignoring the rest.

Best Model

From the tibble, we conclude that the elastic net model is the winning. Let's look at which are the value of the parameters of our best elastic net model.

Hide

```
elastic_roc_auc
```

```
## # A tibble: 1 × 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  0.0215   0.667 roc_auc binary     0.809    10 0.00419 Preprocessor1_Model064
```

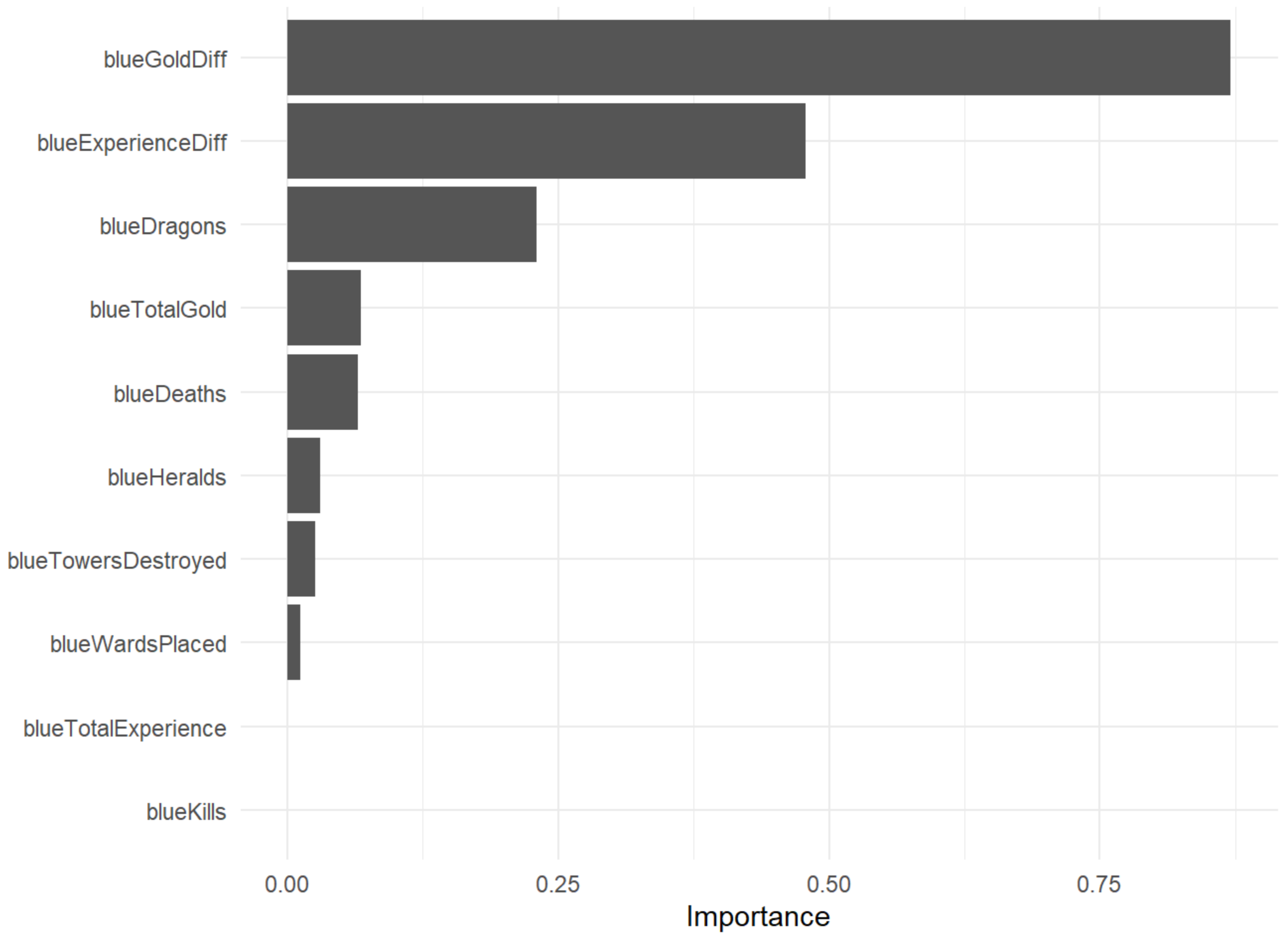
So our best model is an elastic net model with `penalty = 0.02154435` and `mixture = 0.6666667` !

Variable Importance

Since my intention for this project is to know what contributes the most in the early game to winning, we can construct a variable importance plot to see which variables are the most crucial in predicting the outcome.

Hide

```
elastic_final_fit <- finalize_workflow(elastic_wkflow, elastic_roc_auc)
elastic_final_fit <- fit(elastic_final_fit, data = lol_train)
elastic_final_fit %>% extract_fit_parsnip() %>%
  vip() + theme_minimal()
```



From the plot, we see that lead in gold and experience for the blue team is the most important factor, which makes total sense since huge gold and experience difference simply means the blue team champions are much stronger so they will most likely to win every team fights and eventually enjoy victory.

Fitting Best Model to Test

Now let's finally put our best model to use! We will test its performance on the testing data.

Hide

```
augment(elastic_final_fit, new_data = lol_test) %>%
  roc_auc(Win, .pred_No)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.812
```

We have a ROC AUC of 0.8125 which is slightly less than that on the training data but is totally acceptable. We have a decent model!

Conclusion

Throughout this project, we have explored, analyzed, and filtered our data and built several models to predict whether the blue team will win a game based on their performance in the first ten minutes of the game. After training and testing, we obtained an elastic net model which did the best at predicting the outcome of a typical lol game.

As for future improvements, one method that could potentially improve the prediction accuracy is using a neural network model. Neural networks have been shown to be effective in handling complex and nonlinear relationships between input features and output targets, which is particularly useful for gaming data where there are many factors that can influence the outcome of a game. Neural networks can also handle large amounts of data and can learn from patterns and trends in the data, which can lead to more accurate predictions. However, neural networks can be more complex to build and may require more computational power compared to some of the models we've already explored. In addition, during many League of Legends professional games, I noticed that they have an artificial intelligence that monitors the game and can produce prediction of the chances of winning of the blue and the red team at a certain time stamp based on their performance before that time. I am very curious about how it works so I will do some further research on that and try to mimic the model if I could.

I also learned a lot by doing this project. This is probably the first time that I obtained data and applied machine learning techniques to predict an outcome all by myself, which was a great opportunity for me to develop my research and data analysis skills and accumulate experience in machine learning. I found myself become more comfortable working with data which will help me a lo in future projects or career.

Finally, I am glad that I was able to predict the outcome of a lol game with decent accuracy. Now I know what should do in the early game in order to win!