League of Data: Predicting Game Outcome and Community Growth

Kirkland Water, Fall 2019

Don Agiro, Robert Hutchinson

Abstract

As esports become more popular across the globe, the value of predictive models and data analysis in the field has skyrocketed. In this paper, we explore the predictability of League of Legends' matches, with the goal to determine which team will win a given match. Predicted outcome will be determined through key factors such as secured objectives, team damage, and minion kills. Using stepwise logistic regression, the best fit model was chosen based off the lowest AIC value. Furthermore, we will model and forecast subscriber growth in the League of Legends subreddit. Using monthly data spanning seven years, we built and selected the ARIMA model with the lowest AIC. Using this model, the predicted subscriber count a year from now is 4.6 million, nearly a 33% growth in users. All data cleaning and transformation was conducted in Excel, and models and forecasts were calculated in R.

Introduction

When developers at Riot Games published League of Legends in 2009, the idea of a global sensation was just a farfetched dream. Yet ten years later, the game sits at the top of the leaderboards, with a massive following of over 120 million players.

The game's objective is simple: the first team to destroy the opponent's nexus – a structure at the heart of the enemy team's base – wins. However, getting there is not easy. Each team is comprised of five unique characters called champions, and they must simultaneously defend their own base while attacking the opponent's defenses. Key map objectives include towers (static structures that defend each team's base) and monsters (neutral enemies that spawn throughout the map), both of which provide gold when defeated. With gold, players can buy items to increase their stats, such as health and damage, which increase the likelihood of winning brawls against enemy champions.

Due to this feedback loop, it is understandable that matches can potentially snowball and favor the team that starts off on the right foot. For instance, if the blue team captures the first objective, they will receive an influx of gold, allowing them to not only push farther into the red team's base but also strengthen their own champion's stats. This will likely give the blue team an advantage during future teamfights, accelerating the rate of their growth.

Our goal will be to predict match outcome given key information, such as gold difference and objectives taken. The final model should be able to predict match outcome *before* the match is complete, relying on data available throughout the game. As a result, the model's prediction accuracy is expected to increase as game time increases.

Data

The original data was pulled using Riot Game's API and includes observations on 1000 challenger-ranked games[1]. Players compete on a ranked ladder, which assigns players a title based on their skill level. Challenger is the highest tier, followed by Master, Diamond, and Platinum, respectively[2]. Therefore, this dataset is comprised of extremely high-skill matches, and teams often contain at least one Challenger player. Approximately .08% of players are Challenger or Master, 3.5% are Diamond, and 15% are Platinum. The remaining players are ranked either Gold or below and will not appear in our data.

To prepare for the modeling phase, the data was first cleaned and transformed. Duplicate rows were removed, leaving 695 of the initial 1000 observations. Since the

[1] Data was retrieved from Kaggle, an online community where users explore and publish free-to-use datasets

[2] As of late 2018, a Grandmaster tier was added between Challenger and Master. However, since our dataset was retrieved before this change, we will not account for this.

average League of Legends game lasts for 28 minutes, we removed all sub-10 minute matches from our dataset. Longer matches are not typically considered outliers, as these games are more common and provide valuable information, unlike their shorter counterparts.

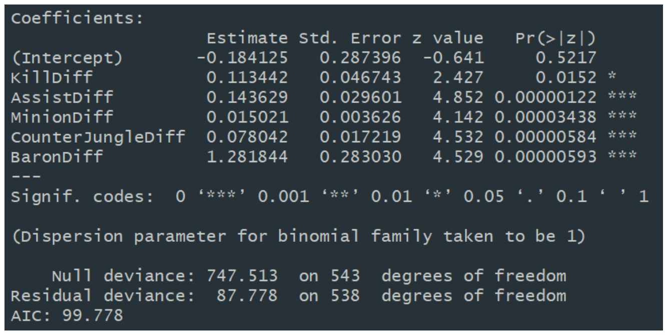
The downloaded dataset contained 551 variables, but the majority of these were removed due to multicollinearity. These variables did provide any additional benefit – for instance, total team gold was dropped because this information is captured through other variables such as minion[3], tower, and champion kills. The cleaned dataset consists of 12 predictor variables including vision score[4], team damage, and total kills. These variables trend upward as the game goes on, so transformed variables were created. For example, the columns 'blue team assists' and 'red team assists' were combined into one variable, calculated as the difference between the blue and red team's values. Other variables, such as 'first team to destroy a tower' are binary and were not transformed. The y-variable, 'win', is also binary – a '1' is used for a blue team victory, and a '0' for red. A detailed list of variables and definitions is provided in Appendix A.

Logistic Methods

Using logistic regression, we can model the win probability of a given team. Explanatory variable estimates are measured in odds ratios and the predicted match outcome value lies between 0 (blue loss) and 1 (blue win). Match outcome is affected by hundreds of factors, but these complexities can be categorized into two buckets: quantitative and qualitative data. While the bulk of our dataset is quantitative, competitive rank is a qualitative factor based off player skill. Even though ladder rank is both a calculated and ordered field, there are still differences in player skill within the same tier. Therefore, we began by testing the significance of player skill on match outcome, using two separate methods.

In the first test, we directly input each player's skill level, using four tiers defined as 'Challenger', 'Master', 'Diamond', and 'below'. In the second test, we created a new variable calculated as the difference between the team's player's skills. However, in both models, player skill did not appear as a significant variable. Upon further investigation, these results began to make sense. When creating a match, the League of Legend's algorithm auto-balances teams and assigns similarly skilled players to each team. Therefore, player skill acts as a constant variable, and other factors such as damage and towers impact game outcome much more significantly. This regression model summary is provided in Appendix B.

Moving forward, we made a few changes to our dataset. The previous regression was run on a set of late-game variables, meaning even a beginner player could likely predict match outcome. Since the goal is to create a model more accurate than humans, we pivoted and used a more robust 12 variable dataset. Then, we partitioned the data into two sets for training and validation. After running a stepwise logistic regression on the train data, the model with the lowest AIC was chosen.



```
Coefficients:
                   Estimate Std. Error z value   Pr(>|z|)
(Intercept)       -0.184125   0.287396  -0.641     0.5217
KillDiff           0.113442   0.046743   2.427     0.0152 *
AssistDiff         0.143629   0.029601   4.852 0.00000122 ***
MinionDiff         0.015021   0.003626   4.142 0.00003438 ***
CounterJungleDiff  0.078042   0.017219   4.532 0.00000584 ***
BaronDiff          1.281844   0.283030   4.529 0.00000593 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 747.513  on 543  degrees of freedom
Residual deviance:  87.778  on 538  degrees of freedom
AIC: 99.778
```

As seen above, 5 of the 12 explanatory variables are statistically significant in predicting match outcome. Since the explanatory variables are all calculated as the difference between blue and red team values, it makes sense that all regression estimates are positive. For every kill that blue team has over red team, blue team's log odds chance of winning increases by 0.113. Using a specific example, in a game where KillDiff = -6, AssistDiff = 4, MinionDiff = -7, CounterJungleDiff = 0, and BaronDiff = 1, the calculated blue team win probability is 70%.

Next, we checked the assumptions of our model by running diagnostics. Unlike linear models, logistic models do not require homoscedasticity and the residuals do not need to be normally distributed. However, there are still a few assumptions that should be met. To begin with, our sample size n should meet the requirement

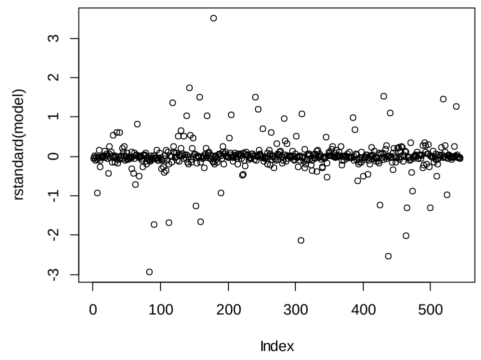
$$n = 10k/p$$

where k is the number of independent variables and p is the expected probability of the least frequent outcome. Plugging in our parameters, the calculated minimum sample size is

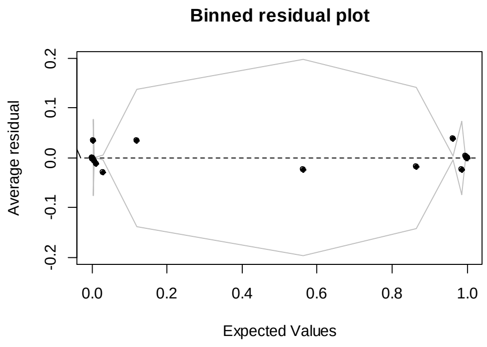
[3] Red and blue minions spawn from their respective team's nexus and drop gold upon death.

[4] Portions of the map are covered with fog and impair team vision of the affected area. Players can purchase temporary vision-granting items to clear up the fog.

100, and we provided over 500 data points. As seen below, the standardized residuals of our model hover around 0, implying that the model is quite accurate in making predictions.

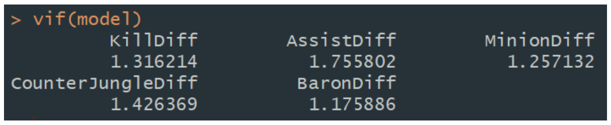


To further assess fit, a custom binned residual plot was used, since traditional plots offer less benefit with logistic models.



The gray standard error bands are expected to contain 95% of the observations. While the bunched variables on the right are hard to discern, the majority of our fitted values fall within the bounds.

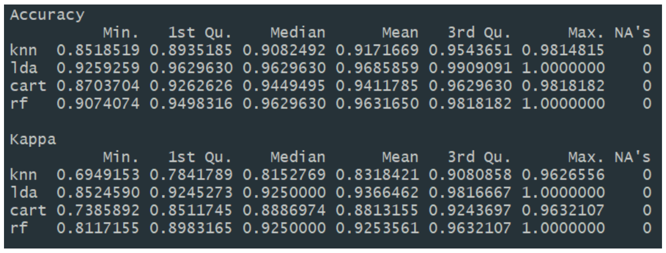
Furthermore, we checked for multicollinearity between our variables by using the variable inflation factor test (VIF). Using a VIF threshold of 3, we can conclude that no variables in our model exhibit multicollinearity. This step is crucial, as this positive result is one of the few requirements of a reliable logistic model.



```
> vif(model)
         KillDiff          AssistDiff          MinionDiff
         1.316214            1.755802            1.257132
CounterJungleDiff           BaronDiff
         1.426369            1.175886
```

Finally, we can check overall model accuracy by making predictions on our validation dataset. Using a threshold of 50% – meaning any prediction value greater than 50% counts as a blue team win – the calculated accuracy value of the model was 94%.

Out of curiosity, we wanted to explore if a machine learning model would perform better than our lowest AIC logistic model. Using the *caret* packsage in R, we cross-trained on four separate machine learning algorithms and chose the model with the highest average accuracy. A detailed description of these algorithms is provided in Appendix C.



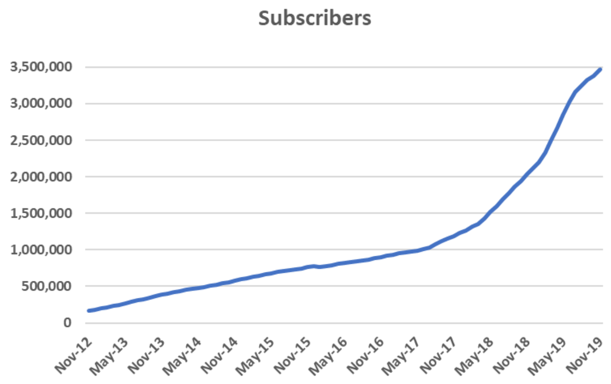
```
Accuracy
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
knn  0.8518519 0.8935185 0.9082492 0.9171669 0.9543651 0.9814815    0
lda  0.9259259 0.9629630 0.9629630 0.9685859 0.9909091 1.0000000    0
cart 0.8703704 0.9262626 0.9449495 0.9411785 0.9629630 0.9818182    0
rf   0.9074074 0.9498316 0.9629630 0.9631650 0.9818182 1.0000000    0

Kappa
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
knn  0.6949153 0.7841789 0.8152769 0.8318421 0.9080858 0.9626556    0
lda  0.8524590 0.9245273 0.9250000 0.9366462 0.9816667 1.0000000    0
cart 0.7385892 0.8511745 0.8886974 0.8813155 0.9243697 0.9632107    0
rf   0.8117155 0.8983165 0.9250000 0.9253561 0.9632107 1.0000000    0
```

Linear discriminant analysis (LDA) came out on top, with the highest accuracy and kappa values. The LDA model's accuracy under the validation dataset reached 97.8% accuracy, a slight improvement upon the logistic model.

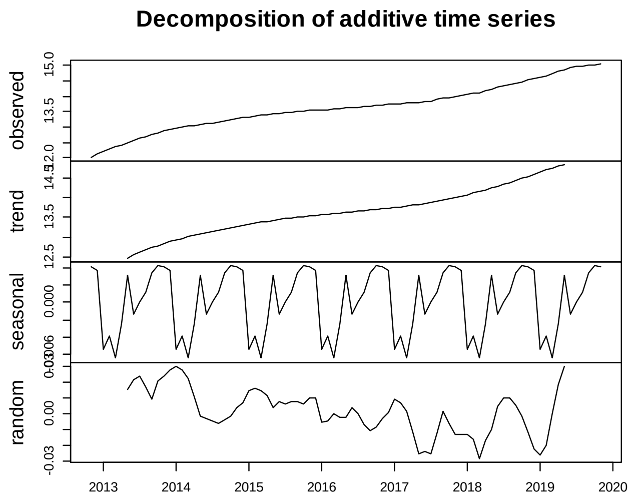
Overall, these models tell us that given a few key values, the outcome of a match can be predicted with a very high accuracy. But there are some flaws to be addressed. For instance, although our revised dataset removed several late-game variables, the data was still composed of post-game values. As a result, the model will be less accurate in the early-game, and some values will be unavailable altogether. For instance, the BaronDiff variable is only meaningful after 20 minutes into the game, since that is when Baron Nashor spawns. Nonetheless, these models provide helpful insight into what really matters in a match, and the benefit of analysis League of Legend's competitive scene continues to grow.

Time Series

Next, we explored the growth of r/leagueoflegends, the largest online gathering of League of Legends players. Built on Reddit's community system, the subreddit was established in January of 2010 and has since amassed over 3.5 million subscribers.



To better understand the subreddit's growth and make forecasts, we used a dataset consisting of 85 monthly observations, each representing a given month's total subscriber count. The exponential growth seen in recent years has skewed the overall distribution, so we first log-transformed the subscriber count variable. Then, by applying an additive decomposition on our transformed data, we can plot each component of our series.



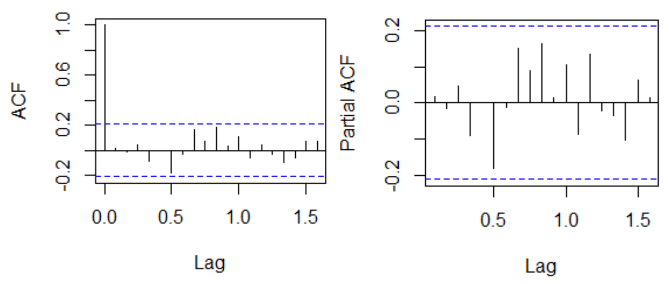
This breakdown tells us that our data follows a continuous positive trend and appears to exhibit no seasonality. This narrows down our model type into one of two types – ARIMA and ARMA. The difference between the models depends on the series' stationarity, which can be assessed through the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) unit root test. The calculated p-value was less than 0.01, so we can reject the null hypothesis that the series is stationary.

```
        KPSS Test for Level Stationarity

data:  subscribers
KPSS Level = 2.1028, Truncation lag parameter = 3, p-value = 0.01
```

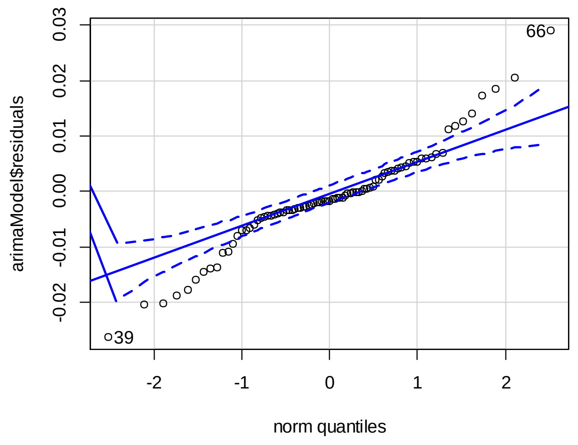
The best fit model was chosen based on the lowest AIC, resulting in an ARIMA (1,2,0) model, shown below.

```
Series: subscribers
ARIMA(1,2,0)

Coefficients:
          ar1
      -0.2362
s.e.   0.1079

sigma^2 estimated as 0.00008304:  log likelihood=274.4
AIC=-544.81   AICc=-544.66   BIC=-539.97
```

However, before we can reliably use this model to make forecasts, we must first conduct diagnostics for the fundamental model assumptions. By calculating the ACF and PACF of our residuals, we can confirm that our lowest-AIC model is accurate.

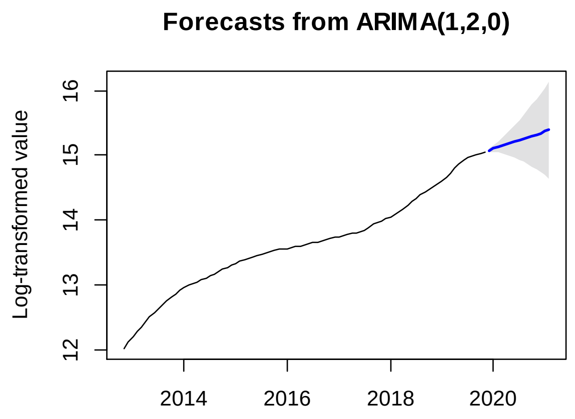


The graph below shows the Q-Q plot of our model's residuals.



Although the middle portion of the Q-Q plot looks normal, the left and right sides are outside the bounds. This indicates that our data is fat-tailed, and in future analyses a t-distribution may be a better fit for our residuals.

Finally, after confirming the validity of our time series model, we can make forecasts of subreddit growth. Using a confidence level of 99.5%, we predicted the next fifteen months of growth. These values are log-transformed, meaning the actual subscriber count needs to be calculated with e^x. Exact values are provided in Appendix D. The forecasted value a year from now is 4.6 million members.



Conclusion

League of Legend's success has spread worldwide, boosting it to the peak of global competitive gaming. The recent Worlds 2019 final match set another record, achieving 1.75 million concurrent viewers on Twitch. This paper explored the predictability of League of Legends matches and the growth of the game's subreddit.

A plethora of variables impact the performance of teams and individual players. Even 20 minutes into a game, it is not always clear which side will win. Therefore, using a dataset of nearly 700 ranked games, we developed and tested a regression model predicting match outcome. The final logistic model's significant variables included champion kills and assists, minion and monster score, as well as objective captures. Next, to forecast the growth of the League of Legends' subreddit, we used a time-series ARIMA model. This gave us a reliable estimate into future subscribers, and the results are in line with the current expansion of the game's player base.

Regression and time-series analysis are the basic stepping stones into more advanced research techniques, and this project provided our group a great study into the benefits of analyses such as these. All analysis in this paper was completed using R, and the full script is provided in Appendix E. In the future, we would like to explore the combination of time-series data with win probability. Here, we would look at what variables impact match outcome, and at what point in the game is each component most crucial. However, this analysis may prove difficult due to the lack of existing data and the complex process to record new data. Furthermore, a more in-depth look at various machine learning models would be both educational and applicable, as more industries implement these techniques into their daily processes.

References

Assumptions of Logistic Regression. (n.d.). Retrieved October 29, 2019, from https://www.statisticssolutions.com/assumptions-of-logistic-regression/.

How Many People Play League of Legends in 2019? (n.d.). Retrieved October 25, 2019, from https://rankedkings.com/blog/how-many-people-play-league-of-legends.

Sato, G. (2019, June 18). League of Legends - Challenger Ranked Games. Retrieved October 24, 2019, from https://www.kaggle.com/gabisato/league-of-legends-challenger-ranked-games.

This esports giant draws in more viewers than the Super Bowl, and it's expected to get even bigger. (2019, April 14). Retrieved October 25, 2019, from https://www.cnbc.com/2019/04/14/league-of-legends-gets-more-viewers-than-super-bowlwhats-coming-next.html.

Webb, J. (2017, September 3). Course Notes for IS 6489, Statistics and Predictive Analytics. Retrieved October 25, 2019, from https://bookdown.org/jefftemplewebb/IS-6489/logistic-regression.html#logistic-regression-coefficients-as-odds-ratios.

Appendix

Appendix A – Clean Data Breakdown

Win – the variable to be predicted, is ‘1’ if blue team wins, ‘0’ if red team wins

FirstBlood – whichever team claimed the first kill of the match, ‘1’ for blue team, ‘0’ for red team

FirstTower– whichever team claimed the first tower of the match, ‘1’ for blue team, ‘0’ for red team

KillDiff – blue team champion kills minus red team champion kills

AssistDiff – blue team champion assists minus red team champion assists

DamageDiff – blue team total damage dealt minus red team total damage dealt

TrueDamageDiff – blue team total true damage dealt minus red team total true damage dealt (True damage ignores armor, magic resistance, and other forms of damage reduction.)

HealDiff –blue team total heals minus red team total heals

CCTimeDiff – blue team total crowd control time dealt minus red team total crowd control time dealt, measured in seconds (Crowd control is any form of debuff that diminishes an enemy’s control over itself, such as a slow, stun, or silence.)

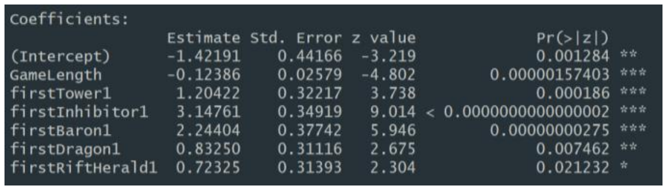
MinionDiff – blue team minion kills minus red team minion kills

CounterJungleDiff – blue team’s counter jungle monster kills minus red team’s counter jungle monster kills (Counter jungling occurs when a player hunts monsters in the opposing team’s jungle, depriving the enemy team of buffs, gold, and experience.)

DragonDiff – blue team dragon kills minus red team dragon kills (The dragon is a neutral monster that provides gold, experience and buffs to the team that slays it, and will respawn after five minutes. It spawns 5 minutes into the game.)

BaronDiff – blue team baron kills minus red team baron kills (Baron Nashor is a neutral monster that provides gold, experience and buffs to the team that slays it, and will respawn after six minutes. It spawns 20 minutes into the game.)

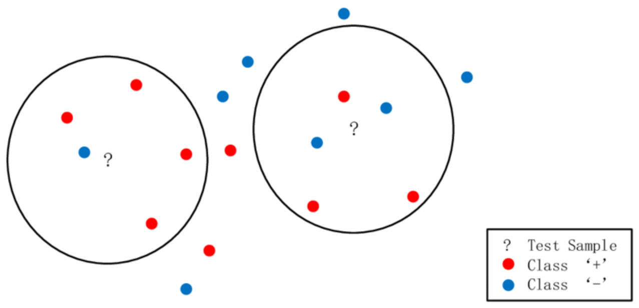
Appendix B – Initial Regression Output



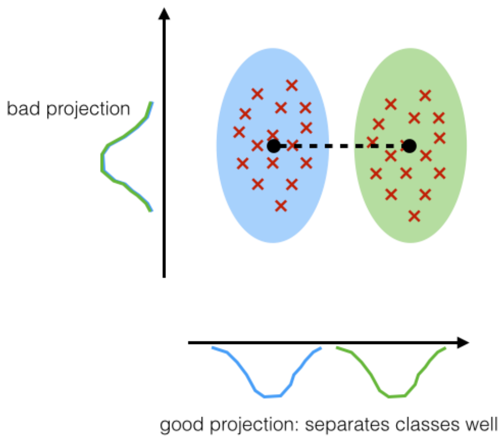
```
Coefficients:
                  Estimate Std. Error z value             Pr(>|z|)
(Intercept)       -1.42191    0.44166  -3.219             0.001284 **
GameLength        -0.12386    0.02579  -4.802         0.00000157403 ***
firstTower1        1.20422    0.32217   3.738             0.000186 ***
firstInhibitor1    3.14761    0.34919   9.014 < 0.0000000000000002 ***
firstBaron1        2.24404    0.37742   5.946         0.0000000275 ***
firstDragon1       0.83250    0.31116   2.675             0.007462 **
firstRiftHerald1   0.72325    0.31393   2.304             0.021232 *
```

Appendix C – Machine Learning Models

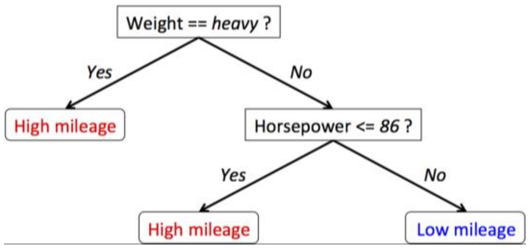
K-nearest-neighbors – This model assumes that similar data points exist near each other, such as in the image below. It finds the distance between a query and all points in the data, then counts the number K points nearest to the defined query. The KNN algorithm runs several times then votes for the K-value with the least errors.



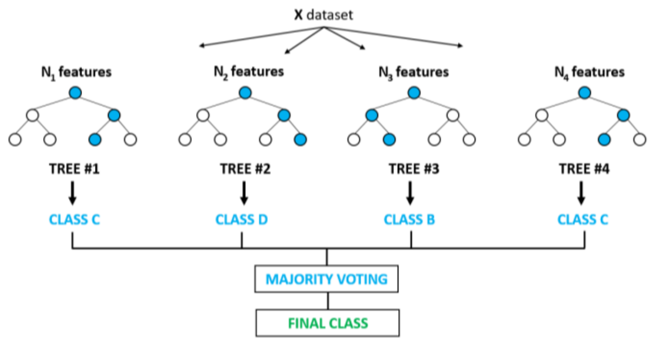
Linear Discriminant Analysis – Based off of Fisher’s model, LDA creates a function that maximizes variance between groups and minimizes variance within groups. Effectively minimizing group overlap, this method is often used in classification problems.



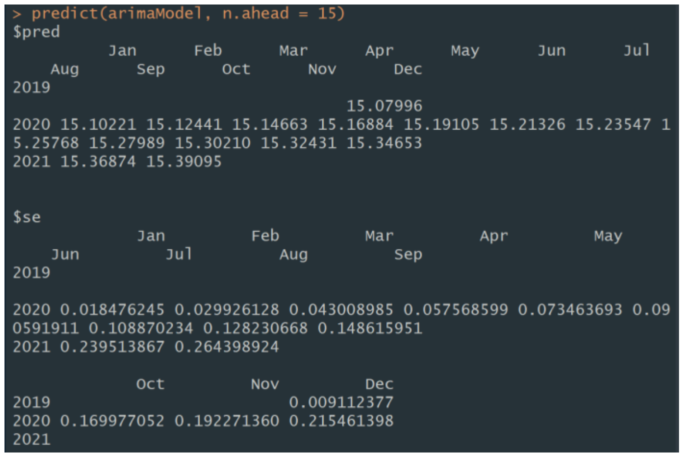
Classification and Regression Trees – The CART model utilizes decision trees, which classify data points by segmenting it into different sets depending on how it reacts to different nodes. Data is sent through a flowchart structure – first through a root node, then down to internal nodes where decisions are made, and finally to a leaf node with a classification result.



Random Forest – A more complex implementation of decision trees, RF has a few main differences from CART. First, it tests each data point using random selection, whereas CART uses the entire dataset. This means RF utilizes more decision trees and averages the result. Furthermore, RF randomly selects a subset of features from the dataset to test. Ideally, this reduces correlation between trees, but also results in longer computation times.



Appendix D – Forecasted Subscriber Growth Values



```
> predict(arimaModel, n.ahead = 15)
$pred
          Jan      Feb      Mar      Apr      May      Jun      Jul
    Aug      Sep      Oct      Nov      Dec
2019
                                   15.07996
2020 15.10221 15.12441 15.14663 15.16884 15.19105 15.21326 15.23547 1
5.25768 15.27989 15.30210 15.32431 15.34653
2021 15.36874 15.39095


$se
             Jan         Feb         Mar         Apr         May
    Jun         Jul         Aug         Sep
2019

2020 0.018476245 0.029926128 0.043008985 0.057568599 0.073463693 0.09
0591911 0.108870234 0.128230668 0.148615951
2021 0.239513867 0.264398924

             Oct         Nov         Dec
2019                         0.009112377
2020 0.169977052 0.192271360 0.215461398
2021
```

Appendix E – R Code

```r
library(dplyr)
library(plyr)
library(read_csv)
library(readr)
library(MASS)
library(caret) # Machine Learning
library(car) # VIF and QQ plot
library(tidyr)
library(tseries) # Unit root test for time series
library(forecast) # ARIMA


### LOGISTIC MODEL
# Supress scientific notation
options(scipen = 4)
# Import main data and summoner spell data
file <- "C:/Users/aaron/Documents/LeagueData/ChallengerNewEditFinal.csv"
data <- read_csv(file)
data <- data %>% drop_na() # Remove NA rows

# Convert binary variables to factor
data$win <- as.factor(data$win)
data$firstBlood <- as.factor(data$firstBlood)
data$firstTower <- as.factor(data$firstBlood)

# Partition data into two sets, training and validation
set.seed(61)
trainSamples <- createDataPartition(data$win, p = .8, list = F)
trainData <- data[trainSamples,]
validationData <- data[-trainSamples,]

# Create Model
model <- glm(win~., family = binomial(link = 'logit'), data = trainData) %>%
  stepAIC(trace = F)
summary(model)
plot(rstandard(model)) # plot the standardized residuals

# Check model accuracy
validationData$probability <- predict(model, validationData, type = "response")
# If calculated blue team win % > 50, set to 1
validationData <- validationData %>% mutate(prediction =  1*(probability > 0.5), actual = 1*(win == "1"))
validationData <- validationData %>% mutate(accuracy = 1*(prediction == actual))
sum(validationData$accuracy)/nrow(validationData) # Calculate accuracy
View(validationData)
vif(model)

binnedplot(fitted(model),
           residuals(model, type = "response"),
           nclass = NULL,
           xlab = "Expected Values",
           ylab = "Average residual",
           main = "Binned residual plot",
           cex.pts = 0.8,
           col.pts = 1,
           col.int = "gray")

# Machine learning models
# Run algorithms using 10-fold cross validation
control <- trainControl(method = "cv", number = s10)
metric <- "Accuracy"

fit.knn <- train(win~., data = trainData, method = "knn", metric = metric, trControl = control)
fit.lda <- train(win~., data = trainData, method = "lda", metric = metric, trControl = control)
fit.cart <- train(win~., data = trainData, method = "rpart", metric = metric, trControl = control)
fit.rf <- train(win~., data = trainData, method = "rf", metric = metric, trControl = control)
results <- resamples(list(knn = fit.knn, lda = fit.lda, cart = fit.cart, rf = fit.rf))
summary(results)

predictions <- predict(fit.lda, validationData)

confusionMatrix(predictions, validationData$win)
```

```r
# TIME SERIES MODEL
file2 <- "C:/Users/aaron/Documents/LeagueData/RedditSubscribers.csv"
subscribers <- read_csv(file2)
subscribers <- subscribers[,3] # Use log-normalized data
# Transform data into time series data
subscribers <- ts(subscribers, start = c(2012,11), frequency = 12)

# Use additive since data is log-normalized
components <- decompose(subscribers, type = 'additive')
plot(components)

# Unit root test for stationarity
# Null hypothesis is stationarity, small p-value rejects null
kpss.test(subscribers, null = "Level") # Data appears to be non-stationary

# Auti-ARIMA model with lowest AIC
arimaModel <- auto.arima(subscribers)
arimaModel

qqPlot(arimaModel$residuals)

predict(arimaModel, n.ahead = 15)

acf(arimaModel$residuals)
pacf(arimaModel$residuals)

forecast <- forecast(arimaModel, h = 15, level = c(99.5)) # add axis
plot(forecast, ylab = "Log-transformed value")
```