Statistical Analysis and Feature Selection of League of Legends Player In-Game Stats

By Jay Choi

Jay1.Choi@ryerson.ca

<https://github.com/torontosj/League-of-Legends-Capstone>

**Introduction**

League of Legends is a multiplayer online battle arena (MOBA) game played by millions of players around the world. 32 million unique active users participate every month with over 12 million daily active players. Users log over 1 billion hours of play per month globally.[[1]](#endnote-1) Attempting to pinpoint the specific factors of the game’s growth can be a convoluted task and cannot be answered with absolute certainty. However, a factor which has contributed to the growth of the game is the emergence of professional leagues in which the top players in a region compete in structured league and tournament formats. The publisher of the game, Riot Games, have made efforts to validate League of Legends as a competitive sport through these leagues and tournaments.

There are several features of any game that must be satisfied to be considered within the same regard as traditional sports. The end result, or probability of success, should be determined by inherent and acquired skill. Luck, circumstance, and happenstance often affects the end result of real sports games. However, a team or participant of greater skill should win more often than the less skilled opponent if the game instance is enumerated over a large sample size with the same controls.

This project is an attempt to quantify the in game registered statistics of events which occur in the game. Due to the sheer volume of games played as well as the limitations of the dataset extraction process, a specific subset criteria was defined prior to beginning the analysis. The dataset was curated by accessing the game publisher’s API using Python scripts, an API wrapper and storing the JSON files into a MySQL database. The datasets were then exported into comma separated values format. Initial exploratory analysis began with understanding which features were available and its returned values. The returned values of a single match include various events from within the game that correspond to individual player and team events. The recorded events included Boolean, continuous and categorical values all representing a unique trait about a single match. Each match included 10 participants with two designated teams determined by a unique match making system which matches users by MMR. MMR is a hidden Elo rating system which calculates the relative skill levels of players in a competitor versus competitor games such as chess.[[2]](#endnote-2) This MMR system accounts for past performance and uses a hidden algorithm used by game publishers in an attempt to match users with other users of the same relative skill. The specific criteria for determining which matches were extracted will be described further in the dataset section. The primary task then focused on using a general linear model for use in feature selection for multivariate regression.

**Dataset Curation Process and Description**

**Step 1 - Discovering Riot Games API**

Any registered account can access the Riot Games API(application program interface) after accepting the API usage guidelines outlined by the publisher of the game. The API is made accessible for application developers interested in the game to create websites or services that enhance users’ experience of the game. Some of the existing services include reviewing a user’s match history, general patch trends examining strength of champions, and current game info. The full list of returned variables can be found in the MatchEndpoint document in the repository.

**Step 2 - Using R to Access API**

Extensive efforts were made to use R to create an appropriate dataset using R. Initial success of access the API using the packages httr and RCurl were tempered when it was realized that the returned JSON files were extensively nested. The size and complexity of the JSON file made it difficult to iterate the process of extracting and storing the match data.

**Step 3 - Cassiopeia- Python Wrapper**

A wrapper for a Rest API can be defined as code that is written by a programmer to automate required processes for executing specific functions of an API. Cassiopeia is a Python 3 wrapper written by Rob Rua[[3]](#endnote-3). Multiple wrappers exist for the Riot Games API due to the popularity of the game. They are maintained by users who have created a website which use the wrappers. Pycharm IDE and the Anaconda distribution was used to access the Cassiopeia library.

Creating a sufficiently robust dataset required several steps due to the design of the Riot API endpoints. Based on comments from the official forums, there are several different approaches to acquiring a list of matches to call from the API. The approach in which to acquire the matchlist, in some ways, is the initiation of the subset selection because a specific method can discriminate the average MMRs of the players involved in the matches.

The method used to curate the match database is described in the flowchart below.

The Challenger tier represents the top 200 players of a given region. The Korean region was used because it is perceived to contain the highest skilled players in the world. The match histories of these players were acquired using a specified start date of September 21st 2015 which represents the day after patch 5.18. Patches are released periodically by Riot Games to modify game parameters. It is done to tune or balance the game in an effort to remove what is considered by Riot to be exploitive strategies. This process is somewhat arbitrary can have a big impact on defining optimal strategies in terms of item builds and champion select.

**Step 4 - Storing to MySQL Database**

SQLAlchemy library was used to store the pulled match data into a local database. The library automatically defines the MySQL schema based on the JSON file. Many tables were created within the database and each table was reviewed to determine the contents of the table. The table that contained the dataset for use in the primary analysis was called matchparticipantstats. The entire table was exported which contained 53452 rows with each row representing an instance of a player in a match. A different table called matchteam was used to verify dataset veracity. Matchteam table showed 10700 entries of team events representing 5350 unique matches. Since each team contains 10 players, the player stats table was determined to have 8 missing player entries which was a tolerable margin of error in match extraction.

**Step 5 - Exporting MySQL Tables to CSV**

The task of exporting the matches was done in MySQL Workbench using the Table Data Export Wizard. File name, columns to be exported, and field separator were defined for the table and converted into CSV(comma separated values) format.

**Approach**

**Step 1 – Check for Missing Values**

The full dataset was checked for any missing values. The dataset was found to have no missing values.

**Step 2 – Preliminary Feature Selection**

The exported dataset contained 65 feature variables. X\_id and x\_participant\_id were removed because they are non-descript ordinal values. Six item values were removed because they were categorical values referencing a specific item that a player can purchase in the game. Item values could be used in factor analysis in a different study. Ten additional categories were removed because they are stats that are recorded in a different game mode. All ten values returned values of 0. FirstBloodAssist was removed because the values were returned as 0. NeutralMinionsKilledTeamJungle were removed as the coefficient values in the summary of the GLM would return NA values. It is unclear why this is the case as the stat is a continuous variable that does not return 0 values in the dataset. The final dataset totals 44 features which includes the response variable.

**Step 3 – Creating Training and Testing for Fold Validation**

The original dataset of 53452 instances were reduced to 50000 and divided into 5 folds. Corresponding training sets were created by combining the remaining unused instances. The same folds were used for all tests performed. Slice function from the dplyr package[[4]](#endnote-4) is used to divide the subsets. Random sampling was not used because the order within the dataset is non-discriminatory and could be problematic for reproducing results.

**Step 4 – GLM Formula and Functions**

The GLM using a logit function was trained using all 43 features. The winner is defined as the response variable for all GLMs. The GLM function is included in the base stats package of R. All GLM functions include the argument of family as binomial. This indicates to the function that the GLM is performing a logistic regression as the winner result contains dichotomous variables of 0 or 1. The predict.glm function is used to create a prediction object which includes the test set. This process is repeated for each model of unique feature sets.

fullfeatureglm <- glm(winner ~., data = train,family = binomial)

fullfeatureglmpred <- predict.glm(fullfeatureglm, newdata = test, type = "response")

The summary() function can be used for the results of the GLM to produce information about each feature used in the GLM.

glm(formula = winner ~ ., family = binomial, data = train)

Deviance Residuals:

Min 1Q Median 3Q Max

-4.7911 -0.3197 0.0003 0.2266 4.7673

Coefficients:

Estimate Std. Error z value Pr(>|z|)

(Intercept) -3.555e+00 1.538e-01 -23.108 < 2e-16 \*\*\*

assists 3.101e-01 7.728e-03 40.128 < 2e-16 \*\*\*

champLevel 2.502e-01 2.239e-02 11.173 < 2e-16 \*\*\*

deaths -5.461e-01 1.241e-02 -44.007 < 2e-16 \*\*\*

doubleKills -9.338e-02 4.453e-02 -2.097 0.035996 \*

firstBloodKill -8.527e-03 6.494e-02 -0.131 0.895535

firstInhibitorAssist 2.473e+00 6.980e-02 35.428 < 2e-16 \*\*\*

firstInhibitorKill 1.202e+00 1.604e-01 7.496 6.60e-14 \*\*\*

firstTowerAssist 5.587e-01 9.991e-02 5.592 2.25e-08 \*\*\*

firstTowerKill -5.785e-02 8.163e-02 -0.709 0.478525

goldEarned 1.302e-03 4.912e-05 26.501 < 2e-16 \*\*\*

goldSpent -1.077e-03 3.137e-05 -34.329 < 2e-16 \*\*\*

inhibitorKills 1.048e+00 1.152e-01 9.101 < 2e-16 \*\*\*

killingSprees 1.932e-02 3.690e-02 0.524 0.600462

kills 2.666e-01 1.973e-02 13.513 < 2e-16 \*\*\*

largestCriticalStrike -4.166e-04 8.254e-05 -5.047 4.48e-07 \*\*\*

largestKillingSpree -7.584e-02 1.769e-02 -4.287 1.81e-05 \*\*\*

largestMultiKill 3.057e-01 5.219e-02 5.857 4.71e-09 \*\*\*

magicDamageDealt 5.479e-02 3.240e-02 1.691 0.090863 .

magicDamageDealtToChampions -3.445e-02 3.477e-02 -0.991 0.321715

magicDamageTaken -3.682e-02 3.235e-02 -1.138 0.255003

minionsKilled -7.268e-03 9.257e-04 -7.851 4.12e-15 \*\*\*

neutralMinionsKilled -3.409e-02 2.475e-03 -13.773 < 2e-16 \*\*\*

neutralMinionsKilledEnemyJungle 1.704e-01 6.610e-03 25.776 < 2e-16 \*\*\*

pentaKills -1.327e-01 7.003e-01 -0.190 0.849696

physicalDamageDealt 5.479e-02 3.240e-02 1.691 0.090853 .

physicalDamageDealtToChampions -3.446e-02 3.477e-02 -0.991 0.321582

physicalDamageTaken -3.673e-02 3.235e-02 -1.136 0.256158

quadraKills -1.247e-01 2.660e-01 -0.469 0.639269

sightWardsBoughtInGame 3.763e-02 1.239e-02 3.037 0.002386 \*\*

totalDamageDealt -5.479e-02 3.240e-02 -1.691 0.090858 .

totalDamageDealtToChampions 3.436e-02 3.477e-02 0.988 0.322993

totalDamageTaken 3.667e-02 3.235e-02 1.134 0.256901

totalHeal 1.120e-05 6.500e-06 1.723 0.084875 .

totalTimeCrowdControlDealt -1.557e-04 4.403e-05 -3.536 0.000406 \*\*\*

totalUnitsHealed -1.935e-02 3.428e-03 -5.646 1.64e-08 \*\*\*

towerKills 4.640e-01 2.710e-02 17.120 < 2e-16 \*\*\*

tripleKills -2.781e-01 9.840e-02 -2.826 0.004713 \*\*

trueDamageDealt 5.479e-02 3.240e-02 1.691 0.090852 .

trueDamageDealtToChampions -3.440e-02 3.477e-02 -0.989 0.322445

trueDamageTaken -3.690e-02 3.235e-02 -1.141 0.253985

visionWardsBoughtInGame -1.952e-02 1.409e-02 -1.385 0.166060

wardsKilled -1.205e-01 7.921e-03 -15.213 < 2e-16 \*\*\*

wardsPlaced -2.512e-02 2.942e-03 -8.539 < 2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 55452 on 39999 degrees of freedom

Residual deviance: 18967 on 39956 degrees of freedom

AIC: 19055

Number of Fisher Scoring iterations: 7

**Step 5 – Create Prediction Object Using Winner as Class**

The ROCR[[5]](#endnote-5) package was used to create predictions of winners for each instance in the model. The prediction function is called to accomplish this task.

fullfeaturepred <- prediction(fullfeatureglmpred,test$winner)

**Step 6 – Plot ROC and Measure AUC**

Performance of the model is measured by using the performance() function from ROCR package in R. The measure and x.measure arguments are called to define the performance measure for use in evaluation. The first example below measures the true positive rate and the false positive rate. Plotting this object will result in the plot of the ROC. This step was repeated for each model and each fold. Defining the measure argument as AUC will calculate the area under the curve of the ROC. This was used as the primary measurement component of performance as the AUC accounts for sensitivity and specificity.

fullfeatureROC <- performance(fullfeaturepred, measure = "tpr", x.measure = "fpr")

fullfeatureauc <- performance(fullfeaturepred, measure = "auc")

**Step 7 – Test Alternative Models Using GLM**

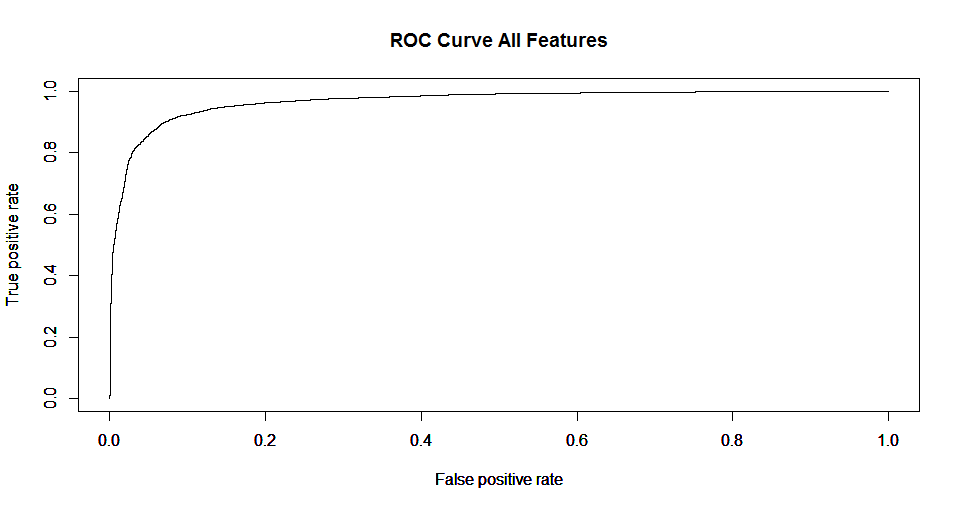
The steps listed above were repeated for general linear models using different formulas. The formula indicates which features are included for use in the model. The first experimental formula was including 22 of the highest positively and negatively correlated values. The second experimental formula was using 10 of the same criteria. The final model used was a formula derived from backward stepwise selection process using the step() function in R. The GLM used in the step() function included the entire dataset to avoid bias resulting from the training folds. When the formula was compared between the full sample and fold training set, the full sample formula contained one extra feature. This formula comparison can be found in the git repository in the file BackwardStepFormulaComparison. The step function is a method to find the subset of features which returns the best results of prediction performance. This is a critical component of feature selection and feature reduction. Once the formula was defined, the previous steps were repeated with 5 folds to test for performance variance. The author of this paper was dubious on the success of the initial results so models containing one feature and three features were used in an attempt to create models with lower performance. The single variable formula used was kills and another formula containing kills, assists, and deaths was used.

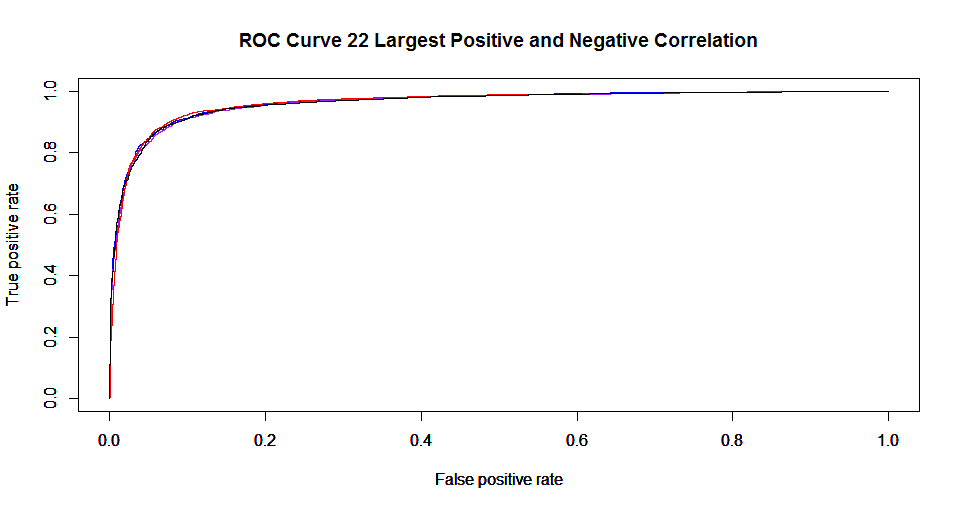
**Results**

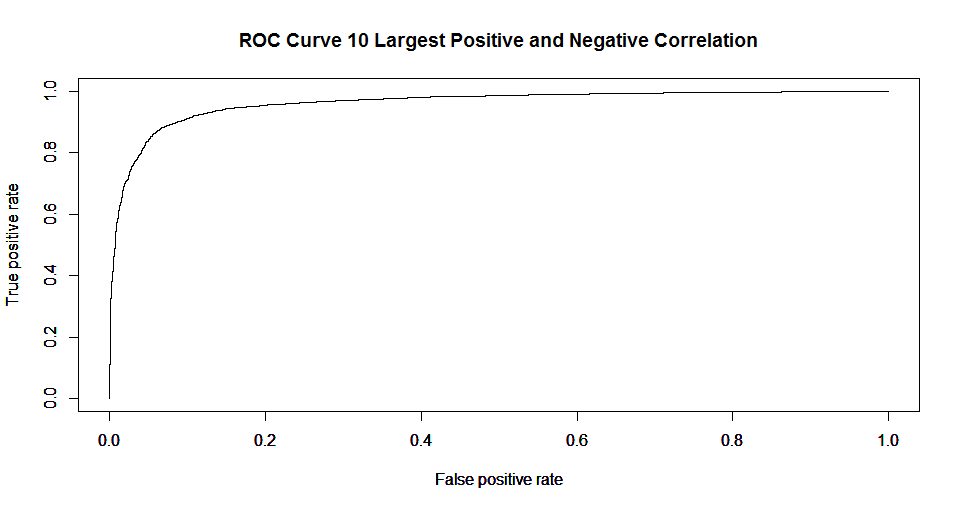
The ROC plots displays the performance of prediction for each GLM model used. The corresponding plot for all features and 10 features display what seem to be one plot but they are in fact the plots of all folds. The plots provide a visual representation of prediction performance. Nominal performance variance can be observed for some of the ROC plots for each fold as indicated by the colours of the plot lines. The first boxplot displays the performance comparison of the models which yielded high success rates. The second boxplot displays the performance of all models tested. The table of AUC values for all tests indicates the exact performance results. The highest performing formula was from the backward stepwise selection as indicated by the highest average AUC value of 0.965650. This was the expected result as the stepwise function is an attempt at finding the best subset of features for prediction. This was still nominally similar to the results of the full feature set model which yielded average AUC value of 0.965596. The dataset used in this research has a unique characteristic in that the response variable is evenly distributed since each match will result in an equal number of winners and a losers. The table of correlation indicates the values used to determine the formula in the reduced feature set model.

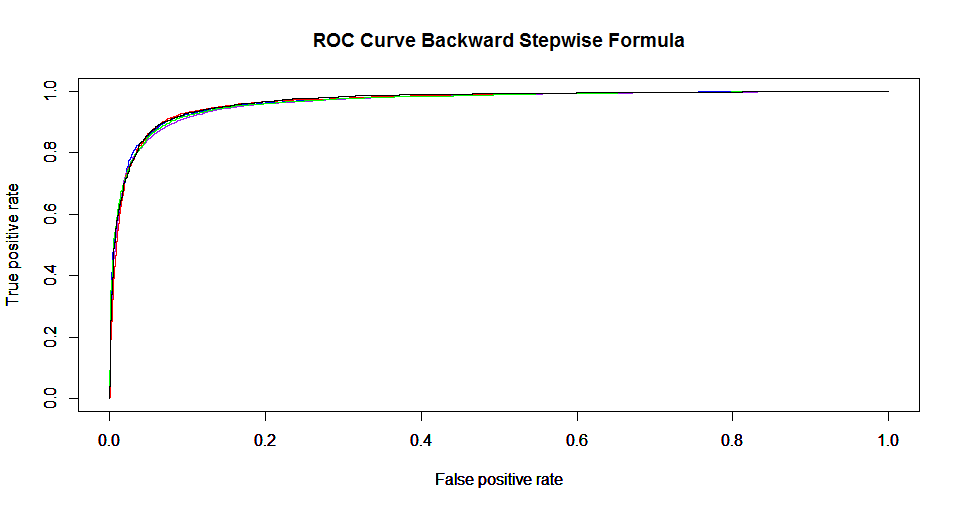
The initial exploratory analysis included other forms of analysis such as analysis of variance, correlation, co-variance to list a few as an example. It is important to note the application of domain expertise throughout the entire project. The initial feature reduction process included removing categorical item values which could have reduced the performance of the general linear models. The full dataset returned by the Riot Games match API endpoint included additional time series data, categorical data, and other team related information. A K-nearest neighbour model was used to evaluate prediction performance of team events with the response variable set as the winner. This model can be found in the file knnteam R script in the GitHub repository. The results of this model are not included in this report but the model achieved a true positive rate of prediction of approximately 85-88 percent. It is hypothesized that joining this information with the dataset which was used for analysis could result in a higher performance for the GLMs.

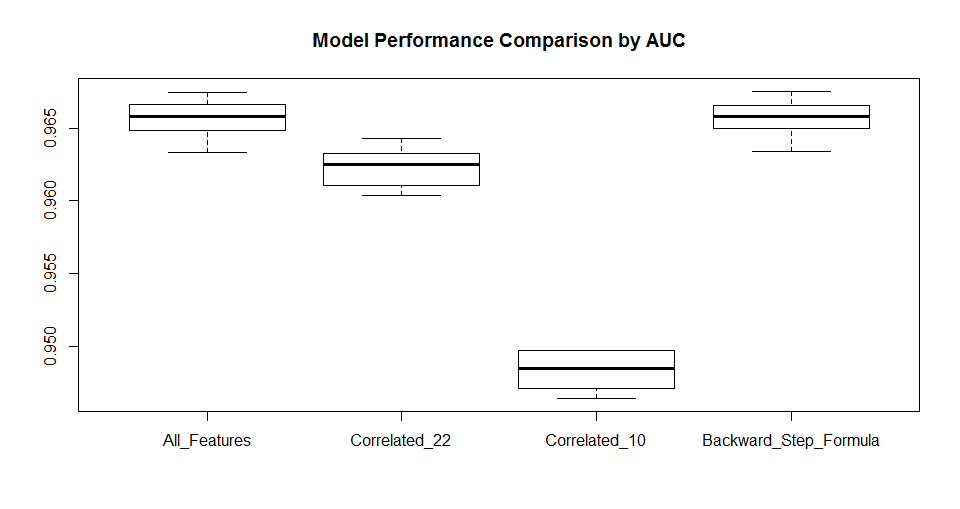
All attempts have been made to allow the results to be reproducible. Please follow the instructions in the R script to reproduce results.

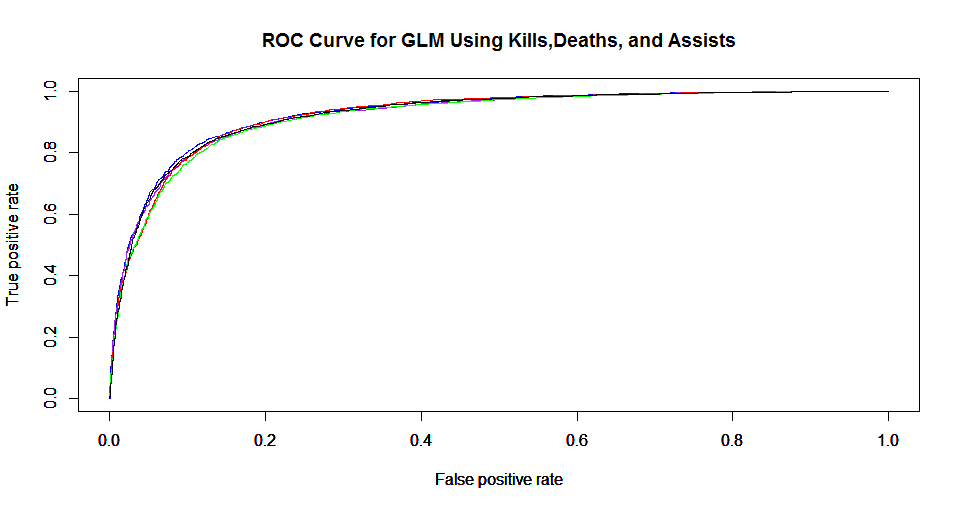


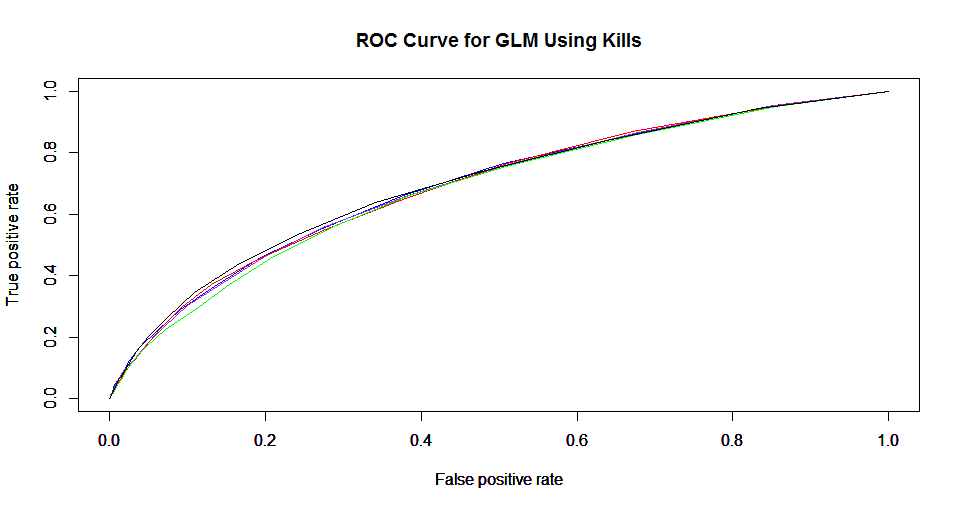


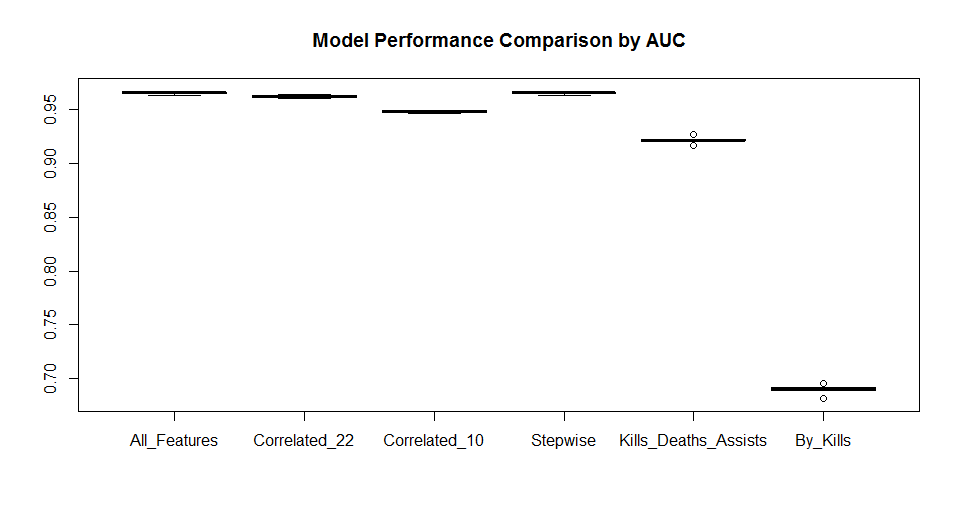












|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Performance of GLMs by AUC** | | | | | | |
|  | **All\_Features** | **Correlated\_22** | **Correlated\_10** | **Backward\_Step\_Formula** | **Kills\_Deaths\_Assists** | **By\_Kills** |
| **Fold 1** | 0.966598 | 0.963243 | 0.94975 | 0.966579 | 0.926741 | 0.691691 |
| **Fold 2** | 0.96331 | 0.960374 | 0.946437 | 0.963371 | 0.921556 | 0.689095 |
| **Fold 3** | 0.965787 | 0.962523 | 0.948514 | 0.965801 | 0.922596 | 0.690045 |
| **Fold 4** | 0.964823 | 0.961084 | 0.947168 | 0.964963 | 0.916727 | 0.681298 |
| **Fold 5** | 0.967463 | 0.964262 | 0.949777 | 0.967538 | 0.921968 | 0.695731 |
| **Mean** | 0.965596 | 0.962296 | 0.948329 | 0.965650 | 0.921917 | 0.689572 |

|  |  |  |
| --- | --- | --- |
| **Feature Number** | **full\_cor.winner** | **row.names.full\_cor.** |
| **44** | 1 | winner |
| **6** | 0.43482588 | firstInhibitorAssist |
| **1** | 0.41229887 | assists |
| **16** | 0.37799913 | largestKillingSpree |
| **36** | 0.36480611 | towerKills |
| **14** | 0.32207006 | kills |
| **10** | 0.29518296 | goldEarned |
| **13** | 0.29112879 | killingSprees |
| **12** | 0.26387576 | inhibitorKills |
| **17** | 0.26021403 | largestMultiKill |
| **23** | 0.24906348 | neutralMinionsKilledEnemyJungle |
| **4** | 0.24342334 | doubleKills |
| **2** | 0.2330426 | champLevel |
| **7** | 0.23189613 | firstInhibitorKill |
| **11** | 0.21110277 | goldSpent |
| **31** | 0.1590799 | totalDamageDealtToChampions |
| **37** | 0.14013851 | tripleKills |
| **30** | 0.1378184 | totalDamageDealt |
| **33** | 0.12361938 | totalHeal |
| **9** | 0.10280352 | firstTowerKill |
| **26** | 0.10229195 | physicalDamageDealtToChampions |
| **25** | 0.09659761 | physicalDamageDealt |
| **8** | 0.08844158 | firstTowerAssist |
| **19** | 0.0813944 | magicDamageDealtToChampions |
| **15** | 0.07402504 | largestCriticalStrike |
| **5** | 0.0734539 | firstBloodKill |
| **28** | 0.07142821 | quadraKills |
| **18** | 0.06610738 | magicDamageDealt |
| **21** | 0.06150163 | minionsKilled |
| **22** | 0.05758024 | neutralMinionsKilled |
| **34** | 0.05416148 | totalTimeCrowdControlDealt |
| **39** | 0.05220796 | trueDamageDealtToChampions |
| **42** | 0.04012759 | wardsKilled |
| **41** | 0.03951821 | visionWardsBoughtInGame |
| **38** | 0.03542284 | trueDamageDealt |
| **24** | 0.03491565 | pentaKills |
| **43** | 0.03235006 | wardsPlaced |
| **35** | 0.02591005 | totalUnitsHealed |
| **29** | -0.01148918 | sightWardsBoughtInGame |
| **27** | -0.03603912 | physicalDamageTaken |
| **40** | -0.07961434 | trueDamageTaken |
| **32** | -0.09224281 | totalDamageTaken |
| **20** | -0.13466616 | magicDamageTaken |
| **3** | -0.44237633 | deaths |

Backward Stepwise Formula

|  |
| --- |
| winner ~ |
| assists + |
| champLevel + |
| deaths + |
| doubleKills + |
| firstInhibitorAssist + |
| firstInhibitorKill + |
| firstTowerAssist + |
| firstTowerKill + |
| goldEarned + |
| goldSpent + |
| inhibitorKills + |
| kills + |
| largestCriticalStrike + |
| largestKillingSpree + |
| largestMultiKill + |
| magicDamageDealt + |
| magicDamageDealtToChampions + |
| magicDamageTaken + |
| minionsKilled + |
| neutralMinionsKilled + |
| neutralMinionsKilledEnemyJungle + |
| physicalDamageDealt + |
| physicalDamageDealtToChampions + |
| physicalDamageTaken + |
| sightWardsBoughtInGame + |
| totalDamageDealt + |
| totalHeal + |
| totalTimeCrowdControlDealt + |
| totalUnitsHealed + |
| towerKills + |
| tripleKills + |
| trueDamageDealt + |
| trueDamageDealtToChampions + |
| trueDamageTaken + |
| wardsKilled + |
| wardsPlaced |

**Conclusion**

The primary component of this project can be described as feature selection using multivariate logistic regression using a general linear model. It was an attempt at finding the highest performing subset of features in the GLM. It should be noted that a significant number of hours were invested into the data curation process. Refining the approach to dataset acquisition required the author to learn the Python environment, Riot API endpoints, database management, and various data frame manipulation techniques. The data curation steps included the process of establishing discriminatory conditions such as when the matches were played, by which region, and the skill level of the players.

The success of the GLMs prediction performance can also be attributed to the fact that all variables are shown to have a linear relationship with the response variable. Regression models containing independent variables with a non-linear relationship to the response variable may be inclined to use other techniques to define the relationship between independent and response variables.

As described in the text, An Introduction to Categorical Data Analysis by Alan Agresti,

*The selection process becomes more challenging as the number of explanatory variables increases, because of the rapid increase in possible effects and interactions. Cautions that apply to building ordinary regression models hold for any GLM. For example, models with several predictors often suffer from multicollinearity – correlations among predictors making it seem that no one variable is important when all the others are in the model.*[[6]](#endnote-6)

The participant dataset contained a sufficient number of player instances to avoid the problem of overfitting. In order to absolve the model of a potential overfitting problem, a separate dataset of the same features extracted from a different region was used to test a GLM of the full Korean dataset. The AUC value returned was similar to the highest performing models in the experiment with a value of 0.956303. The replication for this code can be found in the final section of the Final Report R script found in the GitHub repository.

While the GLMs provided a high prediction success rate as indicated by the AUC, the functional utility of the use of these models by the general player population should be discussed. It should be noted that the match participant stats can only be retrieved after the game has been completed. Players of various skill level currently use online tools which cumulatively measure the past performance of their play. One practical function the GLMs can provide is to indicate to a player whether their performance in a played match should have resulted in a win.

1. http://www.riotgames.com/articles/20121015/138/league-legends-growth-spells-bad-news-teemo [↑](#endnote-ref-1)
2. https://en.wikipedia.org/wiki/Elo\_rating\_system [↑](#endnote-ref-2)
3. http://robrua.github.io/cassiopeia/ [↑](#endnote-ref-3)
4. https://cran.r-project.org/web/packages/dplyr/dplyr.pdf [↑](#endnote-ref-4)
5. https://cran.r-project.org/web/packages/ROCR/ROCR.pdf [↑](#endnote-ref-5)
6. An Introduction to Categorical Data Analysis by Alan Agresti 1996 Wiley [↑](#endnote-ref-6)