Using machine learning for predicting League of Legends match outcome

Jakob Maležič
Faculty of Computer and Information Science
University of Ljubljana
Ljubljana, Slovenia
Email: jm6421@student.uni-lj.si

***Abstract*—We compare different machine learning classifiers for predicting the match outcome of a popular online multiplayer game *League of Legends*. Features that are used are extracted from a Kaggle data set that contains 108,000 ranked games from Korea. The data encapsulates champion picks, participants' usernames, players' roles and in-game players' statistics. We process the data in such way, that each case contains data about participants of one team ordered by their roles. By doing so, we created a data set with 156344 cases. We find out, that we can accurately predict outcome of a match with the provided data using different classifiers.**

I. Introduction

League of Legends is a multiplayer online battle arena (MOBA) game developed by Riot Games. Players are assigned a team with 4 teammates and play head-to-head versus enemy team, which consists of 5 players as well. The goal of the game is to destroy the enemy base and their main structure, called The Nexus. It is currently the world's most popular video game with over 100 million active monthly players and a vast and widely competitive e-sports scene. Because of a such huge community, predicting match outcomes for casual players as well as tournament games is very interesting and could be very valuable to players and fans.

A. Game Overview

In a classic *League of Legends* game, two teams of five players, called summoners, battle on a map called Summoner's rift. Contrary to MMORPG (Massively Multiplayer Online Role-Playing Game) games, there is no unit construction nor there is a massive number of players on a map at a time in a MOBA game. Much of the strategy revolves around cooperative team play and individual performance.

The map is split into three lanes: top lane, mid lane and bottom lane, as shown in figure 1. Between the lanes there is jungle with neutral monsters. Map is split by a river in the middle.

Each player is assigned a different role before the game starts. There are five different roles: top, jungle, mid, bottom and support, also called utility. This plays an important role on player's choosing of a champion, since champions are categorized by roles as well. A champion that does well on top lane does not necessarily do well in the bot lane. This is important, because players usually tend to specialize one role and sometimes even only one champion.

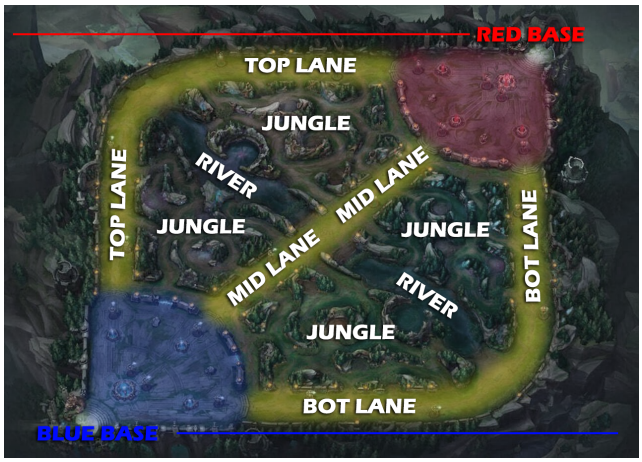

Fig. 1. Summoner's rift.

Each player chooses a unique champion from a pool of over one hundred different champions. Champions differ by their characteristics and abilities. Some champions are faster, some provide utility, some provide sustain, some are close range fighters, some are long ranged, etc. Each champion has four different unique abilities.

Champions are chosen in the draft phase, as shown in figure 2, before the start of the game. Each of the players first bans one champion from the game. Banned champions cannot be played by either of the teams. Players then alternately pick their champions. After everyone has picked there is a 30 second countdown till the start of the game. In this time any of the players can choose to leave the game. This is also called dodging, because the player dodges the game. By doing this, he cancels the match and players need to find new teammates and opponents to play.

League of legends has a ranking system, matching the players of a similar skill level to play with and against each other. It comprises nine tiers which indicate the skill level of players. Players within each division are ranked using a system of points called League Points (LP).

Each tier from Iron to Diamond is divided into four divisions, depicted by a roman numeral starting from IV (4 being the lowest) to I (1 being the highest). Master tier and higher do not have divisions, they are instead exclusively reliant on

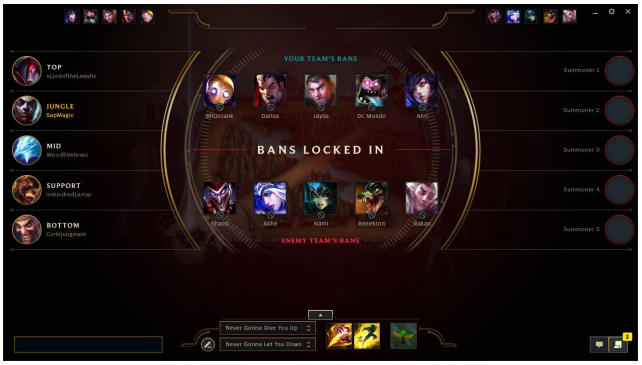

Fig. 2. Draft phase.

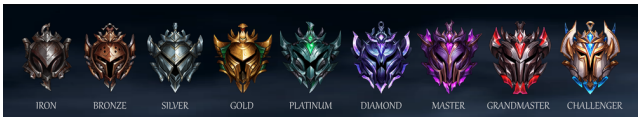

Fig. 3. Ranking system - tiers.

LP and the population of players within these rank tiers. Tiers are shown in figure 3.

The player earns League Points when they win ranked games and lose them when they lose ranked games. The amount earned or lost depends on the player's hidden Match Making Rating (MMR). The higher the MMR, the more LP earned per win and the less LP lost per loss [1].

The penalty for the player that leaves the game in the draft phase is losing some amount of LP, but his MMR stays the same. This makes it very valuable to dodge games, if the player can predict a loss before the game starts.

II. Related work

Since *League of Legends* is a popular game, there have already been a couple of researches that tried to predict the match outcome.

Lucas Lin used gradient boosted trees and gradient boosted trees with logistic regression. He used one-hot encoding to transform data to a feature vector. By using these methods, he tried to predict the outcome of a match based on in-game statistics. He described this in his article *League of Legends Match Outcome Prediction* [2].

Ani R, Vishnu Harikumar, Arjun K Devan, O.S. Deepa tried to predict match outcome by doing feature selection and ensemble models of classification algorithms. Their work is presented in the article *Victory prediction in League of Legends using Feature Selection and Ensemble methods* [3].

Yifan Yang, Tian Qin and Yu-Heng Lei tried to predict match outcome of a similar game, called Dota 2. They used pre-match features as well as real-time (during-match) features at each minute as the match progressed. They used logistic regression, Attribute Sequence Model and their combinations as the prediction models. They presented this in their article *Real-time eSports Match Result Prediction* [4].

Kenneth T. Hall used deep neural networks, based upon the historical game play statistics of match participants for predicting the winning team of the match. His most successful network achieves greater than 80 percent win prediction accuracy on testing data. His work is described in article *Deep Learning for League of Legends Match Prediction* [5].

III. Data collection

Riot Games provide a public API for accessing historical and live data of League of Legends matches. If we want to get statistic data for a summoner, we need his summoner name. We then need to make a request to the API, to get summoner's id. With this id, we can then get list of summoner's match references, 100 per request. Each match reference contains match id, which references one particular match. We then need to make another request to get all data about this particular match. The data contains participant identities and statistics for every participant in that match. This statistics encapsulates' data like: kills, assists, deaths, damage per minute, gold per minute, vision score, wards bought, champion, etc. This is exactly the data that we need for building a model for predicting match outcome.

However, Riot API only allows us to make 50 request per minute. This is not enough for the data that we need, because we need historical data of all the players of the team. That would require us to make a requests for each match the player has played. That is why we decided to build a model on a data set from Kaggle[1]. This data set consists of 108941 games from master, grandmaster and challenger players from Korea.

Since the model will be build only on games from high ranked games from Korea, we need to keep in mind that the model might not work as well in different region and rank.

IV. Data processing

The data we got from Kaggle is split into three files:

- *match_data* - contains all statistical data about matches
- *match_winner_data* - contains match references of winning matches
- *match_loser_data* - contains match references of losing matches

The statistical data about a match contains data for the winning and the losing team. This means, that the *match_winner_data* references the same games as the *match_loser_data*. This is the reason why we only need to use one file of match references and not both.

First we read *match_data* and *match_winner_data* in Python[2] using Pandas[3] library and merge match references with match data. By doing this we create a table of 108941 rows, where each row contains all statistical data about specific match. We then sort this table by match's timestamp. This is important to preserve the actual sequence of the matches.

We then loop through the matches and extract the features we are interested in. Since we want each our case to present

[1]https://www.kaggle.com/
[2]https://www.python.org/
[3]https://pandas.pydata.org/pandas-docs/stable/index.html

one team of the match, we need features for each player of the team in the same case. Because the columns always need to be in the same order, we sorted players by their role: top, jungle, mid, bottom and support. We did this by using a python library Cassiopeia[4], which uses champion's statistical data to decide player's role. The features that we extracted for each participant are:

- **Total games** - the amount of matches played.
- **Wins** - the amount of matches won.

And the features we extracted for each participant's role are:

- **Kills** - the amount of enemy champions killed in a match. This feature is important, because killing enemy champions grant the player extra gold, which is used for buying items and therefore getting and edge over the enemy. This feature together with deaths and assists forms the kills, deaths and assists ratio (KDA) which is the most often used statistic in statistical analysis of League of Legends matches. It's the quickest way of knowing whether or not the player played well.
- **Deaths** - the number of times the player got killed by the enemy. This feature is the inverse of the kills feature, since getting killed by the enemy, gives gold to the enemy player.
- **Assists** - the number of times the player assisted at killing an enemy champion. Similarly to kills, this feature is important, because assisting at killing an enemy champion grants gold to the player.
- **Gold earned** - the total amount of gold earned in a game by the player.
- **Total damage dealt to champions** - the amount of damage dealt to enemy champions.
- **Total minions killed** - the number of minions killed in the match. Minions are units that comprise the main force sent by the Nexus. They spawn periodically from their Nexus and advance along a lane towards the enemy Nexus, automatically engaging any enemy unit or structure they encounter. They are controlled by artificial intelligence and only use basic attacks [6]. The feature is important, because, similarly to kills, killing a minion gives gold to the player. Twelve minion kills equals to one champion kill.
- **Vision score** - stat that indicates how much vision a player has influenced in the game, this includes the vision they provided and denied. League of Legends uses a fog of war which only reveals terrain features and enemy units through a player's reconnaissance [7]. Without a unit actively observing, previously revealed areas of the map are subject to a shroud through which only terrain is visible, but changes in enemy units are not.
- **Vision wards bought** - the number of control wards bought in a match. Control wards are consumable items in League of Legends that grant sight over the surrounding area and reveal enemy wards and hidden traps.. Consumable items are one-time use items that disappear or lose effect after the first use, and can be used again only by buying them again. [8].
- **Total games** - the amount of matches played with a given role.
- **Wins** - the amount of matches won with a given role.

For each match we update the player's statistics by calculating the mean for each feature. This way we only use statistics of past matches. We picked these features by our domain knowledge and statistical data that is usually shown in professional e-sports broadcasts. In conclusion, each our case now has 260 features.

V. Methodology

We tested four different machine learning classifiers that we thought would best fit the problem.

A. Gaussian naive Bayes

Bayesian classifiers assign the most likely class to a given example described by its feature vector. Learning such classifiers can be greatly simplified by assuming that features are independent given class. Although independence is generally a poor assumption, in practice naive Bayes often competes well with more sophisticated classifiers [9].

Naive Bayes is a classification algorithm that works for binary and multi-class classification problems. The technique works best when using binary or categorical input values, however, it can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution. Since our features are real values, we use the Gaussian naive Bayes.

B. Decision trees

A decision tree is a classifier and it consists of nodes that form a rooted tree, meaning it is a directed tree with three types of nodes:

- **root** - a node that has no incoming edges, there is exactly one such node in a tree
- **internal or test node** - a node with exactly one incoming edge and outgoing edges
- **leaf or terminal node** - a node with exactly one incoming edge and no outgoing edges

In a decision tree, each internal node splits the instance space into two or more sub-spaces according to a certain discrete function of the input attributes values. Each leaf is assigned to one class representing the most appropriate target value [10]. Figure 4 shows an example of a decision tree.

Induction of an optimal decision tree from a given data is considered to be a hard task. It has been shown that finding a minimal decision tree consistent with the training set is a NP-hard. Because of this, heuristic methods have to be used for solving the problem [11].

In most of the cases, the functions in internal nodes are univariate, meaning that an node is split according to the value of a single attribute. Consequently, the tree building algorithm searches for the best attribute upon which to split. There exist many univariate criteria, these are some of them:

[4]https://github.com/meraki-analytics/cassiopeia

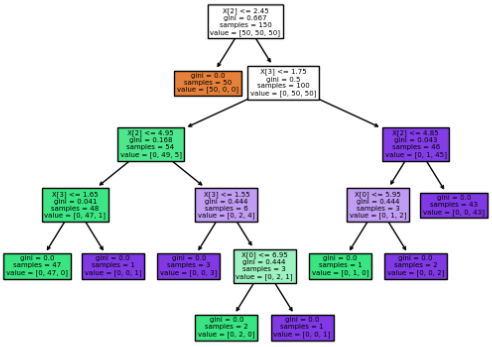

Fig. 4. Decision tree example.

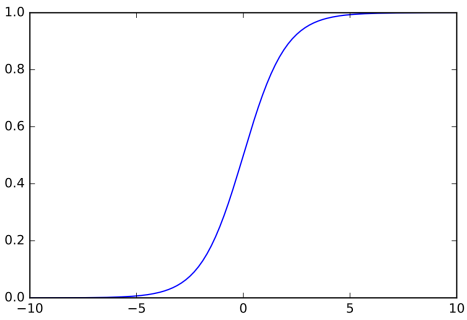

Fig. 5. Logistic function.

- **Information Gain** - an impurity-based criterion that uses the entropy measure as the impurity measure [11].
- **Gini Index** - an impurity-based criterion that measures the drivergences between the probability distributions of the target attribute's values [12].
- **Gain Ratio** - the gain ratio "normalizes" the information gain as follows [11]:

$$\text{GainRatio}(a_i, S) = \frac{\text{InformationGain}(a_i, S)}{\text{Entropy}(a_i, S)} \quad (1)$$

C. Logistic regression

Logistic regression is a statistical model that in its basic form uses a logistic function, figure 5, to model a binary dependent variable. However, unlike linear regression, the response variables can be categorical or continuous, as the model does not strictly require continuous data, as seen on picture 6. To predict group membership, logistic regression uses the log odds ratio rather than probabilities and an iterative maximum likelihood method rather than a least squares to fit the final model. Because of this, the method is more appropriate for non-normally distributed data.

D. Random forest

Random forests are a scheme for building a classification ensemble with a set of decision trees that grow in randomly

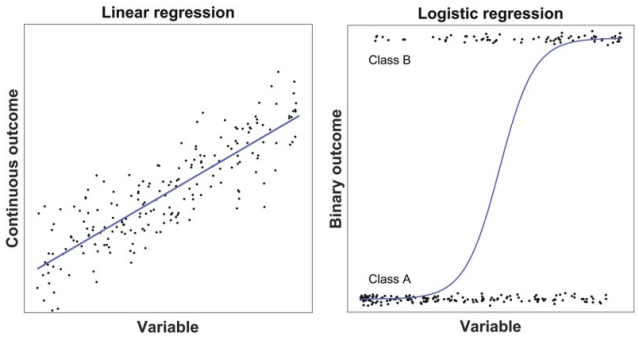

Fig. 6. Linear regression compared to logistic regression.

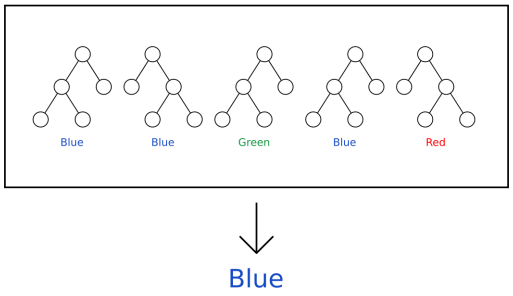

Fig. 7. Random forest example.

selected subspaces of data [13]. Experimental results have shown that random forest classifiers can achieve high accuracy in classifying data in domains of high dimensions with many cases [13], [14].

Since our data consists of huge number of features and cases, we tried to use random forest for predicting the match outcome.

E. K-Fold Cross Validation

Cross-validation is a resampling procedure used for evaluating machine learning models. The procedure takes one parameter k that refers to the number of groups that a given data sample is to be split into. That is why, it's often called k-fold cross validation.

It is mainly used to evaluate the machine learning model on unseen data. That is, to build a model on a limited sample of data in order to estimate how well does it perform in general, when the model is built on data that is not used during the training phase. The general procedure is as follows:

- Shuffle the dataset randomly.
- Split the dataset into k groups, also called folds.
- For each group:
  - Take the group as test data set
  - Take the remaining groups as training data sets
  - Fit the model on the training data sets and evaluate it on the test set
  - Retain evaulation score and discard the model
- Summarize the score of the model using the sample of model evaluation scores

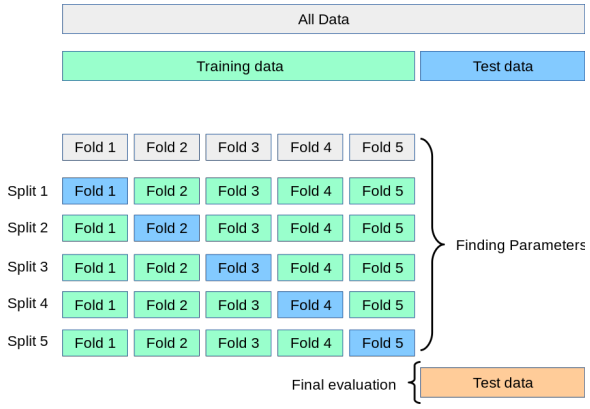

Fig. 8. K-Fold Cross Validation.

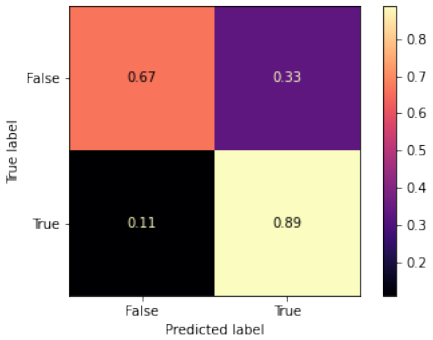

Fig. 9. Naive Bayes confusion matrix.

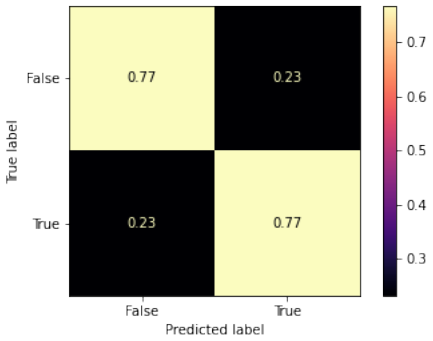

Fig. 10. Decision tree confusion matrix.

F. Confusion matrix

Confusion matrix is a matrix that helps us evaluate the accuracy of classification. It enables us to visualize the performance of an algorithm, usually a supervised learning one. Each row in the confusion matrix represents an instance of predicted class while each column represents the instance of the actual class [15].

In our case, the target class is a binary value: true, if the match outcome is win and false if the match outcome is loss. Therefore, all our confusion matrices will be size 2 by 2:

- **true positive** - When prediction was a win and the actual result was also a win.
- **true negative** - When prediction was a loss and the actual result was also a loss.
- **false positive** - When prediction was a win, but the actual result was a loss.
- **false negative** - When prediction was a loss, but the actual result was a win.

The value written in each square shows a normalized number of cases of that instance.

VI. Results

For building and testing models we used the *scikit-learn*[5] machine learning library in Python. We did the following for all algorithms:

- Read the data using Pandas
- Split the data to train and test data sets in a ratio of 3 : 2
- Run the 10-Fold Shuffle Cross Validation with the chosen model
- Calculate the mean of cross validation accuracies
- Fit the model on whole train data set
- Predict match outcomes on test data set
- Calculate the accuracy of the model

Gaussian naive Bayes turned out to work pretty well in comparison with other algorithms. It's accuracy was third best. Figure 9 shows it's confusion matrix. This was not expected, since it assumes that features are independent. However, this might not be a huge factor in our case, because looking at features independently doesn't make a big difference. For an example, a player with high win rate will in most cases still play well, no matter the values of other features. Similar results are presented in *League of Legends Win Predictor* [16]. Training of Gaussian naive Bayes was very fast, but that is expected, since only the probability of each class and the probability of each class given different input values need to be calculated.

Decision tree classifier with entropy as the univariate criterion performed the worst. The reason for that might be a huge number of features. We tested the Gini index criterion as well, but it gave similar results. The confusion matrix can be seen in figure 10.

The best classifier turned out to be Logistic regression. This was expected, since it does well with binary target class and non-normally distributed data. Figure 11 shows it's confusion matrix. Training a logistic regression model took the longest, although that is expected with such huge number of real valued features.

With a bit longer learning time but better accuracy, random forest algorithm worked good as well. It's confusion matrix can be found in figure 12

Summary of the results of all the classifiers can be found in table I and scatter plot in figure 13.

[5]https://scikit-learn.org/stable/index.html

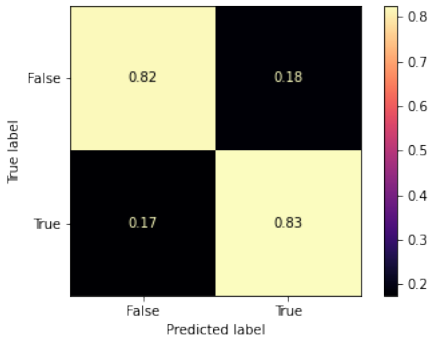

Fig. 11. Logistic regression confusion matrix.

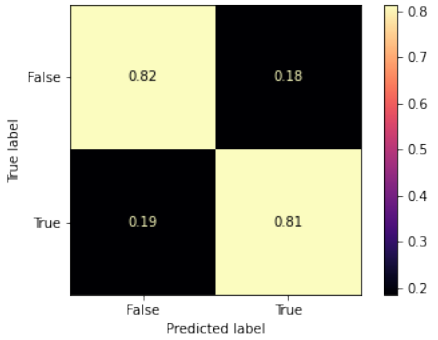

Fig. 12. Random forest confusion matrix.

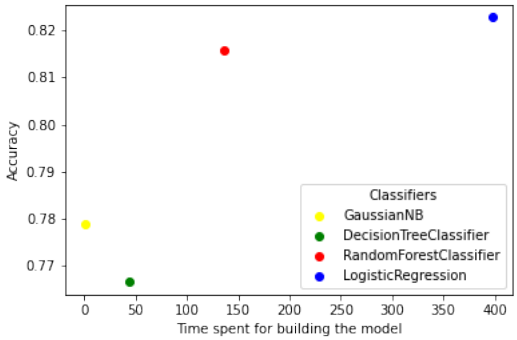

Fig. 13. Methods compared by time and accuracy.

TABLE I
RESULTS

Classifier	Cross validation accuracy [%]	Test accuracy [%]	Model building time [s]
Gaussian naive Bayes	78.2	78.2	1
Decision tree	76.5	76.8	43
Logistic regression	82.4	82.3	397
Random forest	81.5	81.5	136

VII. CONCLUSION

Riot Games may need to adjust their matchmaking algorithms. The accuracies that we got on test data show a strong performance at predicting the match outcome. League of Legends players depend on fair matchmaking which gives them and enemy opponents fair chance of winning the game. The fact that a machine learning classifier can predict the winning team with such accuracy is worrisome. The only problem for using such system in real time is getting the data needed for predictions, but Riot Games's matchmaking system has access to this data and could use it to make fair matches.

The models could definitely still be improved. It would be interesting to do a principal component analysis to see which features are the most important. There is also a lot more of statistical data that is available and could be used to further improve the predictions. The biggest of them are probably champion picks that have been completely ignored in this article.

Overall this article shows that predicting match outcome of a League of Legends game using machine learning classifiers with statistical data that we can get before the game is definitely possible and it can be used to improve player's chance of winning.

REFERENCES

[1] RiotGames, *Rank*, 2020 (accessed December 30, 2020), https://leagueoflegends.fandom.com/wiki/Rank.
[2] L. Lin, "League of Legends Match Outcome Prediction," 2016.
[3] R. Ani, V. Harikumar, A. K. Devan, and O. Deepa, "Victory prediction in League of Legends using Feature Selection and Ensemble methods," in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, no. Iciccs. IEEE, may 2019, pp. 74–77.
[4] Y. Yang, T. Qin, and Y.-H. Lei, "Real-time eSports Match Result Prediction," *arXiv*, no. Nips, pp. 1–9, dec 2016.
[5] K. T. Hall, *Deep Learning for League of Legends Match Prediction*, 2020 (accessed December 30, 2020), https://github.com/minihat/LoL-Match-Prediction/blob/master/FINAL%20REPORT.pdf.
[6] RiotGames, *Minions*, 2020 (accessed December 30, 2020), https://leagueoflegends.fandom.com/wiki/Minion.
[7] Wikipedia, *Fog of war*, 2020 (accessed December 30, 2020), https://en.wikipedia.org/wiki/Fog%20of%20war.
[8] RiotGames, *Consumable item*, 2020 (accessed December 30, 2020), https://leagueoflegends.fandom.com/wiki/Consumable%20item.
[9] I. Rish, "An empirical study of the naive Bayes classifier," *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 22230, 2001.
[10] L. Rokach and O. Maimon, *Decision Trees*. Boston, MA: Springer US, 2005, pp. 165–192.
[11] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, mar 1986.
[12] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees*. Taylor & Francis, 1984. [Online]. Available: https://books.google.si/books?id=JwQx-WOmSyQC
[13] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, 2001.
[14] R. E. Banfield, L. O. Hall, K. W. Bowyer, and W. Kegelmeyer, "A Comparison of Decision Tree Ensemble Creation Techniques," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 1, pp. 173–180, jan 2007.
[15] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, jun 2006.
[16] Huang, Thomas, Kim, David, Leung, Gregory, "League of legends win predictor," https://thomasythuang.github.io/League-Predictor/.