Victory Curve: Realtime Analysis of League of Legends

Grand Valley State University
College of Computing

Benjamin Groseclose, Ira Woodring

Abstract

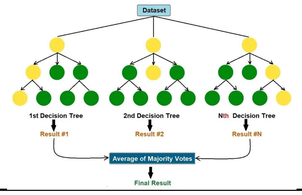
Analyzing *League of Legends* games in real time is challenging even for experienced players. This project aims to create a consistent and data-driven method to predict the outcome of matches. This desktop application leverages machine learning algorithms to predict the likely winner based on real-time game data. By automating this analysis, the tool provides valuable insights and helps players make better in-game decisions.

Introduction

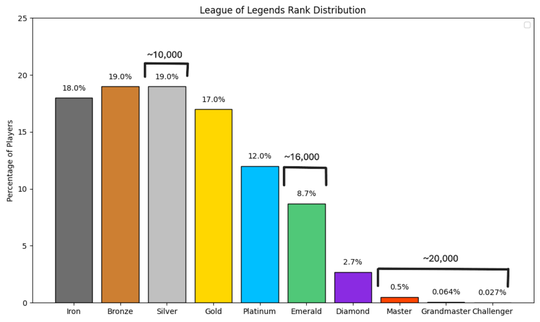
League of Legends is a five vs five strategic online video game, where players take control of a specific character. The ultimate goal is to destroy the enemy team's base, which is achieved by increasing your character's strength, via capturing various objectives or defeating the enemy team's characters. According to *Priori Data* in the last 30 days, over 100 million players globally have played *League of Legends*. There are over 170 characters (*champions)* a player can choose to play and eight different objectives that can be captured. No two games are the same. I have played League of Legends for many years and have always wondered while playing, "Are we winning or losing right now?". This question motivated me to work on this project. I was curious if my years of experience playing would align with what the machine was telling me.

This project aims to create a machine learning-based system that can predict the winning team in real time based on game data. This is achieved by utilizing historical match data, training a machine learning model, and then using that trained model to predict the outcome. For this project, I attempted several common machine learning algorithms including k-nearest neighbors (kNN), Decision tree, and Random Forest. I found that random forest provided the best results.

Random Forest classifier is an ensemble algorithm meaning they combine multiple "weaker" models to create a stronger model. In Random Forest's case, it uses multiple decision trees to classify the data, hence the word "forest". In addition, each tree is taken from a random subset of the data. Random Forest is ideal for reducing overfitting and bias.

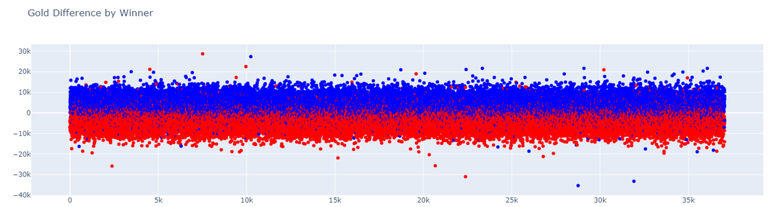


Results/Implementation



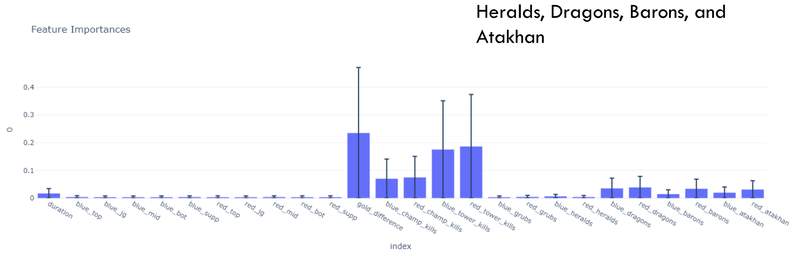
Data

In any machine learning project data is key. You need the right features and a sufficient amount to build a strong model. League of Legends has several crucial data points are worth tracking. Fortunately, *RIOT Games* provides an API that allows you to fetch detailed match information. Initially I focused on highly rated games (Master tier and above) under the assumption that these would provide reliable data. However, later I expanded the dataset to include other tiers. In total I collected approximately **36k matches**. The API provided a massive amount of data points. I begin with just *champions*, realized that is not consistent enough to produce accurate results. So, I incorporated additional features such as gold difference and key objectives. This significantly boosted the model accuracy.



In total, **26 features** were used to train the model, this included 10 champions (5 per team), gold difference, kills, towers, Grubs, Heralds, Dragons, Barons, and Atakhan

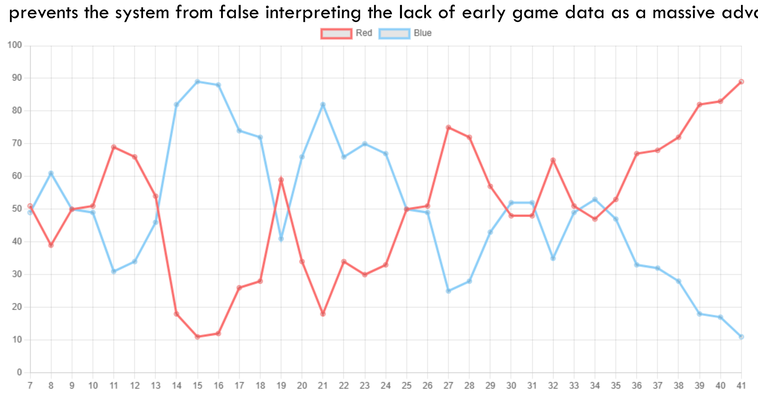
Model



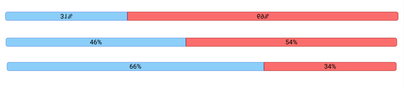
The random forest model was selected due to its ability to handle complex data, ability to identity the importance of features, and reduce the risk of overfitting. After training the model, I achieved an **98% accuracy**. I used 50% of the dataset for training and remaining 50% for testing. The model identified **gold difference** and **tower kills** as the strongest predictors of match outcome. This result is not surprising given gold is awarded to the team as they progress towards winning the game, and a prerequisite to defeating the enemy base is destroying at least 5 towers, but this is usually much higher. In contrast, other objectives like Dragons or Barons may provide strategic advantages but do not directly result in a win.

Application

The final product of this project was a desktop application designed to allow users to analyze their game in real-time. *RIOT Games* offers a "live client" which provides live match data while a game is in progress. With a single click users can start tracking their match through the application. However, this API provides data already available to the player. It does not provide information they do not already know, to avoid giving users any unfair advantage. Because of this limitation the model starts after a three-minute delay. This prevents the system from false interpreting the lack of early game data as a massive advantage for the user's side. After the initial Delay, the application fetches all necessary features and processes them using the trained model. The results are then visualized in a real-time graph. I found this to be help, while playing games myself. League of Legends is a complex game with many factors that are easy to overlook during gameplay.



The image to the left is an example of a game I played (red side).

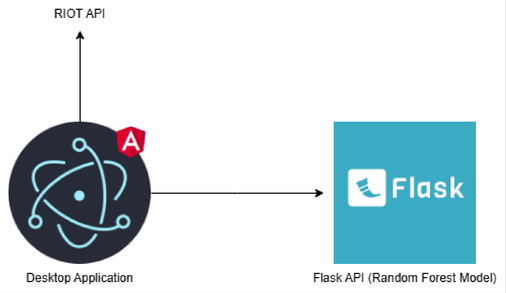


Technical Details

For this project, I used several software components to bring everything together. I began with *Jupyter notebooks* to fetch and store the data from the *RIOT API*. For training and testing the random forest machine learning model I used *sciket-learn* python library.

Once the model was trained and the data thoroughly analyzed, I developed the desktop application using **Electron**, a popular framework for building cross-platform desktop apps with web technologies like JavaScript, HTML, and CSS. On top of that, I used **Angular** to structure and build the front-end interface.

The desktop app communicates with a **Flask**-based Python API. This API exposes a single endpoint that accepts the required features and returns a prediction from the trained model.



Future Work

I believe this application would be most effective in an esports setting, where it could function as a spectator. This approach eliminates the point-of-view limitation. As a spectator, the model would have complete, unfiltered access to all in-game information, enabling it to provide the most accurate and comprehensive analysis.

In addition, I believe adding more features would only increase the model's strength such as gold difference at various points in the match.

Bibliography

Data sourced from *RIOT API*
Model created using *scikit-learn* package