COMP 560: League Of Legends Rank Predictor

Al Pagar, Kaw Bu, Jennifer Lee, Navya Katraju

November 20, 2025

Abstract

The aim of our project is to utilize a dataset containing match and player data of the game League of Legends from Kaggle in order to train a machine learning classifier to predict the rank of the player based on match statistics. After data exploration and feature engineering, we train a suite of machine learning classifiers to evaluate which had the best performance. The models we train are as follows: Logistic Regression, Ridge Regression, Decision Tree, Random Forest, AdaBoost, and Gradient Boosting. We then implement the selected model into a Streamlit application utilizing the selected model to predict ranks.

1 Introduction

Competitive multiplayer games generate large volumes of behavioral and performance data, offering an opportunity to study how these statistics relate to player skill. League of Legends, with more than 4 million daily players [Act25], provides extensive match-level statistics that reflect strategic decision-making, mechanical proficiency, and team coordination. Accurately modeling player rank from these metrics can provide beneficial use, such as the support of player development tools, matchmaking research, and broader studies of skill expression in complex online environments.

This project investigates the predictive relationship between match statistics and player rank using a publicly available Kaggle dataset [Sma25] containing player and match history records from League of Legends. After conducting exploratory analysis and preparing the data set for training, we evaluated multiple machine learning classifiers to determine which model has the best ability to predict rank based on the statistics given. The models tested include Logistic Regression, Ridge Regression, Decision Tree, Random Forest, AdaBoost, and Gradient Boosting. The best-performing model is then deployed in a Streamlit application that allows users to input match statistics and receive a predicted rank.

2 Models Background

The following is a background of the models we chose to implement and evaluate against the dataset.

2.1 Logistic Regression

Although Linear Regression is the standard algorithm for predicting continuous values, it is generally unsuitable for classification tasks. Linear Regression minimizes the sum of squared errors:

$$\text{Loss} = \tfrac{1}{2}\sum(y - \hat{y})^2.$$

This loss function is sensitive to outliers and does not constrain predictions to a valid probability range (values may fall below 0 or exceed 1).

To address this, Logistic Regression adapts the linear model by applying the sigmoid activation function to the linear output.

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

This transformation maps the predictions to a suitable probability range for classification. Furthermore, instead of minimizing squared error, Logistic Regression minimizes the Cross-Entropy Loss (Log Loss), which can be written as:

$$\mathcal{L} = -\left[y \ln(\hat{p}) + (1 - y) \ln(1 - \hat{p})\right].$$

This loss function exponentially penalizes high-confidence incorrect predictions, making it more effective for optimizing categorical decision boundaries than standard linear approaches.

2.2 Ridge Regression

Ridge Regression is fundamentally a regression algorithm that introduces L2 regularization to the standard loss function. By adding a penalty term proportional to the square of the magnitude of the coefficients,

$$\text{Penalty} = \lambda \|w\|^2,$$

the algorithm prevents any single feature from dominating the model and effectively handles multicellularity among features.

For classification tasks, the Ridge Classifier adapts this approach by converting class labels into binary targets (e.g., $\{-1, 1\}$) and minimizing the regularized Mean Squared Error. Unlike Logistic Regression, Ridge Classification does not optimize probabilities via Log Loss but instead focuses on finding a hyperplane that minimizes the squared distance to the class labels. This often results in faster training times and improved robustness against noise, while still assuming a linear relationship.

2.3 Decision Tree

Unlike the linear models discussed previously, Decision Trees are non-linear classifiers that partition the feature space into rectangular regions. The model learns a hierarchy of "if–then" rules by recursively splitting data at specific thresholds.

In our implementation, the model utilized Gini Impurity as the splitting criterion, which aims to maximize the homogeneity of the target rank in the resulting nodes:

$$\text{Gini} = 1 - \sum_i p_i^2.$$

This structure allows the model to capture complex, conditional interactions between features. For example, the tree can learn that a high KDA combined with high CS/min predicts a Diamond rank, whereas a high KDA combined with low CS/min might predict a Silver rank. These are contextual nuances that linear models cannot easily represent.

2.4 Random Forest

Random Forests are an ensemble learning technique that build multiple Decision Trees using two layers of randomness: bootstrap sampling of rows and random feature selection at each split. By averaging the predictions of many decorrelated trees, Random Forests drastically reduce the variance present in a single large Decision Tree. This makes them highly robust to noise and better suited for modeling complex, non-linear relationships such as interactions between KDA, CS/min, gold income, and objective control.

2.5 AdaBoost

AdaBoost (Adaptive Boosting) builds an ensemble of weak learners, usually shallow Decision Trees, by training each new model to focus increasingly on the errors made by previous ones. Misclassified samples receive higher weights, which forces later trees to concentrate on the most difficult parts of the feature space. This sequential pattern makes AdaBoost effective when data contains subtle patterns, but it also makes the model sensitive to noise and highly vulnerable to class imbalance.

2.6 Gradient Boosting

Gradient Boosting builds an ensemble step-by-step, where each new tree is trained to predict the residual errors (negative gradients) of the current model. Unlike AdaBoost, Gradient Boosting uses a more stable optimization process and typically incorporates strong regularization parameters such as learning rate, sub-sampling, and maximum tree depth. These mechanisms make Gradient Boosting more flexible and stable than AdaBoost, while still capable of capturing intricate nonlinear structure.

3 Data Analysis

3.1 Overview of Initial Data

The dataset consisted of multiple `.csv` files, with `MatchStatsTbl.csv` serving as the primary table containing over 100,000 entries and approximately 50,000 unique matches. Additional files such as `ChampionTbl.csv` were used for mapping latent numerical identifiers to interpretable values (e.g., mapping `ChampionId = 1` to the champion "Annie"). Each row represented a player's match-level statistics, and only features describing in-game performance were retained for model training.

3.2 Data Cleaning

To ensure consistency, the dataset was filtered to include only "classic" gamemodes, excluding fast-paced or non-standard modes where gameplay statistics differ significantly from ranked matches. Since the raw data did not correctly distinguish between the "support" and "bottom" player roles, we manually mapped champions typically associated with these roles to correct the labeling. Additional cleaning steps included removing HTML tags from item-related features, discarding entries with invalid champion identifiers, and handling infinite values assumed to originate from data collection errors. Finally, all features were converted to appropriate data types and standardized in naming for clarity.

3.3 Feature Engineering

To enhance both visualization and model performance, several derived features were created. These included KDA ratio (a measure of kills and assists relative to deaths) and other performance metrics such as *GoldPerMin*, *CSPerMin*, *DmgPerMin*, *VisionPerMin*, *DmgPerGold*, and *DmgEfficiency*. Means of these variables were also computed, as they served as important aggregated indicators of player performance for the training phase.

3.4 Data Findings and Insights

Visualization of the cleaned dataset revealed that champion pick rates were relatively uniform, aside from a few popular outliers likely influenced by the meta during the data collection period. We also identified moderate to strong positive correlations between several performance metrics (e.g., vision score, kills, KDA, CS per minute) and match victories, as well as strong negative correlations between deaths and winning. Distributions for metrics such as KDA, total damage, and game duration appeared approximately normal. Lastly, the dataset contained a disproportionate number of "Master" ranked games compared to other ranks, an imbalance that was taken into consideration during model training and evaluation. Source code and graphs can be located at this Github Repo [AP25].

4 Model Analysis

Multiple models were implemented and trained on the processed dataset, including vanilla Logistic Regression, Ridge Classifier CV, and several tree-based methods such as decision trees, random forests, AdaBoost, and gradient boosting classifiers. Each model was evaluated using standard metrics including accuracy, balanced accuracy, F1 score, recall, and precision. Confusion matrices were visualized to assess class-specific performance, revealing that logistic regression models struggled with class imbalance and frequently misclassified minority ranks. Table 1 and 2 show the accuracies and balanced accuracies while other metrics and confusion matrices can be found on the Github Repo [AP25]

	LogReg	Ridge	DecTree	RandForest	AdaBoost	GradBoost
Accuracy	0.453	0.490	1.000	1.000	0.502	0.730
Balanced Accuracy	0.165	0.193	1.000	1.000	0.205	0.596

Table 1: Training set accuracy and balanced accuracy for each model.

	LogReg	Ridge	DecTree	RandForest	AdaBoost	GradBoost
Accuracy	0.449	0.490	0.939	0.903	0.504	0.714
Balanced Accuracy	0.163	0.196	0.901	0.817	0.205	0.556

Table 2: Testing set accuracy and balanced accuracy for each model.

4.1 Logistic Regression

The Logistic Regression model served as our baseline linear classifier, but its performance indicated significant limitations. The model achieved a training accuracy of 45.3% and a testing accuracy of 44.9%. The close proximity of these two scores suggests that the model is underfitting; the linear decision boundary is simply too rigid to capture the complex, non-linear nuances of League of Legends game play. Furthermore, the model's balanced accuracy on the test set was only 16.3%. This low score reveals a severe struggle with class imbalance, as the model likely defaulted to predicting the most frequent ranks (such as Master or Gold) while failing to correctly identify minority classes like Iron or Challenger.

4.2 Ridge Regression

Using Cross-Validation (CV) to select the optimal regularization parameter, the Ridge Classifier offered a modest improvement over the baseline. The model achieved a testing accuracy of 49.0% and a balanced accuracy of 19.6%. While the approximately 4% boost in accuracy suggests that regularization helped generalize the model better than standard Logistic Regression, the results ultimately confirm the limitations of linear models in this domain. The mapping from game statistics (such as KDA, CS/min, and vision score) to player rank appears to be non-linear, meaning that no straight line or hyperplane can effectively separate the ten distinct ranks.

4.3 Decision Tree

The Decision Tree model demonstrated a dramatic improvement in performance compared to linear baselines, identifying it as the superior approach to this task. The model achieved a training accuracy of 100% and a testing accuracy of 93.9%. Although a perfect training score is often a sign of overfitting, the exceptionally high test accuracy indicates that the model successfully learned robust generalizable rules rather than simply memorizing noise. The balance of precision of the test was also notably high at 90.1%. These results prove that the relationship between game statistics and rank is defined by strict statistical thresholds and non-linear interactions, which the Decision Tree was able to map with high precision. We show the accuracy and balanced accuracy below for our discussion, while other metrics can be found within the GitHub repository of our work.

4.4 Random Forest

The Random Forest model did not outperform the single Decision Tree in this dataset. Although a 100% training accuracy indicates extremely high model capacity, the 9% drop to 90.3% on the test set suggests mild overfitting. The model still performed well overall, but it did not match up to the Decision Tree's 93.9% test accuracy. The lowered balanced accuracy indicates that Random Forests struggled more than expected on minority classes such as Grandmaster and Challenger. The likely cause is that bootstrapping diluted these minority ranks even further, making it harder for trees to learn the patterns in these ranks. Overall, Random Forests provided a strong generalization, but the dataset's rank imbalance favored a single decision tree.

4.5 AdaBoost

AdaBoost performed poorly in this task relative to other tree-based models. The incredibly low balanced accuracy tells us that AdaBoost almost completely failed to identify minority ranks and defaulted to the majority ranks. Its training accuracy is just over 0.5, meaning the model struggled even before generalization was considered. The main issue likely stems from the dataset's heavy class imbalance, combined with AdaBoost's aggressive weighting strategy. Once the algorithm concentrates too heavily on rare, noisy, or overlapping samples, each additional stump adds onto earlier mistakes,

which deteriorates performance. Overall, AdaBoost was not well-suited for this dataset due to its high sensitivity to imbalance and noise.

4.6 Gradient Boosting

Gradient Boosting had moderate performance, better than the linear model and AdaBoost, but it was much weaker than Decision Trees and Random Forests. This tells us that the model was able to successfully learn meaningful structure, but it wasn't able to model the full complexity of rank prediction. The drop from training to testing accuracy reflects some overfitting, though not as extreme as pure decision trees. Gradient Boosting captured non-linear patterns more effectively than linear classifiers, but lacked the capacity that was necessary to match the high performance of the Decision Tree model. The moderate balanced accuracy shows improvement over AdaBoost, but still some difficulty separating rare ranks. Gradient Boosting was solid but not competitive with the best-performing tree-based methods that were observed.

5 Best Model Selection

Among the models, vanilla decision trees demonstrated the most consistent performance across both training and testing sets. The decision tree model achieved high accuracy while maintaining a better balance across different ranks, as observed in its confusion matrices. The model was able to capture non-linear relationships between features and effectively distinguish between ranks, making it the most suitable choice for this task. Based on these results, the vanilla decision tree was selected as the primary model for predicting player ranks in League of Legends matches.

6 Streamlit Application + Submission

Finally, utilizing the selected model a Streamlit application was created. The purpose of the application is to showcase the ability of the top performing model. One feature is a user can upload their league of legends stats and a machine learning model will attempt to predict the rank of the user per game. Another feature is that you can input a match ID and the application will predict the ranks of each player in the game.

A video demo can be located with this link. [Pag25]
Github code can be located with this link. [AP25]
Presentation for class can be located with this link.

References

[Act25] ActivePlayer.io. "league of legends.". https://activeplayer.io/league-of-legends/, Nov 2025.

[AP25] Jennifer Lee Navya Katraju Saaketh Akula Al Pagar, Kaw Bu. "comp560 final project.". https://github.com/Icefirez1/COMP560_Final_Project, Nov 2025.

[Pag25] Al Pagar. "comp560 final project demo.". https://youtu.be/csOy3wOU28U, Nov 2025.

[Sma25] Nathan Smallcalder. "lol match history & summoner data – 80k matches.". https://www.kaggle.com/datasets/nathansmallcalder/lol-match-history-and-summoner-data-80k-matches/data, Nov 2025.