Experimental Comparison of Classifiers for Predicting League of Legends Match Outcomes at 15 Minutes

Rubem Valadares de Almeida

Graduate Program in Computer Science, Federal University of Espírito Santo (UFES), Vitória, ES, Brazil

Abstract

This paper presents an experimental comparison of five machine learning algorithms for the task of predicting the winner in League of Legends (LoL) matches. The distinctive feature of this study lies in the use of a restricted dataset, containing only information collected at the 15-minute mark, simulating a real-time prediction scenario with partial information. The evaluated classifiers were: Decision Tree (DT), K-Nearest Neighbors (KNN), Multi-Layer Perceptron (MLP), Random Forest (RF), and a custom *ensemble* called Heterogeneous Boosting (HB). The experimental procedure was conducted using nested cross-validation, with 3 repetitions of 10 external folds for testing and 4 internal folds for hyperparameter optimization via Grid Search. Results indicate that MLP achieved the highest average accuracy (75.31%), closely followed by HB (75.16%) and RF (74.66%). Hypothesis tests (Nadeau and Bengio's t-test and Wilcoxon) confirmed that although MLP, RF, and HB performed statistically better than DT and KNN, there was no significant statistical difference among them, suggesting that all three are equally effective for this early prediction task. The complete source code and data used are publicly available for experiment reproduction.

Keywords: Machine Learning, League of Legends, Pattern Recognition, Classification, Intelligent Systems, Sports Prediction

1. Introduction

Esports represent a global phenomenon, moving a billion-dollar industry and attracting millions of viewers [1]. Among the most popular games, League of Legends (LoL), a Multiplayer Online Battle Arena (MOBA), stands out for its strategic and competitive complexity. The ability to predict the outcome of a match based on its initial state is of great interest to analysts, teams, and the betting market.

This work focuses on predicting the winning team in a LoL match using a restricted subset of

Email address: rubem.almeida@edu.ufes.br (Rubem Valadares de Almeida)

data, collected specifically at the 15-minute mark. This approach simulates a practical scenario where decisions must be made with partial information, long before the match outcome is decided.

The main objective is to conduct a rigorous experimental comparison among five classification techniques: Decision Tree (DT), K-Nearest Neighbors (KNN), Multi-Layer Perceptron (MLP), Random Forest (RF), and a proposed ensemble method, Heterogeneous Boosting (HB). Through a nested cross-validation procedure and hypothesis testing, we aim to identify not only the model with the best average performance but also to assess whether the superiority of one method over the others is statistically significant.

2. Dataset

The dataset used in this study is fundamental for the analysis and training of the machine learning models. Below, we detail its main characteristics.

2.1. Domain Description

The study is based on data from League of Legends (LoL) matches, one of the most popular esports in the world. In LoL, two teams of five players compete to destroy the enemy base. Team performance can be measured through various metrics, such as accumulated gold, experience (XP), number of kills, among others. Predicting outcomes based on data collected in the early stages of the match is a relevant challenge, as it can indicate victory trends before an advantage becomes irreversible.

2.2. Definition of Classes and Features

The problem is modeled as a binary classification task. The target class, 'result', indicates the match winner, where '1' represents a blue team victory and '0' a red team victory.

For Task IV, only 5 features were used, representing the performance difference between the two teams at 15 minutes:

- **golddiffat15**: Gold difference.
- **xpdiffat15**: Experience points difference.
- **csdiffat15**: Creep score (minions killed) difference.
- **killsdiffat15**: Kills difference.
- **assistsdiffat15**: Assists difference.

All features are numeric and were standardized (mean 0 and standard deviation 1) before model training to avoid scale bias.

2.3. Number of Instances

The complete dataset contains 8152 instances, where each instance corresponds to a unique match from the 2021 season.

2.4. Class Distribution

The dataset presents a slight class imbalance:

- **Class 1 (Blue team win)**: 4336 instances (53.19%).
- **Class 0 (Red team win)**: 3816 instances (46.81%).

This imbalance was considered in the experimental procedure by using stratified cross-validation, ensuring that the class proportion was maintained in each training and test fold.

3. The Heterogeneous Boosting Method

For this comparative analysis, a custom *ensemble* method, Heterogeneous Boosting (HB), was implemented. HB was designed to combine the predictions of multiple heterogeneous base classifiers, seeking a more robust and generalist model.

HB consists of four distinct machine learning algorithms, implemented using the Scikit-learn library [2] and used with their default parameters:

- Decision Tree (DecisionTreeClassifier)
- Gaussian Naive Bayes (GaussianNB)
- Multi-Layer Perceptron (MLPClassifier)
- K-Nearest Neighbors (KNeighborsClassifier)

The HB prediction process is based on simple majority voting. For a given instance, each of the N ensemble estimators (where N is a hyperparameter) makes a prediction. The final class is the one that receives the most votes. In case of a tie, the tiebreaker is the majority class observed in the training set, a simple strategy to resolve ambiguities without adding complexity to the model. The method's pseudocode is presented below.

```
procedure HBEnsemble(X_train, y_train, X_test, n_estimators)
    // Training
    base_learners = [DT, NB, MLP, KNN]
    n_base_learners = length(base_learners)
    trained_models = []

    for i from 1 to n_estimators:
        learner = base_learners[i % n_base_learners]
        model = train(learner, X_train, y_train)
        add model to trained_models

    // Prediction
    all_predictions = []
    for model in trained_models:
        predictions = predict(model, X_test)
        add predictions to all_predictions

    final_predictions = []
    for each sample_index:
        votes = get_votes_for_sample(all_predictions, sample_index)
        if tie_in_votes(votes):
            predict majority_class_from_y_train
        else:
            predict most_voted_class

    return final_predictions
end procedure
```

4. Description of the Experiments and Results

The experimental procedure was designed to ensure a robust and unbiased evaluation of the classifiers, using Scikit-learn [2] implementations and nested cross-validation for hyperparameter optimization and testing.

The evaluation was performed through 3 repetitions of stratified 10-fold cross-validation (external loop). For each external fold, a grid search with 4-fold cross-validation (internal loop) was used to find the best hyperparameters for each model. The tested hyperparameters were:

- **DT**: `criterion` (`gini`, `entropy`), `max_depth` (5, 10, 15, 25).
- **KNN**: `n_neighbors` (1, 3, 5, 7, 9).
- **MLP**: `hidden_layer_sizes` ((100,), (10,)), `alpha` (0.0001, 0.005), `learning_rate` (`constant`, `adaptive`).
- **RF**: `n_estimators` (5, 10, 15, 25), `max_depth` (10, None).
- **HB**: `n_estimators` (5, 10, 15, 25, 50).

To ensure reproducibility, random seeds (`random_state`) were set to `36854321` for cross-validation and `13` for classifiers with stochastic components.

4.1. Classifier Results

Table 1 summarizes the performance of the five classifiers in terms of average accuracy, standard deviation, and the 95% confidence interval over the 30 test runs.

Table 1: Accuracy results of the classifiers.

Method	**Mean**	**Std. Dev.**	**Lower Bound (95% CI)**	**Upper Bound (95% CI)**
DT	0.7500	0.0146	0.7448	0.7552
KNN	0.7362	0.0141	0.7311	0.7412
MLP	0.7531	0.0158	0.7474	0.7587
RF	0.7466	0.0153	0.7411	0.7521
HB	0.7516	0.0168	0.7456	0.7576

MLP achieved the highest average accuracy, slightly outperforming the HB and RF *ensemble* methods. KNN had the lowest performance among the evaluated models. Figure 1 illustrates the distribution of accuracies for each classifier through a boxplot.

4.2. Statistical Analysis

To determine whether the observed performance differences are statistically significant, paired hypothesis tests were performed: the corrected t-test of Nadeau and Bengio [3] and the Wilcoxon test [4]. Table 2 presents the obtained p-values, with values rejecting the null hypothesis ($p < 0.05$) in bold.

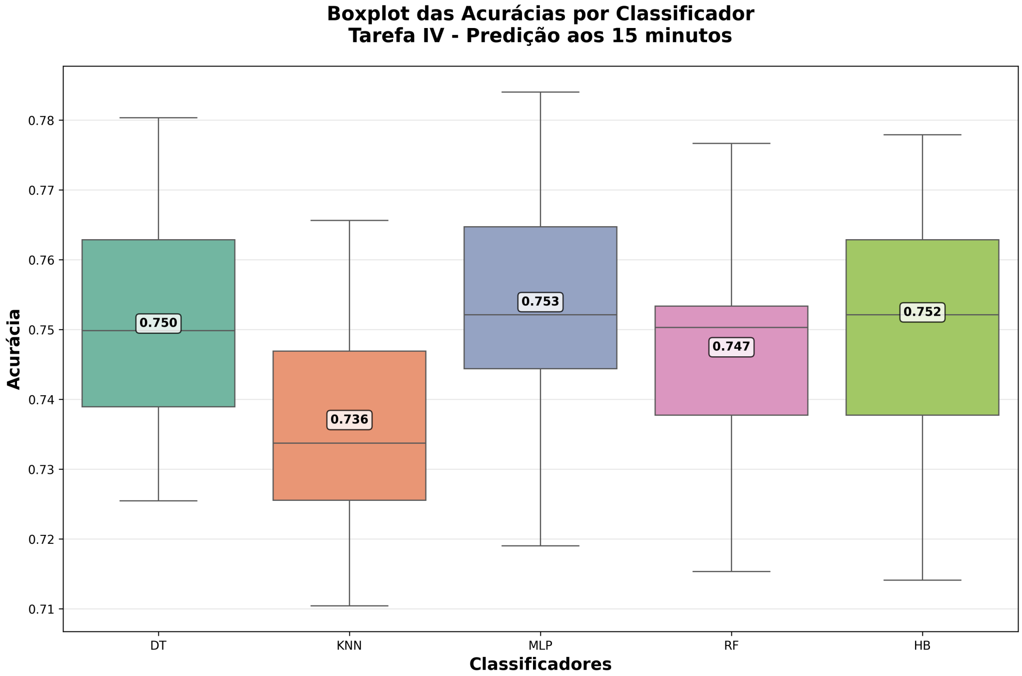

Figure 1: Boxplot of classifier accuracies over the 30 test folds.

Table 2: P-value table. Upper triangle: corrected t-test. Lower triangle: Wilcoxon test.

	DT	**KNN**	**MLP**	**RF**	**HB**
DT		**0.0000**	0.3014	0.1693	0.6128
KNN	**0.0000**		**0.0000**	**0.0000**	**0.0000**
MLP	**0.0381**	**0.0000**		**0.0016**	0.5898
RF	**0.0002**	**0.0000**	0.0617		**0.0076**
HB	0.2882	**0.0000**	0.3204	**0.0011**	

The statistical analysis reveals that MLP, RF, and HB were significantly superior to DT and KNN. However, the null hypothesis that there is no performance difference among MLP, RF, and HB could not be rejected by most tests, indicating that, statistically, their performances are equivalent for this task.

5. Conclusions

This work conducted a detailed experimental analysis for predicting outcomes in League of Legends matches using only data from the first 15 minutes of play.

5.1. General Analysis of Results

The results showed that it is possible to achieve an accuracy of approximately 75% using only a limited subset of early-game features, demonstrating the high predictive power of these variables. The MLP model achieved the highest average accuracy, although statistical tests did not indicate significant superiority over the RF and HB *ensembles.* This suggests that, for this specific task, all three models are competitive choices. The simpler models, DT and KNN, performed statistically worse, indicating that the complexity and generalization capacity of more advanced models are advantageous.

5.2. Contributions of the Work

The main contribution of this study is the rigorous and comparative evaluation of different classification approaches in a challenging and practical early prediction scenario. Additionally, the implementation and evaluation of the Heterogeneous Boosting (HB) *ensemble* demonstrated that combining heterogeneous models can achieve competitive performance, comparable to established methods such as Random Forest [5].

5.3. Improvements and Future Work

As future work, it is suggested to explore a broader set of features, including player-specific data or map events. The use of deep learning techniques, such as recurrent neural networks (RNNs) to analyze the sequence of game events, may also offer performance improvements. Finally, applying these models in a real-time prediction system would be a natural step to validate their effectiveness in a production environment.

Code and Data Availability

To ensure the reproducibility of the experiments and facilitate future research, all source code used in this work, including algorithm implementations, experimentation scripts, and statistical analysis, is publicly available in the GitHub repository: `https://github.com/rubemalmeida/ml-lol-match-prediction.git`

The repository contains:

- Complete implementation of the Heterogeneous Boosting (HB) *ensemble*
- Scripts for nested cross-validation and hyperparameter optimization
- Code for generating result tables and plots
- Statistical tests (Nadeau and Bengio's t-test and Wilcoxon)
- Jupyter Notebook with complete exploratory analysis
- Detailed documentation for experiment reproduction
- Dataset used (`jogosLoL2021.csv`) with 8152 matches from the 2021 season

References

References

[1] S. Schudy, O. D. Kothgassner, eSports: a new industry, SHS Web of Conferences, 92 (2021) 04002.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12 (2011) 2825-2830.

[3] C. Nadeau, Y. Bengio, Inference for the Generalization Error, Machine Learning, 52(3) (2003) 239-281.

[4] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics bulletin, 1(6) (1945) 80-83.

[5] L. Breiman, Random Forests, Machine Learning, 45(1) (2001) 5-32.