Predicting LOL ARAM games outcomes using Decision Tree & Random forest classifier

October 2023

1. Introduction

League of Legends (LOL) is an online competitive battle arena game. The game consists of two teams with five players each. In the ARAM game mode, the map has only one lane and a team wins by destroying the other team's "Nexus". ARAM game mode is skill based, relying less on the economy and strategy aspect of the game, compared to the regular mode. As a result, a team's strength is represented by the combat skill of the players. This application focuses on predicting ARAM games outcomes using supervised machine learning methods. The next part will discuss problem formulation and explain the data points, features and labels used in this ML problem. The third part goes into details about the used methods.

2. Problem Formulation

2.1 Problem summary

The goal of this application is to predict game outcomes using supervised machine learning methods with participants' data that exist before the game starts.

2.2. Dataset

The dataset is collected and aggregated by the author using different techniques. There are 1670 datapoints, each representing a game match.

2.3. Features and labels

The game ids, player ids, and player names are first collected from Riot Games official API. The API can only gather matches data for a specific player in a query. To generate a large enough dataset, a short list of players is created initially. This list is looped through and appended with new players, which are the queried matches' participants. The process is repeated until a large enough dataset is generated, given its time-consuming nature. This API, however, does not provide detailed players' aggregated statistics. There are many dedicated LoL statistics sites and blitz.gg proves to be most suitable for this application. Web scraping is thus implemented using BeautifulSoup to collect players' data. These individual statistics are then aggregated using the general formula:

$$\sum_{\text{team0 statistics}} - \sum_{\text{team1 statistics.}}$$

This is done in order to reduce dimensions and maintain focus on the difference in strength between the teams.

Features	Explanation	Type
Winrate	Difference in players' winrate between two teams	float
Kill death assist ratio	Difference in average kill death ratio between two teams ((K+A)/D)	float

Damage dealt per minute	Difference in average damage per minute between two teams	float
Mastery	Difference in mastery points between two teams.	float

The features are chosen with focus on teams' strength, given their availability. The application is meant to predict the game outcomes using only existing data from the participants. The four features chosen are most impactful and widely used in representing strength. Although winrate, kda ratio, and mastery points are straightforward statistics, the damage dealt per minute is suitable in the setting of ARAM games, where it has more impact to the outcome than in regular game mode.

The labels are binary values representing which teams won the game. The positive case is when the first team won (team0). There are 780 points belonging to class 0 and 890 in class 1.

Classes (values)	Meaning
0	Team 1 wins
1	Team 0 wins

3. Methods

3.1 Constructing training, validation, and test sets

The widely used ratio 60-20-20 for training, validation, and test is used. The data is splitted randomly into 80% training-validation set, and 20% test set. This separate test set ensures that the model is assessed on unseen data. Because the dataset is small and simple, with few features, k-fold cross-validation is employed during the training process to assess model performance. k=4 is chosen to split 60-20 between the training and validation set.

3.2 Decision tree classifier

To choose the most suitable model, scatterplots between the features and the label are constructed using matplotlib to see if there are linear relationships between them.

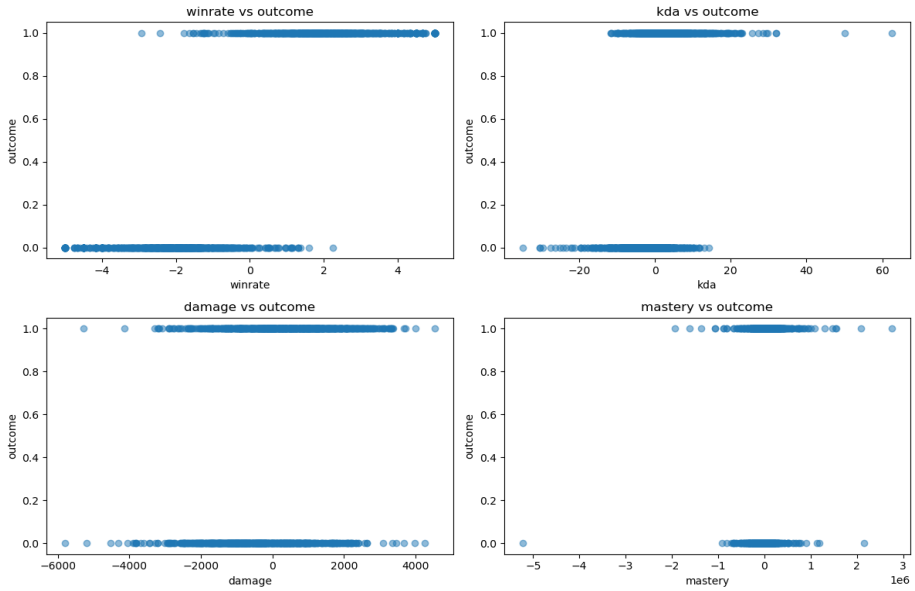


Figure 1: Scatterplots between the features and the label

It appears that there is no clear linear relationship between the features and the label. Therefore, the decision tree classifier is chosen in this situation.

Decision trees create tree-like structures to split the data based on the features. Each branch represents a decision. The data is splitted until the algorithm decides to stop. The final nodes in the tree are called leaf nodes, and they represent the class labels. (Jung, p. 97-98) The application utilizes the python Sklearn library built-in decision tree model.

Given the dataset's inherent balance in class distribution, the choice between Gini impurity and information gain does not significantly impact the model performance. Hyperparameter tuning is conducted to confirm this theory. The model used is sklearn decision tree classifier. The model is initialized twice with the 'criterion' parameter alternating between 'gini' (gini impurity) and 'entropy' (information gain). The models are trained and validated using k-fold. After multiple runs, it shows no noticeable difference which loss function is used.

```
Tree results with gini
Training accuracy score average:  1.0
validation accuracy average:  0.9191579750553625
Tree results with entropy
Training accuracy score average:  1.0
validation accuracy average:  0.9185727972645806
```

Figure 2. Result of training decision tree with each loss function

As a result, the default Gini impurity is used as the split quality measure. Gini impurity measures the probability of misclassifying a randomly chosen element if it were randomly classified according to the class distribution in the dataset. It is calculated using the following formula:

$$1 - \Sigma i \, pi^2$$

3.3 Random forest classifier

The second method used is random forest classifier. Random forest is an algorithm that combines output of multiple decision trees. Its output is the class selected by most trees. This method is chosen because of the lack of linear relationship between the features and label. Furthermore, random forests tend to obtain better results than decision trees, given its ensemble approach. The sklearn random forest classifier is used,

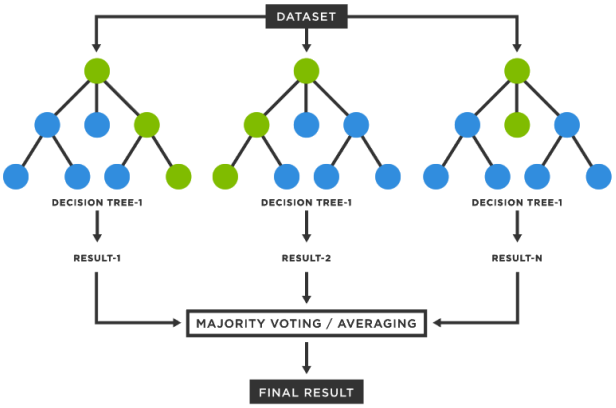


Figure 3. Random forest model [6]

Random forest is a collection of decision trees, thus the loss function is the same as for decision trees. A similar test is again conducted as above. Two random forest classfier models are fitted using different loss functions. The results again show no difference which loss function is used. The default gini impurity is thus chosen.

```
Forest results with gini
Training accuracy score average:  1.0
validation accuracy average:  0.9467057932601287
Forest results with entropy
Training accuracy score average:  1.0
validation accuracy average:  0.9473067479031129
```

Figure 4. Result of training random forest with each loss function

4. Results

Since this is a classification problem, accuracy score is used instead of training and validation error to measure model performance. The models are cross validated using k-fold, and the accuracy score of each run is obtained. The average of these scores is then calculated to measure the model overall performance. Furthermore, the score standard deviation is calculated to avoid having high variation in model performance in different runs.

```
Tree results:
Training accuracy score average:  1.0
Training accuracy score deviation:  0.0
validation accuracy average:  0.9167555907427167
validation accuracy standard deviation:  0.010578075233472962
```

Figure 5. Decision tree training and validation accuracy score

```
Forest results:
Training accuracy score average:  1.0
Training accuracy score deviation:  0.0
validation accuracy average:  0.9484914460775876
validation accuracy standard deviation:  0.014663367784924311
```

Figure 6. Random forest training and validation accuracy score

Although both models perform well on the data, obtaining 100% accuracy on the training data and above 90% validation accuracy scores, the random forest classifier outperforms the decision tree by around 3%. Random forest classifier has a 94% average validation accuracy score. Both models also have low standard deviation in accuracy across different runs. This indicates that the model performance does not vary highly. As a result, the random forest classifier is chosen as the final model. The model is then trained again using the whole training and validation set. The test set set aside in the beginning is now used to estimate the model performance. The model has a 94.6% accuracy on this new unseen data. This is a great result.

5. Conclusion

This report applies decision tree classifier and random forest classifier to predict an ARAM match outcome, using the teams' players average strength metrics. The models are chosen to

account for non-linear relationships in the data. The features are chosen based on domain knowledge. Given the results, it is clear that the game outcome can be predicted fairly satisfactorily. However, the results are quite optimistic and leave room for improvement. Firstly, the process of data collection (as mentioned in features and labels section) leads to the dataset containing many matches from the same set of players. Nevertheless, a complete dataset containing random matches from random players requires prolonged data collection, or special permission from Riot Games. Moreover, the features selection and engineering process may benefit from more sophisticated methods. For example, more related features could be used. Dimensionality reduction methods, such as PCA, can be conducted with new features. Such data would take more time to collect and clean. In conclusion, although the result is quite satisfactory, a more complete dataset can greatly benefit the performance of predicting unseen data.

6. References

[1] Blitz.gg 2023, 'League of Legends Builds, Runes, and Stats', blitz.gg, https://blitz.gg/lol , (Accessed: September 15, 2023).
[2] Jung, A. "Machine Learning: The Basics," Springer, Singapore, 2022. Access date March 10, 2022
[3] Riot Games 2023, 'Riot Games API Documentation', Riot Games Developer Portal, https://developer.riotgames.com/apis , (Accessed: September 15, 2023).
[4]Sklearn.tree.DecisionTreeClassifier. Retrieved September 15, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[5]Sklearn.ensemble.RandomForestClassifier. Retrieved. October 9, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[6] Spotfire.com What is a random forest? https://www.spotfire.com/glossary/what-is-a-random-forest

7. Appendix

The code for collecting data from Riot API and for web scraping is done in my personal project. Link to the project repo: https://github.com/anhtuan18602/ARAMprediction
The code for querying from Riot API is in the “riot.py” file. The code for web scraping is in the “scraper.py” file. The data is then transformed, as stated in chapter 2.3, and saved into a SQLite database. For simpler model evaluation, later work will be done in a jupyter notebook. The dataset is extracted to a csv file. The model training and testing code is appended below:

league

October 9, 2023

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, KFold
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, mean_squared_error,␣
 ↪confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

[2]:
```
data = pd.read_csv("league.csv")
data.head(5)
X = data[["m_winrate","m_kda", "m_dmg", "m_mastery"]]
X.columns = ["winrate","kda", "damage", "mastery"]
y = data["m_win"]
```

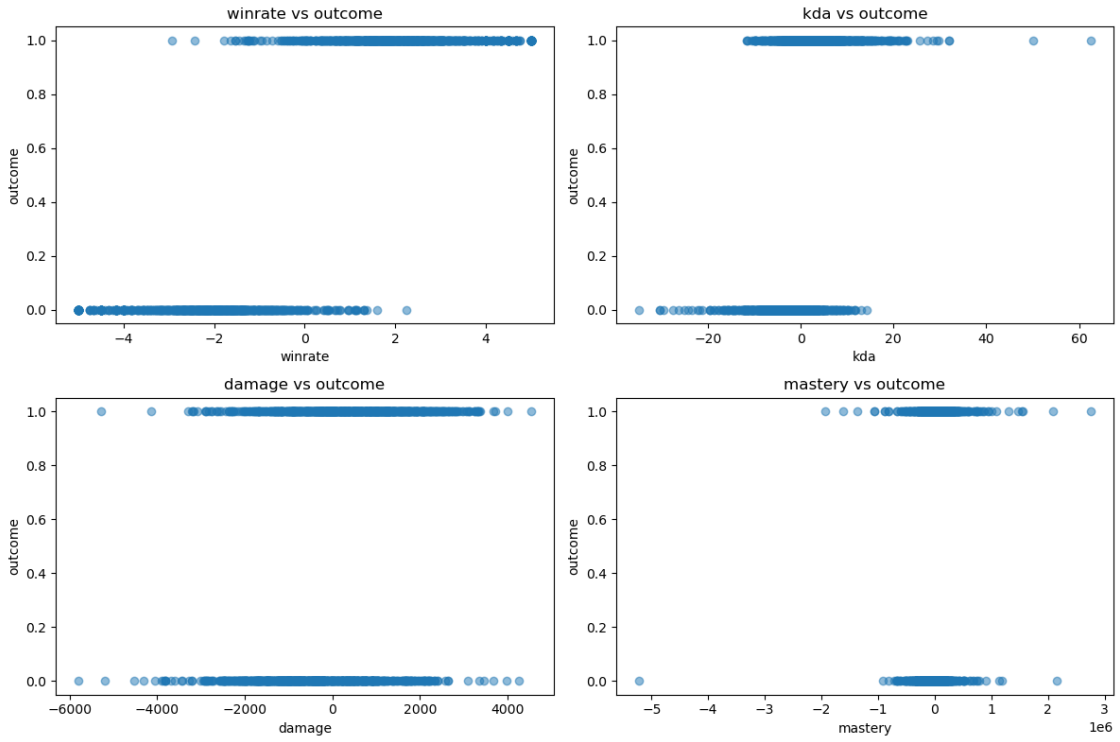
[3]:
```
num_features = len(X.columns)
num_rows = num_features // 2 + num_features % 2  # Calculate the number of rows␣
 ↪for subplots
fig, axes = plt.subplots(num_rows, 2, figsize=(12, num_rows * 4))  # Create␣
 ↪subplots

# Flatten axes for easy iteration
axes = axes.ravel()

# Loop through feature columns and create scatter plots
for i, feature in enumerate(X.columns):
    ax = axes[i]
    ax.scatter(X[feature], y, alpha=0.5)
    ax.set_title(f"{feature} vs outcome")
    ax.set_xlabel(feature)
    ax.set_ylabel("outcome")

plt.tight_layout()

# Show the plots
plt.show()
```



[4]:
```
X_train_val, X_test, y_train_val, y_test = train_test_split(X,y, test_size = 0.
↪2)
k = 4
```

[15]:
```
# Hyperparameter tuning
for criteria in ["gini", "entropy"]:
    kf = KFold(n_splits=k, shuffle=True)

    tree_training_scores = []
    tree_val_scores = []
    for train_index, test_index in kf.split(X):
        dec_tree = tree.DecisionTreeClassifier(criterion = criteria)
        X_train, X_val = X.iloc[train_index], X.iloc[test_index]
        y_train, y_val = y.iloc[train_index], y.iloc[test_index]

        dec_tree.fit(X_train, y_train)

        y_pred = dec_tree.predict(X_val)

        training_scores = accuracy_score(y_train, dec_tree.predict(X_train))
        val_scores = accuracy_score(y_val, y_pred)

        tree_training_scores.append(training_scores)
```

```
        tree_val_scores.append(val_scores)
    print("Tree results with", criteria)
    tree_scores_tr = np.array(tree_training_scores)
    tree_tr_avg = tree_scores_tr.mean()
    print("Training accuracy score average: ",tree_tr_avg)


    tree_scores_val = np.array(tree_val_scores)
    tree_val_avg = tree_scores_val.mean()
    print("validation accuracy average: ",tree_val_avg)
```

```
Tree results with gini
Training accuracy score average:  1.0
validation accuracy average:  0.9191579750553625
Tree results with entropy
Training accuracy score average:  1.0
validation accuracy average:  0.9185727972645806
```

[22]:
```
kf = KFold(n_splits=k, shuffle=True)

tree_training_scores = []
tree_val_scores = []
for train_index, test_index in kf.split(X):
    dec_tree = tree.DecisionTreeClassifier()
    X_train, X_val = X.iloc[train_index], X.iloc[test_index]
    y_train, y_val = y.iloc[train_index], y.iloc[test_index]

    dec_tree.fit(X_train, y_train)

    y_pred = dec_tree.predict(X_val)

    training_scores = accuracy_score(y_train, dec_tree.predict(X_train))
    val_scores = accuracy_score(y_val, y_pred)

    tree_training_scores.append(training_scores)
    tree_val_scores.append(val_scores)

print("Tree results:")
tree_scores_tr = np.array(tree_training_scores)
tree_tr_avg = tree_scores_tr.mean()
tree_tr_sd = tree_scores_tr.std()
print("Training accuracy score average: ",tree_tr_avg)
print("Training accuracy score deviation: ",tree_tr_sd)

tree_scores_val = np.array(tree_val_scores)
tree_val_avg = tree_scores_val.mean()
tree_val_sd = tree_scores_val.std()
```

```
print("validation accuracy average: ",tree_val_avg)
print("validation accuracy standard deviation: ",tree_val_sd)
```

```
Tree results:
Training accuracy score average:  1.0
Training accuracy score deviation:  0.0
validation accuracy average:  0.9167555907427167
validation accuracy standard deviation:  0.010578075233472962
```

[18]:
```
for criteria in ["gini", "entropy"]:
    kf2 = KFold(n_splits=k, shuffle=True)

    forest_training_scores = []
    forest_val_scores = []
    for train_index, test_index in kf2.split(X):
        forest = RandomForestClassifier(criterion = criteria)
        X_train, X_val = X.iloc[train_index], X.iloc[test_index]
        y_train, y_val = y.iloc[train_index], y.iloc[test_index]

        forest.fit(X_train, y_train)

        y_pred = forest.predict(X_val)

        training_scores = accuracy_score(y_train, forest.predict(X_train))
        val_scores = accuracy_score(y_val, y_pred)

        forest_training_scores.append(training_scores)
        forest_val_scores.append(val_scores)

    print("Forest results with", criteria)
    forest_scores_tr = np.array(forest_training_scores)
    forest_tr_avg = forest_scores_tr.mean()
    forest_tr_sd = forest_scores_tr.std()
    print("Training accuracy score average: ",forest_tr_avg)

    forest_scores_val = np.array(forest_val_scores)
    forest_val_avg = forest_scores_val.mean()
    forest_val_sd = forest_scores_val.std()
    print("validation accuracy average: ",forest_val_avg)
```

```
Forest results with gini
Training accuracy score average:  1.0
validation accuracy average:  0.9467057932601287
Forest results with entropy
Training accuracy score average:  1.0
validation accuracy average:  0.9473067479031129
```

[19]:
```
kf2 = KFold(n_splits=k, shuffle=True)

forest_training_scores = []
forest_val_scores = []
for train_index, test_index in kf2.split(X):
    forest = RandomForestClassifier()
    X_train, X_val = X.iloc[train_index], X.iloc[test_index]
    y_train, y_val = y.iloc[train_index], y.iloc[test_index]

    forest.fit(X_train, y_train)

    y_pred = forest.predict(X_val)

    training_scores = accuracy_score(y_train, forest.predict(X_train))
    val_scores = accuracy_score(y_val, y_pred)

    forest_training_scores.append(training_scores)
    forest_val_scores.append(val_scores)

print("Forest results:")
forest_scores_tr = np.array(forest_training_scores)
forest_tr_avg = forest_scores_tr.mean()
forest_tr_sd = forest_scores_tr.std()
print("Training accuracy score average: ",forest_tr_avg)
print("Training accuracy score deviation: ",forest_tr_sd)

forest_scores_val = np.array(forest_val_scores)
forest_val_avg = forest_scores_val.mean()
forest_val_sd = forest_scores_val.std()
print("validation accuracy average: ",forest_val_avg)
print("validation accuracy standard deviation: ",forest_val_sd)
```

```
Forest results:
Training accuracy score average:  1.0
Training accuracy score deviation:  0.0
validation accuracy average:  0.9484914460775876
validation accuracy standard deviation:  0.014663367784924311
```

[27]:
```
#Train the RandomForestClassifier on the entire training and validation data
forest = RandomForestClassifier()
forest.fit(X_train_val, y_train_val)
y_pred = forest.predict(X_test)
print("accuracy score on test set:", accuracy_score(y_test, y_pred))
```

```
accuracy score on test set: 0.9461077844311377
```

[ ]: