How do League of Legends Champions in Various Roles Differ in How They Contribute to a Game?

By: Aurelia Iskandar, Weng Jae Chin, Xin Yu Chuah, Yong Qi Soh

1 Aim **2**

2 Datasets **2**

3 Pre-processing and Data Wrangling **3**

4 Analysis Methods **4**

4.1 Methods 4

4.2 Training-Test Split 4

4.3 Preliminary Analysis 4

4.4 Modelling 5

4.4.1 Decision Trees 6

4.4.2 KNN 6

4.4.3 Final Evaluation 8

5 Discussion **8**

6 Evaluation **9**

1 Aim

“How do champions in various roles differ in how they contribute to a game?”

The aim of this investigation is to automatically identify the roles of cha09mpions in a game of League of Legends given match performance and to investigate the behavioural characteristics of each role. The results from this research may be beneficial to professional League of Legends e-sport teams to identify strong champions for each role to fill and round out a team composition for professional games. It will also be useful for League of Legends players to choose champions for each role given their champions’ strengths and weaknesses.

2 Datasets

The datasets used were match statistics from one randomly chosen player in different Challenger ranked matches in League Of Legends. We were given 3 datasets initially in CSV format recorded from different regions: North America, Korea, and Europe, in January 2022. The datasets given were of fairly good data quality, and only came with some missing values which are resolved in ‘Pre-processing and Data Wrangling’.

The match statistics consisted of data such as the champion used, the player’s deaths, kills, assists, and many more as shown in Figure 2.1.

Feature	Description	Type Check	Boundary Check
d_spell	summoner spell on d key	float	between 0.0 and 21.0, inclusive
f_spell	summoner spell on f key	float	between 0.0 and 21.0, inclusive
champion	champion being played	string	within our list of 141 champions that exist in the game
side	side of map player is on red/blue	string	either "Side.red" or "Side.blue"
assists	number of assists in match	integer	between 0 and 41, inclusive
damage_objectives	damage to objectives	integer	between 0 and 44.2k, inclusive
damage_building	damage to buildings	integer	between 0 and 70k, inclusive
damage_turrets	damage to turrets	integer	between 0 and 44.2k, inclusive
deaths	deaths in game	integer	between 0 and 21, inclusive
kda	k/d/a ratio in game	float	between 0 and 33, inclusive
kills	kills in game	integer	between 0 and 30, inclusive
level	level in game	integer	between 1 and 18, inclusive
time_cc	time crowd controlling others	integer	between 0 and 259, inclusive
damage_taken	total damage taken in game	integer	between 2594 and 78.9k, inclusive
turret_kills	turret kills in game	integer	between 0 and 9, inclusive
vision_score	vision score in game	integer	between 1 and 174, inclusive
damage_total	total damage in game	integer	between 4493 and 505k, inclusive
gold_earned	gold earned in game	integer	between 3093 and 28.6k, inclusive
role	role being played	string	either "Other" or "TopLane_Jungle"
minions_killed	total minions killed in game	string (enum)	either "Many" or "Few"

Figure 2.1: Features of the datasets

3 Pre-processing and Data Wrangling

To ensure the accuracy, completeness, consistency, timeliness, believability, and interpretability of the data is efficient, we performed multiple data wrangling techniques.

First, we ensure the believability of our data by checking if all values are of the correct type and format, and are of a believable value. The parameters we will be checking for each feature is based on the dataset provided and is described in Figure 2.1. We also used our domain knowledge in the game of League of Legends to ensure the accuracy and believability of our data.

To handle missing data, missing/null values were replaced with the median of each column. The median was chosen to ensure that data such as ‘kills’ and ‘deaths’ will remain whole.

Next, we standardised the data based on the player’s level at the end of the game and then normalised all values to be between 0 and 1 to be able to compare the other match statistics with each other. This is important because a higher level represents more time spent in a League of Legends match, which would result in likely having higher match statistics such as ‘damage_total’, ‘kills’, ‘assists’ when compared with statistics from a lower level.

We removed columns such as ‘side’, ‘champion’, and ‘damage_objectives’ based on domain knowledge and due to repetition in data. We determined that the side was unrelated to the role using our domain knowledge. ‘Damage_objectives’ have identical values as ‘damage_turrets’, only with more missing values. Champions were removed because it was too closely related to the role (each champion has a specific role) and we did not want our model to determine the role of the game based on the champion.

4 Analysis Methods

4.1 Methods

As our data predict discrete data from the column “role”, a classification technique is required. This is due to how classification techniques are only utilised when the expected output is in a particular category. Regression techniques were also not used as they predict continuous values whereas we are assigning records into discrete categories (“TopLane_Jungle” and “Other”). By operating a classification technique, we will be able to observe whether a player’s game performance can identify a player’s role in the match. Hence, this research uses algorithms such as K-nearest neighbours and decision trees.

4.2 Training-Test Split

Prior to the application of algorithms, the data was initially split into two subsets, training, and test data. The training dataset included 80% of the original data, while the testing dataset consists of the remaining 20%. The model fitting was done in the training data, while the test dataset was used to evaluate the model’s performance in predicting the player’s role based on their game performance. K-fold cross-validation was also used to ensure that our model is not reliant on the initial training-test split.

4.3 Preliminary Analysis

To further understand the relationship between a player's game performance and their roles, we initially analysed each of the columns in the data and its correlation with the column 'role'. We did this by calculating the mutual information scores for each column and by domain knowledge. The results can be seen in the table below. A high mutual information score would indicate a high correlation with the column 'role'.

From Figure 4.3.1, it can be seen that 'normalised_damage_total' has the highest correlation to role out of all the columns, whereas 'normalised_turret_kills' had the lowest correlation. Although it can also be said that the columns 'normalised_damage_building', 'normalised_damage_turrets', 'normalised_damage_taken', and 'normalised_gold_earned' show a correlation with 'role'. This implies that damage and gold could be good indicators for a role.

Features	**Mutual Information Scores**
normalised_d_spell	0.1507243663227542
normalised_f_spell	0.15235526966789417
normalised_assists	0.05351089781651618
normalised_damage_building	0.5670495733051201
normalised_damage_turrets	0.44394297210757716
normalised_deaths	0.016466505714185332
normalised_kda	0.1012102380031556
normalised_kills	0.025487556607684224
normalised_time_cc	0.06725963778379788
normalised_damage_taken	0.5875015014160732
normalised_turret_kills	0.011621519523195723
normalised_vision_score	0.10057504126357317
normalised_damage_total	0.6388727235441769
normalised_gold_earned	0.5373370121980837

Figure 4.3.1 Table of Features and Mutual Information Scores

4.4 Modelling

In this research, we utilised the k-nearest neighbour algorithm and decision trees.

We ran the KNN and decision tree with varying hyper-parameters to obtain the best combination of values that yield the best results. Namely, we varied the k for k-fold cross validation's k parameter to be between 5 inclusive and 12 exclusive, $k_{kfold} = [5, 12)$; mutual information (MI) threshold for feature selection to be between 0 inclusive and 0.3 exclusive, $MI = [0, 0.3)$; and KNN's k parameter to be between 3 inclusive and 7 exclusive, $k_{KNN} = [3, 7)$. Both algorithms were run with the varied parameters and we measured the accuracy and precision of each fold as well as calculated the mean accuracy and precision of all folds of a given combination; Figure 4.4.1 illustrates an example of how the overall process..

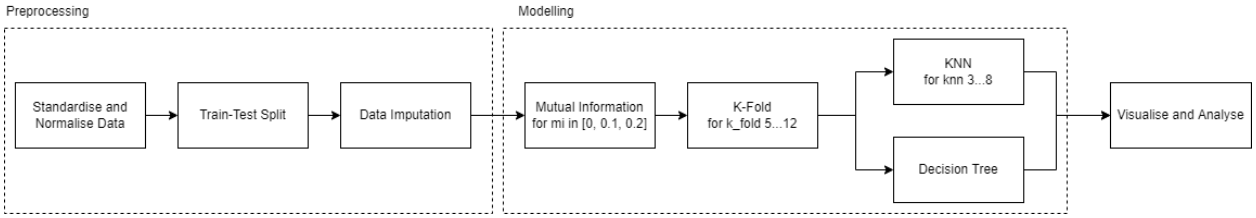

Figure 4.4.1 Illustration of our KNN and Decision Tree pipeline

4.4.1 Decision Trees

Since decision trees did not have any additional hyperparameters to tune, it was operated through thresholds {0, 0.1, 0.2} for feature selection.

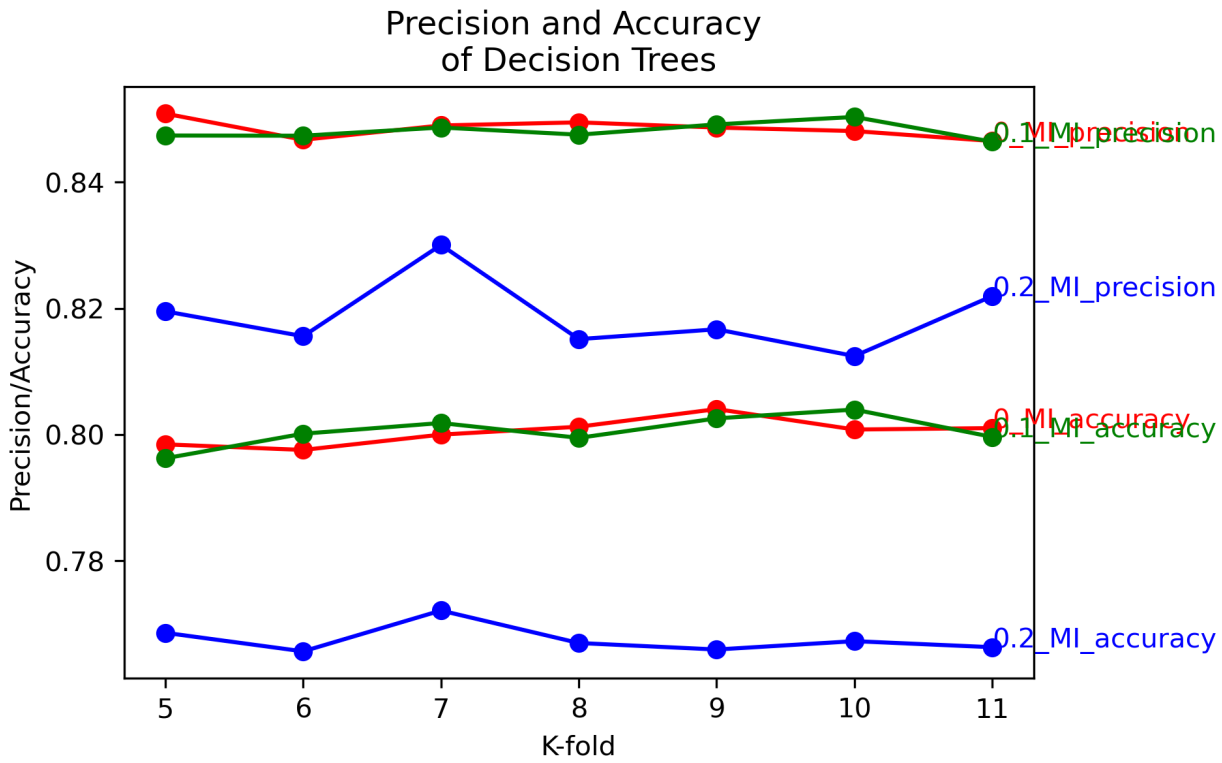

Figure 4.4.1: Results of running the Decision Tree algorithm with varying hyperparameters (ie. K-fold k parameter and Mutual Information Thresholds).

As seen from Figure 4.4.1 above, the model with MI Thresholds of 0 and 0.1 had better precision and accuracy. This implies that all or almost all of the features were used to determine role.

4.4.2 KNN

KNN resulted in $7 \times 3 \times 4 = 84$ hyperparameter combinations; the results of which are shown in Figure 4.4.2a.

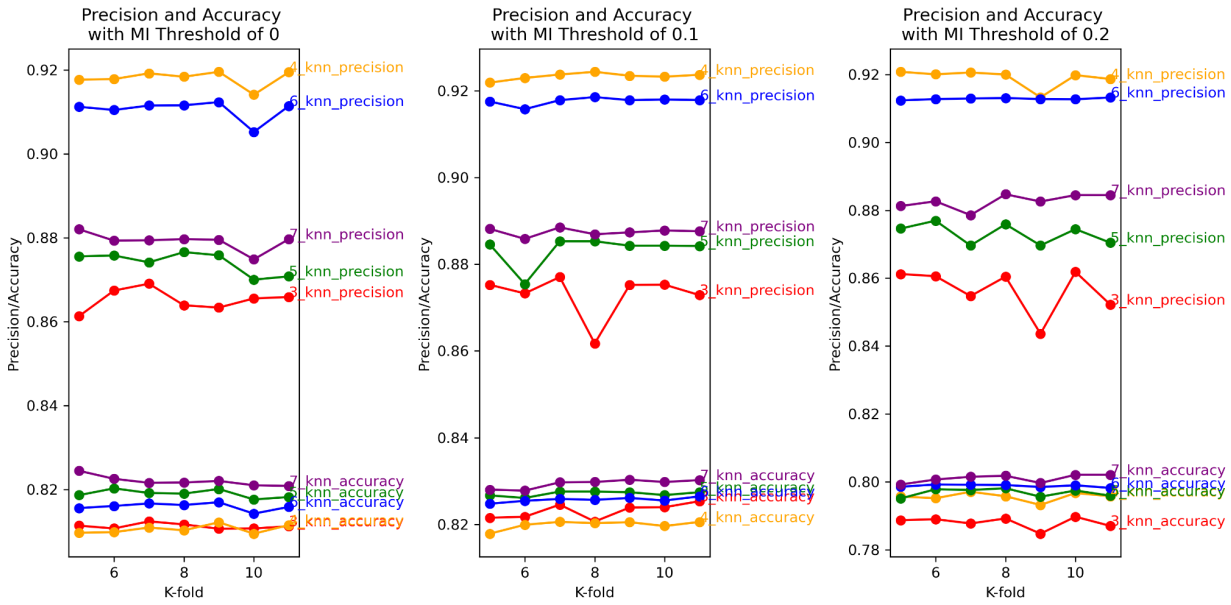

Figure 4.4.2a: Results of running the KNN algorithm with varying hyperparameters (ie. K-fold k parameter, Mutual Information Thresholds and KNN k parameter).

From the results obtained in Figure 4.4.2a, we concluded that the model with 4-NN and an MI Threshold of 0.1 had the best precision; however, the model with 7-NN and an MI Threshold of 0.1 had the best accuracy.

Comparing the results from 4.4.2a and 4.4.1, we can see that the KNN algorithm is more performant overall when evaluated on precision and accuracy. As such, we continued analysing and tuning our KNN models by simplifying Figure 4.4.2a to better compare the differences between the different k_{KNN}. If we value both precision and accuracy equally, and calculated the average of both ($\frac{precision + accuracy}{2}$), this produces Figure 4.4.2b.

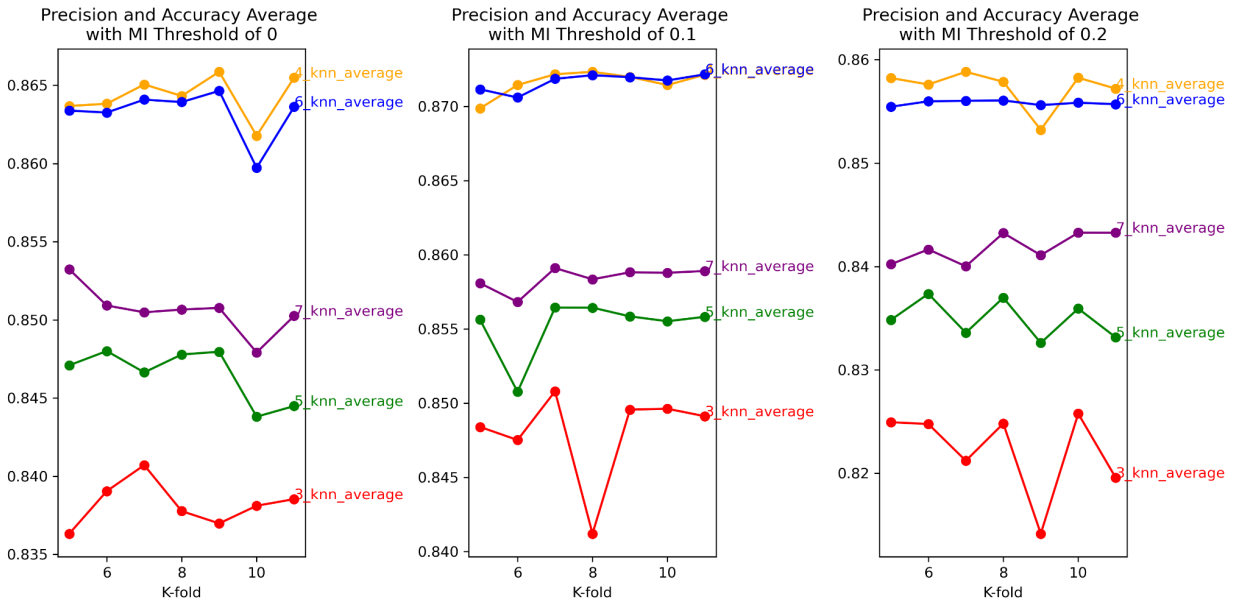

Figure 4.4.2b: Precision and Accuracy Average (ie. adding precision and accuracy and dividing by 2) with varying hyperparameters; equivalent to taking the average of the two lines from Figure 4.4.2a; for easier comparison of k_{KNN}.

From Figure 4.4.2b, it can be seen that the 4NN (yellow) and 6NN (blue) models with MI Threshold of 0.1 are most performant when evaluated on both precision and accuracy equally.

We believe that the 4NN model is better because, looking at Figure 4.4.2, the 4NN model has higher precision, and we want higher precision because it indicates lower false positives and we do not want false positives because it may indicate that champions were labelled with the wrong role, which would result in a false champion recommendation to the user. Furthermore, since k_{KNN} is smaller, the KNN algorithm will run faster than the 6NN model on bigger datasets.

4.4.3 Final Evaluation

	KNN	Decision Tree
Accuracy	0.8205959745341922	0.8008094985443232
Precision	0.9237393494843367	0.8511996452033571

Figure 4.4.3: Compares the best models from KNN and Decision tree

5 Discussion

In this section, we interpret the models and discuss the significance of the results.

The mutual information scores table (Figure 4.3.1) implies that some features from game performance may have a correlation with the roles in League of Legends. Some of these features are: ‘normalised_damage_total’, ‘normalised_damage_taken’ and ‘normalised_gold_earned’ (0.63887, 0.58750, 0.53734, respectively). This preliminary analysis suggests that damage and gold earned can be indicators of roles. However, since we have adapted the thresholds of feature selection for the KNN and decision tree models (to display the best possible accuracy and precision), the features that were used in our final model did not only include damage and gold. This suggests that game performance is not exclusively based on the damage dealt and gold earned but also on other different features such as spell usage, KDA, and vision score.

Between the KNN and decision tree models, it can be seen that KNN has a higher accuracy and precision score (0.8206, 0.9237) compared to the decision tree (0.8008, 0.8512). After evaluating the selected KNN model on the test set we acquired an accuracy and precision score of 0.8511 and 0.7830 respectively. From our selected KNN model, we observed fairly high accuracy and precision metrics. This indicates that the relationships between game performance and role are statistically significant, hence, we will be able to automatically predict a role based on the player’s game statistics.These results are valuable as it allows professional E-sport teams to be able to choose champions for each role with the best game performance.

6 Evaluation

In this section, we identify and examine the limitations of our experiment design and propose possible avenues for further research. Firstly, the distance between features in the k-nearest neighbours algorithm is limited as data is standardised on champion levels, which may not correlate well with some features. Future studies could incorporate game length in the datasets and standardise values on game length or other features that are more indicative of game progression. Secondly, there may be a significant correlation between features that we did not identify, this might cause more weight to be put on certain features. Future studies may wish to compare the correlation between features and ensure that only linearly independent features are in their models. Moreover, we are using a constant random seed for our k-fold cross-validation, which might cause bias in our model selection. Future extensions could run k-fold cross validation multiple times, or across different seeds to get an optimal performance score. Finally, the model we are using, k-nearest neighbour, tends to fall victim to the “curse of

dimensionality", which means that it doesn't perform well on high-dimensional inputs. Future studies could improve the performance of the model by using dimensionality reduction techniques such as Principal Component Analysis.