Ecamania Esports Project Report

AI Lab

First Name :	**Last Name :**	**Matricule :**
Jurquet	Nicolas	21211

Haute École ICHEC - ECAM - ISFSC

Academic year 2024 – 2025

1 Introduction

Ecamania Esports Project is a project that aims to find the best *League of Legends (LoL)* team to recruit for the ECAM team. It is asked to analyse a dataset of LoL games by training Machine Learning (ML) and Deep Learning (DL) models to:

— Determine, based on in-game statistics of a single player, whether their team won or lost the match.

— Identify the most important quantifiable variables that players should focus on to increase their chances of winning.

— Determine the most talented players Ecamania could recruit for the next season.

Considering the lack of time and setup problems I had, only the first point was completed and analysis/explainations were not developped as much as intended.

The methodology adopted was to first visualize and understand the dataset, then see which features could be relevant to predict the outcome of a game or its interaction with other features, a preprocessing step was then used, and finally models were trained and evaluated to predict the outcome of a game.

All the code created during the project is available on my GitHub repository.

2 Dataset visualization

Upon displaying some of the first rows of the `game_players_stats.csv` file, **28** columns were identified and a total of **374554** rows of player games. A total of **37459** unique games were played, with a total of **4953** players, and within **844** different teams.

game_id	player_id	player_name
team_id	team_name	team_acronym
role	win	game_length
champion_name	team_kills	tower_kills
inhibitor_kills	dragon_kills	herald_kills
baron_kills	player_kills	player_deaths
player_assists	total_minions_killed	gold_earned
level	total_damage_dealt	total_damage_dealt_to_champions
total_damage_taken	wards_placed	largest_killing_spree
largest_multi_kill		

Table 1: Dataset columns

Figure 1 shows the distribution of all numerical features in the dataset. It can directly be noted that the `game_length` feature seems to have outliers as all the data is on the left side of the graph. We also have the same problem with the number of wards placed in a match reaching values of 200 which seems to be very unlikely. Limiting those values to a meaningful maximum value result in a better distribution of the data shown in Figure 2.

The dataset also contained some missing values but only for the `team_acronym` column (0.077425% of it) which does not impact plyers stats and performance. Missing values

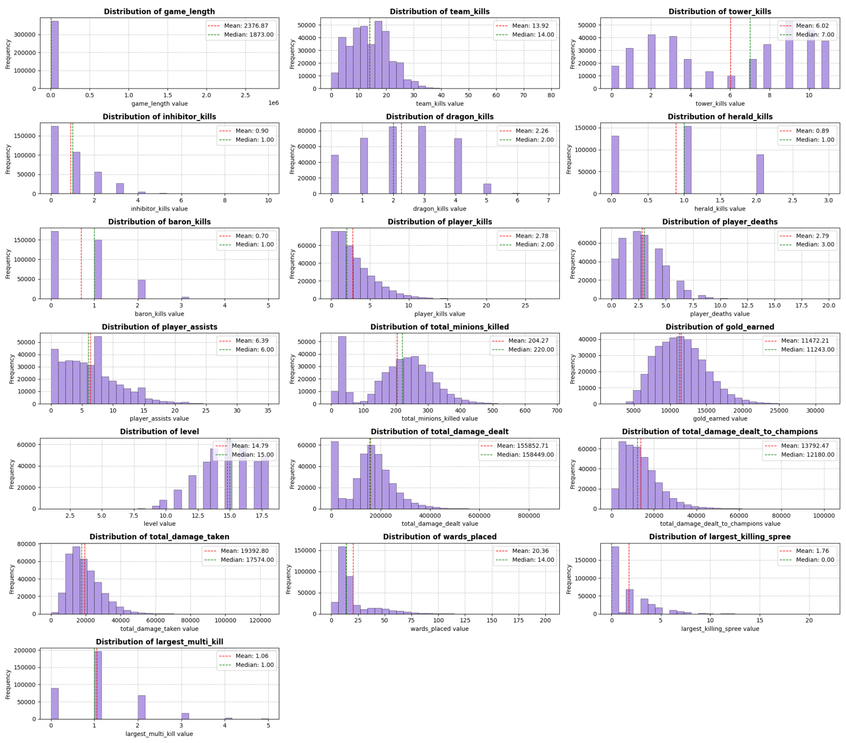


Figure 1: Distribution of all numerical features in the dataset

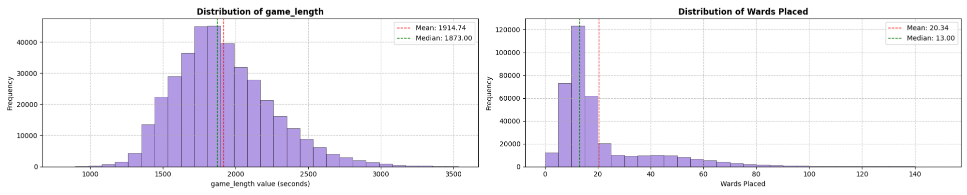


Figure 2: Distribution of game length and wards placed without outliers

will be addressed later in the preprocessing step.

An interesting statistic is the total number of games played and by each player represented on Figure 3.

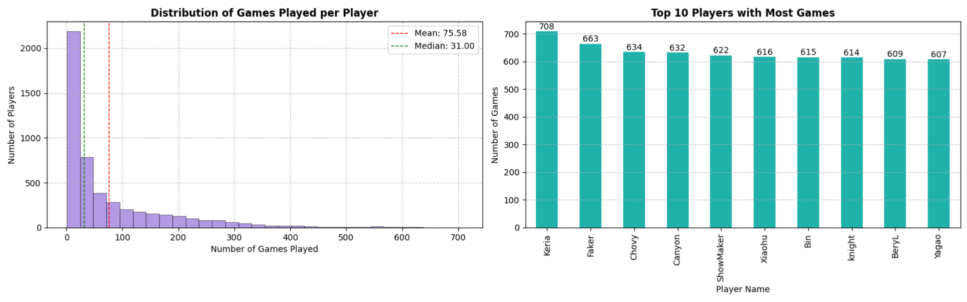


Figure 3: Total number of games played by each player

The tendency is showing that players considered as very good players are playing a lot of games, while there is a lot of players playing only a few games. Filtering the dataset to only keep players with a minimum amount of games could be a good idea to avoid considering players that are probably not playing at a high level.

There is also 168 different champions played across the dataset, showed in Figure 4 and Figure 5.

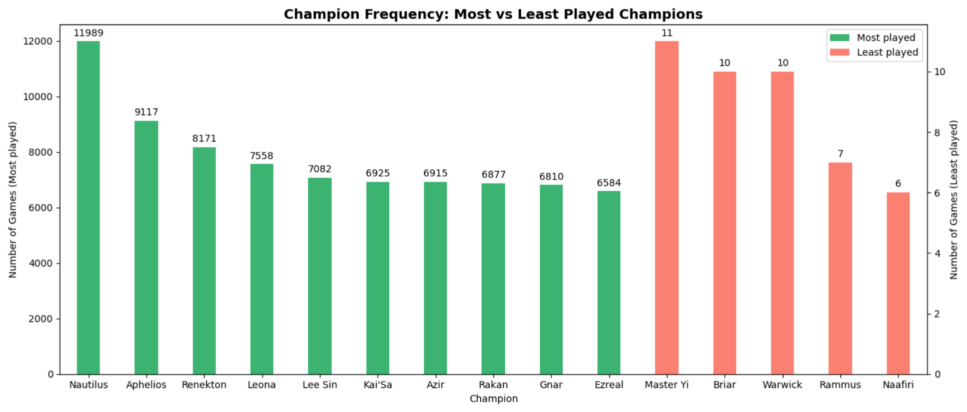


Figure 4: Most vs. Least played champions

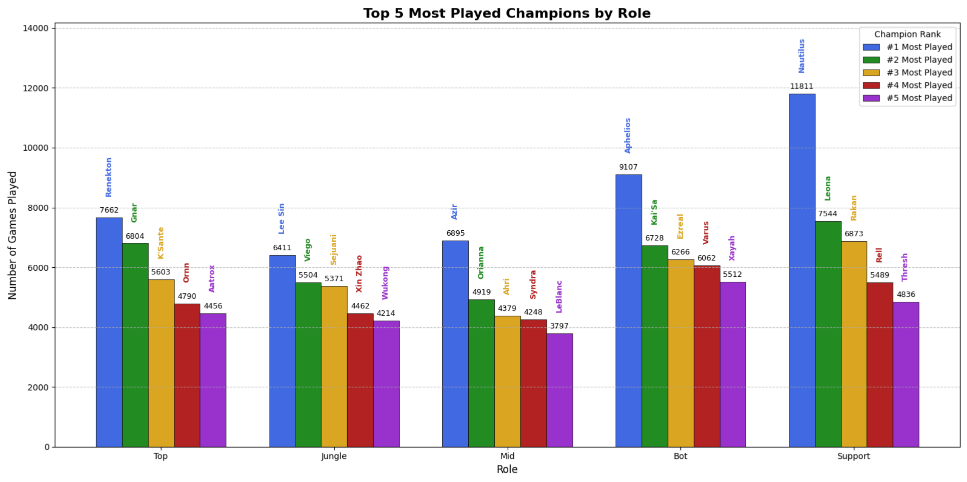


Figure 5: Most played champions per role

3 Feature analysis

To analyse the features and know which one are worth to be interested in, a Mutual Information (MI) graph was drawn.

A correletion heatmap was also used in Figure 7 to also see the correlation between features.

Some interesting features of MI scores were represented in histograms and available in Appendix A.

Some new features were also created based on their correlation and the LoL knowledge I have, to predict if a player won the match or not. They are all represented in Appendix B. It can clearly be seen that there is a distinction between winning and loosing based on the value of these features. As some of them were going towards the same direction, some features like KLA ($(Kills + Assists)/(Deaths + 1)$) were created

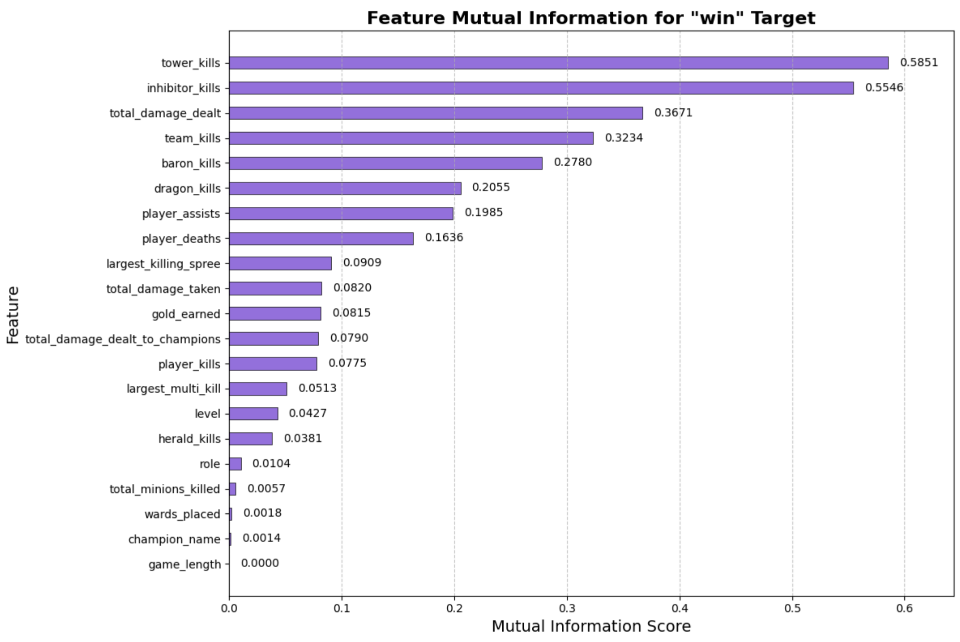


Figure 6: Mutual Information scores for win target

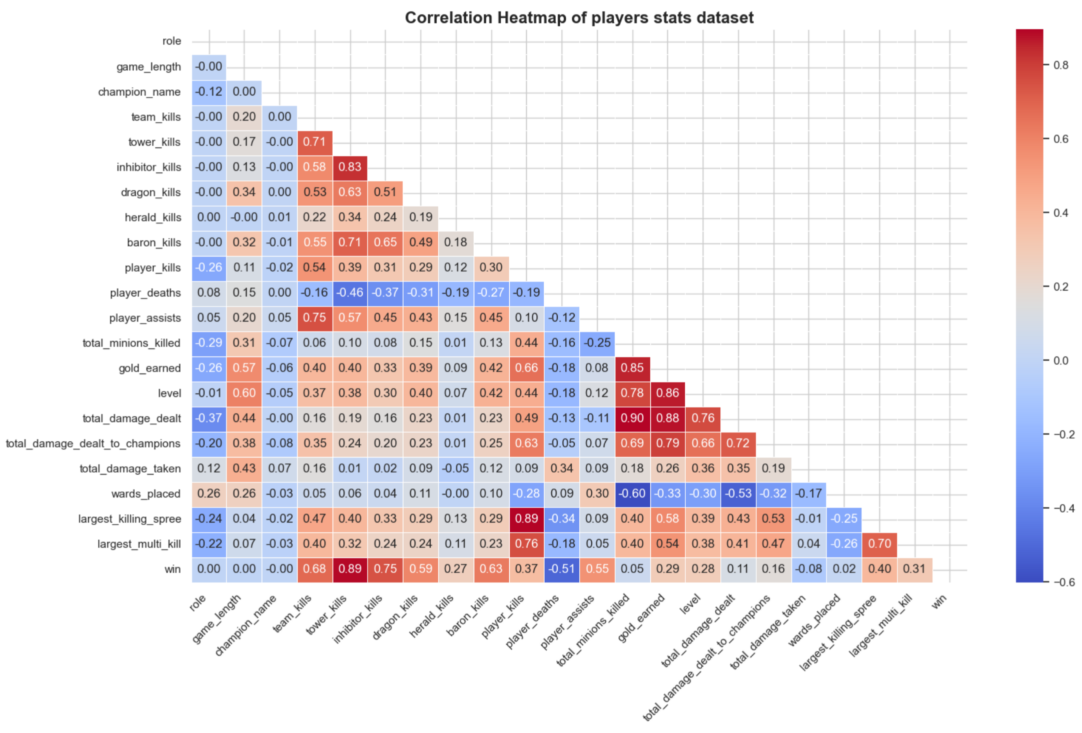


Figure 7: Correlation heatmap

by combining them reducing at the same time the number of useful features. It's interesting to note that Support role does not benefit from CSing minions because it is the role of the ADC, and every damage related feature is not relevant for them. There seems to also have a big impact for the win when objectives are taken, that is, towers (and inhibitors) and dragons/heralds/barons. Although the game length has no impact on the win, it is necessary to make other features relative to it so the model do not get fooled by the game length.

4 Preprocessing

The first step was to get rid of missing values and outliers in the dataset. The team acronym column was therefore completely removed / imputed as it does not provide any value for determining the outcome of a game and is not important for this project. The way outliers were managed was simple, it just keeps the rows of the dataset where the game length is lower than 1 hour (unlikely to be higher and would mean both teams are high level, making it difficult to determine the winner) and the number of wards placed is lower than 120.

Here no categorical features are needed so One-Hot Encoding was not used. Most of them are more of a metadata or player information.

Then every new interesting feature was added to the dataset. Gold, level, CS, and all damage features were replaced by their relative value to the game length, so every one of them per minute of game.

All numerical features (without indexes like player id) were then standardized so that each feature contributes equally to the model (in terms of value range).

$$\text{Standardized feature} = \frac{\text{feature} - \text{mean(feature)}}{\text{std(feature)}}$$

The obtained Dataframe shows an average of nearly 0 and a standard deviation of 1 for each numerical feature.

5 Model training

Three different models were trained to predict the outcome of a game:

— **Random Forest Classifier**

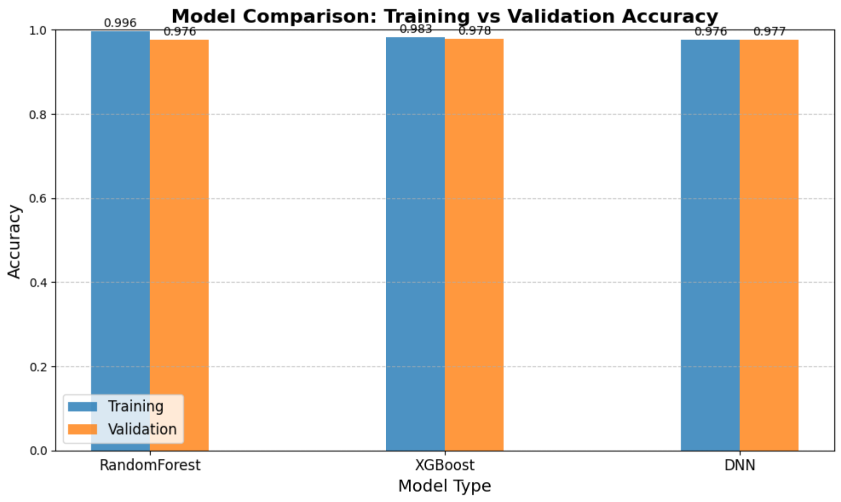
— **XGBoost Classifier**

— **Neural Network**

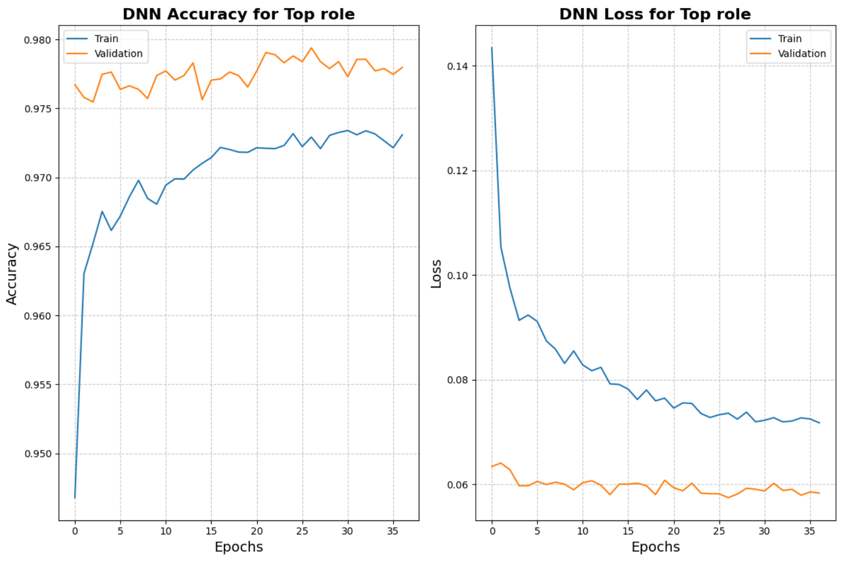
The choice of these models was made by considering that Random Forest is a very common model for lots of *smaller* applications, XGBoost was the model used in the paper to get the best results, and Neural Networks are powerful models capable of learning complex relationships between features and highly used nowadays.

Each model was trained with various hyperparameter configurations to identify the best performer within each model type. The overall best model was then saved for testing.

Initially, models were trained using K-Fold Cross Validation (CV) but several problems were encountered. The already implemented functions of the `sklearn` library did not permit to get the control needed to manage NN history. Implementing a custom CV function did not work as expected and was multiplying by 5 (when using 5 folds) the training time of each model. The training and validation sets and the different folds seemed to be correlated and not independant leading to results of Figure 8. We see that accuracies on the training and validation sets are near 100% which is not good. Models are overfitting and can also be seen on the NN loss and accuracy graphs where the training loss is higher than the validation loss but most importantly, where the validation is already established at the beginning epochs. The validation is clearly not isolated from the training set.



(a) Comparison of top models with K-Fold Cross Validation



(b) Top model accuracy and loss with K-Fold Cross Validation

Figure 8: K-Fold Cross Validation results

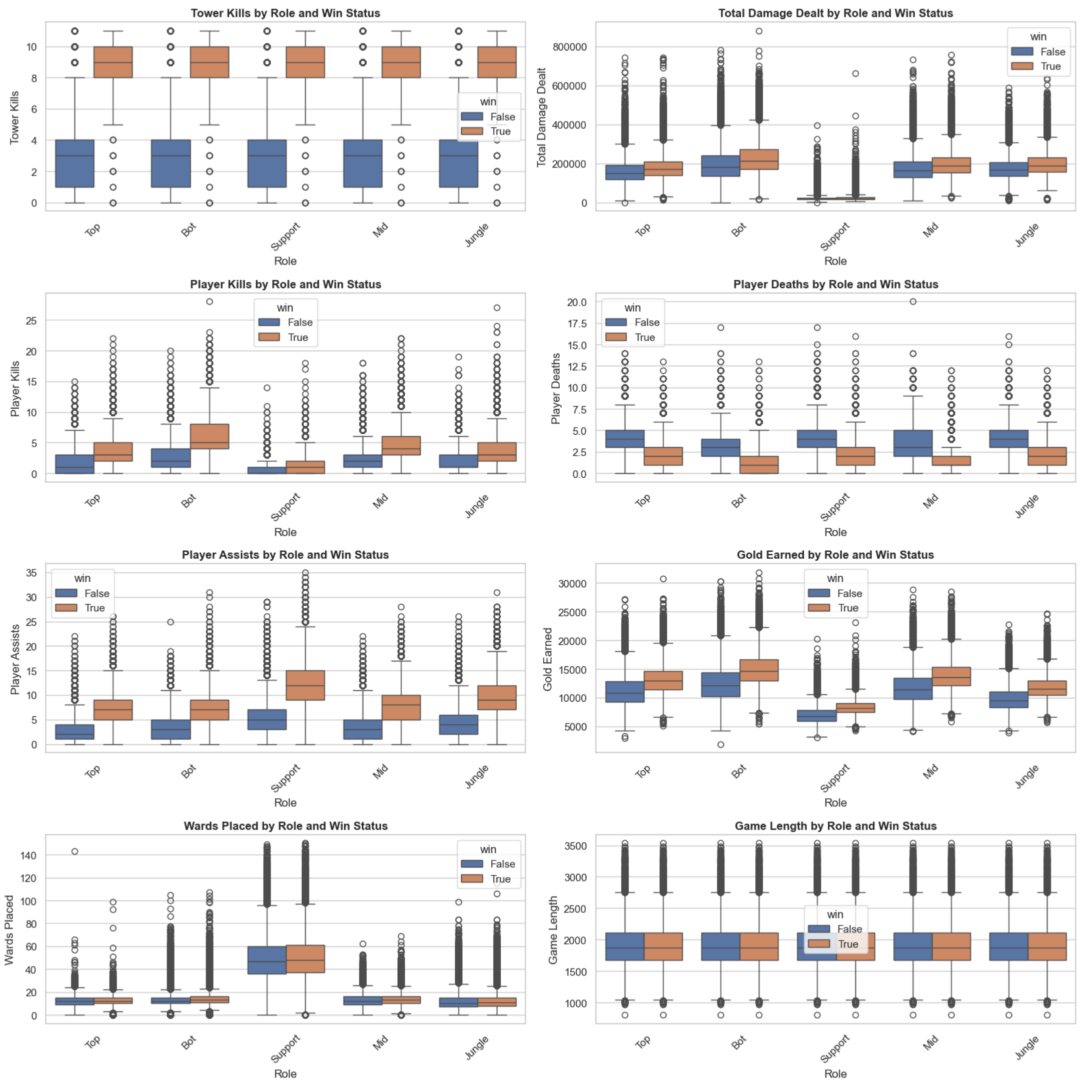
The solution implemented was to train the NN model with different hyperparameters using *for* loops and other models by using Cross Validation Grid Search so it only train 1 time on assured separated training/validation sets. The Random Forest model was trained using range of values for the number of trees in the forest, the maximum depth of the trees, and the minimum number of samples required to split an internal node. Note that it was also tested with `max_depth=None` so nodes are expanded until all leaves are pure, and got better results but was increasing the training time a lot. The XGBoost model was trained using different values of the number of boosting rounds, max depth, and learning rate. The Neural Network was trained using different values of neurons in hidden layers, dropout values, and learning rates. Every hidden layer uses ReLU activation function and are applied batch normalization and dropout. To make sure that the model is not overfitting, L2 regularization of 0.1 was used on each hidden layer and, in fact, really do have made better results.

The metric used to evaluate the models was the accuracy on the validation set. If a model has a higher accuracy, it keeps that model as the best current model and repeat the operation until finding the best overall model.

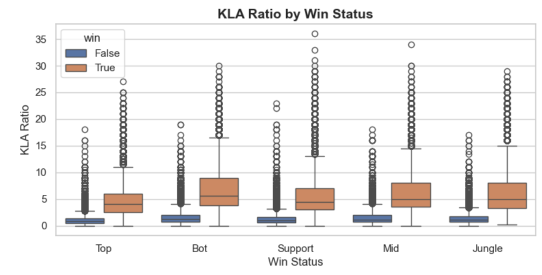
The results for each game role are shown in Appendix C for top, Appendix D for jungle, Appendix E for mid, Appendix F for bot, and Appendix G for support. Results shows good accuracies for training and validation troughout all roles and NN accuracy/loss rapidly converges to them. But I would still suspect that there still seems to be something wrong with the validation set as the accuracies are pretty high for not highly optimized models. All of the roles except Mid role find best accuracies with the XGBoost model. Mid got better results with the Neural Network model. However, based on the results, they are all very close to each other and the differences are not significant. Other parameters, features and ML concepts could be used like proper cross validation or PCA analysis for better features to see which one is really the best.

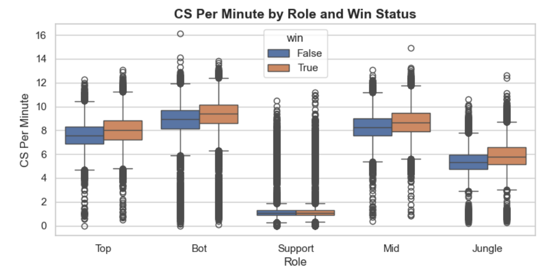
The final model is then used to predict the outcome of a game and the probability of winning or loosing on the test set. The test set was created by splitting the dataset into 75% training, 15% validation, and 10% test sets. Surprisingly, the test set accuracy is close to the traning and validation accuracies. This should not be the case, so there is still something that had been done wring in the training process or the dataset splitting.

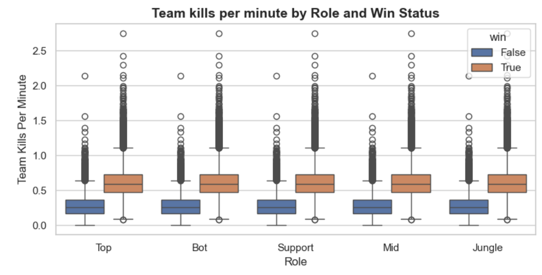
A Histogram representation of key features

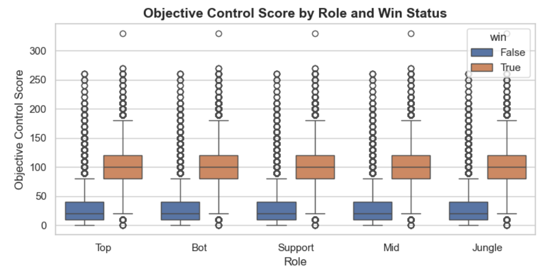


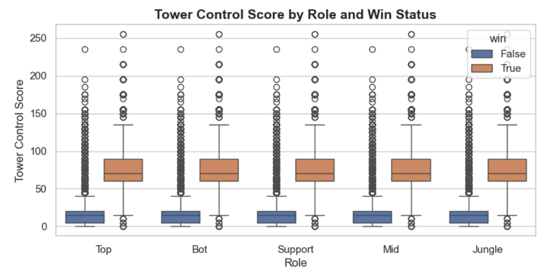
B Histogram representation of new features

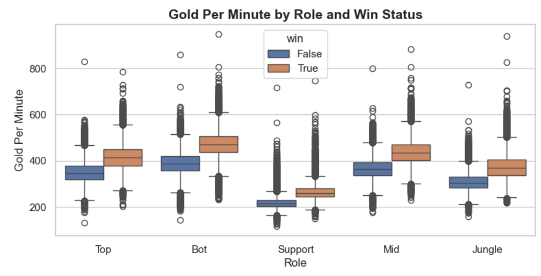


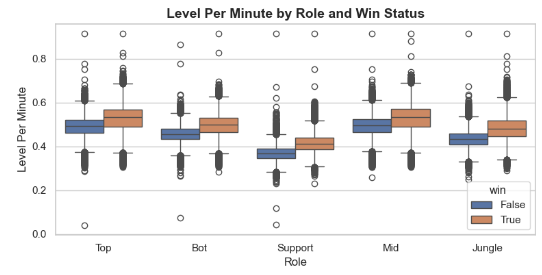


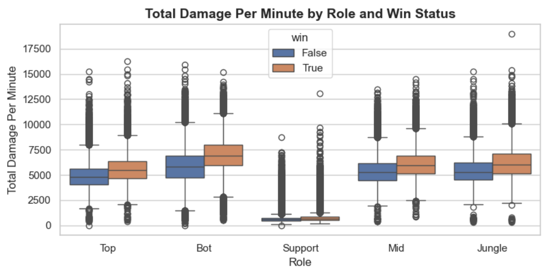


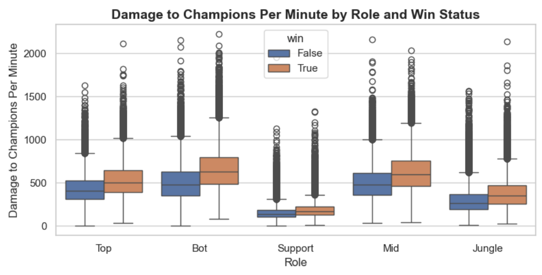


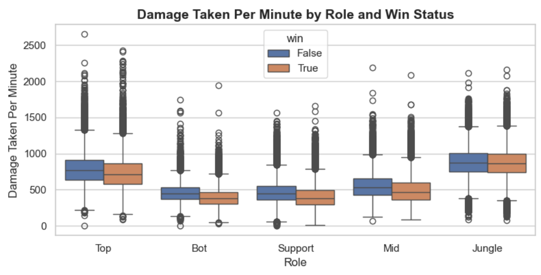




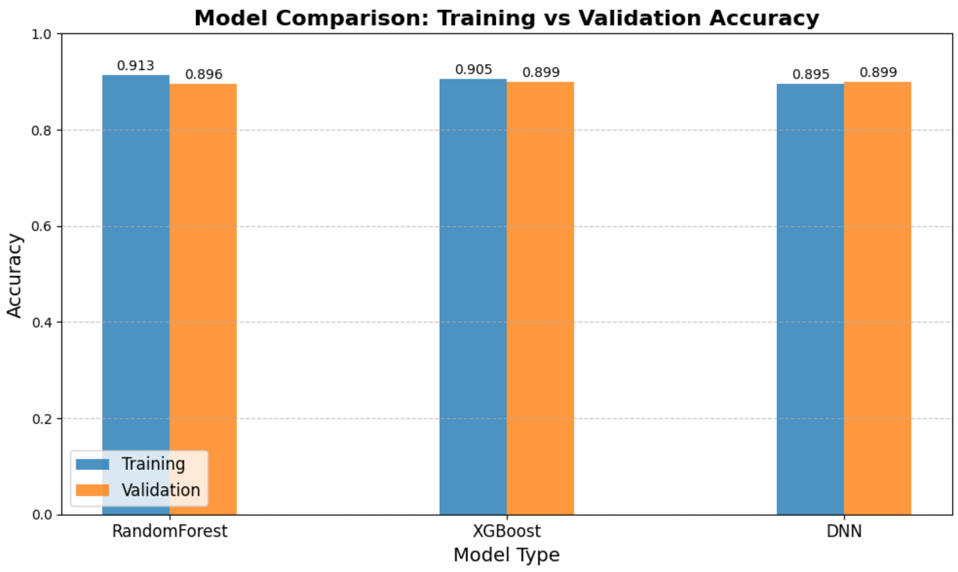




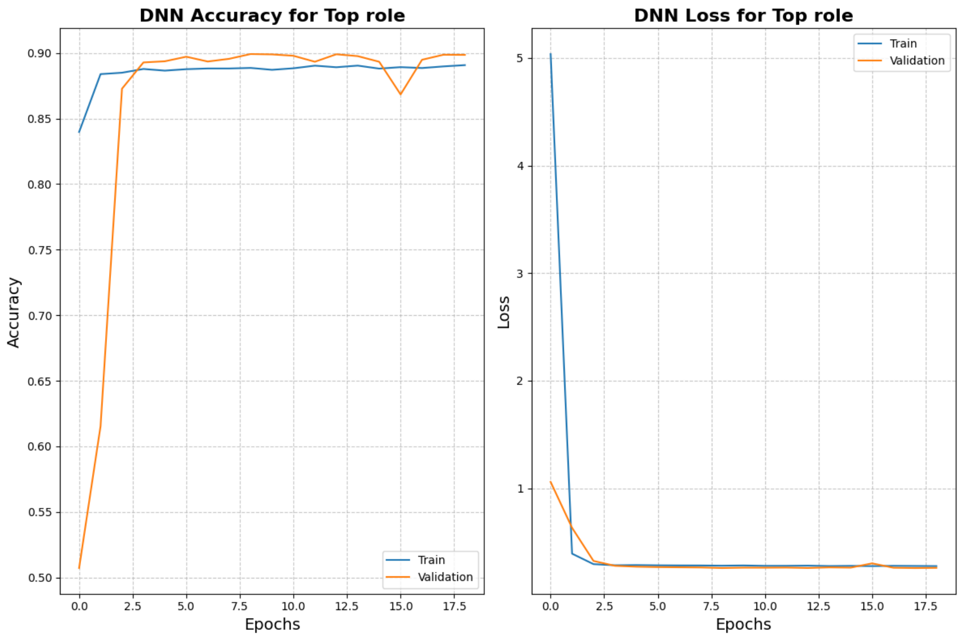




C Top model training and validation

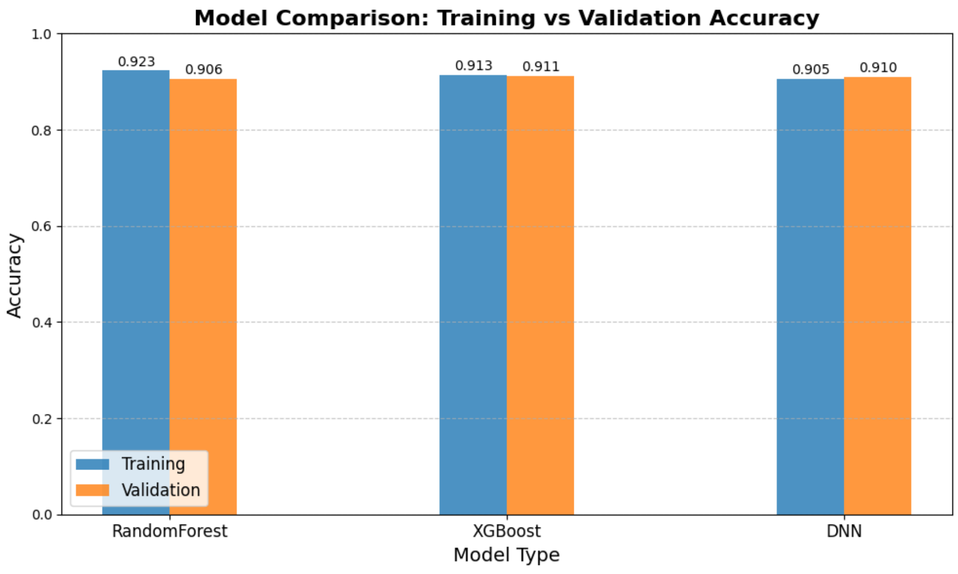


(a) Comparison of top models

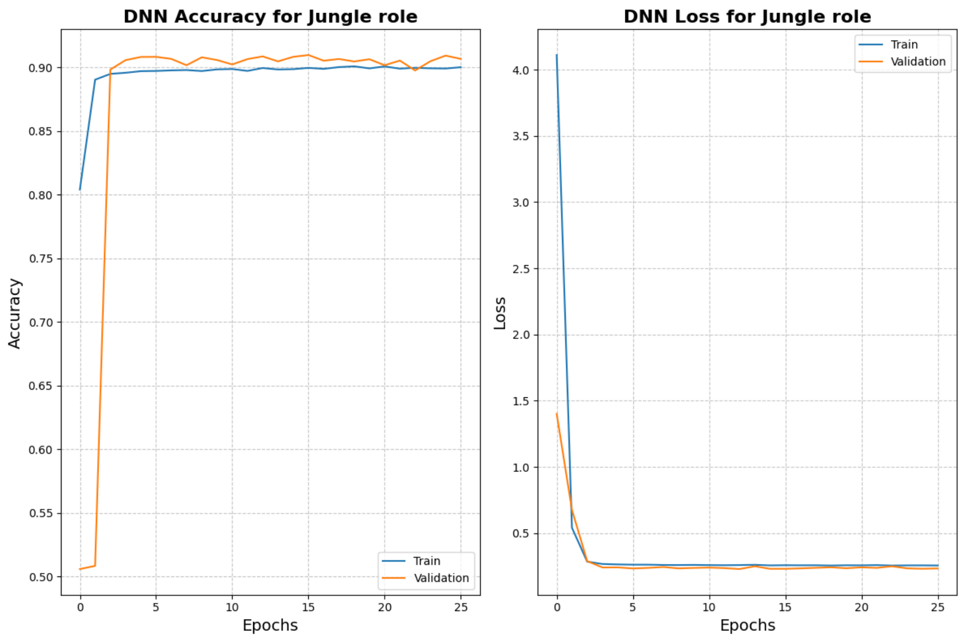


(b) Top model accuracy and loss

D Jungle model training and validation

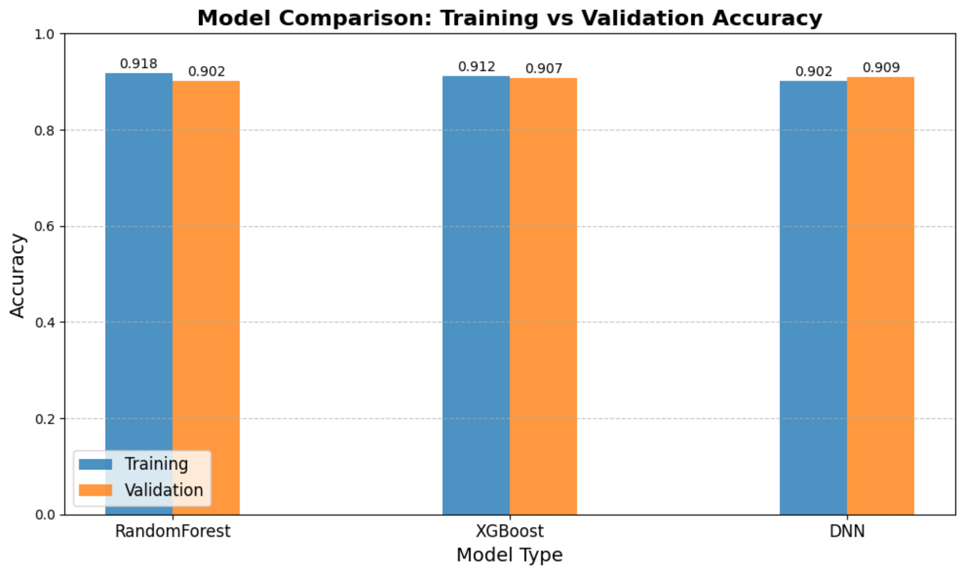


(a) Comparison of jungle models

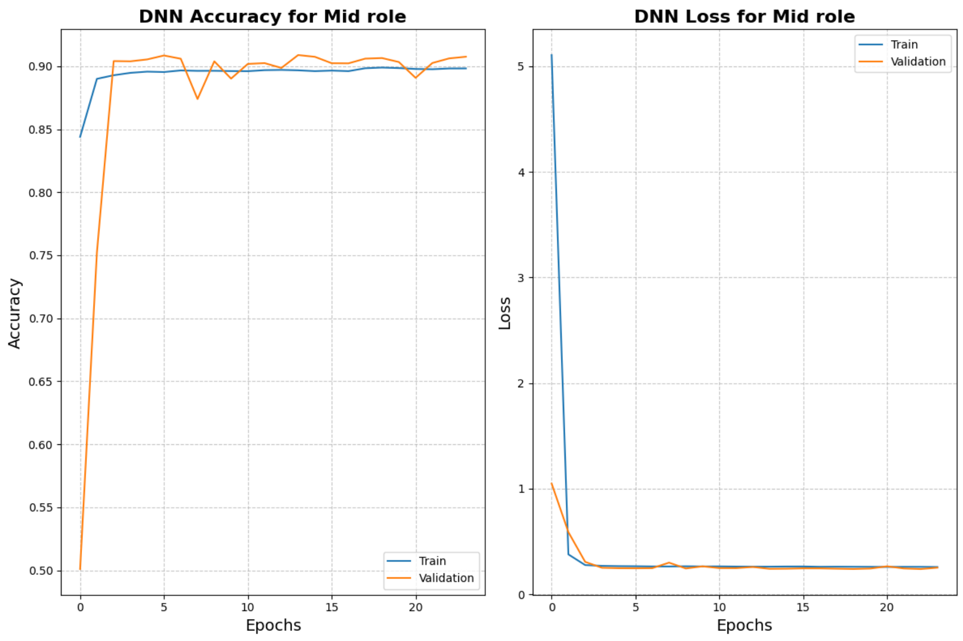


(b) Jungle model accuracy and loss

E Mid model training and validation

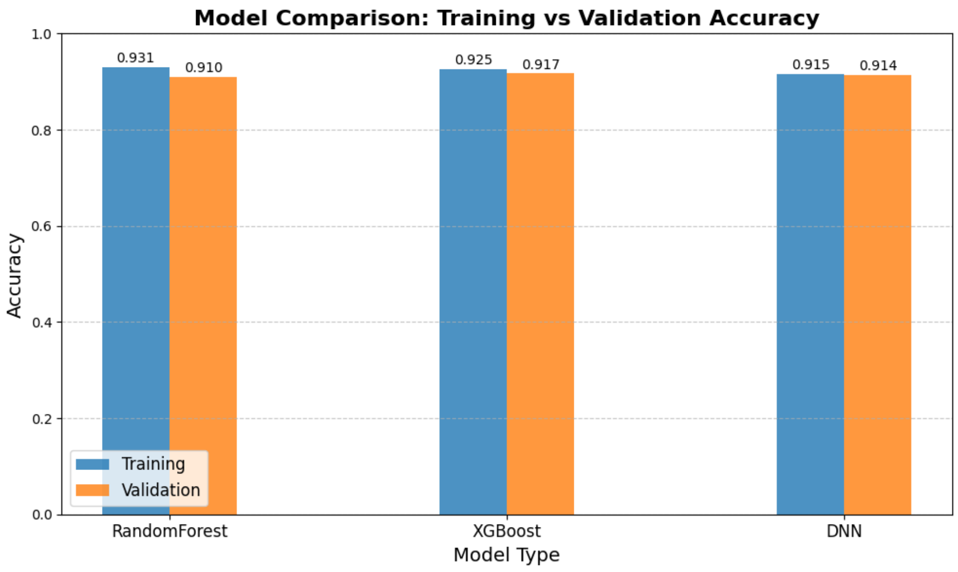


(a) Comparison of mid models

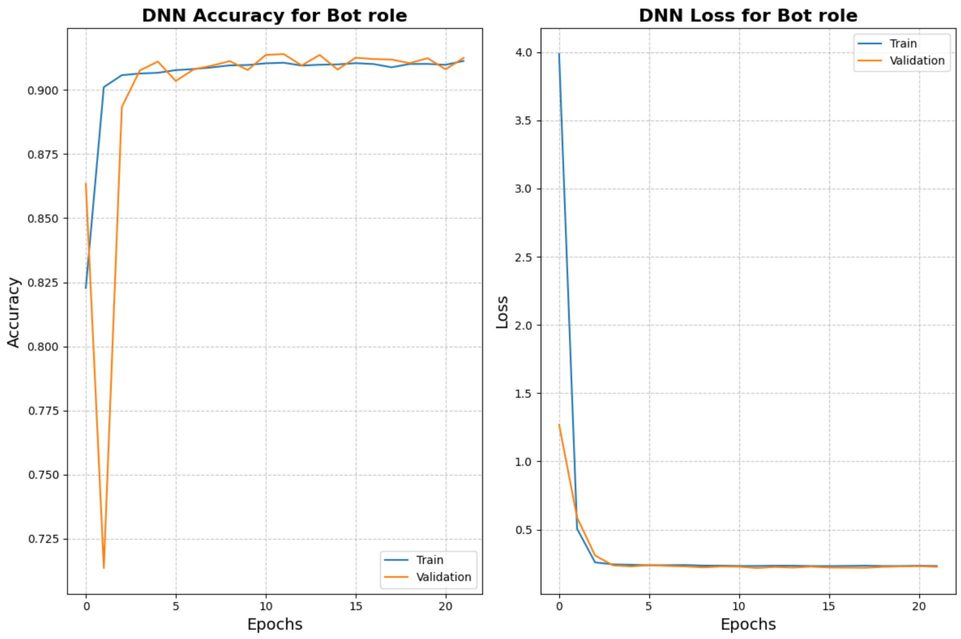


(b) Mid model accuracy and loss

F Bot model training and validation

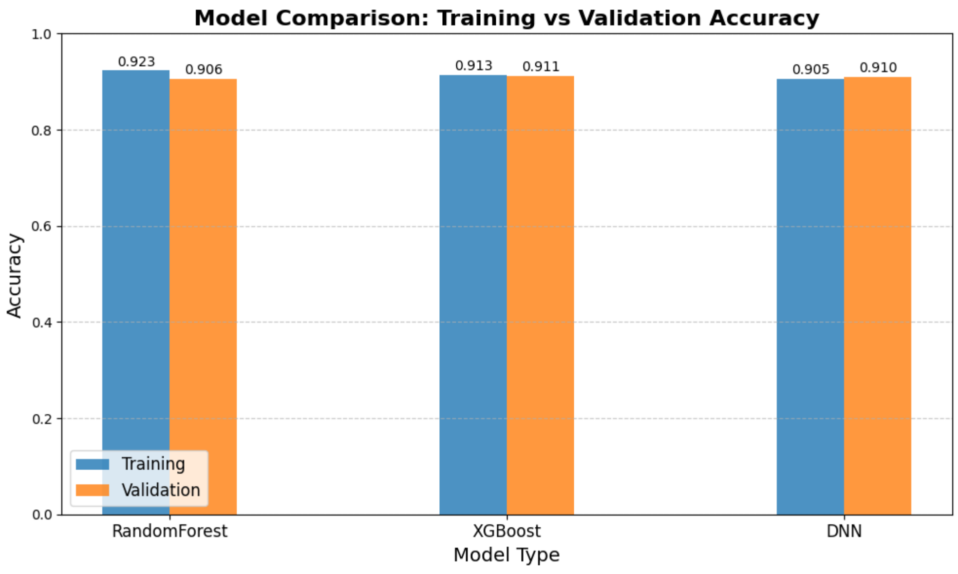


(a) Comparison of bot models

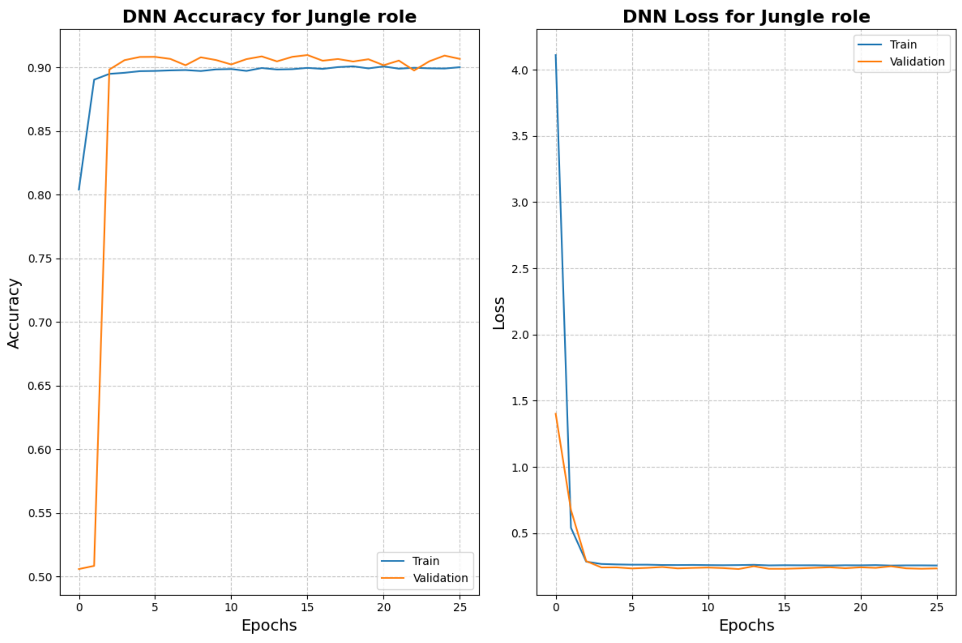


(b) Bot model accuracy and loss

G Support model training and validation



(a) Comparison of support models



(b) Support model accuracy and loss