Capstone Project 1: Machine Learning Analysis

Introduction

As a recap, the capstone project will utilize Riot Game's API to pull data on player matches from the 2018 season. The matches come from high ranking players in Diamond tier and above. When initially cleaning the data and forming a Pandas Data Frame, each row of data representative one match. Each row, initially comprised of over a 1000 features, including post game stats and player choices (champion picks, runes, summoner spells, etc). I decided I wanted to predict the outcome of a match just based on player choices before a game. There were two target predictions, red side and blue side win. The outcome for either prediction would be 'Win' or 'Fail'. There are many tools online already online that utilize post game statistics to help players create the most optimal build or sites that display many game statistics and analytics. Predicting which team will win based on pre-match choices is something I have not seen online. Initially, I thought it would prove challenging because of the variance in player skill levels. Therefore, I ended up at the idea of creating a predictive model for high level matches only. This made more intuitive sense because players at the highest level all had a very high understanding of game play and game mechanics. Players at a high level are also more determined to win a game versus playing casually to have fun. With that assumption, outcomes could be better predicted as a result of team composition with the optimal summoner spells and runes.

The result of this model could develop into an application where players can analyze match ups and team combinations. League of Legends is a very popular game in Esports right now. This model could develop into a tool used by professional teams and their coaches to gain insight on developing new team composition strategies versus common picks in professional play. This could also be a useful and fun tool for shout casters to add to the repertoire of pre-game match statistics.

The GitHub link to this project:
https://github.com/jltsao88/Capstone_Project_1/blob/master/Machine_Learning.ipynb

The Machine Learning Problem

The problem is a classification problem where the model is trying to classify matches as either a 'Win' or 'Fail' for a specific team. The Data Frame being used initially contained these features: Champion pick for each player, Summoner Spell of each player, the primary and secondary runes (labeled as perks in the data), the role of the champions picked, number of that type of role in game, red team win, and blue team win. The features 'blue_team_win' and 'red_team_win' were removed and stored as the target to predict. You will see I pulled out game length as well, but did not end up using it. The data was further split up into two feature sets. Feature Set 1 included the champion roles and number of champions. Feature Set 2 had roles and role counts removed. The intuition for 2 feature sets was that champion role features may end up just being noise in the model and I wanted to test that intuition. The features were all in

categorical form and pre-processing was needed to create dummy variables for every feature. Pandas' .get_dummies() function was used to achieve this.

Picking the Machine Learning Models

Since this is a classification problem, I decided to test 4 different classification models. They were Logistic Regression, SGD Classifier, Support Vector Machine Classifier, and Random Forest Classifier. This project utilizes Python's sklearn package to utilize machine learning.

First, I created a training and test sets for Feature Set 1 and 2. Next, for all 4 models I initially used the default implementation without setting any hyper parameters to see how each model performed out of the box to predict Red Side wins. Feature Set 1 was used as the initial test for these models. Each model was fitted to the training data and tested on the test data. The default accuracy score was used to get a idea of model performance. The out of the box scores were as followed: SGD: 0.536, SVC: 0.505, Logistic Regression: 0.566, Random Forest: 0.807. There was already a clear winner in performance based on accuracy. I also looked at the classification report for each model, looking at precision, recall, and f1-scores. Still, Random Forest came out on top. Just to make sure Random Forest was the best, I tried hyper parameter tuning and cross validation non the models. I also looked at AUC score for the ROC. Random Forest was still the best performer. With that, I went more in depth with fine tuning the hyper parameters of the Random Forest Model.

Because Random Forest performed so well I decided to tune the hyper parameters: 'n_estimators', 'max_features', 'max_depth', 'min_samples_split', 'min_samples_leaf', and 'bootstrap'. Look at the GitHub link for the details. I used a randomized grid search with 3 fold validation due to the training time and computational limits of my personal computer. I found optimal hype parameters which slightly improved accuracy but saw great improvements in precision, recall, and f1-score. I decided to plot the ROC curve to did a visually sense of where the model was at. I trained the model again using the second feature set and saw slight improvements and decided the champion roles added noise to the model. From thereon, I used the Feature Set 2 for the models. With Feature Set 2, the AUC score was 0.93. I repeated the steps again with predicting Blue Side win with similar results. The AUC score for predicting blue side win was slightly better at 0.932. Based on the results, Random Forest does a good job at classifying wins for pre-game match features. I tried using a smaller test and training set to speed up the process, but the results were far worse by doing so.

Random Forest is an ensemble model and aggregates a bunch of decision trees together and uses that to make predictions. There are around 140 champions, 10 commonly used summoner spell, and a variety of rune choices. This makes the number of team compositions enormous. Intuitively, Random Forest would be the best in finding these choices to lead to a win or fail. Also, most of the features when split into dummy variables are binary choices. The other models used some sort of regression or draw lines/hyperplanes to predict the data. With the way the features are chosen and currently set up, it would seem the other models have a harder time at classifying. In

the case of SVM, the run time to fit the model was extremely long and I could not test different hyper parameters with my current machine.

Final Thoughts and Intuition

I want to reinforce the idea that successful out come of the Random Forest model would most likely be applicable in high level games or professional play. I would assume that the model would perform far worse in games where players are playing for fun, have inexperience, and high variation of skill levels. Overall, I am happy with how the model performed. I developed some intuition afterwards about why the model could not make better predictions. For one, the game is 10 players and not all player can control their emotions in a match. I common occurrence in games are 'rage quitting' or 'intentionally feeding'. When that happens, one or a few players will get upset and basically throw the match. Unfortunately, it was difficult to find those matches were players intentionally threw the game.

Another afterthought was taking into account 'one tricks'. These are players who predominantly play one champion and therefore have higher win rates and play rates of a certain champion. This could also skew certain features of the data. In hindsight, it would have been better to remove known 'one trick' player, but compiling a list would be a daunting task. Including those 'one tricks' in the current data may have led to some over fitting in the model. Making a generalized model would be difficult due to various skill over the different tiers of players. Ideally, a separate model would need to be made for each player tier and every region. I do not believe a generalized model would be practical and lead to poor performance. I in making a tool for shout casters, you could perform the steps in the model but used the data from matches of professional play only.