Power Rankings Hackathon: GNAR-Score

This is an entry for the DevPost Power Ranking Hackathon.

The PDF Version can be found here.
The video showcase of this entry can be be found here.

Introduction

This is an entry to the Devpost Power Rankings Hackathon 2023. I am a IT Consultant based in Berlin, Germany and have been a long time player of League of Legends and follower of it's esports scene.
The combination of micro decision and skill expression through fast, reactive game play and cool combos, and also big picture, strategic thinking and organized team work which make League of Legends a fascinating game to analyze.

The GNAR(Gaussian Naive Adjusted Ranking)-Score is based on a Elo formula, whose predictive properties have been improved using the output of a Gaussian Naive Bayes classifier trained on historical data.

Entry

My Entry into this hackathon is in the form of an API. This API is available under https://usm38g8rwj.execute-api.eu-central-1.amazonaws.com/api/global_rankings where all the required endpoints are available. The first 20 teams of the global rankings system can be fetched with the following curl command: here is a curl command for each endpoitn

```
# get the global ranking (first 20 teams)
curl 'https://usm38g8rwj.execute-api.eu-central-
1.amazonaws.com/api/global_rankings?number_of_teams=20'

# get rankings for a list of team_ids
curl 'https://usm38g8rwj.execute-api.eu-central-1.amazonaws.com/api/team_rankings?
team_ids=98767991926151025,107563714667537640,98767991853197861,100205573495116443
'


# get the rankings for teams in a tournament by tournament_id
curl 'https://usm38g8rwj.execute-api.eu-central-
1.amazonaws.com/api/tournament_rankings/110733838935136200'
```

A frontend to explore the rankings in the browser is also availabe at: http://power-rankings-frontend.s3-website.eu-central-1.amazonaws.com/

Tech Stack

- The API is written in Python with the Chalice framework from AWS
- AWS Lambda and ApiGateway for hosting the API
- S3 for accessing hackathon data and hosting data for the API
- Athena for exploring and exporting data
- Python Pandas and Sckit-Learn for data exploration and model building
- Terraform to configure required infrastructure and permission

Python was chosen because of my familiarity with the language and because it's ecosystem offers good libraries for both api development and data science tasks. Chalice is a Python library that allows easy deployment to AWS with lambda and API Gateway, using syntax that is similar to other common python api frameworks like flask and fast-api. The Data from S3 was used to create tables in Athena as per the guide provided by Devpost.

Running locally

The `generate.py` script generates the global rankings and rankings for each tournament and stores them in an S3 bucket. The name of the s3 bucket can be configure in `/chalicelib/const.py`

```
RANKINGS_BUCKET="power-rankings-hackathon"
RANKINGS_DIR="rankings/"
```

After configuring the S3 bucket, the rankings can be generated like so (can take a long time):

```
python3 generate.py
```

Afterwards the api can be started locally by install chalice and running

```
chalice local
```

Basic Statistical Indicators

To begin building a predictive model we can start by looking at some basic statistics indicators. A few somewhat obvious choices are first blood, first tower and first inhibitor. Out of the 25255 games in the dataset, Blue won 13350 games and red 11903. Therefore the blue side has a 64% win rate after getting first blood, while the red side has 0.59. The following table also includes the win rates per for first tower and first inhibitor.

Side	Total Game	Wins	First Blood WR	First Tower WR	First Inhib WR
Blue	25255	13352	0.63953	0.69689	0.94578
Red	25255	11903	0.58693	0.67633	0.94511

These stats might be interesting, but are only use for predicting the outcome of the game live as it is being played out. And event like the first inhibitor only happen quite late into the game, ideally we would like to make predictions earlier, even before the games starts

Predictive Model

The following indicators have been used to train the predictive model:

- first_blood, tower_tower and first_inhibitor
- total kills and deaths
- level and minion score advantage in the early game
- shutdown gold generated and collected

All data points were aggregated into a single dataframe, after which a rolling average over the last 6 games was calculated for each stat. The process of preparing this data is detailed in the preprocessing jupyter notebook.

The resulting data was used as input to train the Gaussian Naive Bayes classifier included in the scikit-learn python library. The data was split into train and test batches to avoid over fitting.
On the test data, the trained classifier achieves and accuracy score of around 80%.
The code for training and testing this model can be found in the model jupyter notebook

Elo System

The main component of the GNAR-Score is an Elo system. These kinda of ranking systems are relatively simple and robust. They make no assumptions about which strategies or game play patters are desirable, since the attribution of points depends only on a teams ability to win. This approach is a convenient way to cut through all the complexity and variance of a competitive game like league and easily derive and objective rating the accurate reflects relative strength base on past performance.

In this elo system, each team starts with 1500 points, and all matches in the last 180 days are considered.

A teams elo is updated base on the following formula:

```
k_factor = 50
expected_score = 1/(1+10**((opponent_elo-old_elo)/480))
Elo = old_elo + (K_FACTOR*(1-expected_score))
```

The expected score, is the expected outcome of the game based on the difference it in elo. It is a value between 1 and 0 with 1 representing victory and 0 representing a loss.
The predictive ability of this elo system can be judged by calculating the squared_error using the expected outcome base on elo difference and actual result.

```
squared_error = (expected_score_blue-outcome_blue)**2 + (expected_score_red-
outcome_red)**2
```

By averaging out the squared_error over all prediction, the base Elo system has a `mean_squared_error` of around `44%`.

Combining Statistical and Elo Model

The superior predictive ability of the earlier model can be used to improve the elo system by incorporating its predictions into the expected out come. The formula above can be updated, so the expected score is combined with the weighted output of the gaussian statistical model.

```
expected_score = 1/(1+10**((opponent_elo-old_elo)/480))
expected_score = (w * expected_score) + ((1 - w) * prob_outcome)
```

Where `prob_outcome` is the prediction of out model and `w` is the weight, which is adjusted in proportion to the confidence of our prediction (high confidence, high weight).

After incorporating the models predations the `mean_squared_error` of the systems predictions is around `37%`. This is an increase of about 7 percentage points, which may not sound like a lot, but is a good result for a highly randomized environment of a League of Legends match.

The script `mean_squared_error.py` will calculate and output the mean suared error for the elo system with and without the prediction model.

Created by Jan Wendel jan@wendel.berlin
Credit frontend: Xander Van den Bossche - https://github.com/Xandervdb1/league-front/tree/main