Social Networks Final Project - League of Legends

Yakir Hadad 313250276 and Chen Mashali 204377915

July 2020

1 Introduction

League of Legends (LoL) is a multiplayer online battle arena video game developed and published by Riot. the players assume the role of a "champion" with unique abilities and battle against a team of other player- or computer-controlled champions. The goal is usually to destroy the opposing team's Nexus, a structure that lies at the heart of a base protected by defensive structures, although other distinct game modes exist as well with varying objectives, rules, and maps. Each League of Legends match is discrete, with all champions starting off relatively weak but increasing in strength by accumulating items and experience over the course of the game. There is one map in this game and it is separated to 3 lanes and the reason that this game is so popular is the variety of champions that's makes every game to be differ from its previous.

Each champion has its own special abilities so it almost impossible to know which combination of champions will give the most advantage over the other team. The main goal in our research is that we want to predict how well 2 champions might play together based on previous games that already played (Without necessarily a game with those two). We used Leskovec methods for this prediction and extended it with to be more directed to League Of Legend domain.

2 Main Goal

The Main Goal Right now from our data we could extracted 148 champions with 10799 relations between them from 14265 matches. We will explain on the next chapter how we build our graph but for now we will say that each champion is a node and a relation between 2 nodes mean that they play together. Ideally our graph will be full connected with enough data that each edge is based over multiple games. Well, that's not the case and this is where our research comes to help. A fully connected undirected graph with 148 nodes will have $\binom{148}{2} = 10,878$ edges, means that we don't know about $10,878–10799 = 79$ edges, so we were close to find relations about all champions. First we wanted to find out about *if* 2 champions are playing well together, similar to Leskovech research, but after getting our result which was really really good we decided to extend it and check whether if we can actually predict *how* well these 2 champions play together.

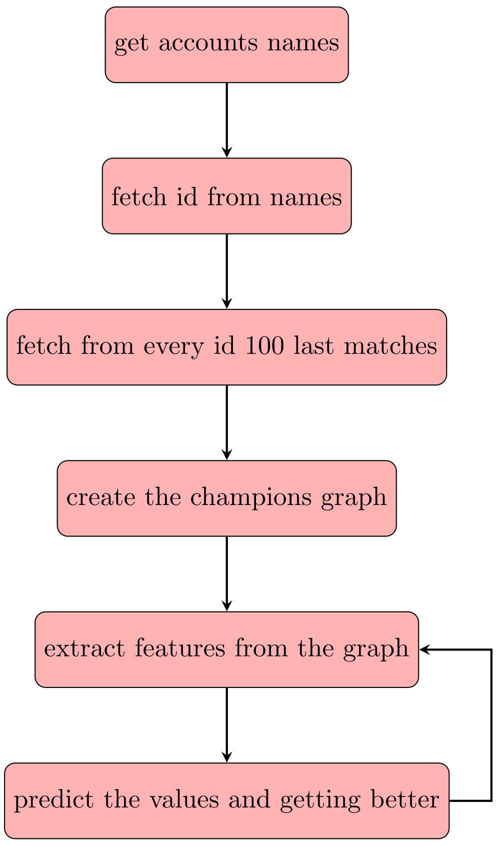
3 Method

3.1 The Steps from Data to Prediction

Riot gives the developers an API with a lot of information about their game including matches of every player we just want. The players we use for our research are some of our friend's accounts names and most of them from the ranking page https://eune.op.gg/ranking/ladder.

1. run script that crawl all names in ranking pages
2. fetch account ids by names.
3. fetch 100 last games from every account
4. create the champions graph
5. extract features from the graph to classify/predict the relation between every 2 champions and build a DB for a machine learning purpose.
6. Learn the result and try to understand if we can improve that.

We can draw our process in our research as this flowchart:


get accounts names
fetch id from names
fetch from every id 100 last matches
create the champions graph
extract features from the graph
predict the values and getting better

3.2 Data Source

All our data have been extracted using RIOT API. Following their tutorial https://developer.riotgames.com/docs/lol was easy enough. With python libraries that make it even simpler, we fetch all the data we needed by giving it an account name. To give our script enough names we create a simple crawler that read names from lol ranking page https://eune.op.gg/ranking/ladder. The names we took from variety of pages and not only professional players. In total we got 145 players. We just had to make sure we regenerate our API Key for new data if we needed. The full API can be found here https://developer.riotgames.com/apis

There are a lot of type of requests, but we used only 2 of them for our purpose.

1. ⟨ Account name ⟩ to ⟨ account id ⟩
2. Last N recent matches of ⟨ account id ⟩

In our case we choose N to be 100 to compliance the conditions of riot limitation to developers product of 20 requests every 1 second / 100 requests every 2 minutes. The response from our requests is returned in a JSON format, each match has almost 1500 fields, so we add an example with this report (example.json).
The JSON file is build of array of matches, the fields that used in our cases are only:

- Match[‘gameid’] : Int
- Match[‘participants’] : Array of participant
- Participant[‘stats’][‘win’] : Boolean if win

3.3 The Graph Construction

Our algorithm to create the graph is quite simple.

Algorithm 1: How to build the champions graph

```
Result: Weighted graph
new undirected graph G;
for every match do
    add champions to G;
    add 1 to edge between winning champions;
    subtract 1 to losing champions;
    C(e)++;
end
```

Because we fetch last matches from multiple players, we make sure to remember the matches id to avoid duplication. We add a new field named C(e) = Count of Matches between edge e. Each node is a champion and the edges are the relation how well they play together, the higher the weight the better their performance are.

To avoid underfitting on edges with not enough matches we test our model with multiple threshold of matches > th to call the link as edge, these is the graphs that were built:
Here is a graph of number of edges as a function of threshold.

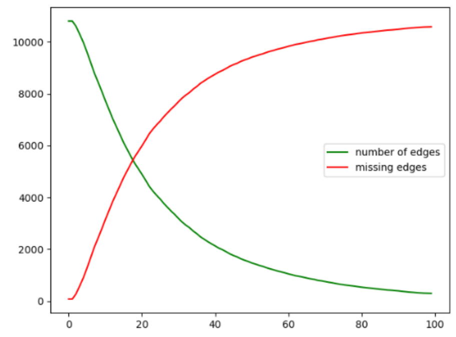


Figure 1:

We notice some linearity in the first 20 threshold and we found it interesting that this is the intersection point.

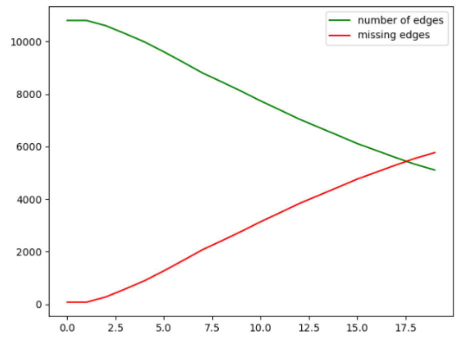


Figure 2:

This table below show the parameters depends on the threshold.

threshold	edges	missing edges
0	10799	79
5	9606	1272
10	7739	3139
15	6115	4763
20	4893	5985
25	3935	6943
30	3186	7692

3.4 Our Model

We use ML algorithm to classify whether 2 champions are working well together or not. From our simple research the best regressor that exist these days is gradient boosting and XGBoost is the most recommended library for this purpose. Gradient boosting is type of a Random Forest algorithm that convert weak learners into strong learners. The DB is a pandas DataFrame that each row represents an edge. We run the program and each time we learn something more and try to get a better result. We split our data to 80% training set and 20% test set using sklearn function.

3.4.1 Phase 1

At the beginning we did like Leskovec's paper W(u) = sum all edges weight that connected to node u.

3.4.2 Phase 2

We add features of average distance with 2 edges and 3 edges from u to v. We wanted to add distance of 4 edges too but the run time for this was too long. We decided to do that because as Leskovec mentioned Heider theory of balanced network. So if champion A play well with champion B and champion B play well with champion C than we believe that probably A play well with C. And because teams are built from 5 players we wanted all the average distance of with 1, 2, 3 players between u and v.

3.4.3 Phase 3

Just add variance to the distances

3.4.4 Phase 4

We have decided instead of having a number on edge that doesn't really represent how well champions play well together, for example in case of 2 champions play 10 times and win all this matches the edge will weight 10, another case is 2 champions play 100 times and have 55 wins and 45 loses so the edge will weight 10 either but any one with common sense understand that they not the best choice. This is why we choose to have a ratio of edge/ matches instead, this way our numbers are more comparable to know which champion is better.

3.5 Phase 5

Use data that is not from the graph.

4 Results

We will say it again we got results and try to get better results, first we try to predict *if* champions play well together by a sign value:

4.1 Phase 1 - sum the edges

First the result was horrible, We can say even pure random:

threshold	mse	% accuracy
0	0.911	49.44
5	0.881	51.30
10	0.914	50.12
15	0.987	47.99
20	0.945	50.25
25	0.900	52.73
30	0.888	52.97

4.2 Phase 2 - average distance

We take all average distance between u and v with 1 node between them and 2 nodes between them (d3 and d4):

threshold	mse	% accuracy
0	0.689	59.81
5	0.743	59.26
10	0.773	57.68
15	0.806	56.74
20	0.844	55.36
25	0.785	57.94
30	0.832	56.26

4.3 Phase 3 - variance distance

We add the variance to the average distance between u and v with from phase 2 (var3 and var4) (doesn't really improve):

threshold	mse	% accuracy
0	0.716	59.25
5	0.822	54.37
10	0.815	55.03
15	0.819	55.68
20	0.859	54.13
25	0.817	56.92
30	0.799	57.05

4.4 Phase 4 - centrality and weighted centrality

Another 2 features we add to the model are centrality by edges and centrality by the weight of the edges:

threshold	mse	% accuracy
0	0.643	62.00
5	0.725	59.19
10	0.697	60.31
15	0.672	61.68
20	0.712	59.81
25	0.751	58.92
30	0.772	58.44

We expect a better results, so we try to understand why we fail to get closer to the 80% accuracy so we check how we can improve that even more than that 10

4.5 Phase 5 - roles and lanes of nodes

Another 2 features we add to the model are champions role and lane: We study the game a little deeper and found out that every player have its own lane in the map: BOTTOM, TOP, MIDDLE, JUNGLER. Every game "must" to have 1 TOP 1 MIDDLE 1 JUNGLER and 2 BOTTOM (from all of the tutorial we saw any other combination of champions probably lose) so we add to each edge the lane of each champion. The champions also have roles if this champion is more supportive character or offensive and there are more types, we add it to the model, the results were encouraging.

threshold	mse	% accuracy
0	0.543	72.01
5	0.576	69.89
10	0.463	78.07
15	0.512	74.68
20	0.499	76.37
25	0.563	70.04
30	0.543	72.12

From all of our experiments threshold of 10 gives the best result, probably without overfitting of too lower threshold and without underfitting of too high threshold.

5 Summary

From our results we can see that we cant yet predict really by only 2 players if they are going to win or not, but we have been getting closer for that. Lol is known to be one of the most strategy game that are out there these days, to know if 2 players are playing well together has to be tough task or this game would be really boring. Also we don't know much of this game and maybe with more knowledge about the champions attributes we could extract more features to make this more accurate.

We know that there are some bots that worth a lot of money and the professionals teams using to help them know if a team have more success rate to win than the other team. It can be very interesting to extend our model from see if 2 players play well to how 5 players will play and if it will be better than the other team.

There was a little disappointment that we couldn't reproduce Leskovec results but we learn a lot from it.