League Of Legends Data analysis:

Project Documentation

#### 

| Asmaa IsmailID: 201600780Anhar Hassan ID: 201601171 |  |
| --- | --- |

# 

# DATA COLLECTION:

In this stage, we used the python REQUESTS library to send api requests to the Riot developer’s portal and receive data. To collect the match JSON objects we had to follow this pipeline:

## 1- get a collection of summoner’s names :

We chose to get the summoners from the ‘NA1: americas’ server. We tried to have a realistic distribution of summoner ranks in the collected summoners set, and to do so we referred to the statistics publicly available on [League Of Graphs](https://www.leagueofgraphs.com/rankings/rank-distribution) to know the popularity ratios between the different ranks. We ended up collecting 6000 summoners decently distributed across ranks.

## 2- get match ids from summoner:

Before we can use the summoners names obtained in step 1 we need to convert them into globally unique PUUIDS by querying the summoners API. After that we used the 6000 puuids to get a total of 120 thousand match ids, which after accounting for some duplicates came down to a little shy of 117 thousand. We made sure to get only 20 recent matches for each summoner to avoid version crossovers and to limit the contribution of each single summoner in the data to eliminate potential skill bias.

## 3- get match objects from match ids:

This is by far the most time consuming step since it entails querying the Riot api over 100 thousand times which is extremely slow under the api rate limit.

# Data preprocessing and cleaning:

In this step, the goal is to simplify the complex nested JSON structure and separate out the parts that we will be using in our analysis to simplify the analysis logic and not repeat codes. We simplify the structure and separate out units without discarding any of the data.

## 1- extracting all-participants data frame and saving to csv file:

The all participants dataframe is the backbone on which most of the analysis tasks are performed. We get it by extracting the participants entry from the match JSONs and exploding it. So instead of having a list of 10 participant dictionary per row, we now have 10 times the number of rows and each row is a single participant JSON. We added the game id to the participants’ entries and made it into a composite key along with the team id so that the match and team that each participant belonged to can be easily identified.

## 2- extracting the bans information dataframe from the teams’ entry and saving it:

This is a small dataframe that has the champion names and ids in a column, and the number of times they were banned in another. We obtained it by exploding the teams entry to get a single team per row dataframe. Then selecting the bans entry and exploding it again to have the champion id and pick turn in every row. From there it was easy to group by the champions and count the pick turns.

## 3- extracting useful information from the datadragon and saving it in a csv for ease of access.

Maily we extracted the champion classes associated with each champion name / id , which wasn’t included in our dataset. We also extracted the items names and tags which were likewise not indicated in the dataset. We also used some of the images in the datadragon , mainly the champion and item images, to add to our visualizations.

### 

### Challenges and optimizations:

1- The biggest challenge in the project was the extremely slow and inefficient data collection process. For us the main limitation was the api request rate limit:

To combat this problem, we made 10 accounts with 10 api keys to collect the data concurrently in segments.

2- the very nested structure of match objects and the flood of information that we mostly don't need:

Instead of discarding any data, we took some time to do some de-nesting, exploding, and splitting the of the data into smaller units that we can easily access when needed, ex: all\_teams.csv, all\_participants.csv, champion\_bans.csv, ..etc.

# Data analysis:

### 1- REQUIREMENT 1: Champion win, pick and ban rates

We calculated these statistics as following:

We start with the all participants dataframe. To get the counts of each champion we simply aggregate by champion name & id then count. The pick rate can now be calculated as mentioned above. We also aggregate the sum of wins, then divide that by the counts to get the win rate. To get the ban rate we use the bans dataframe extracted in the data cleaning and simply divide the ban count column by the number of matches.

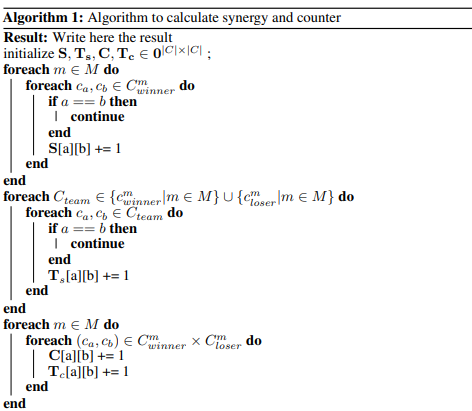
### 2-REQUIREMENT 2:champion synergy

We referred to this [paper](https://courses.cs.washington.edu/courses/cse547/21sp/old_projects/kim_etal.pdf) from the University of Washington in the calculation of the champion synergy. They define the synergy between two champions c1 and c2, belonging to the same team m1 or m2 as:

Which means the probability of the champion pair to belong to the same winning team over the probability of the champion pair to occur in any team (ie, its overall occurance).

We can simplify this in terms of counts instead of probabilities:

Which they provide a simple algorithm to calculate from a dataset:



Our own calculation steps:

1- Extract the relevant columns from the participants dataframe:

[gameAndTeam, champion id/name, win, role]

Then we filter out the DUO rows only.

2- we group the dataframe by the game+team composite key and aggregate the champions in a list. We now have a dataframe where each row is a single team in a match and has entries: list of champions who played duo in the team, and the win status of the team.

3- We then generate the combinations of every two non-identical elements in the champions list to convert the list of champions into a list of pairs where each pair is a tuple.

4- We now explode the champion pairs column to have every tuple pair in a single row.

5- we then sort the pair alphabetically to guarantee pair uniqueness irrelevant of order, then group by the pairs and aggregate their counts, and the sum of their win statuses. We now have a dataframe that looks something like this.

| championPairs | duo Occurrence Count | wins |
| --- | --- | --- |
| (jhin, zoe) | 15 | 10 |

6- we can simply calculate the synergy score by dividing the wins column by the counts column X 100% as indicated by the equations above.

### 3- REQUIREMENT 3: item win, and pick rates

Here we used the participants dataframe and extracted the following relevant columns: [items 0:through:5 , win]

1- this gives a dataframe with 7 columns; 6 columns for every item index and one column for the win status. We start by melting the 6 item columns into one item column where each row of 6 items is stacked vertically into one column and added to the new column ‘items’.

2- now we can group by the items and aggregate the count(items) and the sum(win). We get a data frame that looks something like this:

| items | countItems | TotalWinsPerItem |
| --- | --- | --- |
| 1003 | 1500 | 900 |

3- we calculate the pick and win rate exactly the same way we did in requirement one with the champions; the pick rate = counts col/total items, and win rate = total wins/ counts.

4- now the one step left is to translate the given numeric indices/ids representing items in our dataset into proper item names. We do this using the simplified items names data frame which we extracted from the datadragon in the preprocessing step.

### 4-REQUIREMENT 4:

### Here we had two tasks:

### -first: calculating item-champion synergies, we first extracted relevant columns from all\_participants- namely, the item, the champion, and win state, then melting the item lists again like above, then we create two necessary separate columns; one which aggregates the sum over all the values based on the pair of the item index and the champion name, and the other aggregates over these two in addition to the win state, then we drop the state of losing “where win = False”, join these together with the initial dataframe, then divides the later over the former “count where win = True/ total count”.

### - Second: calculating item-class synergies; this is similar to item-champion except that we first extracted classes from the dataset, joined them with our dataframe, exploded it , as there can be more than one class for each champion, and then proceeded with the same steps as above.

### 

### We had to calculate in the first task, the champion pick rate for each item as it helps in Task 5.

### 5- REQUIREMENT 5:

### We had to calculate in the first task, the champion pick rate for each item as it helps in Task 5.

So we randomly chose two champions”champs = ['Alistar', 'Anivia']”, we did some pre-processing: where we joined the dataframe in task1 from requirement 4 with the dataframe from task 2, where we care about 4 columns, general item pick rate and win rates, and champ-item synergy, and champ-item pick rate, we then aggregated these with item class from the datadragon, and created a column for #classes the current item belongs to, and finally made our heuristic.

After trials and errors, we thought this relation may result in the best suggestion: “ h = 0.75\*float(chSynergy)/float(ch\_item\_pick) + 0.25\*float(win)/float(pick) -5\*float(n)”

Where n = #.classes for the item.

The rational beyond this is that the item-champ synergy should have the highest weight, but reduced by dividing it by how frequent the champ picks the item, while the general item winning rate can also has a less effect, and decreased by its popularity as well, and the number of classes adds a penalty as even if the item classes are not within the restricted classes, they still add restrictions for the upcoming possible item candidates.

We have two functions after that:

* is-valid; is passed the item, and the current restricted list, then checks the item tags and checks whether the choice of this item is valid or not, and returns the new restricted tags in the first case.
* Suggestion: the main function, is passed the name of the champion, then it checks all its-associated elements, sorts according to the heuristic, checks the validity of the items with the highest heuristics, if yes, it updates the list of restricted classes, if not, it moves on to the next item, and so on, until it has a list of 6 items.

### 6-REQUIREMENT EXTRA-1: Match result prediction MLlib:

Here, we attempt to use some of the match info data in our dataset as predictive features to train a simple classifier ML model to predict match outcomes. The classification output is either 1 if the blue team (id 100) wins, and a zero if the red team (id 200) wins (the blue team loses).the features we selected are the following:

**'assists',**

**'damageDealtToObjectives',**

**'damageDealtToTurrets',**

**'damageSelfMitigated',**

**'deaths',**

**'kills',**

**'magicDamageDealt',**

**'neutralMinionsKilled',**

**'physicalDamageDealt',**

**'timeCCingOthers', 'wardsPlaced',**

**'doubleKills',**

**'tripleKills',**

**'pentaKills',**

**'dragonKills',**

**'goldEarned',**

**'goldSpent',**

**'turretKills',**

**'turretTakedowns',**

**'turretsLost',**

**'unrealKills',**

**'visionScore'**

We start by extracting the features from the participants dataset as usual- along with the win column which will produce our labels. Next we combine all of the feature columns into one column ‘features’ that contains all the features in a list. We then group by game+team ids composite key to aggregate each team’s features in one row. We then group and aggregate again by the game id to have every match info in one row. By now we have a data frame of 2 main columns; features, and win status in the following format:

| game ID | features | win |
| --- | --- | --- |
| 1234567 | [ [list of team1 feats] , [list of team2 feats]] | Team1 win? =1 or 0 |

We then vectorize the features using pysparks’s vector assembler from the ml package. We split the data into train and test sets using pyspark’s randomsplit function then we fit a decision tree classifier on the data using pyspark MLlib’s built in classifiers.

### 7-REQUIREMENT EXTRA-2: distribution of champion win rate aver different lane and the best lane for each champion:

The goal of this analysis step is to both find the best (associated with highest win rate) lane for every champion and to visualize the overall win rate distribution for each champion on different lanes. We start from the participants df and extract the champions, lane, and win columns. We then group by both the champion and lane pairs and aggregate the wins for each unique pair. We save this data frame for visualization of the win/lane distribution. We then find the maximum win lane by windowing the df over the unique champions in the champion column, and aggregate in every window by finding the max of wins. We finally retrieve the record with the highest win rate by filtering out the windows for win == max(win).

----------------------

The project drive, including the data, notebooks, graphs, and outputs, is [here](https://drive.google.com/drive/folders/1163mhhRzGaDZe19zFuymtMoR3xx-tfIo?usp=sharing).