Metashift

Matthew D'Souza & Dongyu Zheng

September 1, 2015

Abstract

Created for the League of Legends API Challenge 2.0, Metashift is a web app which uses Riot Games' API to query the game information and present a graphical analysis of thousands of League of Legends matches before and after the 5.13 patch. Many of the changes were made to the items with respect to the Ability Power stat. The goal of this project is to systematically look for a shift in the 'metagame' - the most popular (and often most successful) characters played both before and after patch 5.13.

Contents

1 Features

1.1 Completed

- Graphical and charted display of change in win rate and popularity per champion
- Charted display of change in champion roles
- Sortable by data type, game mode, region, & rate type

1.2 Our Ideas / Selected for Development

- Purchase/win rates of items with respect to champions
- Time of purchases (earlier/later 'power spikes')
- Look for any emerging meta picks
- More general game trends of before & after: first dragon, first baron, first tower, average kills at a certain time, etc.

1.3 Complications

- High computing time - there is A LOT of data to download and parse through
- Poor script testing - overnight scripts failed multiple times
- No time to work on this - both developers have other work to do

2 Backend

Although our web app is a static site, we used Django help us manage the database as we are both experienced in Python and Django.

2.1 Database

We used SQlite for our databse. We were originally going to use PSQL, but neither of us could host and we did not want to spend money renting a server. The database schema was set up using Django models, and there are tables for match data, champions, and items.

The match table has columns for match_id, region, version, gamemode, and data. The data field is a JSON string dump of the data we receive from the API.

Both the champion and item tables have region, version, gamemode, picks, wins, and name. Champion has the extra field of roles which will store a JSON dump of its K-Means Clustering result.

2.2 Downloading

We ran Python scripts to download and populate the database with data required.

2.3 Counting

To count how many picks/wins a champion or item has, we would iterate through each match of the specified region, version, and gamemode to count and save the picks and wins.

2.4 K-Means Clustering

A simple machine learning algorithm was implemented to see if a champion's in-game role differed between the two game versions.

There are 5 roles (K=5): fighter, mage, marksman, support, and tank. Each role is assigned their archetypal items, ie. mage has Void Staff and Rabadon's Deathcap while tank has Warmog's Armor, etc. A 'closeness function' was defined for scoring a set of item's closeness to a role. An item set's score was decided to be calculated by iterating through each item in the item set and summing its popularity within a role - the percentage of players who buy the item as such role. That player's item set would be clustered into its highest scoring role.

To make this more clear, observe the following example:

- A player's item set includes: Void Staff, Rabadon's Deathcap, and Athene's Unholy Grail.
- The following are roles, items and their popularities within that role:
  - Marksman: Void Staff = 0.03; Rabadon's Deathcap = 0.008; Athene's Unholy Grail = 0.01
  - Mage: Void Staff = 0.23; Rabadon's Deathcap = 0.17; Athene's Unholy Grail = 0.18

- This player's item set would score (0.03 + 0.008 + 0.01) = 0.048 points for Marksman while scoring (0.23 + 0.17 + 0.18) = 0.58 points for Mage.
- Clearly, the score for the role of Mage is higher, so this player's item set would be put into the Mage cluster.

After iterating through each player of each match, we would now have 5 role clusters with player data within them, including their item set and champion id.

Now that we have clusters, we would generate the next iteration of clustering data with an item's popularity being equal to the number of times it has appeared in that role divided by the total numbers of items bought in that cluster. We repeated this process 8 times.

To determine a champion's role, we would group each player of each match into the 5 clusters, then we would divide the number of times a champion id appears within a cluster over the total number of times that champion was selected.

2.5 Exporting

To export champion data after counting the number of picks and wins and cluster the roles for a given version/gamemode/region, we would iterate through each champion to produce a JSON dump into a file to be used in our front end.

3 Frontend

The app is hosted on a static GitHub page, and thus relies exclusively on client-side processing with JavaScript and some of its libraries.

3.1 Graphing

Graphing was achieved using Highcharts. We aimed to make the whole process as modular as possible to adapt to different queues (both ranked and normal), regions (NA, LAS, KR, etc.), and other factors. This involved writing various functions, including one to aggregate all the selected regions' and queues' JSON files into one combined one.

Finding the ideal graph type to represent the data proved to be quite a challenge. The graph we chose definitely needed to show the change in some way, but with information like win rates, the actual value (especially with respect to the ideal 50%) also ends up being very relevant.

The graph we ended up deciding on was a columnrange graph - essentially a bar graph confined to a maximum and minimum value. This still proved to be less intuitive than we desired. We were able to improvise, using symbols to plot triangular points and create makeshift arrows. Doing this gave us a much more easily understandable graph.

3.2 Table

Tables were generated using DataTables. Once the combined JSON file is aggregated from the graphing step, the data is then passed and put into a DataTable. This was also done with modularity in mind, and information about win rate/popularity as well as roles can be toggled on or off. Every property is sortable, to offer a better opportunity for analysis/inquiry.

4 Hindsight

At the start, we were both pretty ambitious about our goals for the project. It quickly became clear that we were ill-prepared for a lot of the issues we would run into. These issues, as tedious as they may have been, gave us valuable first-hand lessons in managing such a large project.

- Using a local SQLite database and synchronous API calls is probably not the best way to get information for 400,000 matches.
- Starting a project with limited free time and a deadline will inevitably lead to a sacrifice in either sleep or project quality.
- Putting more emphasis on items - admittedly the actual topic of the challenge - would probably have been a good idea.
- Ignoring generic items such as boots and trinkets would have improved the accuracy of our K-Means Clustering.
- Being familiar with the libraries/APIs you are using can make a project's development infinitely easier.