Allen Roush

CIS 453 Project Proposal

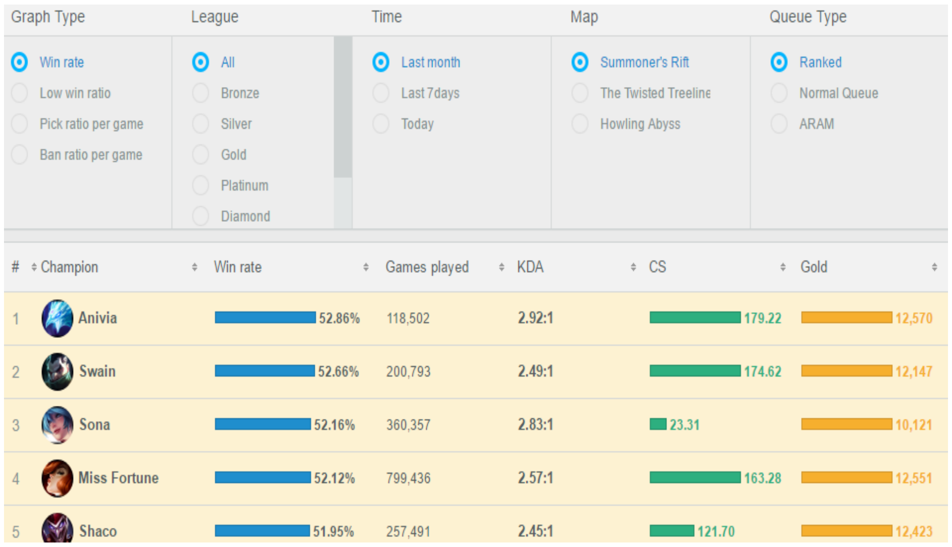
6/4/2017

League of Data: Making Intelligent Game Decisions Using Data.

The problem:

League of Legends is the most popular online PC game in the world at this moment. With over 67 million players, *Riot games* (the creators of League of Legends) created a "Big Data" division that tackled the complex task of categorizing and recording games played by each player[1]. This "Big Data" Division also does internal data-mining work that is used to balance the game. Every conceivable raw statistic (and a considerable number of computed ones) are available for anyone willing to learn how to use the Riot API.

The Riot API is designed for developers who are interested in using their own tools to work with the results. Many websites have been set up that take advantage of the Riot API to give players access to certain services, such as average statistics about a champion[2] or what they are currently "ranked as" in game modes that don't tell you your rank[3]. Large numbers of players find these statistics useful (including the author), and in op.gg's case, have extremely large userbases. Both websites use data-mining techniques to generate actionable information for their readers.



Graph Type: [✓ Win rate] Low win ratio, Pick ratio per game, Ban ratio per game
League: [✓ All] Bronze, Silver, Gold, Platinum, Diamond
Time: [✓ Last month] Last 7days, Today
Map: [✓ Summoner's Rift] The Twisted Treeline, Howling Abyss
Queue Type: [✓ Ranked] Normal Queue, ARAM

#	Champion	Win rate	Games played	KDA	CS	Gold
1	Anivia	52.86%	118,502	2.92:1	179.22	12,570
2	Swain	52.66%	200,793	2.49:1	174.62	12,147
3	Sona	52.16%	360,357	2.83:1	23.31	10,121
4	Miss Fortune	52.12%	799,436	2.57:1	163.28	12,551
5	Shaco	51.95%	257,491	2.45:1	121.70	12,423

Fig 1: Sample Statistics provided by na.op.gg for the current patch

[1] (more details found here: http://www.riotgames.com/departments/big-data also see their section for the riot API).
[2] http://champion.gg/
[3] https://na.op.gg/

For starters, the amount of games that each website curate through is currently around 2.5 million (and increases nearly every month). That's *Per Patch* which happens around every 2 weeks. That is a *tiny, tiny* slice of the total number of games of League of Legends played. Each website stores it's statistics for previous patches as well. This presents the player base with a magnificent opportunity to mine this massive amount of data for information. We present a number of examples of the novel benefits of data mining League of Legends data.

League of Legends is a 5 vs 5 "Multiplayer Online Battle Arena". Each player on a team will select a "Champion" out of <200 available choices to play for the duration of the game. These champions are picked in a "Drafting" setting, that includes 6 bans before the match. Many champions have abilities which "Synergize" with each other, and many of these "Synergies" are known to players within the community. Champions are divided up into "Archetypes" that are designed to play with specific other Archetypes. Unfortunately, no amount of player learned game-knowledge will suffice for the power of a computer to association mine strong rules about what champions are "successful" when picked with others.

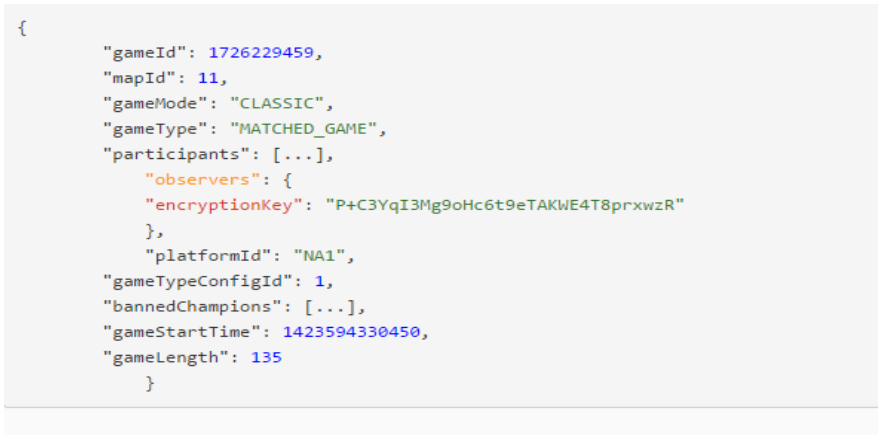
Another Mechanic within the game are "Item sets". When the players get into a game with their selected champion, they will have choices about which "Items" to build. There are over 100 items in the game, and many of them have their own synergies and weaknesses. Some items are plain bad on champions that don't effectively use them, and the study of optimal item builds (which changes each 2 week patch) is one of the main reasons why Professional League of Legends teams hire Data Mining professionals as "Analysts" for their team.

Unfortunately, Riot Games strictly enforces a policy of them being the final curators and distributors of their own data[4]. This prevents any of the websites that curate and beautify the data from independently making it available (like in a zip file). This means that barring the creation of a very good webscraper, the only way that the raw game data will be available to a developer is through using the riot API to retrieve it myself.

The Project Goal:
As such, existing web services are not good enough for anyone that wants to do their own independent analysis of data on their own time. Many Professional League of Legends teams have vested interests in this realm, and the organizations that sponsor them hire professional analysts who very likely utilize riot API data to help their players perform better. I intend to Association Mine and Classify useful League of Legends data.

[4] https://developer.riotgames.com/general-policies

The Data: Conveniently, the Riot API stores and makes it's data available using the JSON format:



```
{
        "gameId": 1726229459,
        "mapId": 11,
        "gameMode": "CLASSIC",
        "gameType": "MATCHED_GAME",
        "participants": [...],
            "observers": {
            "encryptionKey": "P+C3YqI3Mg9oHc6t9eTAKWE4T8prxwzR"
            },
            "platformId": "NA1",
        "gameTypeConfigId": 1,
        "bannedChampions": [...],
        "gameStartTime": 1423594330450,
        "gameLength": 135
            }
```

Fig 2: Sample Data

Fig 2: shows an example table. This is an object that stores a currently played game. The total database schema that the RiotAPI can talk to is listed online. This table acts as my starting point for mining additional data from them, so I'll go into some detail about it.

gameId: is a unique number that identifies the specific game in the database. "Crawling the database" will therefor be either be generating random gameId's or using a large list of known ones.

mapId: corresponds to the ID of the map that the game was played on. This is almost always dependent on the gameMode.

gameMode: Usually "CLASSIC", though sometimes other things depending on if it's a custom game or special rule set.

gameType: Either "Matched_Game" or "Custom" depending on what it's set as

Participants: This is an object that gives access to an encryption key that can be used to spectate the game as it's running. I'm not immediately sure how this will be representable in SQL, but it will give richness to the data being processed by both Database Languages.

BannedChampions: this is an array of champion ID's, that correspond to specific banned champions for that game. There are 6 banned champions per match.

gameLength: is stored in seconds and returns a static value at the time of the query. I may need to write some logic to throw out anomously high values (games that have crashed on the server side and are not properly shut down).

Data's Use-Value:

This table would be useful to an analyst. If you wrote a crawler that could return the information of *all* the currently going games, you'd very quickly have a large sample size of games played. This way I could start to generate ban rates based on what champions are banned by each game.

Within a specific “gameID”, another table labeled “Champions” lists the exact champions picked by each of the 10 players in the match, labeled with which team picked what champion and in what order they were picked in. I can use this to directly farm a large number of “Team compositions” (sets of 5 champion picks which corresponds to the draft of a single team within a single game). I intend to mine the associations between each champion.

Of course, there are many available tables and lots of data available to be used. Listing it all would take a book. Much of my choices for data and quarries will be based on finding interesting association and classification rules.

Project Methodology:

I’ve worked with the RiotAPI before in the CIS 452 project, and I have experience setting up and using a MongoDB database on my home machine. This will easily store the RiotAPI data that I gather. Converting this eventual data into comma separated values may be tricky, but I’m sure that this is a well solved problem.

Regrettably, Riot games has changed the way that the RiotAPI returns data ever since I last used it. The specifics of the change are documented here[5]. It makes it so that in order for me to farm the match history of a specific player, I’m forced to send O(n) requests instead of O(1) requests to acquire it. Since the average ranked-ready league of legends player has played hundreds of games, this means that I will be spending potentially hundreds of requests to the RiotAPI acquiring this data, when in the past it was a single request for the whole match history. The RiotAPI maintains a 500 request per 10 minute interval cap, which is going to do nothing but complicate the data farming. I have my spare data from the CIS 452 project, but it is not necessarily detailed enough for the analysis I want to do. This will be a significant challenge in my project (but there are most likely workarounds to this)

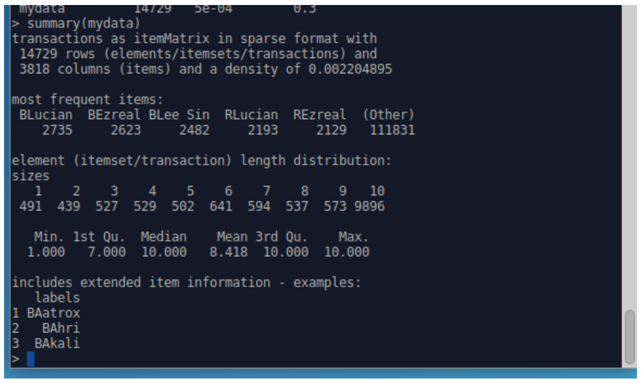
When I finally get the data, I must make choices about how to organize it for learning with. The homework has left me comfortable working with data-sets in the thousands of entries with Excel and other tools. I can use built-in functions found in Weka to convert the data into a readable form.

I imagine that I will have a CSV file listing 10 different champions and try to association mine from that. I run into a problem of enumeration if I want to mark which team a champion is picked on (because it’s important to note if a champion picked in response to a friendly champion, or enemy champion… a champion might be a bad pick on the friendly team but good on the enemy team). I’m not immediately sure how to solve this problem, though researchers in Texas have written a paper that proposes solutions to these problem (Nayak and Cook 2001).

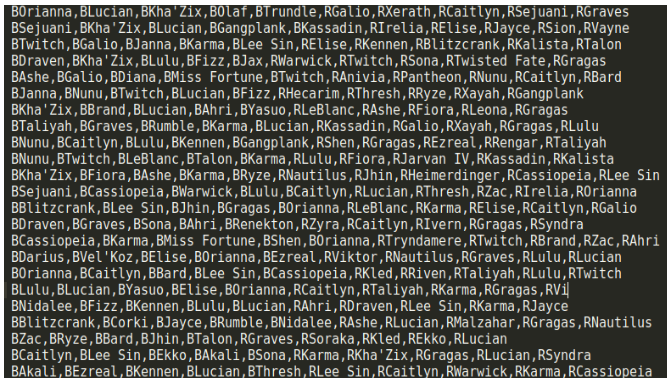
Results:

[5]https://www.reddit.com/r/leagueoflegends/comments/68lakk/changes_to_riot_api_making_everything_harder_for/

I wrote a crawler in Python that scans the top rated master ranked players match history and pulls the champions played in their game. This crawler then looks at the match history of all of the players that played in each game, and pulls match data from those games recursively. I manually appended “B” or “R” to the beginning of each champion name (to distinguish between blue and red side of the map, which also distinguishes between team1 and team2).



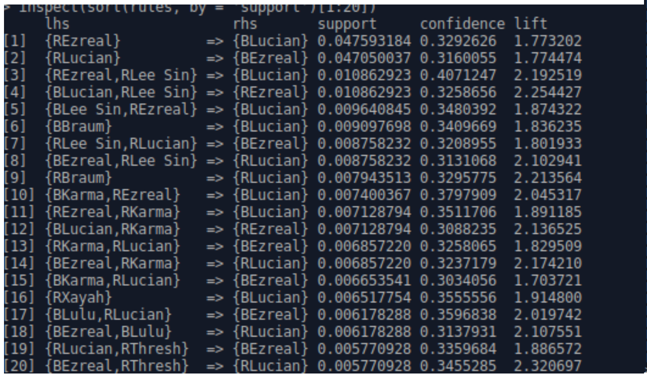
```
  mydata         14729   5e-04        0.3
> summary(mydata)
transactions as itemMatrix in sparse format with
 14729 rows (elements/itemsets/transactions) and
 3818 columns (items) and a density of 0.002204895

most frequent items:
 BLucian  BEzreal BLee Sin  RLucian  REzreal  (Other)
    2735     2623     2482     2193     2129   111831

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10
 491  439  527  529  502  641  594  537  573 9896

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   7.000  10.000   8.418  10.000  10.000

includes extended item information - examples:
   labels
1 BAatrox
2   BAhri
3  BAkali
>
```



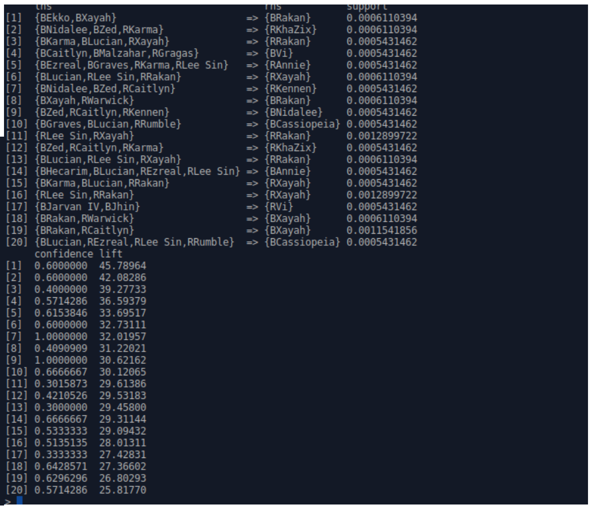
```
BOrianna,BLucian,BKha'Zix,BOlaf,BTrundle,RGalio,RXerath,RCaitlyn,RSejuani,RGraves
BSejuani,BKha'Zix,BLucian,BGangplank,BKassadin,RIrelia,RElise,RJayce,RSion,RVayne
BTwitch,BGalio,BJanna,BKarma,BLee Sin,RElise,RKennen,RBlitzcrank,RKalista,RTalon
BDraven,BKha'Zix,BLulu,BFizz,BJax,RWarwick,RTwitch,RSona,RTwisted Fate,RGragas
BAshe,BGalio,BDiana,BMiss Fortune,BTwitch,RAnivia,RPantheon,RNunu,RCaitlyn,RBard
BJanna,BNunu,BTwitch,BLucian,BFizz,RHecarim,RThresh,RRyze,RXayah,RGangplank
BKha'Zix,BBrand,BLucian,BAhri,BYasuo,RLeBlanc,RAshe,RFiora,RLeona,RGragas
BTaliyah,BGraves,BRumble,BKarma,BLucian,RKassadin,RGalio,RXayah,RGragas,RLulu
BNunu,BCaitlyn,BLulu,BKennen,BGangplank,RShen,RGragas,REzreal,RRengar,RTaliyah
BNunu,BTwitch,BLeBlanc,BTalon,BKarma,RLulu,RFiora,RJarvan IV,RKassadin,RKalista
BKha'Zix,BFiora,BAshe,BKarma,BRyze,RNautilus,RJhin,RHeimerdinger,RCassiopeia,RLee Sin
BSejuani,BCassiopeia,BWarwick,BLulu,BCaitlyn,RLucian,RThresh,RZac,RIrelia,ROrianna
BBlitzcrank,BLee Sin,BJhin,BGragas,BOrianna,RLeBlanc,RKarma,RElise,RCaitlyn,RGalio
BDraven,BGraves,BSona,BAhri,BRenekton,RZyra,RCaitlyn,RIvern,RGragas,RSyndra
BCassiopeia,BKarma,BMiss Fortune,BShen,BOrianna,RTryndamere,RTwitch,RBrand,RZac,RAhri
BDarius,BVel'Koz,BElise,BOrianna,BEzreal,RViktor,RNautilus,RGraves,RLulu,RLucian
BOrianna,BCaitlyn,BBard,BLee Sin,BCassiopeia,RKled,RRiven,RTaliyah,RLulu,RTwitch
BLulu,BLucian,BYasuo,BElise,BOrianna,RCaitlyn,RTaliyah,RKarma,RGragas,RVi
BNidalee,BFizz,BKennen,BLulu,BLucian,RAhri,RDraven,RLee Sin,RKarma,RJayce
BBlitzcrank,BCorki,BJayce,BRumble,BNidalee,RAshe,RLucian,RMalzahar,RGragas,RNautilus
BZac,BRyze,BBard,BJhin,BTalon,RGraves,RSoraka,RKled,REkko,RLucian
BCaitlyn,BLee Sin,BEkko,BAkali,BSona,RKarma,RKha'Zix,RGragas,RLucian,RSyndra
BAkali,BEzreal,BKennen,BLucian,BThresh,RLee Sin,RCaitlyn,RWarwick,RKarma,RCassiopeia
```

Fig 3: Summary of Data and example transactions

I’d originally planned to simply mine some association rules with my own python implementation of the Apriori Algorithim and try to parse through the information by hand. Besides being tedious, this would not necessary give me interesting data. As a result, I came across a paper from (Hahsler and Chelluboina 2017) about a package written in R called “arulesViz” which helps facilitate the visualization of mined association rules. I used this project as an excuse to learn R, and the results have been very interesting. I generated 8000 association rules with minimum support of 7 (0.0005) games and minimum confidence of 0.30



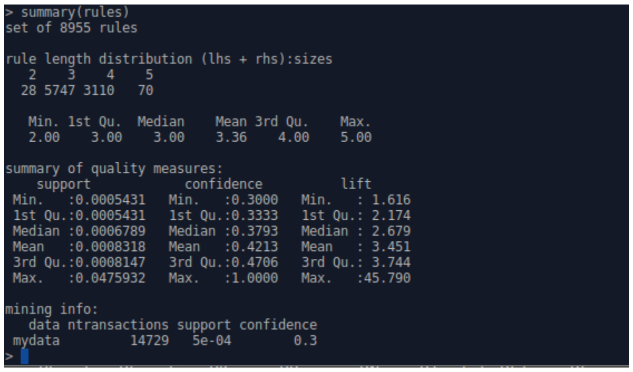
```
> inspect(sort(rules, by = "support")[1:20])
     lhs                  rhs       support     confidence lift
[1]  {REzreal}         => {BLucian} 0.047593184 0.3292626  1.773202
[2]  {RLucian}         => {BEzreal} 0.047050037 0.3160055  1.774474
[3]  {REzreal,RLee Sin} => {BLucian} 0.010862923 0.4071247  2.192519
[4]  {BLucian,RLee Sin} => {REzreal} 0.010862923 0.3258656  2.254427
[5]  {BLee Sin,REzreal} => {BLucian} 0.009640845 0.3480392  1.874322
[6]  {BBraum}          => {BLucian} 0.009097698 0.3409669  1.836235
[7]  {RLee Sin,RLucian} => {BEzreal} 0.008758232 0.3208955  1.801933
[8]  {BEzreal,RLee Sin} => {RLucian} 0.008758232 0.3131068  2.102941
[9]  {RBraum}          => {RLucian} 0.007943513 0.3295775  2.213564
[10] {BKarma,REzreal}  => {BLucian} 0.007400367 0.3797909  2.045317
[11] {REzreal,RKarma}  => {BLucian} 0.007128794 0.3511706  1.891185
[12] {BLucian,RKarma}  => {REzreal} 0.007128794 0.3088235  2.136525
[13] {RKarma,RLucian}  => {BEzreal} 0.006857220 0.3258065  1.829509
[14] {BEzreal,RKarma}  => {RLucian} 0.006857220 0.3237179  2.174210
[15] {BKarma,RLucian}  => {BEzreal} 0.006653541 0.3034056  1.703721
[16] {RXayah}          => {BLucian} 0.006517754 0.3555556  1.914800
[17] {BLulu,RLucian}   => {BEzreal} 0.006178288 0.3596838  2.019742
[18] {BEzreal,BLulu}   => {RLucian} 0.006178288 0.3137931  2.107551
[19] {RLucian,RThresh} => {BEzreal} 0.005770928 0.3359684  1.886572
[20] {BEzreal,RThresh} => {RLucian} 0.005770928 0.3455285  2.320697
```



```
     lhs                                     rhs           support
[1]  {BEkko,BXayah}                       => {BRakan}      0.0006110394
[2]  {BNidalee,BZed,RKarma}               => {RKhaZix}     0.0006110394
[3]  {BKarma,BLucian,RXayah}              => {RRakan}      0.0005431462
[4]  {BCaitlyn,BMalzahar,RGragas}         => {BVi}         0.0005431462
[5]  {BEzreal,BGraves,RKarma,RLee Sin}    => {RAnnie}      0.0005431462
[6]  {BLucian,RLee Sin,RRakan}            => {RXayah}      0.0006110394
[7]  {BNidalee,BZed,RCaitlyn}             => {RKennen}     0.0005431462
[8]  {BXayah,RWarwick}                    => {BRakan}      0.0006110394
[9]  {BZed,RCaitlyn,RKennen}              => {BNidalee}    0.0005431462
[10] {BGraves,BLucian,RRumble}            => {BCassiopeia} 0.0005431462
[11] {RLee Sin,RXayah}                    => {RRakan}      0.0012899722
[12] {BZed,RCaitlyn,RKarma}               => {RKhaZix}     0.0005431462
[13] {BLucian,RLee Sin,RXayah}            => {RRakan}      0.0006110394
[14] {BHecarim,BLucian,REzreal,RLee Sin} => {BAnnie}      0.0005431462
[15] {BKarma,BLucian,RRakan}              => {RXayah}      0.0005431462
[16] {RLee Sin,RRakan}                    => {RXayah}      0.0012899722
[17] {BJarvan IV,BJhin}                   => {RVi}         0.0005431462
[18] {BRakan,RWarwick}                    => {BXayah}      0.0006110394
[19] {BRakan,RCaitlyn}                    => {BXayah}      0.0011541856
[20] {BLucian,REzreal,RLee Sin,RRumble}  => {BCassiopeia} 0.0005431462
     confidence lift
[1]  0.6000000  45.78964
[2]  0.6000000  42.08286
[3]  0.4000000  39.27733
[4]  0.5714286  36.59379
[5]  0.6153846  33.69517
[6]  0.6000000  32.73111
[7]  1.0000000  32.01957
[8]  0.4090909  31.22021
[9]  1.0000000  30.62162
[10] 0.6666667  30.12065
[11] 0.3015873  29.61386
[12] 0.4210526  29.53183
[13] 0.3000000  29.45800
[14] 0.6666667  29.31144
[15] 0.5333333  29.09432
[16] 0.5135135  28.01311
[17] 0.3333333  27.42831
[18] 0.6428571  27.36602
[19] 0.6296296  26.80293
[20] 0.5714286  25.81770
>
```

Fig 4: first 20 association rules generated sorted by relative support (left) and by lift (right)

A quick glance at the summary of the gathered association rules tells me that I’m going to have to find ways to visualize them graphically:



```
> summary(rules)
set of 8955 rules

rule length distribution (lhs + rhs):sizes
   2    3    4    5 
  28 5747 3110   70 

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   2.00    3.00    3.00    3.36    4.00    5.00 

summary of quality measures:
    support           confidence          lift       
 Min.   :0.0005431   Min.   :0.3000   Min.   : 1.616  
 1st Qu.:0.0005431   1st Qu.:0.3333   1st Qu.: 2.174  
 Median :0.0006789   Median :0.3793   Median : 2.679  
 Mean   :0.0008318   Mean   :0.4213   Mean   : 3.451  
 3rd Qu.:0.0008147   3rd Qu.:0.4706   3rd Qu.: 3.744  
 Max.   :0.0475932   Max.   :1.0000   Max.   :45.790  

mining info:
   data ntransactions support confidence
 mydata         14729   5e-04        0.3
> 
```

Fig 5: Statistical Summary of mined association rules, notice the 5 figure summary of Support, Confidence and Lift.

These statistics help to evaluate the data, notably the average lift measure helps us evaluate if a rule is “Interesting” or not relative to the rest.

Still, it’d be best to get a graphical representation of the association rules.

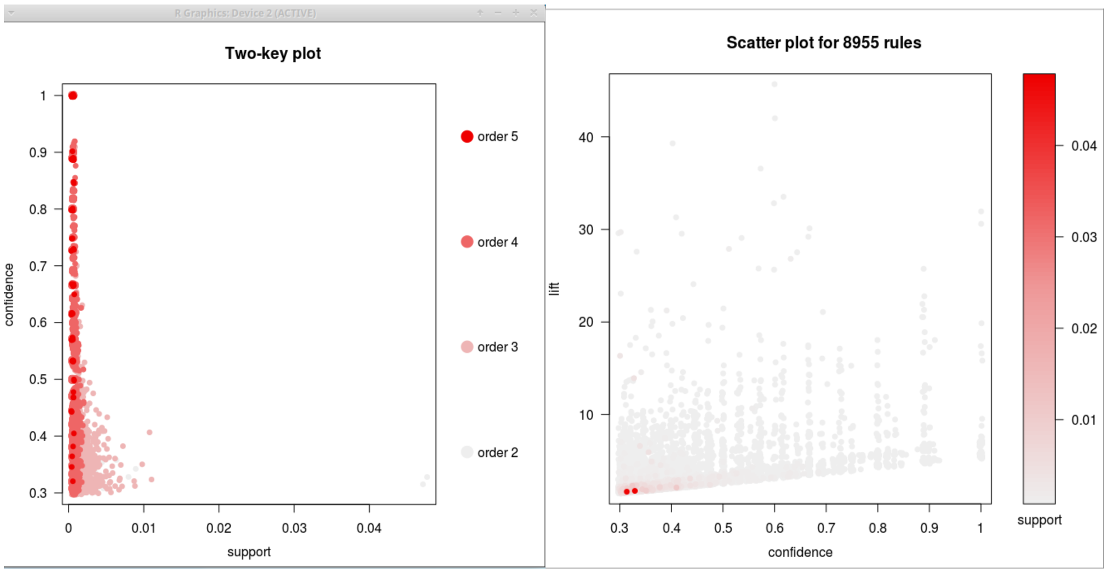


Fig 6: Two-Key Plot of rules (Order = size of rule), Fig 7: Scatterplot of rules, note the two high support rules in the bottom-left.

We see that the rule with high relative support to the rest are mainly grouped in the size 2-3 regions. We also observe that the highest support rules (of which there are 2-3 very strong ones) have a relatively low lift and confidence. These high support red dots correspond to very popular champion in league such as Lucian and Lee-Sin.

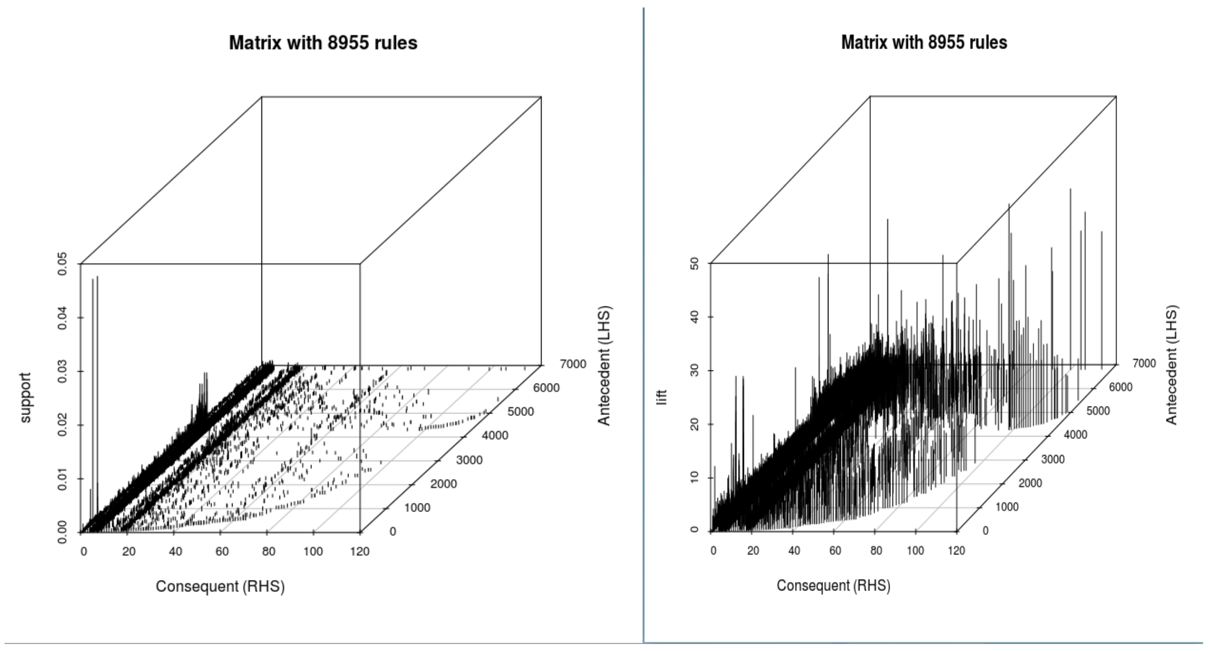


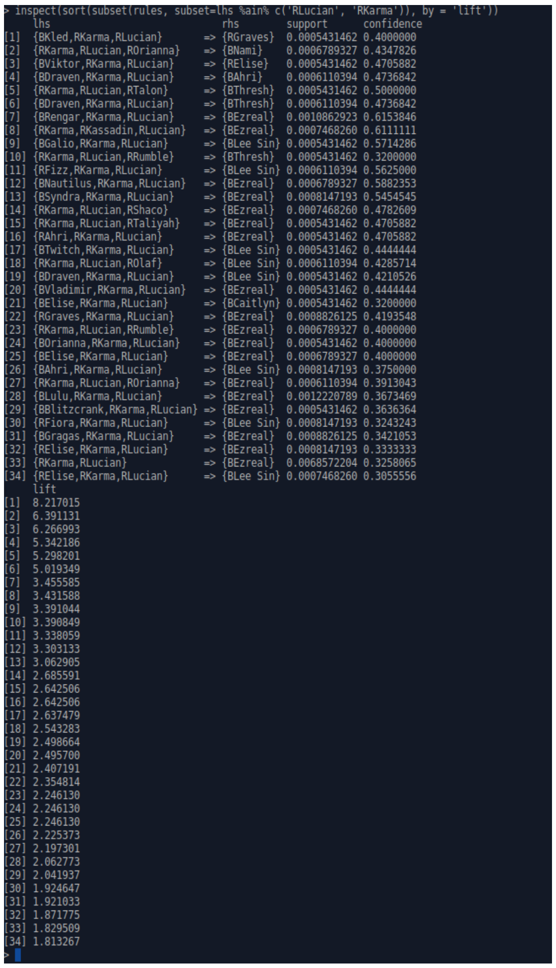
Fig 8: 3D Matrix distribution of rules by Support (left) and Lift (right).

From Fig 8 we observe the same 2 super strong association rules identified in Fig 7 but with a better sense of the support distribution within the remaining rules.

Looking back at this data, we find interesting knowledge within the data. High support champions like Lucian, and Ezreal are found in 4% of analyzed games: considerably higher than the average of 0.2% average champion pick rate.

The high lift rules are for a recently release champion (Xayah) who has a special mechanics with another champion (Rakan) that incentivizes picking them together. The other high lift rules are about certain large team-comps implying less popular champions like Nidalee or Annie.

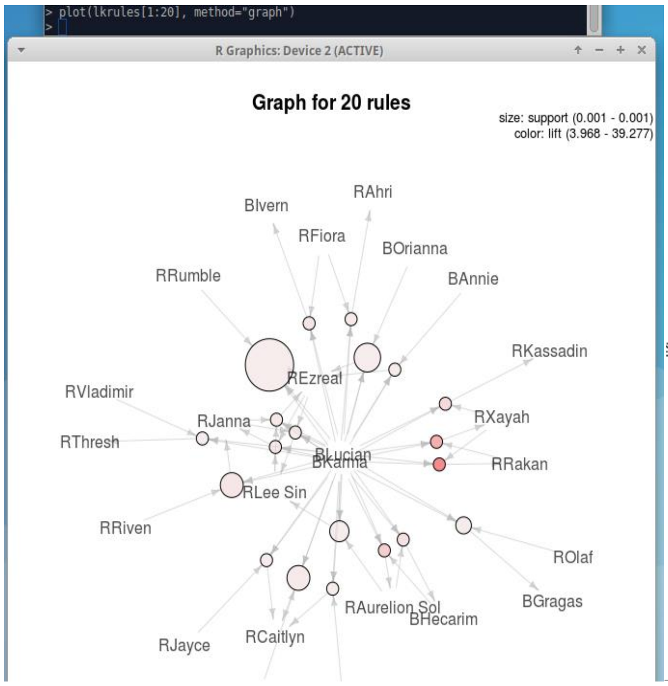
From this point on, I became interested in analyzing subsets of the mined association rules. I choose to analyze the association rules with Red-side Lucian and Red-Side Karma. This netted me 48 rules.



```
> inspect(sort(subset(rules, subset=lhs %ain% c('RLucian', 'RKarma')), by = 'lift'))
     lhs                                rhs        support      confidence
[1]  {BKled,RKarma,RLucian}          => {RGraves}  0.0005431462 0.4000000
[2]  {RKarma,RLucian,ROrianna}       => {BNami}    0.0006789327 0.4347826
[3]  {BViktor,RKarma,RLucian}        => {RElise}   0.0005431462 0.4705882
[4]  {BDraven,RKarma,RLucian}        => {BAhri}    0.0006110394 0.4736842
[5]  {RKarma,RLucian,RTalon}         => {BThresh}  0.0005431462 0.5000000
[6]  {BDraven,RKarma,RLucian}        => {BThresh}  0.0006110394 0.4736842
[7]  {BRengar,RKarma,RLucian}        => {BEzreal}  0.0010862923 0.6153846
[8]  {RKarma,RKassadin,RLucian}      => {BEzreal}  0.0007468260 0.6111111
[9]  {BGalio,RKarma,RLucian}         => {BLee Sin} 0.0005431462 0.5714286
[10] {RKarma,RLucian,RRumble}        => {BThresh}  0.0005431462 0.3200000
[11] {RFizz,RKarma,RLucian}          => {BLee Sin} 0.0006110394 0.5625000
[12] {BNautilus,RKarma,RLucian}      => {BEzreal}  0.0006789327 0.5882353
[13] {BSyndra,RKarma,RLucian}        => {BEzreal}  0.0008147193 0.5454545
[14] {RKarma,RLucian,RShaco}         => {BEzreal}  0.0007468260 0.4782609
[15] {RKarma,RLucian,RTaliyah}       => {BEzreal}  0.0005431462 0.4705882
[16] {RAhri,RKarma,RLucian}          => {BEzreal}  0.0005431462 0.4705882
[17] {BTwitch,RKarma,RLucian}        => {BLee Sin} 0.0005431462 0.4444444
[18] {RKarma,RLucian,ROlaf}          => {BLee Sin} 0.0006110394 0.4285714
[19] {BDraven,RKarma,RLucian}        => {BLee Sin} 0.0005431462 0.4210526
[20] {BVladimir,RKarma,RLucian}      => {BEzreal}  0.0005431462 0.4444444
[21] {BElise,RKarma,RLucian}         => {BCaitlyn} 0.0005431462 0.3200000
[22] {RGraves,RKarma,RLucian}        => {BEzreal}  0.0008826125 0.4193548
[23] {RKarma,RLucian,RRumble}        => {BEzreal}  0.0006789327 0.4000000
[24] {BOrianna,RKarma,RLucian}       => {BEzreal}  0.0005431462 0.4000000
[25] {BElise,RKarma,RLucian}         => {BEzreal}  0.0006789327 0.4000000
[26] {BAhri,RKarma,RLucian}          => {BLee Sin} 0.0008147193 0.3750000
[27] {RKarma,RLucian,ROrianna}       => {BEzreal}  0.0006110394 0.3913043
[28] {BLulu,RKarma,RLucian}          => {BEzreal}  0.0012220789 0.3673469
[29] {BBlitzcrank,RKarma,RLucian} => {BEzreal}  0.0005431462 0.3636364
[30] {RFiora,RKarma,RLucian}         => {BLee Sin} 0.0008147193 0.3243243
[31] {BGragas,RKarma,RLucian}        => {BEzreal}  0.0008826125 0.3421053
[32] {RElise,RKarma,RLucian}         => {BEzreal}  0.0008147193 0.3333333
[33] {RKarma,RLucian}                => {BEzreal}  0.0068572204 0.3258065
[34] {RElise,RKarma,RLucian}         => {BLee Sin} 0.0007468260 0.3055556
     lift
[1]  8.217015
[2]  6.391131
[3]  6.266993
[4]  5.342186
[5]  5.298201
[6]  5.019349
[7]  3.455585
[8]  3.431588
[9]  3.391044
[10] 3.390849
[11] 3.338059
[12] 3.303133
[13] 3.062905
[14] 2.685591
[15] 2.642506
[16] 2.642506
[17] 2.637479
[18] 2.543283
[19] 2.498664
[20] 2.495700
[21] 2.407191
[22] 2.354814
[23] 2.246130
[24] 2.246130
[25] 2.246130
[26] 2.225373
[27] 2.197301
[28] 2.062773
[29] 2.041937
[30] 1.924647
[31] 1.921033
[32] 1.871775
[33] 1.829509
[34] 1.813267
> 
```

Fig 9: Rlucian and RKarma rules by lift

Rule number 5 and 6 from Fig 9 interest me especially. They have lift's that are more than double average, and they predict champions that are played in the same lane against Lucian and karma (The bottom lane of the map). They indicate that if red-side Talon or blue-side draven is in the game, it's likely to see blue-side Thresh picked as a response to it. I then visualized the whole set



```
> plot(lkrules[1:20], method="graph")
>
```

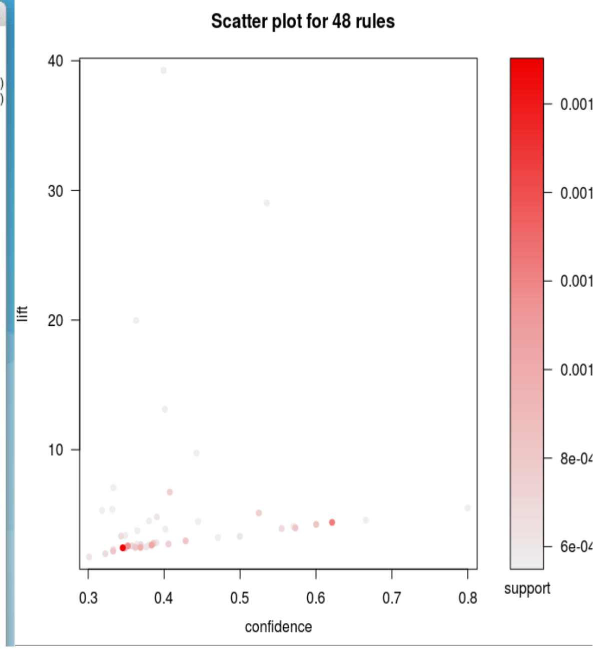


Fig 10: Graphical representation of top 20 rules sorted by lift, Fig 11: confidence-lift plot of full 48 rules.

In Fig 10, the size of the node indicates relative support, the darker color indicates hugger lift. Note the low support but high lift Rakan rules, but high support and low lift Rumble rules. We see these same rules in Fig 11 (there's very faint dots near the 10 lift area).

References:

Hahsler, Michael, and Sudheer Chelluboina. 2017. "Visualizing Association Rules: Introduction to the R-Extension Package arulesViz." Accessed June 4. https://cran.r-project.org/web/packages/arulesViz/vignettes/arulesViz.pdf.

Nayak, Jyothsna R, and Diane J Cook. 2017. "Approximate Association Rule Mining." Accessed May 26. https://www.aaai.org/Papers/FLAIRS/2001/FLAIRS01-050.pdf.