DTU Danmarks Tekniske Universitet

02807 Computational Tools for Big Data

Big Data Study on RIOT GAMES

Teacher and teaching assistant:

David Kofoed Wind,

Finn Årup Nielsen

(Rasmus) Malthe Jørgensen

Ulf Aslak Jensen

Preface

League of Legends(LOL) is an online battle video game developed by RIOT Games, popular in Europe, North America and East Asia. Since its publication, LOL has drawn public attention for its novelty and popularity. Luckily, RIOT Games provide abundant APIs for researchers to study unknown laws in the game. In this project, hence, we will mine the data through APIs and analyze the performance of different summoners or champions, during which process several skills learned in this course like Apache Spark, MongoDB, Map Reduce will be utilized.

Introduction

For a clear structure, the project report is divided into several parts. **Project description** gives a detailed report of the background and what we are going to do with data, telling the specific methods used. **Problem formulation** talks about two main problems we are faced with and key skills to solve them. **Design Specifications** illustrates the structure of the dataset, giving our main idea of the project. **Implementation** is about code but not only the code. This part also combines some notes and description. Some outputs after the execution of codes is shown in **Test**. **Further Improvement** is about what we are going to do with the result and what would be shown next time. And finally we reach our **Conclusion**.

Project description

RIOT gaming has launched a game called League Of Legends(LOL) since 2010. It is a 2D online competitive game that summoner use the characters to take down their opponents. As the popularity increases, the generated data sets become huge by the increasing summoner game behavior.

The RIOT gaming offers an organized API for developers to query the database to registered developers. The API documentation shows a full reference of the data sets. By requesting the specific API developers can fetch related information as they wish to analyze.

In this project, we are mining the data of game statistics information based on given summoner names in the top two leagues. Our group studied how to use the RIOT API to get the related JSON formatted data, for example, the champions' (the characters in the game) win-rate in recent games played by summoners (the game players). The fetched data will be organized by IDs (unique in the game) of each summoner per JSON file, containing their game statistics. We will fetch approximately hundreds of thousands of summoners' game information.

The main goal of our project is to retrieve and analysis RIOT game statistical large data sets. The size of the datasets is approximately 8GB. Our study contains the recursive data retrieved by using RIOT API, Apache Spark job for parallel operation and visualizing the statistical result based on the data.

The dataset is fetched by a home-made library. After downloading the data to local files, we make some initial plots and network analysis to test on part of our data. Then we use spark job to create Resilient Distributed dataset(RDD), and make parallel operation on a cluster to fetch new datasets. Afterwards we save the new datasets which specified into MongoDB for later analysis. Then we will plot the data and show the values to give direct visualization.

Problem formulation

The first problem for our project is retrieving gigabytes level of data set from RIOT database.

Unlike some other projects this is quite a challenge already from the beginning stage. The relatively large datasets among RIOT database is the game table. It mainly contains the game stats of 10 summoners' activity. There are many nested keys with related data which stored as JSON dictionary format. We will collect the summonerID and query the game API by the collection of summonerID. The server will respond with recent 10 games' stats of the summonerID. In each game stats, there are 9 fellow players. We will add the 9 fellow players to the collection of summonerID. In this way we will expand the collection of summonerID. The amount of the game datasets would be increased consequently. These datasets will then be stored in local drive.

The second problem we encountered is processing hundreds of thousands of game-datasets.

We will use spark job RDD API to transform the multiple game datasets into new datasets of championIDs and its stats over certain game. The stats will include for e.g. win/lose, damage dealt by the champion played by certain summoner. Then datasets will be reduced by each championID and the aggregation of its various game stats.

The third problem we are faced with is analyzing the statistical data and plotting them.

As for this problem, we will use the plot library to plot the aggregation of each game statistics over all champions. The plot will show a specific game stat on the y-axis and all champions on the x-axis.

Design overview

The chapter shows our design in a top level. It contains the overview of the design.

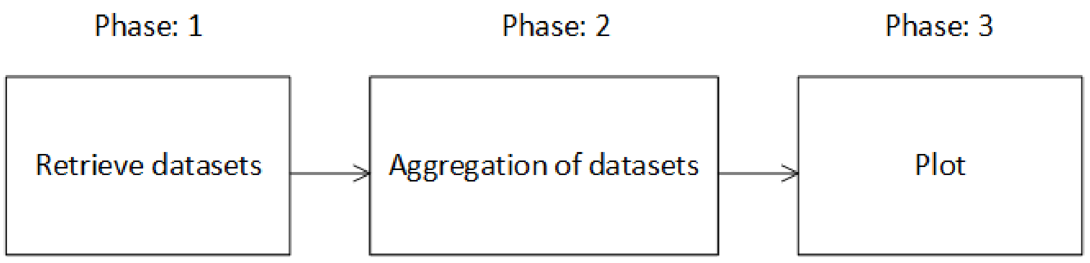


Figure above shows the top level block diagram of the design. It refers the problems listed in the problem formulation.

In the phase 1, we design several functions to request RIOT APIs. These API are summoner, game and champion. We will come back to explain more specifically what we can fetch by doing so.

In the phase 2, we collect game statistics for each champion and aggregate each stat. Then we collect these new datasets and save them into MongoDB.

In the phase 3, we use the aggregated data to do some calculations for e.g. to calculate the win rate of each champions, the frequency of each champion that all summoners has used and the average of the performance of the champions . These calculation will then be plotted as each datasets verse all champions.

Design Specifications

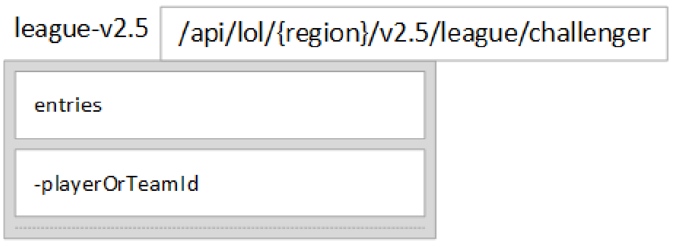
The chapter will show the detailed design specifications for each phase which has been showed in the previous chapter.

Phase 1:

In this phase we make the functions for RIOT API request. As we talked about in the design overview, we will query summoner, game and champion APIs. In this section, we explain the structure of each API and what data we are going to retrieve.

1. League API

This request will be used in a python function for retrieve the offered summonerIDs that RIOT offers in the two top leagues. We will collect these IDs to request the game API that will returns larger datasets of game stats.



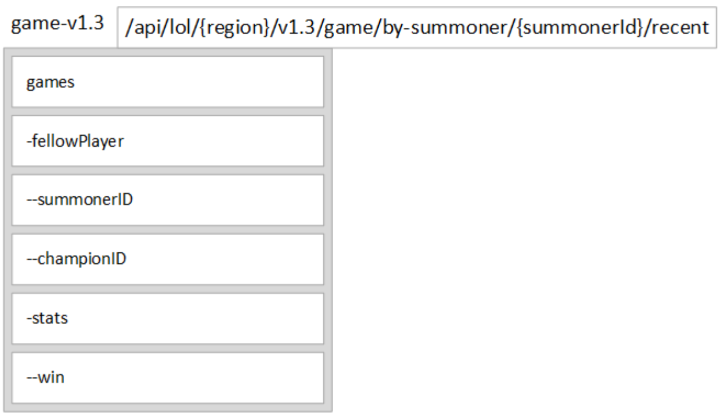
The figure above shows the league API version 2.5. In the figure, 'league-v2.5', /api/lol/{region}/v2.5/league/challenger', 'entries' and '-playerOrTeamId' shows the name of the RIOT API, general content of players in challenger league request, top level key 'entries' and lower level key 'playerOrTeamId'. The entry of key 'playerOrTeamId' contains the summonerID.

```
https://{region}.api.pvp.net/api/lol/{region}//euw/v2.5/league/challenger?api_key={developer API key}
```

The url above is a model of challenger league request. All the content inside curly bracket are the arguments. In the url, 'region' and 'API key' indicates game server region and registered API key.

2. Game API

This request will be used for two purpose. The first one is used for retrieve the offered fellow players which played with a certain summoner. We will collect these IDs to expand the original summonerID collection. The second is to retrieve the game stats of championID and all stats. The game datasets will be expanded in gigabytes level to as more summonerID as we can collect.



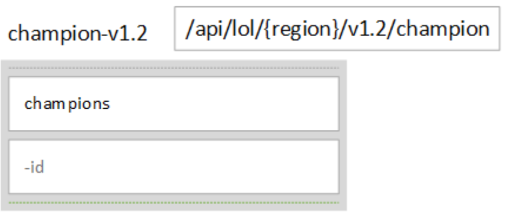
The figure above shows the game API version 1.3. In the figure, 'game-v1.3', '/api/lol/{region}/v1.3/game/by-summoner/{summonerId}/recent', 'games' ,'fellowPlayer', '--summonerID', '-stats', '--win' and '--K/D/A' shows the name of the API, general content of the game data request, the top level key 'games', the lower level key 'fellowPlayer', the 2nd lower level key '--summonerID', the lower level key '-stats', the 2nd lower level key 'win' and the 2nd lower level key 'K/D/A'

```
https://{region}.api.pvp.net/api/lol/{region}/v1.3/game/by-summoner/{summonerID}/recent?api_key={API key}
```

The url above is a model of challenger game stats request. All the content inside curly bracket are the arguments. In the url, 'region', 'summonerID' and 'API key' indicates game server region, specific summonerID and registered API key.

3. Champion API

This request will be used to collect the championIDs. The championIDs are unique. We will be used the championIDs as our analyzed object.



The figure above shows the champion API version 1.2. In the figure, 'champion-v1.2', '/api/lol/{region}/v1.2/champion', 'champions' and '-id' shows the name of the API, general content of the champion data request, the top-level key 'champions' and the lower level key 'Id'.

```
https://{region}.api.pvp.net/api/lol/{region}/v1.2/champion?api_key={API key}
```

The url above is a model of championID request. All the content inside curly bracket are the arguments. In the url, 'region' and 'API key' indicates game server region and registered API key.

Phase 2:

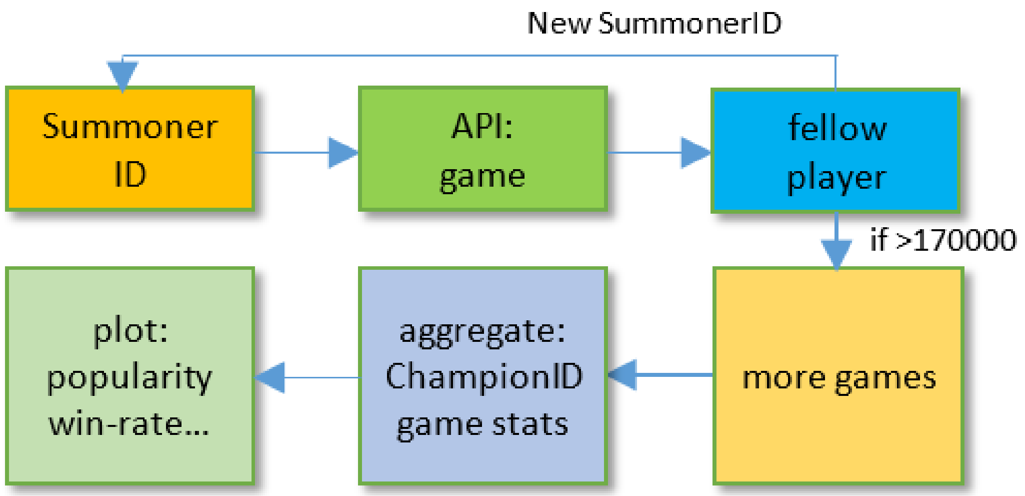
After processing the data we've got using some Hadoop and MapReduce methods, we will store the data into MongoDB, and do some queries using things we've learned. The function of inserting items will be db.insert_one(). The quick sample code is like the following.

```
#Connecting to mongoDB database
client = MongoClient()

db = client['RiotMongoDB']
SummonerID_Collection = db["SummonerID"]
RecentHistory_Collection = db["RecentHistory"]

entryID = SummonerID_Collection.insert_one({str(SummonerID):idURL}).inserted_id
entryID = RecentHistory_Collection.insert_one({str(SummonerID):rdata}).inserted_id

client.close();
```

Phase 3:

As for the result visualization, we will try to format our results in a comprehensible and vivid way. For example, using tables or graph database to present the query results.

Besides, we will consider many game players and the champions they choose in this project. So we will use multiple kinds of plots to present different kinds of results, like histograms, scatter plots, graph plots, etc. You will find more about our plots in the later part!



Implementation

First, we implement all the libraries we need.

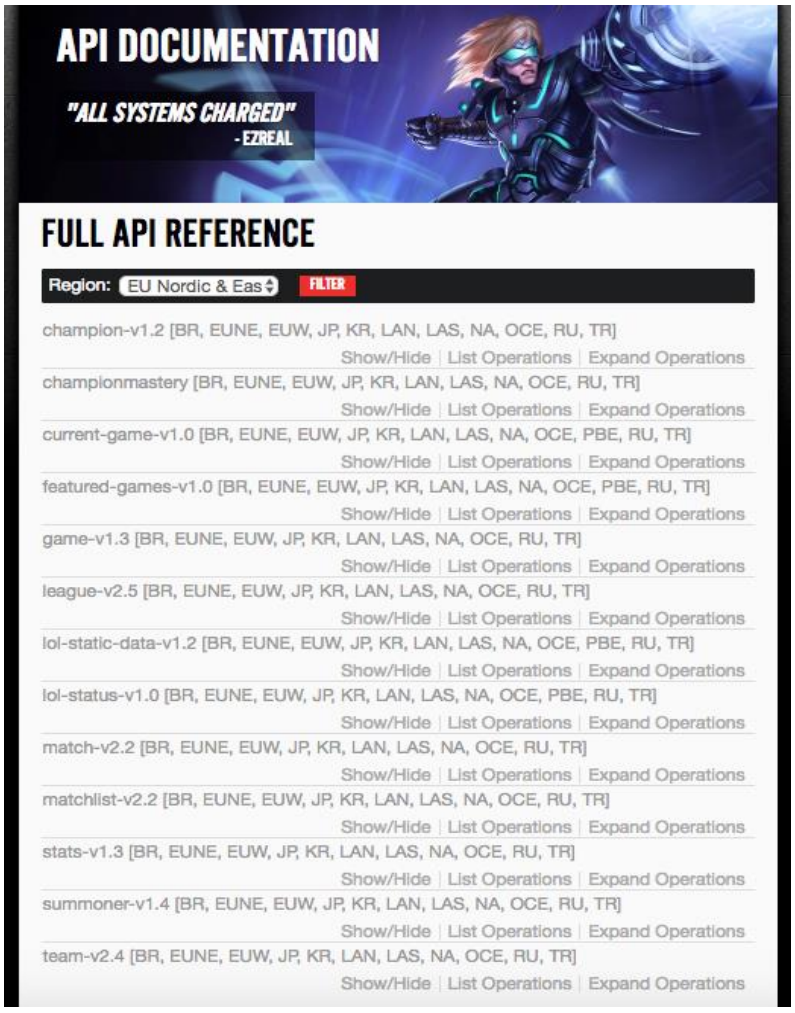
```
# Implement all the libraries we need
%matplotlib inline
import re
import io
import os
import urllib2
import networkx as nx
import community
import json
import glob
import numpy as np
import operator
import matplotlib.pyplot as plt
from matplotlib import pyplot
import math
import time
import networkx as nx
import os.path
from pymongo import MongoClient
from itertools import groupby
from __future__ import division
from pprint import pprint
from os import listdir
import community
```

Then, we defined several functions, in order to get the different APIs and store the json data to our local file.

```
# Get JSON reply using the URL
def getJSONReply(URL):
  response = urllib2.urlopen(URL)
  html = response.read()
  data = json.loads(html)
  return data

# Using the game-v1.3 API to get each player's recent history
def getRecentHistory(SummonerID):
    rURL= "https://" +Region.lower()+ ".api.pvp.net/api/lol/" + \
    Region.lower()+ "/v1.3/game/by-summoner/" + `SummonerID`+ "/recent?api_key=" + Key;
    r_data=getJSONReply(rURL);
    r_data['_id']=r_data['summonerId'];
    r_data.pop('summonerId');
    # Write the data to a local json file, naming it after this Summoner's ID
    with io.open('RecentHistory/%s.json' % str(SummonerID), 'w', encoding='utf-8') as f:
      f.write(unicode(json.dumps(r_data, ensure_ascii=False)))
    return r_data;

# Using the championmastery API to get each player's mastery of the champions
def getChampion(SummonerID):
    rURL= "https://" +Region.lower()+ ".api.pvp.net/championmastery/location/"\
    + Region+ "1/player/" + `SummonerID`+ "/champions?api_key=" + Key
    r_data=getJSONReply(rURL)
    # Write the data to a local json file, naming it after this Summoner's ID
    with io.open('Champion/%s.json' % str(SummonerID), 'w', encoding='utf-8') as f:
      f.write(unicode(json.dumps(r_data, ensure_ascii=False)))
    return r_data

# Using the championmastery API to get each player's mastery of the champions
def getChampion(SummonerID):
    rURL= "https://" +Region.lower()+ ".api.pvp.net/championmastery/location/"\
    + Region+ "1/player/" + `SummonerID`+ "/champions?api_key=" + Key
    r_data=getJSONReply(rURL)
    # Write the data to a local json file, naming it after this Summoner's ID
    with io.open('Champion/%s.json' % str(SummonerID), 'w', encoding='utf-8') as f:
      f.write(unicode(json.dumps(r_data, ensure_ascii=False)))
    return r_data

# Using the summoner-v1.4 API to get each player's identity information by their names
def ReformatJSON(SummonerName,Region,Key):
  idURL = "https://" +Region.lower()+ ".api.pvp.net/api/lol/" +Region.lower()+  "/v1.4/summoner/by-name/" +
SummonerName+ "?api_key=" + Key
  id_data = getJSONReply(idURL)
  idRes=id_data[SummonerName.lower()]
  idRes['_id'] = idRes['id']
  idRes.pop('id')
  return idRes,id_data
```

```
# Using the summoner-v1.4 API to get each player's identity information by their IDs
def ReformatJSONbyid(SummonerID,Region,Key):
  idURL = "https://" +Region.lower()+ ".api.pvp.net/api/lol/" +Region.lower()+ "/v1.4/summoner/"   +SummonerID+
"?api_key=" +Key
  id_data = getJSONReply(idURL)
  idRes=id_data[SummonerID]
  idRes['_id'] = idRes['id']
  idRes.pop('id')
  # Write the data to a local json file, naming it after this Summoner's ID
  with io.open('Summoner/%s.json' % str(SummonerID), 'w', encoding='utf-8') as f:
      f.write(unicode(json.dumps(idRes, ensure_ascii=False)))
  return idRes,id_data
```

Here is a screenshot of the RIOT API full documentation.



FULL API REFERENCE

Region: EU Nordic & Eas FILTER

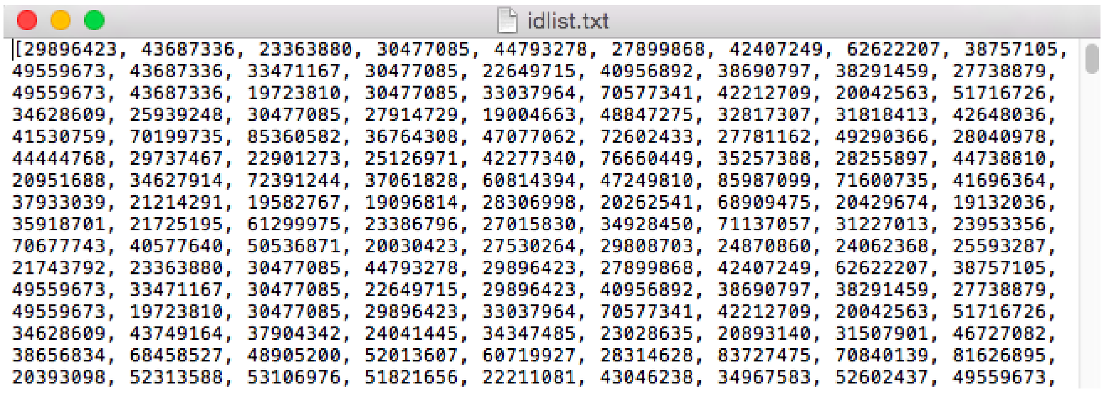
champion-v1.2 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, RU, TR]
Show/Hide | List Operations | Expand Operations
championmastery [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, RU, TR]
Show/Hide | List Operations | Expand Operations
current-game-v1.0 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, PBE, RU, TR]
Show/Hide | List Operations | Expand Operations
featured-games-v1.0 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, PBE, RU, TR]
Show/Hide | List Operations | Expand Operations
game-v1.3 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, RU, TR]
Show/Hide | List Operations | Expand Operations
league-v2.5 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, RU, TR]
Show/Hide | List Operations | Expand Operations
lol-static-data-v1.2 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, PBE, RU, TR]
Show/Hide | List Operations | Expand Operations
lol-status-v1.0 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, PBE, RU, TR]
Show/Hide | List Operations | Expand Operations
match-v2.2 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, RU, TR]
Show/Hide | List Operations | Expand Operations
matchlist-v2.2 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, RU, TR]
Show/Hide | List Operations | Expand Operations
stats-v1.3 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, RU, TR]
Show/Hide | List Operations | Expand Operations
summoner-v1.4 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, RU, TR]
Show/Hide | List Operations | Expand Operations
team-v2.4 [BR, EUNE, EUW, JP, KR, LAN, LAS, NA, OCE, RU, TR]
Show/Hide | List Operations | Expand Operations

Because RIOT doesn't have a list of all the players, we used a Summoner Name randomly chosen from the RIOT top league, retrieving his user information in order to find more summoner ids.

```
# Pre-input using SummonerName provided by RIOT top league
SummonerName="Miladena"
Region="EUW"
Key="RGAPI-20253dda-c325-4a7e-947a-d283af4f8641"

# Retrieving the SummonerID;
idURL,id_data=ReformatJSON(SummonerName,Region,Key)
SummonerID=id_data[SummonerName.lower()]["_id"]
rdata=getRecentHistory(SummonerID)
matchlist=getMatchlist(SummonerID)
```

By finding this summoner's recent games, we get the fellow players' Summoner ID in each game, and iteratively get more and more Summoner IDs.

```
# Generate an id list to store a huge amount of summoner ids
idlist = [SummonerID]

# Iterate calling the API,
# get each summoner's 10 recent games,
# find their fellow players in each game
# save their summoner id to a list, 'idlist'
for k in range(19510):
    rdata=getRecentHistory(idlist[k])
    for i in range(len(rdata["games"])):
     # In some kinds of games, player doesn't have fellow players
     if "fellowPlayers" not in rdata["games"][i].keys():
        continue
     for j in range(len(rdata["games"][i]["fellowPlayers"])):
        idlist.append(rdata["games"][i]["fellowPlayers"][j]["summonerId"])
# Get a list of unique summoner IDs
uniidlist = list(set(idlist))
```



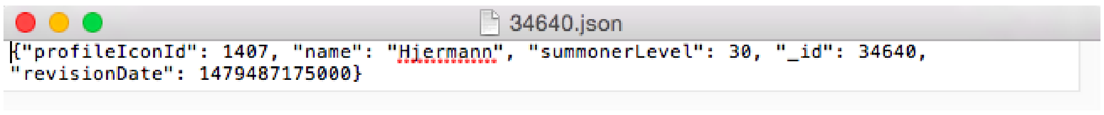
idlist.txt

```
[29896423, 43687336, 23363880, 30477085, 44793278, 27899868, 42407249, 62622207, 38757105,
49559673, 43687336, 33471167, 30477085, 22649715, 40956892, 38690797, 38291459, 27738879,
49559673, 43687336, 19723810, 30477085, 33037964, 70577341, 42212709, 20042563, 51716726,
34628609, 25939248, 30477085, 27914729, 19004663, 48847275, 32817307, 31818413, 42648036,
41530759, 70199735, 85360582, 36764308, 47077062, 72602433, 27781162, 49290366, 28040978,
44444768, 29737467, 22901273, 25126971, 42277340, 76660449, 35257388, 28255897, 44738810,
20951688, 34627914, 72391244, 37061828, 60814394, 47249810, 85987099, 71600735, 41696364,
37933039, 21214291, 19582767, 19096814, 28306998, 20262541, 68909475, 20429674, 19132036,
35918701, 21725195, 61299975, 23386796, 27015830, 34928450, 71137057, 31227013, 23953356,
70677743, 40577640, 50536871, 20030423, 27530264, 29808703, 24870860, 24062368, 25593287,
21743792, 23363880, 30477085, 44793278, 29896423, 27899868, 42407249, 62622207, 38757105,
49559673, 33471167, 30477085, 22649715, 29896423, 40956892, 38690797, 38291459, 27738879,
49559673, 19723810, 30477085, 29896423, 33037964, 70577341, 42212709, 20042563, 51716726,
34628609, 43749164, 37904342, 24041445, 34347485, 23028635, 20893140, 31507901, 46727082,
38656834, 68458527, 48905200, 52013607, 60719927, 28314628, 83727475, 70840139, 81626895,
20393098, 52313588, 53106976, 51821656, 22211081, 43046238, 34967583, 52602437, 49559673,
```

a Quick View of the idlist File

Then we stored this idlist to a local text file. By querying to elements (SummonerIds) in the list, we can get more players' recent 10 games one by one.

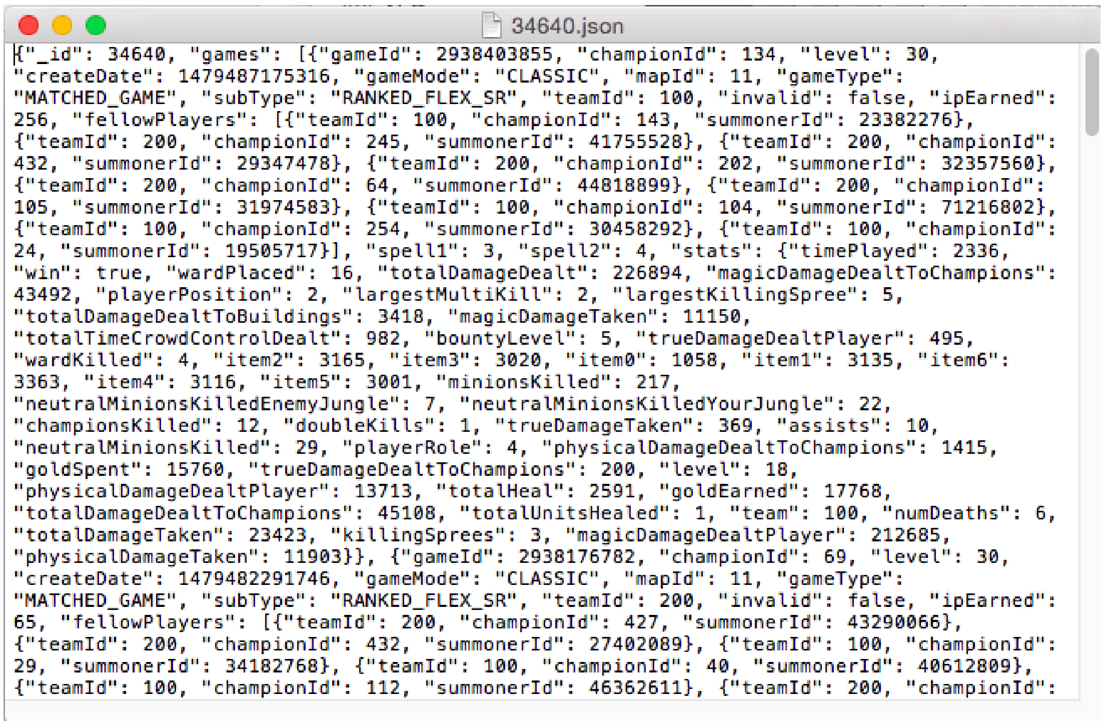
```
# With summoner id list, we get their 10 recent games one by one
for gamer in uniidlist:
   idURL,id_data=ReformatJSONbyid(str(gamer),Region,Key)
   # Because of the API call limit, we have to use time.sleep()
   time.sleep(0.8)
   rdata=getRecentHistory(gamer)
   time.sleep(0.8)
```



34640.json

{"profileIconId": 1407, "name": "Hjermann", "summonerLevel": 30, "_id": 34640,
"revisionDate": 1479487175000}

a Quick View of the Summoner Info File



34640.json

{"_id": 34640, "games": [{"gameId": 2938403855, "championId": 134, "level": 30,
"createDate": 1479487175316, "gameMode": "CLASSIC", "mapId": 11, "gameType":
"MATCHED_GAME", "subType": "RANKED_FLEX_SR", "teamId": 100, "invalid": false, "ipEarned":
256, "fellowPlayers": [{"teamId": 100, "championId": 143, "summonerId": 23382276},
{"teamId": 200, "championId": 245, "summonerId": 41755528}, {"teamId": 200, "championId":
432, "summonerId": 29347478}, {"teamId": 200, "championId": 202, "summonerId": 32357560},
{"teamId": 200, "championId": 64, "summonerId": 44818899}, {"teamId": 200, "championId":
105, "summonerId": 31974583}, {"teamId": 100, "championId": 104, "summonerId": 71216802},
{"teamId": 100, "championId": 254, "summonerId": 30458292}, {"teamId": 100, "championId":
24, "summonerId": 19505717}], "spell1": 3, "spell2": 4, "stats": {"timePlayed": 2336,
"win": true, "wardPlaced": 16, "totalDamageDealt": 226894, "magicDamageDealtToChampions":
43492, "playerPosition": 2, "largestMultiKill": 2, "largestKillingSpree": 5,
"totalDamageDealtToBuildings": 3418, "magicDamageTaken": 11150,
"totalTimeCrowdControlDealt": 982, "bountyLevel": 5, "trueDamageDealtPlayer": 495,
"wardKilled": 4, "item2": 3165, "item3": 3020, "item0": 1058, "item1": 3135, "item6":
3363, "item4": 3116, "item5": 3001, "minionsKilled": 217,
"neutralMinionsKilledEnemyJungle": 7, "neutralMinionsKilledYourJungle": 22,
"championsKilled": 12, "doubleKills": 1, "trueDamageTaken": 369, "assists": 10,
"neutralMinionsKilled": 29, "playerRole": 4, "physicalDamageDealtToChampions": 1415,
"goldSpent": 15760, "trueDamageDealtToChampions": 200, "level": 18,
"physicalDamageDealtPlayer": 13713, "totalHeal": 2591, "goldEarned": 17768,
"totalDamageDealtToChampions": 45108, "totalUnitsHealed": 1, "team": 100, "numDeaths": 6,
"totalDamageTaken": 23423, "killingSprees": 3, "magicDamageDealtPlayer": 212685,
"physicalDamageTaken": 11903}}, {"gameId": 2938176782, "championId": 69, "level": 30,
"createDate": 1479482291746, "gameMode": "CLASSIC", "mapId": 11, "gameType":
"MATCHED_GAME", "subType": "RANKED_FLEX_SR", "teamId": 200, "invalid": false, "ipEarned":
65, "fellowPlayers": [{"teamId": 200, "championId": 427, "summonerId": 43290066},
{"teamId": 200, "championId": 432, "summonerId": 27402089}, {"teamId": 100, "championId":
29, "summonerId": 34182768}, {"teamId": 100, "championId": 40, "summonerId": 40612809},
{"teamId": 100, "championId": 112, "summonerId": 46362611}, {"teamId": 200, "championId":

a Quick View of the Summoner's Recent Game History File

As for the champion mastery of each player, we decide to download it later. Now with 7,455 summoner data downloaded, 74,550 games ready to be analysed, we decide to conduct some tests on these summoner files and see if there's anything interesting.

Later, we will download more summoner data and store them in MongoDB, maybe using MapReduce to improve the working efficiency.

Test

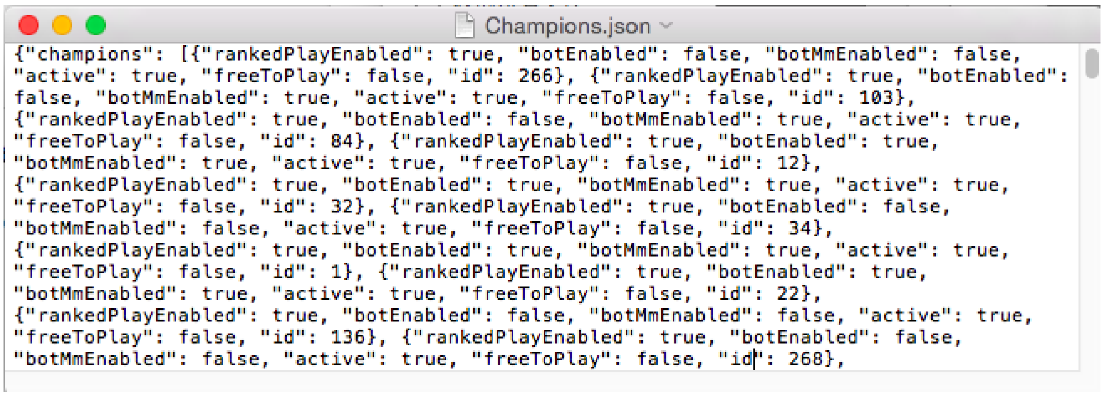
We mainly conduct our tests on 2 parts.

- Analysis about Champions, include their popularities, win-rate
- Analysis about the Summoners' game network

1. Champions

We first queried champions API to get a full dictionary of champions, and store the result to a local json file.

```
URL="https://euw.api.pvp.net/api/lol/euw/v1.2/champion? \
api_key=RGAPI-20253dda-c325-4a7e-947a-d283af4f8641"
data = getJSONReply(URL);

with open('Champions.json', 'w') as f:
  f.write(unicode(json.dumps(data, ensure_ascii=False)))


# Get a list of champion ids
champions = []
for i in range(len(data["champions"])):
  champions.append(data["champions"][i]["id"])
```



{"champions": [{"rankedPlayEnabled": true, "botEnabled": false, "botMmEnabled": false, "active": true, "freeToPlay": false, "id": 266}, {"rankedPlayEnabled": true, "botEnabled": false, "botMmEnabled": true, "active": true, "freeToPlay": false, "id": 103}, {"rankedPlayEnabled": true, "botEnabled": false, "botMmEnabled": true, "active": true, "freeToPlay": false, "id": 84}, {"rankedPlayEnabled": true, "botEnabled": true, "botMmEnabled": true, "active": true, "freeToPlay": false, "id": 12}, {"rankedPlayEnabled": true, "botEnabled": true, "botMmEnabled": true, "active": true, "freeToPlay": false, "id": 32}, {"rankedPlayEnabled": true, "botEnabled": false, "botMmEnabled": false, "active": true, "freeToPlay": false, "id": 34}, {"rankedPlayEnabled": true, "botEnabled": true, "botMmEnabled": true, "active": true, "freeToPlay": false, "id": 1}, {"rankedPlayEnabled": true, "botEnabled": true, "botMmEnabled": true, "active": true, "freeToPlay": false, "id": 22}, {"rankedPlayEnabled": true, "botEnabled": false, "botMmEnabled": false, "active": true, "freeToPlay": false, "id": 136}, {"rankedPlayEnabled": true, "botEnabled": false, "botMmEnabled": false, "active": true, "freeToPlay": false, "id": 268},

a Quick View of the Champions File

Then we used listdir() function to get all the filenames in a folder. This enables us to use the Summoner ID of the summoners that we've already downloaded.

```
filename = listdir("RecentHistory")[1:]
```

Then we get a list of champions that are used in the first 2000 summoners' recent games.

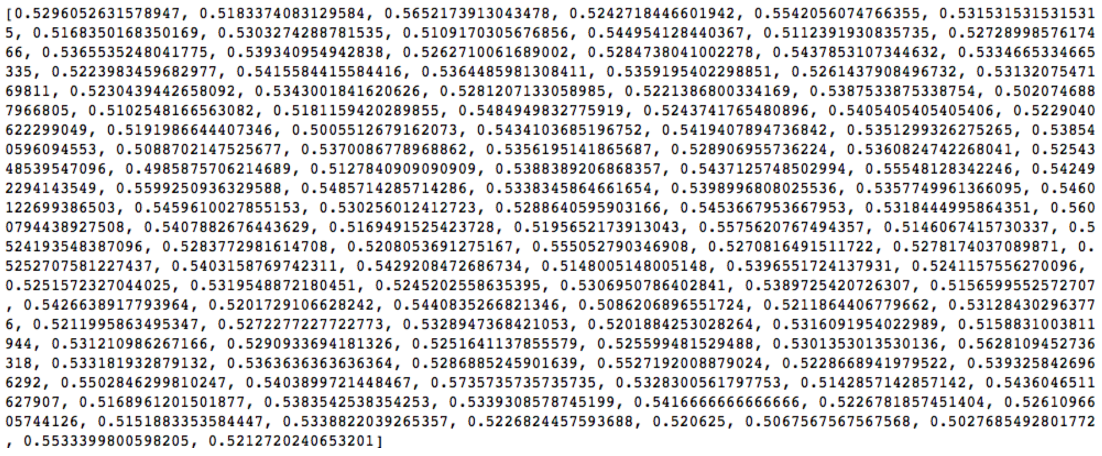
```
# Get all the champions that are used in the first 2000 players' recent games
championid = []
for k in range(2000):
    with open('RecentHistory/%s' % filename[k]) as f:
        data = json.load(f)
    for i in range(len(data["games"])):
        championid.append(data["games"][i]["championId"])
        if "fellowPlayers" not in data["games"][i].keys():
            continue
        for j in range(len(data["games"][i]["fellowPlayers"])):
            championid.append(data["games"][i]["fellowPlayers"][j]["championId"])
```

We took a look at the length of this list, comparing that with the total champions list. Both the lists have 133 elements, and this indicates that in these 2000 summoners' recent games, all of the 133 champions have been used.

Then we calculate the total used times and total win times of each champion, therefore getting their win-rate list.

One thing to mention, as the champions' ID are not sequential, we index the champions' win-rate, win times and total used times by finding their index in the Total Champion List obtained from the previous API. In this way, we can find each champion is in the 4 lists (championid, win_rate, win, total) with the same index.

```
# Initialize 3 lists with value 0, to store the win-rate,
# win times and total used times of each champion
win_rate = [0]*(len(champions))
win = [0]*(len(champions))
total = [0]*(len(champions))

for k in range(2000):
    # Load each summoner's data by reading the local file
    with open('RecentHistory/%s' % filename[k]) as f:
        data = json.load(f)

    for i in range(len(data["games"])):
        # Get the champion id of the summoner in each game
        champid = data["games"][i]["championId"]
        # Find the index of the champion in the champions list,
        # add 1 to the total use frequency of this champion
        total[champions.index(champid)] += 1

        # Find the index of the champion in the champions list,
        # if the summoner wins this game,
        # add 1 to the total win frequency of this champion
        if data["games"][i]["stats"]["win"] is True:
            win[champions.index(champid)] += 1
        # In some games the summoner doesn't have fellow players
        if "fellowPlayers" not in data["games"][i].keys():
            continue

        # Get the champion id of the fellow players in each game
        for j in range(len(data["games"][i]["fellowPlayers"])):
          champid = data["games"][i]["fellowPlayers"][j]["championId"]
          total[champions.index(champid)] += 1
          if data["games"][i]["stats"]["win"] is True:
            win[champions.index(champid)] += 1

# Calculate the win-rate of each champion
for i in range(len(champions)):
    if total[i] == 0:
        win_rate[i] = 0
    else:
        win_rate[i] = win[i]/total[i]

print win_rate
```

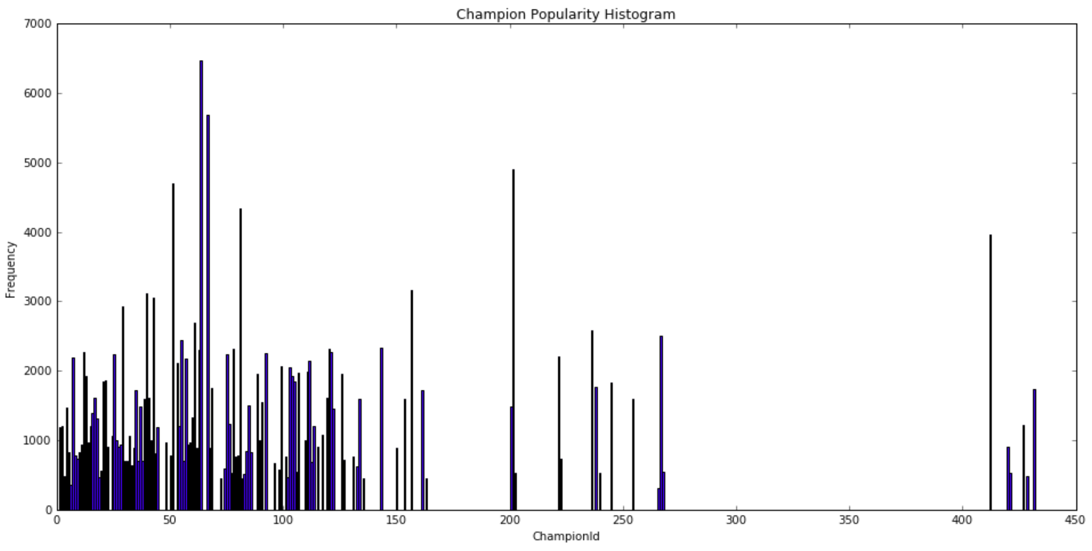


```
[0.5296052631578947, 0.5183374083129584, 0.5652173913043478, 0.5242718446601942, 0.5542056074766355, 0.531531531531531
5, 0.5168350168350169, 0.5303274288781535, 0.5109170305676856, 0.544954128440367, 0.5112391930835735, 0.52728998576174
66, 0.5365535248041775, 0.539340954942838, 0.5262710061689002, 0.5284738041002278, 0.5437853107344632, 0.5334665334665
335, 0.5223983459682977, 0.5415584415584416, 0.5364485981308411, 0.5359195402298851, 0.5261437908496732, 0.53132075471
69811, 0.5230439442658092, 0.5343001841620626, 0.5281207133058985, 0.5221386800334169, 0.5387533875338754, 0.502074688
7966805, 0.5102548166563082, 0.5181159420289855, 0.5484949832775919, 0.5243741765480896, 0.5405405405405406, 0.5229040
622299049, 0.5191986644407346, 0.5005512679162073, 0.5434103685196752, 0.5419407894736842, 0.5351299326275265, 0.53854
0596094553, 0.508870214752567 7, 0.5370086778968862, 0.5356195141865687, 0.528906955736224, 0.5360824742268041, 0.52543
48539547096, 0.4985875706214689, 0.5127840909090909, 0.5388389206868357, 0.5437125748502994, 0.55548128342246, 0.54249
2294143549, 0.5599250936329588, 0.5485714285714286, 0.5338345864661654, 0.5398996808025536, 0.5357749961366095, 0.5460
122699386503, 0.5459610027855153, 0.530256012412723, 0.5288640595903166, 0.5453667953667953, 0.5318444995864351, 0.560
0794438927508, 0.5407882676443629, 0.5169491525423728, 0.5195652173913043, 0.5575620767494357, 0.5146067415730337, 0.5
524193548387096, 0.5283772981614708, 0.5208053691275167, 0.555052790346908, 0.5270816491511722, 0.5278174037089871, 0.
5252707581227437, 0.5403158769742311, 0.5429208472686734, 0.5148005148005148, 0.5396551724137931, 0.5241157556270096,
0.5251572327044025, 0.5319548872180451, 0.5245202558635395, 0.5306950786402841, 0.5389725420726307, 0.5156599552572707
, 0.5426638917793964, 0.5201729106628242, 0.5440835266821346, 0.5086206896551724, 0.5211864406779662, 0.53128430296377
6, 0.5211995863495347, 0.5272277227722773, 0.5328947368421053, 0.5201884253028264, 0.5316091954022989, 0.5158831003811
944, 0.531210986267166, 0.5290933694181326, 0.5251641137855579, 0.525599481529488, 0.5301353013530136, 0.5628109452736
318, 0.533181932879132, 0.5363636363636364, 0.5286885245901639, 0.5527192008879024, 0.522866894197952 2, 0.539325842696
6292, 0.5502846299810247, 0.5403899721448467, 0.5735735735735735, 0.5328300561797753, 0.5142857142857142, 0.5436046511
627907, 0.5168961201501877, 0.5383542538354253, 0.5339308578745199, 0.5416666666666666, 0.5226781857451404, 0.52610966
05744126, 0.5151883353584447, 0.5338822039265357, 0.5226824457593688, 0.520625, 0.5067567567567568, 0.5027685492801772
, 0.5533399800598205, 0.5212720240653201]
```

a Quick View of the Champions Win-rate Output List

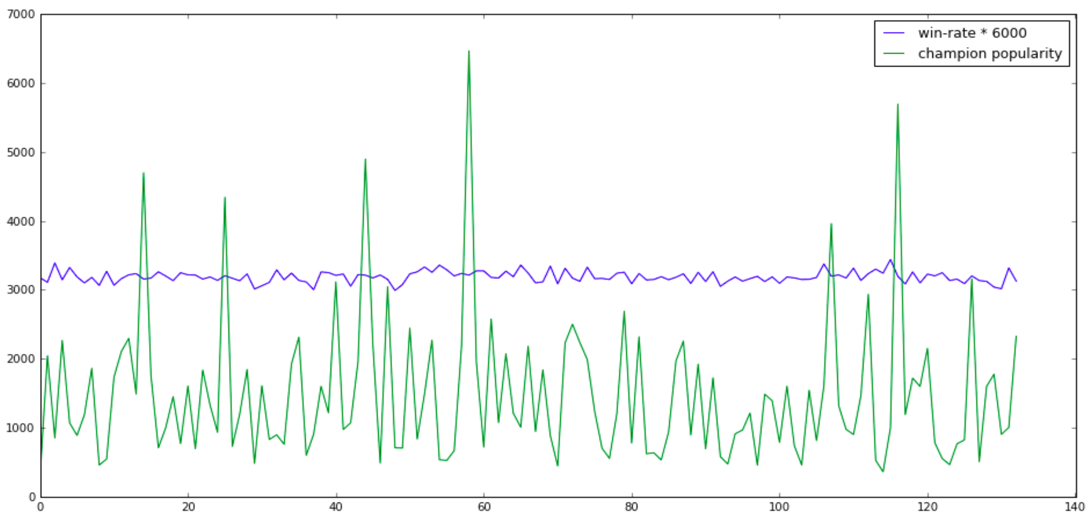
From this list, we can see that all the champions' win-rates here are above 0.5. This is probably because our initial Summoner is chosen from the top players provided by RIOT GAMES, and the advanced player tend to play with other players who have good command of this game.

Of course, if we look at more players, the win-rate of each champion will probably be closer to 0.5.



a Histogram of the Popularity of the Champions

We plotted a histogram of the popularity of the champions. The x axis is the Champion id (so there are some zeros), and the y axis is the frequency that a champion is used in these games. We can see that several champions are really popular with the players, especially the ones with champion id around 70.

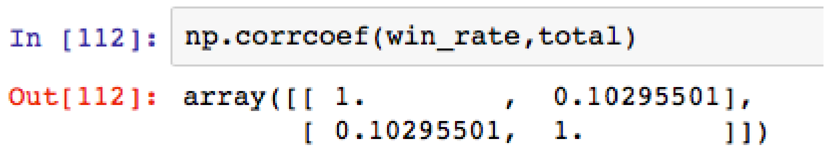


a Plot of the Win-rate and the Popularity of Each Champion

We also plotted a plot with two lines to see if there's any correlation with the win-rate and the popularity of the champions. The blue line shows the win-rate * 6000 (to make it clearer to compare with the popularity), and the green line shows the popularity of each champion.

The x axis is the index of each champion in the Total Champion List.

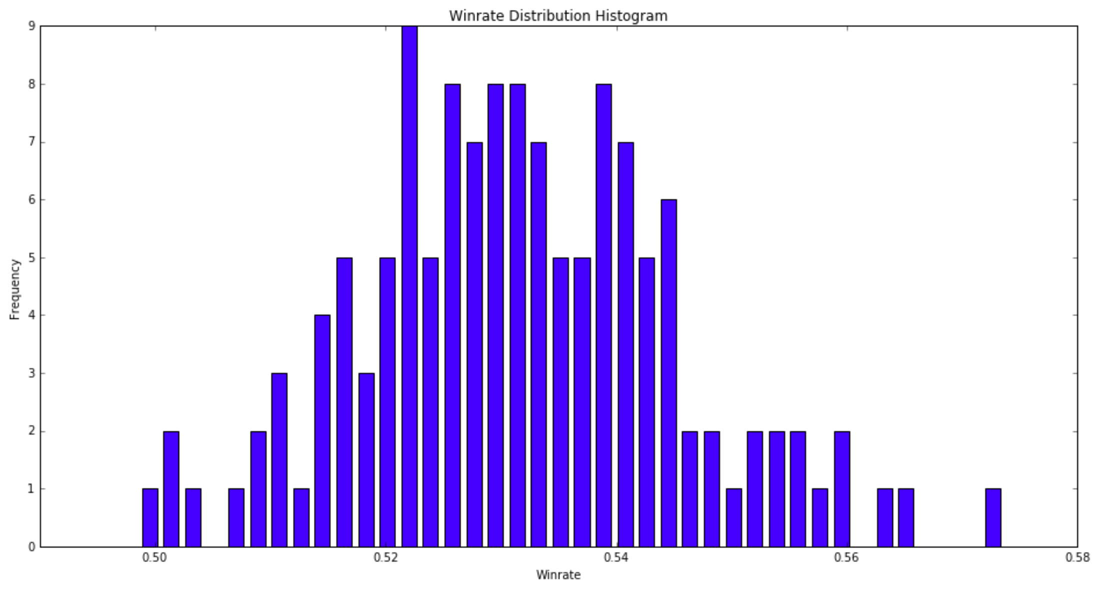
From the plot we can see that the win-rate seems not have much correlation with the champions' popularity. To prove this in a more mathematical way, we calculated the correlation coefficient.



```
In [112]:  np.corrcoef(win_rate,total)

Out[112]:  array([[ 1.        ,  0.10295501],
                  [ 0.10295501,  1.        ]])
```

Result of the Correlation Coefficient

The correlation coefficient is about 0.1, rather low. This indicates that the summoners don't usually choose champions with higher win-rates. We should take more consideration into the behavior of the champions and mastery of the champions of each summoner. This will be interesting to analyze.



The Win-rate Distribution

Also, the win-rate distribution histogram seems to be Gaussian, which is reasonable.

2. Summoners

Next, we want to take a look at the game network of the summoners. The central idea is that we can get all the players in one game, and we can link them together. Therefore, game by game, we can get a huge network with summoners who have played together be linked together.

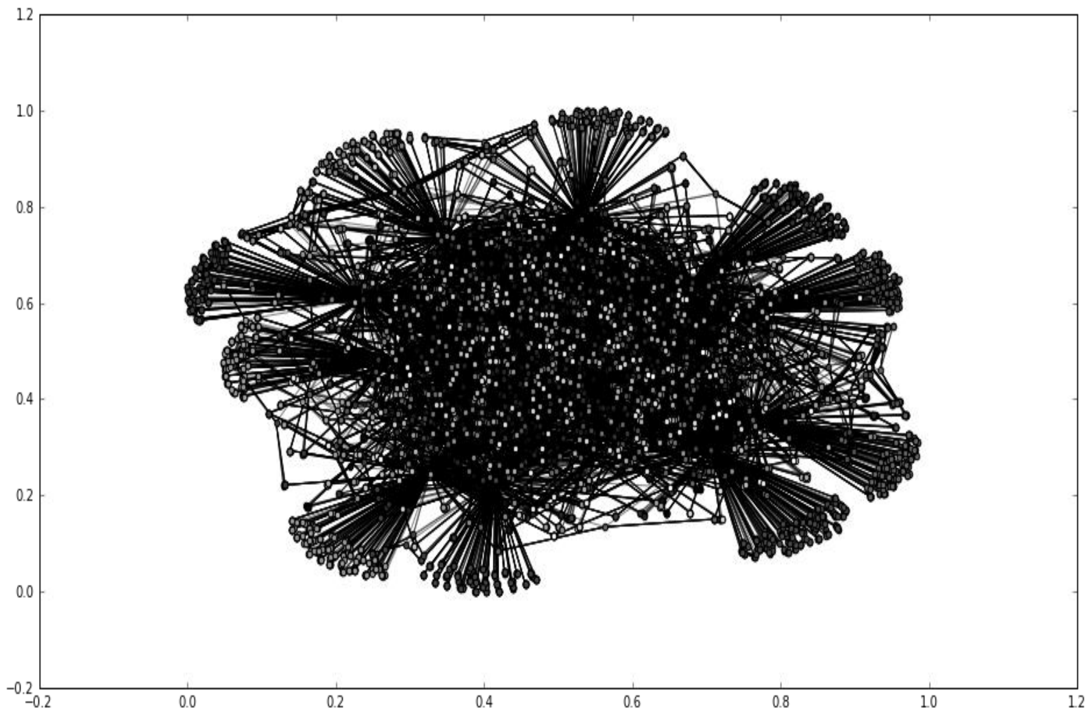
We want to compare this with a real social network, like Facebook. Because we're curious about how players choose their fellow players, and we wonder if that is similar to making friends in the real world.

```
# Creat a list of lists "players",
# store all the players' id in one game into one sublist,
# and then store the sublists to the list
players = []
for i in range(2000):
  with open('RecentHistory/%s' % filename[i]) as f:
    data = json.load(f)

  for i in range(len(data["games"])):
    if "fellowPlayers" not in data["games"][i].keys():
      continue
    # Filename is like "summonerid.json", so we use [:-5] to extract '.json'
    temp = [int(filename[i][:-5])]
    # Append the fellow players to the sublist
    for j in range(len(data["games"][i]["fellowPlayers"])):
      temp.append(data["games"][i]["fellowPlayers"][j]["summonerId"])
    # Append the sublist to the total list
    players.append(temp)
```

Then we created a graph, linking all the players in each game in pairs.

```
# Create a graph using networkx library
G=nx.Graph()
for j in range(1000):
  for i in range(len(players[j])-1):
    for k in range(i+1,len(players[j])):
      G.add_edge(players[j][i],players[j][k])
```

We plotted the graph. The lighter color one node is, the more connected it is.



To create the graph, we used for-loop, which is not very efficient. We will consider using MapReduce to do this job.

We will explore more things later, like the centrality and the degree distribution of the nodes. And we will find out if this network is like the real-world social network using network theories.

Further Improvement

Up till now we have already implemented some expected functions. For example, we have downloaded some information about players and characters in the game from database, which proves that we have understood the structures and functions of APIs. And also, we have found the way to process the data, figuring out some inner laws of the game, such as the relation between winning rate and popularity of the character. Besides, we practiced how to visualize the data by python library *matplotlib*. With all these tools, we have already built the framework of this project.

However, a lot of work still needs to be done for the accomplishment of the whole project. As is shown in the previous part, the data we have processed is only a small fraction of the dataset. Since the source of data is the developers' APIs which have a limit on calls, some measures should be taken to boost the speed of fetching data. And how to restore the data downloaded into a MongoDB is also a problem that should be taken into consideration.

Data processing should also be accelerated. The current implementation, have nothing to do with optimization or acceleration, might be OK when processing small quantity of data. When it comes to the dataset in gigabytes level, however, things will be totally different. For dealing with abundant information, MapReduce will be used to improve the efficiency.

Maybe not too much about plotting is involved in this course. In spite, we still think data visualization is a vital part of the project. A main reason is that in most exercises the output is only several digits while the result of this project would be many numbers and it is the relation of them matters. As a result, in the coming two weeks we will try to find proper ways to show the outputs. Tables and charts are good choice, and more advanced tools might be used, too.

Appendix

Game related words Definitions

summoner - user

summonerID - user ID given by RIOT, it is different with login ID, the user ID used as parameters to query various RIOT API

game - a game consists 10 summoners

champion - character created by RIOT, summoner select a character to play in the game

championID - champion identification is given by RIOT, to distinguish individual champion they have.

damage - damage dealt by the champion which played by certain summoner in the game