武汉大学计算机学院

课程论文

Game Data Analysis of League of Legend

课程名称　　商务智能

专业年级班级　　2016 级 卓越二班

姓名　　刘瑞康

学号　　2016302580242

学期　　2018-2019　　学年　　第一　　学期

成绩＿＿＿＿＿＿＿＿任课教师签名＿＿＿＿＿＿

Content

1. Background

1.1 League of Legends

League of Legends (abbreviated LOL) is a multiplayer online battle arena video game developed and published by Riot Games.

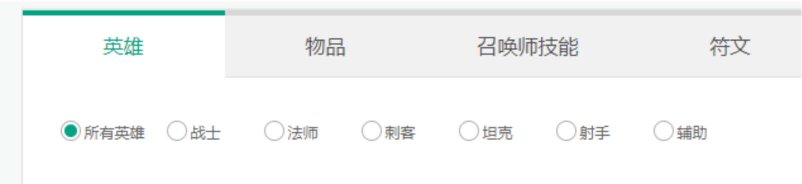
In League of Legends, players assume the role of an unseen "summoner" that controls a "champion" with unique abilities and battle against a team of other players. The goal is usually to destroy the opposing team's "nexus", a structure that lies at the heart of a base protected by defensive structures. Each League of Legends match is discrete, with all champions starting off fairly weak but increasing in strength by accumulating items and experience over the course of the game.

LOL was generally well received upon its release in 2009, and has since grown in popularity, with an active and expansive fan base. In September 2016, the company estimated that there are over 100 million active players each month. The 2017 World Championship had 60 million unique viewers and a total prize pool of over 4 million USD. The 2018 Mid-Season Invitational had an overall peak concurrent viewership of 19.8 million, while the finals had an average concurrent viewership of 11 million.

1.2 Data analysis on games

Data analysis are widely used in game data mining to collect information for game designers and game players. It can be used for game balance, champion positioning and champion recommendation and so on. In fact, the champion team iG takes advantage of the predictive analysis in its ban/pick phase.

2. Requirement Analysis



● 所有英雄 ○ 战士 ○ 法师 ○ 刺客 ○ 坦克 ○ 射手 ○ 辅助

In LOL, there are six official positions for champions: Fighter, Marksman, Mage, Tank, Assassin, Support. However, some of the positions are based on their characteristics like their attack mode, and some are just based on the designer's original intention in designing phase. For example, all the champions with attack mode shooting are regarded as Marksman, and there are many champions which are designed as Tank but sometimes play a role as Fighter or even Assassin. So I’m always curious about their typical roles in the real game instead of the official positions – which of them are more similar and which of them are quite different from the original settings and so on.

3. Data Set

All intermediate data and final data that is used in this project are stored in the directory **data** with json files and a database file.

4. Running Environment and Dependencies and Technolgy Used

The solution projecet is run on JetBrains **PyCharm** Community Edition, based on **python 36** with urllib, sqlite3, numpy, scipy, matplotlib and sklearn, using **hierarchical clustering analysis** and **principal component analysis**

5. Complete Solution & Whole Process

5.1 Data source for analysis

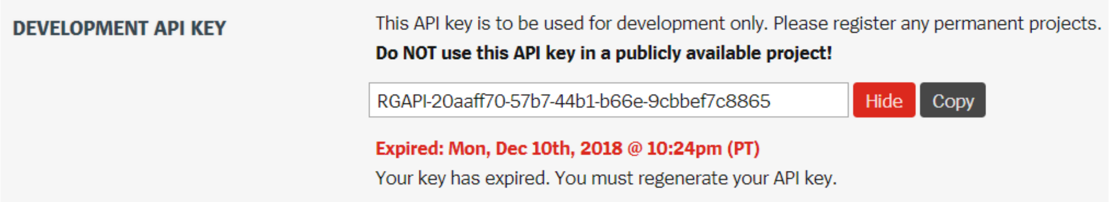
For analysis, an enormous amount of match data are needed. Since I'm a LOL player using chinese client and server, firstly I excepted to get the match data from its agent in China – Tencent. However, after browsing the official website carefully, I'm sure that it doesn't provide any API for the match data.

Fortunately, the North America(NA) server provides its match data free of charge by calling API . So the data used in this project is all from **NA server**.

To use the API, a developer need to register in https://developer.riotgames.com/ and has a game account in NA server. And all the APIs provided are listed in https://developer.riotgames.com/api-methods/.

For each API request, an API key must be included your API key using the api_key parameter on each request, that's why a developer need to register.

Developers can easily generate API key, but the API key string will be **expired in 24 hours** after applying each time, so it need updating mostly every time.



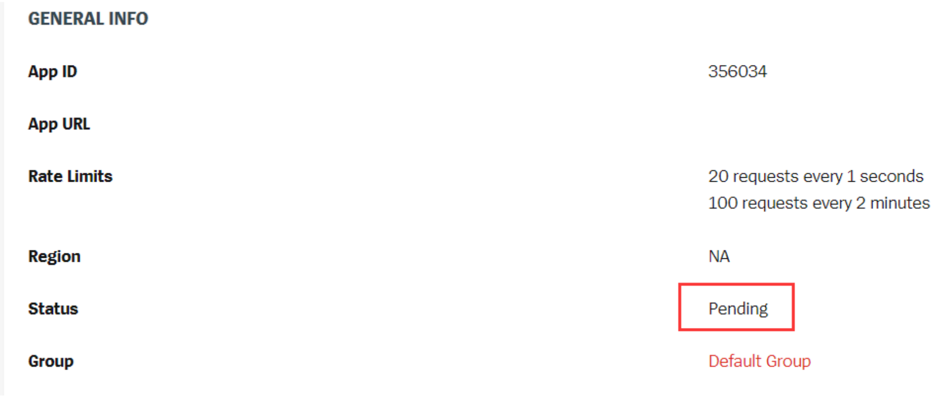
DEVELOPMENT API KEY

This API key is to be used for development only. Please register any permanent projects.
Do NOT use this API key in a publicly available project!

RGAPI-20aaff70-57b7-44b1-b66e-9cbbef7c8865

Expired: Mon, Dec 10th, 2018 @ 10:24pm (PT)
Your key has expired. You must regenerate your API key.

Registering permanent projects can provide a permanent API key, but my application is still pending when I'm writing this report.



GENERAL INFO

App ID	356034
App URL	
Rate Limits	20 requests every 1 seconds 100 requests every 2 minutes
Region	NA
Status	Pending
Group	Default Group

5.2 Data storage

Since the project is based on python36 and the data volume needed is not much huge, sqlite3 is the best choice. All the operations related to database are defined and implemented in **utils/SQLiteUtils.py**.

The database called info.db has 5 data table – account, match, championMatchData, championData and championInfo.

account is used to store the accounts searched or to be searched for match data.

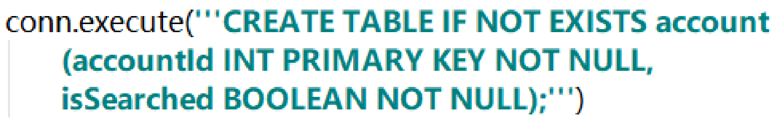
match is used to store the match id of the match data.

championMatchData stores the numerical details of a specific champion in a match.

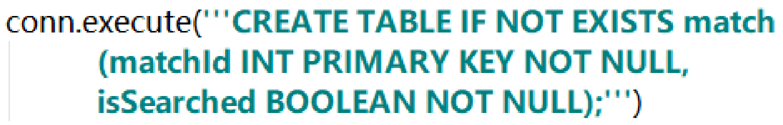
championData is used to store the typical value for each champion

championInfo stores the detailed information for each champion id

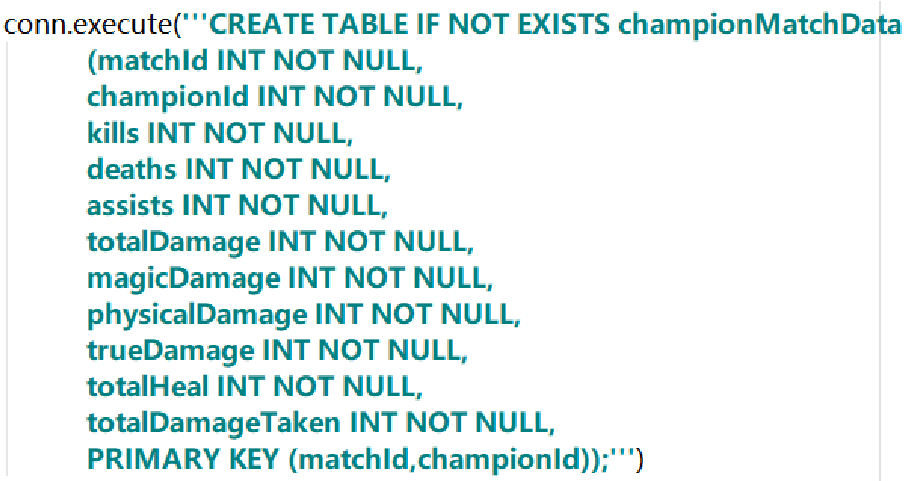
The data table structure are listed below.



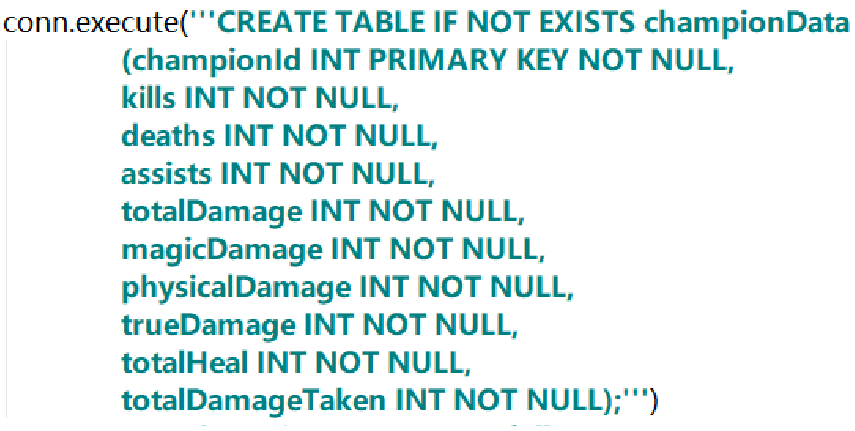
```
conn.execute('''CREATE TABLE IF NOT EXISTS account
     (accountId INT PRIMARY KEY NOT NULL,
     isSearched BOOLEAN NOT NULL);''')
```



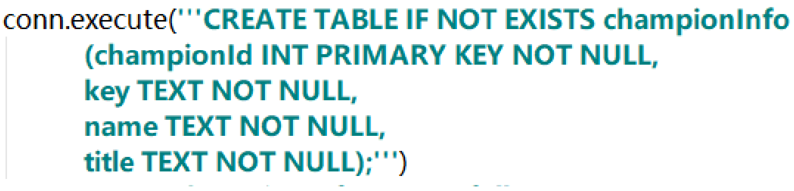
```
conn.execute('''CREATE TABLE IF NOT EXISTS match
        (matchId INT PRIMARY KEY NOT NULL,
        isSearched BOOLEAN NOT NULL);''')
```



```
conn.execute('''CREATE TABLE IF NOT EXISTS championMatchData
        (matchId INT NOT NULL,
        championId INT NOT NULL,
        kills INT NOT NULL,
        deaths INT NOT NULL,
        assists INT NOT NULL,
        totalDamage INT NOT NULL,
        magicDamage INT NOT NULL,
        physicalDamage INT NOT NULL,
        trueDamage INT NOT NULL,
        totalHeal INT NOT NULL,
        totalDamageTaken INT NOT NULL,
        PRIMARY KEY (matchId,championId));''')
```



```
conn.execute('''CREATE TABLE IF NOT EXISTS championData
          (championId INT PRIMARY KEY NOT NULL,
          kills INT NOT NULL,
          deaths INT NOT NULL,
          assists INT NOT NULL,
          totalDamage INT NOT NULL,
          magicDamage INT NOT NULL,
          physicalDamage INT NOT NULL,
          trueDamage INT NOT NULL,
          totalHeal INT NOT NULL,
          totalDamageTaken INT NOT NULL);''')
```



```
conn.execute('''CREATE TABLE IF NOT EXISTS championInfo
        (championId INT PRIMARY KEY NOT NULL,
        key TEXT NOT NULL,
        name TEXT NOT NULL,
        title TEXT NOT NULL);''')
```

5.3 Data retrieval

The main two API used in this project are:

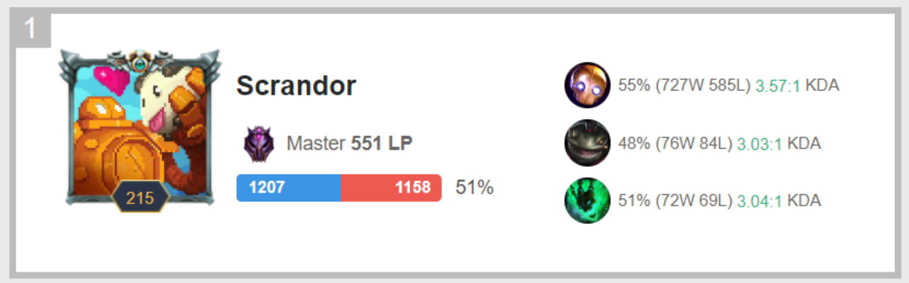
/lol/match/v3/matchlists/by-account/{accountId}

/lol/match/v3/matches/{matchId}

The first one is used to get the details of a specific match with match id matchId, the other is used to get match list of a specific player with account id accountId. These APIs return json files. The sample json file returned by the first API is **data/matchlist.json**, and the sample json file returned by the second one is **data/singlematch.json**.

Since there are no APIs to get a sufficient number of match data, it's required to start from a seed account id, get the match list of the player, and then for each match, get all of the participants of the match, and repeat this process.

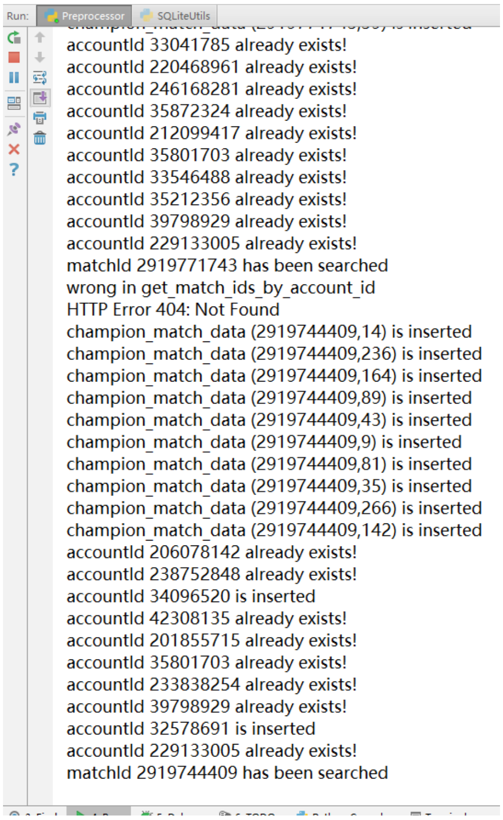
The seed account id chosen is the player with currently ranking #1 in NA server. (The rankings come from http://www.op.gg/). The account id obtained by API /lol/summoner/v3/summoners/by-name/{summonerName} is 37281196.



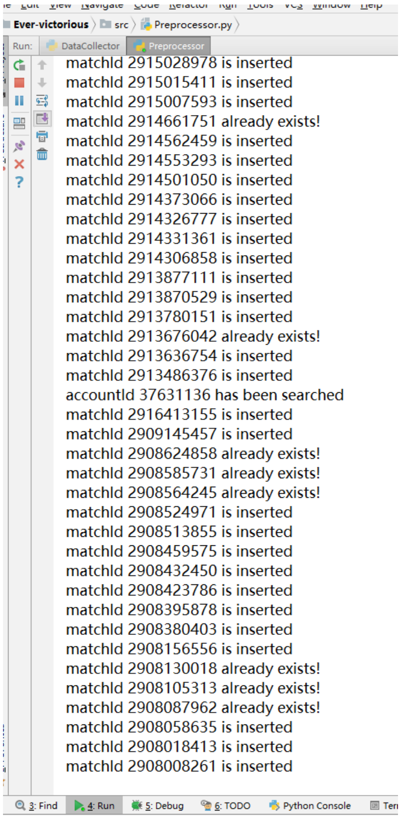
All the operations related to API are defined and implemented in **utils/DataCollector.py**.

In the file **src/Preprocessor.py**, **SQLiteUtils.py** and **DataCollector.py** are used to define methods to implement the loop of getting enough match data and store the data into corresponding data table.

Part of logs during data retrival are as follows:



```
accountId 33041785 already exists!
accountId 220468961 already exists!
accountId 246168281 already exists!
accountId 35872324 already exists!
accountId 212099417 already exists!
accountId 35801703 already exists!
accountId 33546488 already exists!
accountId 35212356 already exists!
accountId 39798929 already exists!
accountId 229133005 already exists!
matchId 2919771743 has been searched
wrong in get_match_ids_by_account_id
HTTP Error 404: Not Found
champion_match_data (2919744409,14) is inserted
champion_match_data (2919744409,236) is inserted
champion_match_data (2919744409,164) is inserted
champion_match_data (2919744409,89) is inserted
champion_match_data (2919744409,43) is inserted
champion_match_data (2919744409,9) is inserted
champion_match_data (2919744409,81) is inserted
champion_match_data (2919744409,35) is inserted
champion_match_data (2919744409,266) is inserted
champion_match_data (2919744409,142) is inserted
accountId 206078142 already exists!
accountId 238752848 already exists!
accountId 34096520 is inserted
accountId 42308135 already exists!
accountId 201855715 already exists!
accountId 35801703 already exists!
accountId 233838254 already exists!
accountId 39798929 already exists!
accountId 32578691 is inserted
accountId 229133005 already exists!
matchId 2919744409 has been searched
```



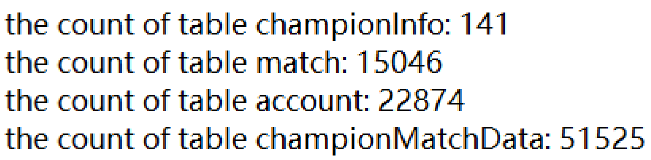
```
matchId 2915028978 is inserted
matchId 2915015411 is inserted
matchId 2915007593 is inserted
matchId 2914661751 already exists!
matchId 2914562459 is inserted
matchId 2914553293 is inserted
matchId 2914501050 is inserted
matchId 2914373066 is inserted
matchId 2914326777 is inserted
matchId 2914331361 is inserted
matchId 2914306858 is inserted
matchId 2913877111 is inserted
matchId 2913870529 is inserted
matchId 2913780151 is inserted
matchId 2913676042 already exists!
matchId 2913636754 is inserted
matchId 2913486376 is inserted
accountId 37631136 has been searched
matchId 2916413155 is inserted
matchId 2909145457 is inserted
matchId 2908624858 already exists!
matchId 2908585731 already exists!
matchId 2908564245 already exists!
matchId 2908524971 is inserted
matchId 2908513855 is inserted
matchId 2908459575 is inserted
matchId 2908432450 is inserted
matchId 2908423786 is inserted
matchId 2908395878 is inserted
matchId 2908380403 is inserted
matchId 2908156556 is inserted
matchId 2908130018 already exists!
matchId 2908105313 already exists!
matchId 2908087962 already exists!
matchId 2908058635 is inserted
matchId 2908018413 is inserted
matchId 2908008261 is inserted
```

5.4 Data cleaning

After data retrieval, invalid data need to be removed. For now, with only the data in the json files, there is no way to judge whether the player is active during the game. So personally conclude that if all of kills, deaths and assists are less than 2, the player is regarded as inactive and the match data of the champion usd by the player is invalid.

The method for data cleaning is defined in **SQLiteUtils.py** and is called by **Preprocessor.py** after data retrieval.

After data cleaning, the data in the database is as follows:

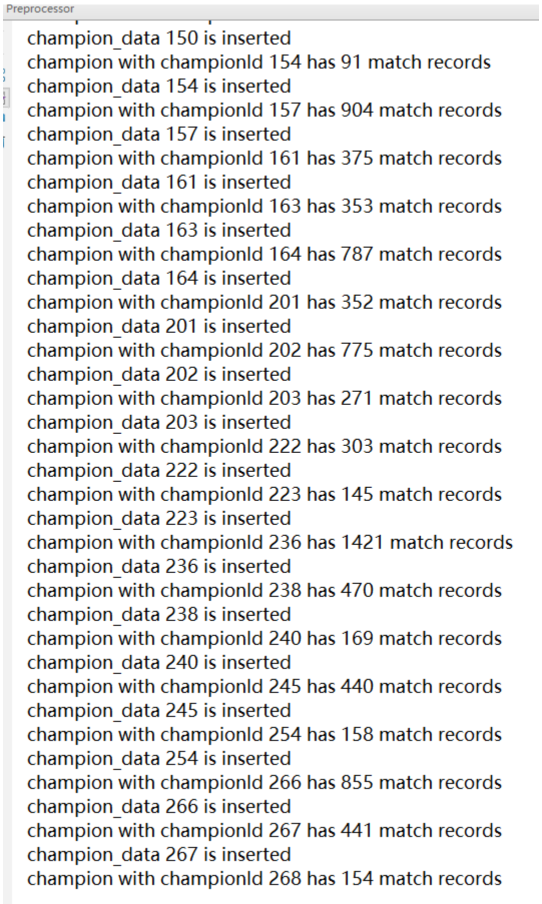


```
the count of table championInfo: 141
the count of table match: 15046
the count of table account: 22874
the count of table championMatchData: 51525
```

5.5 Typical value computing and data normalization

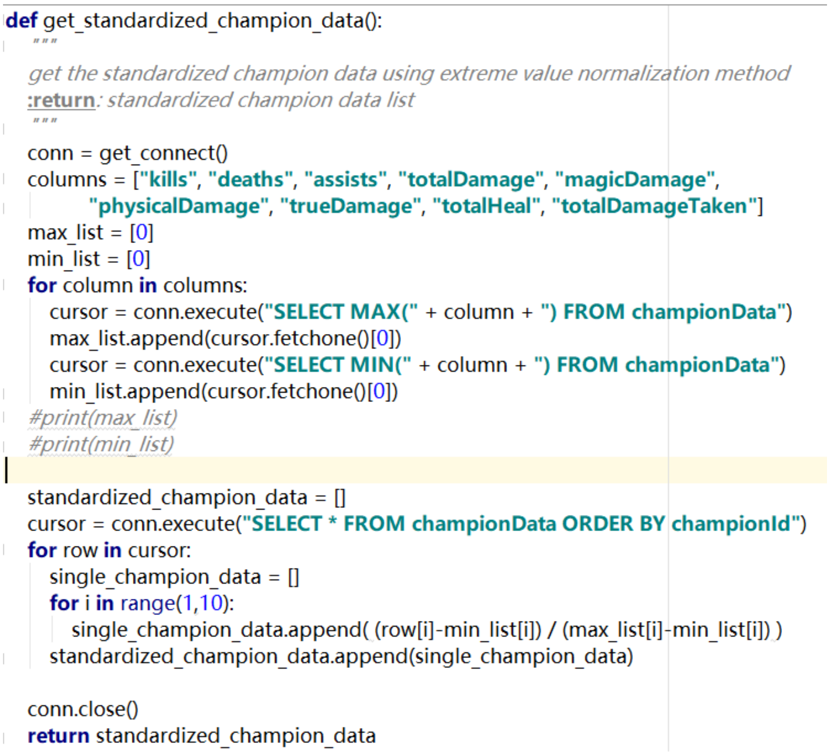
At this stage, simply regard the average value among the matches of a specific champion as its typical value for analysis.

Part of the logs for getting typical value are as follows:



```
champion_data 150 is inserted
champion with championId 154 has 91 match records
champion_data 154 is inserted
champion with championId 157 has 904 match records
champion_data 157 is inserted
champion with championId 161 has 375 match records
champion_data 161 is inserted
champion with championId 163 has 353 match records
champion_data 163 is inserted
champion with championId 164 has 787 match records
champion_data 164 is inserted
champion with championId 201 has 352 match records
champion_data 201 is inserted
champion with championId 202 has 775 match records
champion_data 202 is inserted
champion with championId 203 has 271 match records
champion_data 203 is inserted
champion with championId 222 has 303 match records
champion_data 222 is inserted
champion with championId 223 has 145 match records
champion_data 223 is inserted
champion with championId 236 has 1421 match records
champion_data 236 is inserted
champion with championId 238 has 470 match records
champion_data 238 is inserted
champion with championId 240 has 169 match records
champion_data 240 is inserted
champion with championId 245 has 440 match records
champion_data 245 is inserted
champion with championId 254 has 158 match records
champion_data 254 is inserted
champion with championId 266 has 855 match records
champion_data 266 is inserted
champion with championId 267 has 441 match records
champion_data 267 is inserted
champion with championId 268 has 154 match records
```

Different data indicators often have different dimension and dimension units, which will affect the results of data analysis. In order to eliminate the dimension effect between indicators, data standardization is needed to solve the comparability between data indicators. To get the standardized champion data, extreme value normalization method is used to scale all the data into [0,1]



```
def get_standardized_champion_data():
    """
    get the standardized champion data using extreme value normalization method
    :return: standardized champion data list
    """
    conn = get_connect()
    columns = ["kills", "deaths", "assists", "totalDamage", "magicDamage",
               "physicalDamage", "trueDamage", "totalHeal", "totalDamageTaken"]
    max_list = [0]
    min_list = [0]
    for column in columns:
        cursor = conn.execute("SELECT MAX(" + column + ") FROM championData")
        max_list.append(cursor.fetchone()[0])
        cursor = conn.execute("SELECT MIN(" + column + ") FROM championData")
        min_list.append(cursor.fetchone()[0])
    #print(max_list)
    #print(min_list)

    standardized_champion_data = []
    cursor = conn.execute("SELECT * FROM championData ORDER BY championId")
    for row in cursor:
        single_champion_data = []
        for i in range(1,10):
            single_champion_data.append( (row[i]-min_list[i]) / (max_list[i]-min_list[i]) )
        standardized_champion_data.append(single_champion_data)

    conn.close()
    return standardized_champion_data
```

5.6 Hierarchical clustering analysis

For analysing the similarity and mining, unsupervised learning approach is more suitable for this project.

At first, I tended to use k-means clustering method for analysis since we have learned it in the class. However, after actual implementation, the results expose many shortcomings of k-means clustering method:

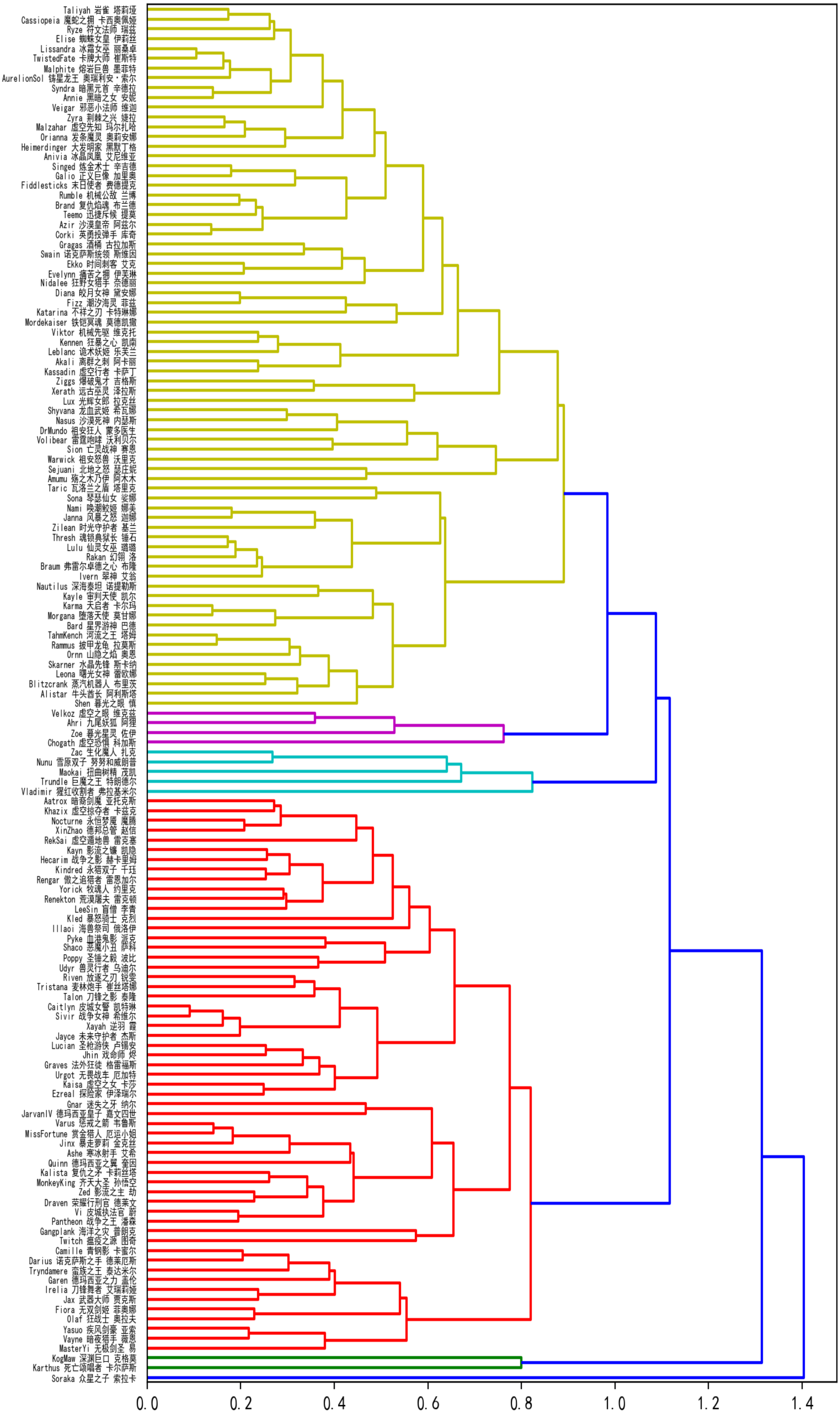
- Clustering results are affected by the initial number of clustering centers
- Clustering results are influenced by initial center selection

- Clustering results are influenced by the order of sample input

- Non-convergence case will occur

For such reasons, hierarchical clustering method are chosen. Hierarchical clustering basically overcomes the shortcomings above. It can display the whole clustering process from discrete sample points to finally a single group with no conception of initial centers, center number, input order and non-convergence.

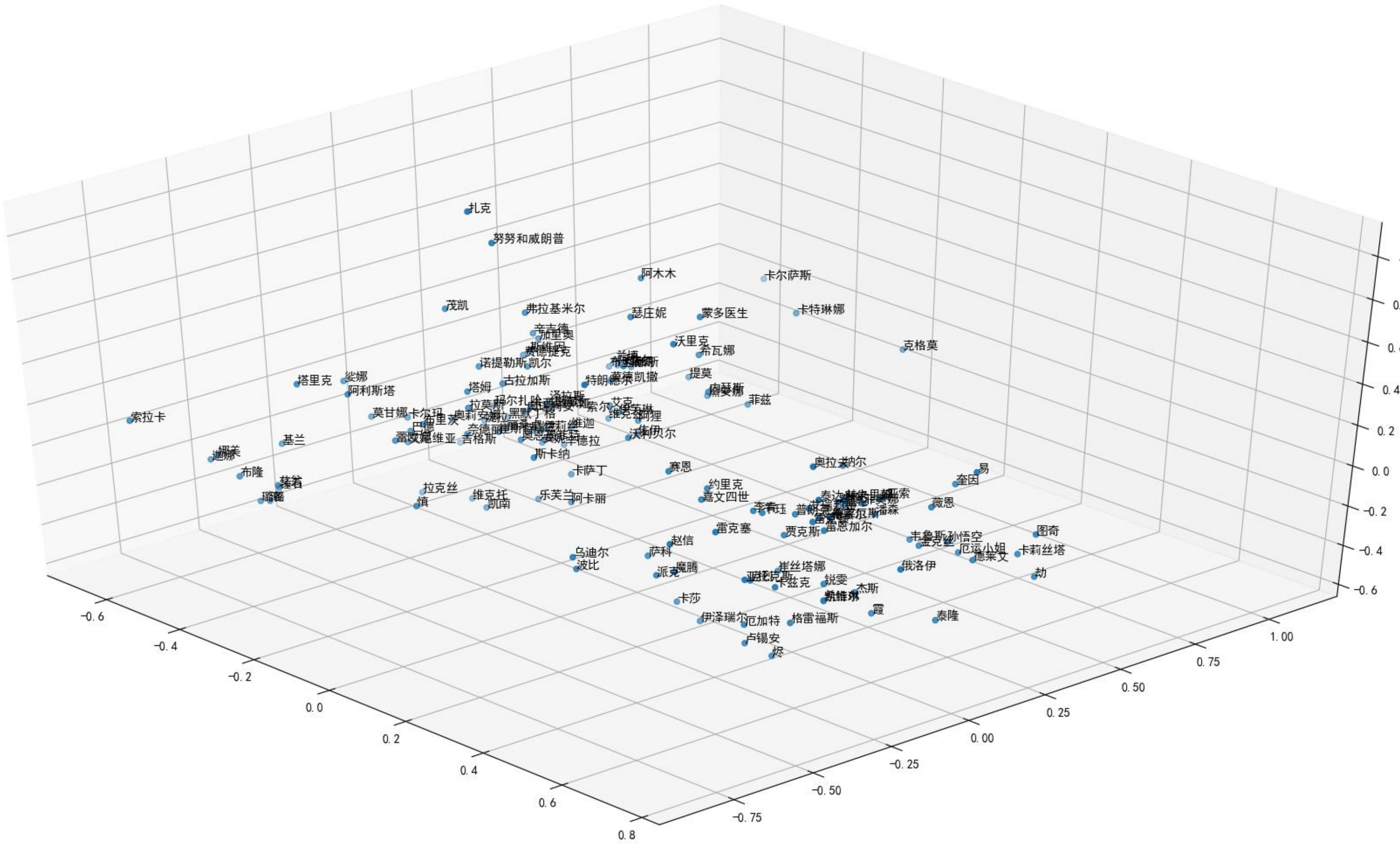
The hierarchical clustering result is as follows and saved in the same directory of this report named **clustering_figure.png**:


Taliyah 岩雀 塔莉垭
Cassiopeia 魔蛇之拥 卡西奥佩娅
Ryze 符文法师 瑞兹
Elise 蜘蛛女皇 伊莉丝
Lissandra 冰霜女巫 丽桑卓
TwistedFate 卡牌大师 崔斯特
Malphite 熔岩巨兽 墨菲特
AurelionSol 铸星龙王 奥瑞利安·索尔
Syndra 暗黑元首 辛德拉
Annie 黑暗之女 安妮
Veigar 邪恶小法师 维迦
Zyra 荆棘之兴 婕拉
Malzahar 虚空先知 玛尔扎哈
Orianna 发条魔灵 奥莉安娜
Heimerdinger 大发明家 黑默丁格
Anivia 冰晶凤凰 艾尼维亚
Singed 炼金术士 辛吉德
Galio 正义巨像 加里奥
Fiddlesticks 末日使者 费德提克
Rumble 机械公敌 兰博
Brand 复仇焰魂 布兰德
Teemo 迅捷斥候 提莫
Azir 沙漠皇帝 阿兹尔
Corki 英勇投弹手 库奇
Gragas 酒桶 古拉加斯
Swain 诺克萨斯统领 斯维因
Ekko 时间刺客 艾克
Evelynn 痛苦之拥 伊芙琳
Nidalee 狂野女猎手 奈德丽
Diana 皎月女神 黛安娜
Fizz 潮汐海灵 菲兹
Katarina 不祥之刃 卡特琳娜
Mordekaiser 铁铠冥魂 莫德凯撒
Viktor 机械先驱 维克托
Kennen 狂暴之心 凯南
Leblanc 诡术妖姬 乐芙兰
Akali 离群之刺 阿卡丽
Kassadin 虚空行者 卡萨丁
Ziggs 爆破鬼才 吉格斯
Xerath 远古巫灵 泽拉斯
Lux 光辉女郎 拉克丝
Shyvana 龙血武姬 希瓦娜
Nasus 沙漠死神 内瑟斯
DrMundo 祖安狂人 蒙多医生
Volibear 雷霆咆哮 沃利贝尔
Sion 亡灵战神 赛恩
Warwick 祖安怒兽 沃里克
Sejuani 北地之怒 瑟庄妮
Amumu 殇之木乃伊 阿木木
Taric 瓦洛兰之盾 塔里克
Sona 琴瑟仙女 娑娜
Nami 唤潮鲛姬 娜美
Janna 风暴之怒 迦娜
Zilean 时光守护者 基兰
Thresh 魂锁典狱长 锤石
Lulu 仙灵女巫 璐璐
Rakan 幻翎 洛
Braum 弗雷尔卓德之心 布隆
Ivern 翠神 艾翁
Nautilus 深海泰坦 诺提勒斯
Kayle 审判天使 凯尔
Karma 天启者 卡尔玛
Morgana 堕落天使 莫甘娜
Bard 星界游神 巴德
TahmKench 河流之王 塔姆
Rammus 披甲龙龟 拉莫斯
Ornn 山隐之焰 奥恩
Skarner 水晶先锋 斯卡纳
Leona 曙光女神 蕾欧娜
Blitzcrank 蒸汽机器人 布里茨
Alistar 牛头酋长 阿利斯塔
Shen 暮光之眼 慎
Velkoz 虚空之眼 维克兹
Ahri 九尾妖狐 阿狸
Zoe 暮光星灵 佐伊
Chogath 虚空恐惧 科加斯
Zac 生化魔人 扎克
Nunu 雪原双子 努努和威朗普
Maokai 扭曲树精 茂凯
Trundle 巨魔之王 特朗德尔
Vladimir 猩红收割者 弗拉基米尔
Aatrox 暗裔剑魔 亚托克斯
Khazix 虚空掠夺者 卡兹克
Nocturne 永恒梦魇 魔腾
XinZhao 德邦总管 赵信
RekSai 虚空遁地兽 雷克塞
Kayn 影流之镰 凯隐
Hecarim 战争之影 赫卡里姆
Kindred 永猎双子 千珏
Rengar 傲之追猎者 雷恩加尔
Yorick 牧魂人 约里克
Renekton 荒漠屠夫 雷克顿
LeeSin 盲僧 李青
Kled 暴怒骑士 克烈
Illaoi 海兽祭司 俄洛伊
Pyke 血港鬼影 派克
Shaco 恶魔小丑 萨科
Poppy 圣锤之毅 波比
Udyr 兽灵行者 乌迪尔
Riven 放逐之刃 锐雯
Tristana 麦林炮手 崔丝塔娜
Talon 刀锋之影 泰隆
Caitlyn 皮城女警 凯特琳
Sivir 战争女神 希维尔
Xayah 逆羽 霞
Jayce 未来守护者 杰斯
Lucian 圣枪游侠 卢锡安
Jhin 戏命师 烬
Graves 法外狂徒 格雷福斯
Urgot 无畏战车 厄加特
Kaisa 虚空之女 卡莎
Ezreal 探险家 伊泽瑞尔
Gnar 迷失之牙 纳尔
JarvanIV 德玛西亚皇子 嘉文四世
Varus 惩戒之箭 韦鲁斯
MissFortune 赏金猎人 厄运小姐
Jinx 暴走萝莉 金克丝
Ashe 寒冰射手 艾希
Quinn 德玛西亚之翼 奎因
Kalista 复仇之矛 卡莉丝塔
MonkeyKing 齐天大圣 孙悟空
Zed 影流之主 劫
Draven 荣耀行刑官 德莱文
Vi 皮城执法官 蔚
Pantheon 战争之王 潘森
Gangplank 海洋之灾 普朗克
Twitch 瘟疫之源 图奇
Camille 青钢影 卡蜜尔
Darius 诺克萨斯之手 德莱厄斯
Tryndamere 蛮族之王 泰达米尔
Garen 德玛西亚之力 盖伦
Irelia 刀锋舞者 艾瑞莉娅
Jax 武器大师 贾克斯
Fiora 无双剑姬 菲奥娜
Olaf 狂战士 奥拉夫
Yasuo 疾风剑豪 亚索
Vayne 暗夜猎手 薇恩
MasterYi 无极剑圣 易
KogMaw 深渊巨口 克格莫
Karthus 死亡颂唱者 卡尔萨斯
Soraka 众星之子 索拉卡
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4

5.7 Principal component analysis

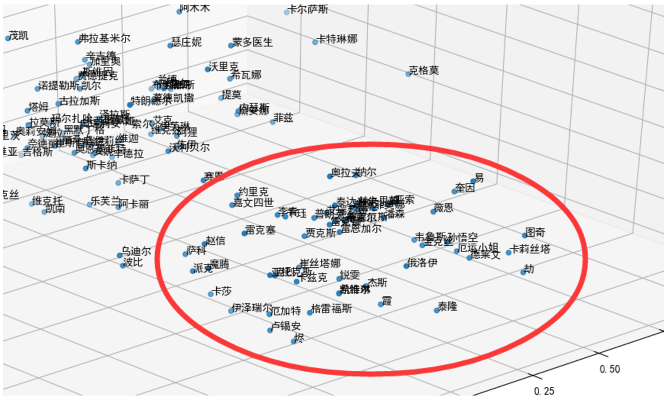
In order to show the similarity among champions in a more intuitive way, principal component analysis is applied to implement data dimension reduction to 3d.

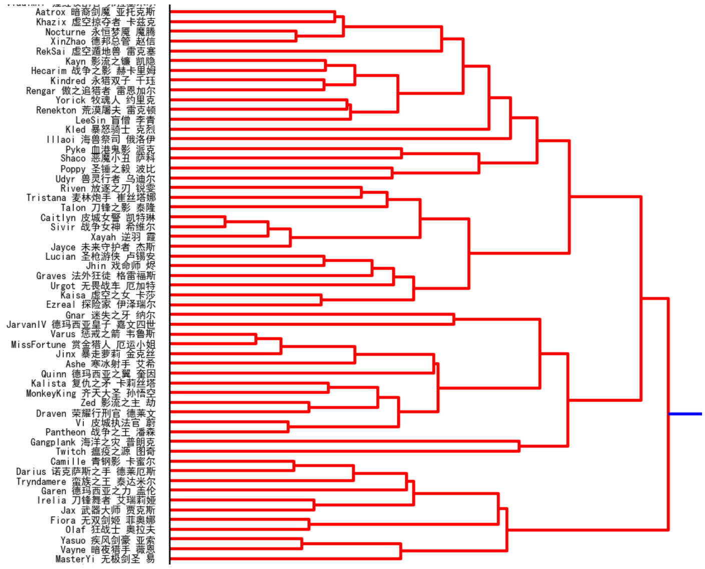
The principal component analysis result is as follows and saved in the same directory of this report named **pca_figure.png**:



6. Conclusions

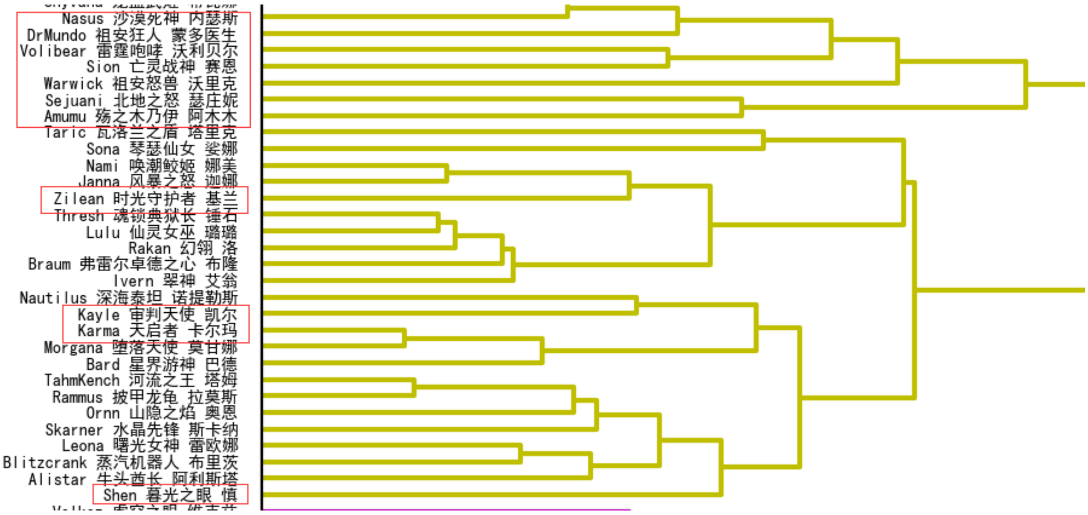
1. In both Hierarchical clustering analysis and Principal component analysis, most of the champions with position Marksman perform similarly with Fighters and Assassins. That means that they typically play the same role in a game, so it’s unadvisable to have too many such champions in one team.



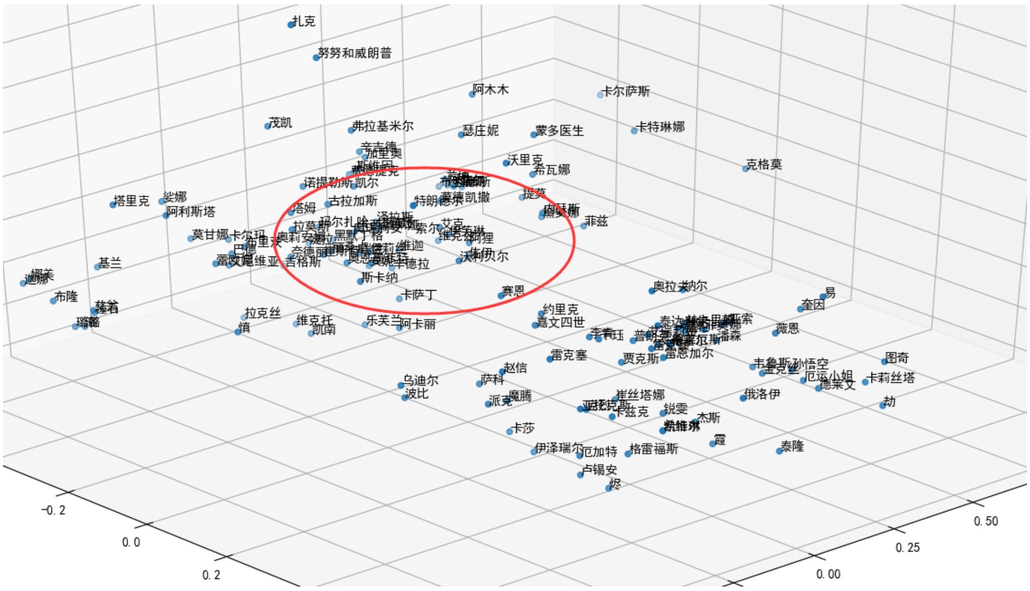


2. Some funcitional Tanks and Mages perform similarly with Supports. The champions with position Support are designed for funcitional supports to other champions. But there are also champions with other positions can do

the same things.

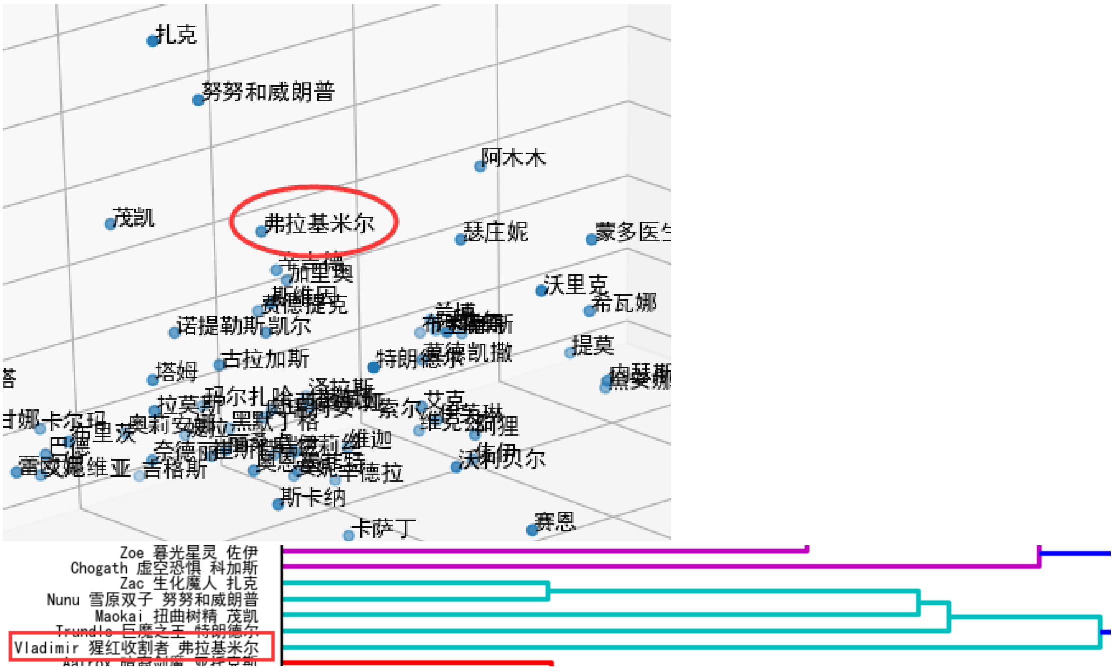


3. Some typical value of champions tend to be neutral. Most of those champions play flexible roles in games based on the items it equiped, which is determined by the game styles and preferences of the players.

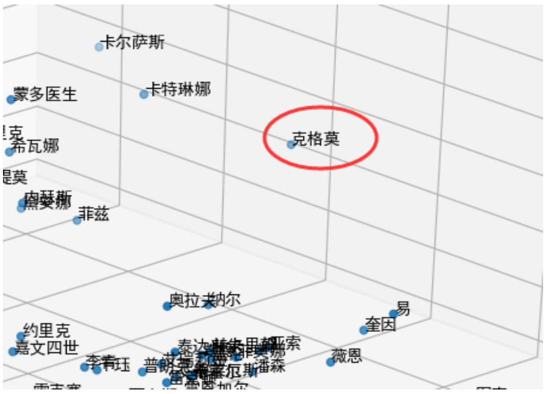


There are some rare extreme cases, too.

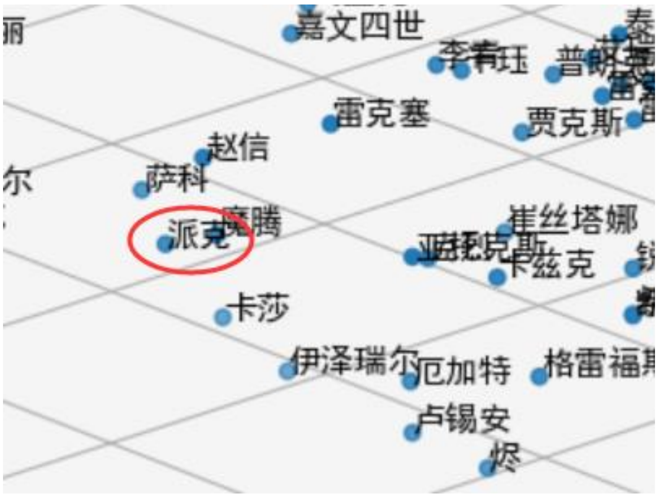
4. Mage “Vladimir “ performs like Tank. This Mage can continuously suck blood from enemy, which make it the ability of taking a lot of damage like Tank



5. Marksman “KogMaw” performs neither like Marksman nor Mage, but in between. That’s because this champion’s skills make it the ability of causing both magic and physical demage.



6. Support “Pyke” performs like Assassin and Marksman. That’s because its unique skill can directly kill enemies with blood less than a certain amount.



And there are many other interesting results.

7. Details And Problems

7.1 About implementation

In the process of implementation, firstly I tend to implement both the Hierarchical clustering analysis and Principal component analysis by myself. After the initial implementation and testing, it seems that the space efficiency and time efficiency of my implementation are not high enough. What’s more is that my implementation of Hierarchical clustering can only display the clustering process in text, which has poor readability.

After searching related information on the Internet, the python libraries **spicy.cluster** and **sklearn.decomposition** meet the needs of the project respectively for Hierarchical clustering analysis and Principal component analysis. That’s also the final version of the approaches adopted.

7.2 About results

In the process of Hierarchical clustering analysis and Principal component analysis, champions with positions Fighter, Assassin and Marksman have a mixing trend from the very beginning. But in real games, those champions are supposed to work together to win the game instead of having many champions of the same position. I suspect that there are two possible reasons for this situation.

The first reason is in my normalization process, each data indicators are scaled to [0, 1], which are used directly to compute the similarity distance. But in fact, these indicators should have different weights since the indicators represent different features of the champions. However, after trying many different weights combination, I still cannot get ideal clustering process that can divide those champions thoroughly, so in the final version of the project, the default weight 1 is remained.

The second reason is the lack of enough indicators. The match data returned by the API only contains values for the whole match. As a matter of experience, the Fighters and Assassins play an important role in the early stages of the game, while the Marksmen play a more crucial part in the later stages. If such information related to different stages can be obtained, at least the Marksman will separate from the mixing trend.

8. How to see results

Each part of the process is separated in its corresponding .py files with condition “if __name__ == ‘__main__’”, so load the project into PyCharm and then it’s easy to run any part of the project by run the related files. Since the final data is stored in the database, it’s also convenient to directly run the **Analysis.py** to see the final analysis results.

For each API request, an API key must be included your API key using the api_key parameter on each request, that’s why a developer need to register.

Developers can easily generate API key, but the API key string will be **expired in 24 hours** after applying each time, so it’s needed to update the **api_key** in the file **utils/DataCollector.py** when wanting to call the methods of this file**.**

The steps for generating new api key:

1. Open https://developer.riotgames.com/
2. Login with your account, or my account with username RuikangLiu and password 123456Lrk
3. Open the DASHBOARD
4. Click the button “:REGENERATE API KEY” in the bottom of the page
5. Copy the new api key into the python file **utils/DataCollector.py** to replace the old string of **api_key**