Jiyuan Zhao
INFO 7374 Big Data Engineering
Northeastern University
Professor Yusuf Ozbek

League of Legends Champion Stats and Win Prediction

Introduction

League of Legends is an online competitive multiplayer strategy game that pits a team of five players against another team of five players with each player selecting a distinct character, known as a champion, from a pool of currently 126 champions. It is currently the most played online game in the world, with over 7.5 million players concurrently during peak hours.

Goal

My goal is to use the historical data given by the official API, to provide players with a comprehensive champion stats overview and predict which team will win before a match begins. This is particularly essential to game playing because (professional) players can make a decision based on the analysis data whether they should considering “dodging” a match before it becomes a definite lost.

Motivation

Right before a match begins, 10 players take turns to select their desired champion from a champion pool of currently 126 champions. During this period, each player can see what the opponent team is selecting. Meanwhile, if one player feels the opponent champions or team composition is too overpowered, he can leave the game a.k.a. “dodging”. Instead of losing 20+ LPs (League Points), dodging will only deduct a few LPs from the player who left. Therefore, it saves the whole team and quite an amount of time. This is a good winning strategy especially for professional players.

Data

Riot Games, the company who created League of Legends, has provided developers with a set of gaming API to retrieve game data from its server. Every developer is granted a unique API key for accessing API. I wrote a LoLDataFetch java program to retrieve data from the server. In order to generate enough data for processing, I have to deploy the LoLDataFetch program on AWS. The program ran continuously and at the end I was able to generate about 10,000 unique matches. These 10,000 unique matches were separate into 100 JSON files with a total of 1.66 GB. Information for each match record includes match ID, match type, team composition, all 10 champions' information and game footprints and milestones at each timestamp. There are 100,000 champion records in total.

Technology

The technologies used in this project is as follows:

- Amazon Web Services
- Hadoop
  - HBase
  - MapReduce
  - Pig
  - Mahout
- Web
  - HTML, CSS, JavaScript
  - Google Chart
  - Bootstrap

WorkFlow

A brief description of the workflow is as follows:

1. **API**

   Riot Games official developer API was used to access gaming data.

   - Input:

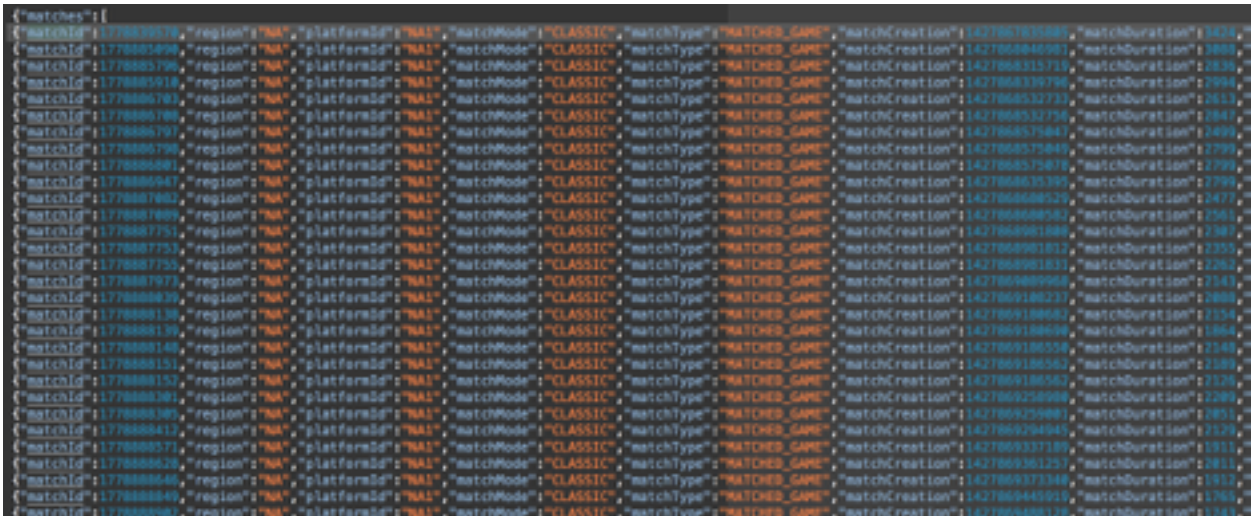
     ```
     https://na.api.pvp.net/api/lol/na/v2.2/match/2033092160?
     api_key=60e01f47-cbf0-4fb0-a9a5-bf89aa642921
     ```

- Program: LoLDataFetcher.java

```java
public class MatchDataFetcher {

    private static String matchList = "https://na.api.pvp.net/api/lol/na/v2.2/match/";
    private static String api_key = "?api_key=60e01f47-cbf0-4fb0-a9a5-bf89aa642921";
    private static String prefix = "{\"matchId\"";

    public static void main(String[] args) throws Exception {

        PrintWriter writer = new PrintWriter("match-1m.json", "UTF-8");

        for (int i = 1807845889; i < (1807845889+1000000); i++) {
            String matchListQuery = matchList + String.valueOf(i) + api_key;
            URL matchListUrl = new URL(matchListQuery);

            try (BufferedReader matchListReader = new BufferedReader(new InputStreamReader(matchListUrl.openStream()))) {
                for (String matchListLine; (matchListLine = matchListReader.readLine()) != null;) {
                    if (matchListLine.startsWith(prefix) && matchListLine.contains("RANKED_SOLO_5x5")) {
                        System.out.println(matchListLine);
                        writer.println(matchListLine);
                        TimeUnit.MILLISECONDS.sleep(1205);
                    }
                }

            } catch (Exception e) {
                continue;
            }

        }

        writer.close();

    }
}
```

I was confronted with a challenging situation that there was a bandwidth limit of approximately 1 fetch per 1.2 seconds. An API key will be blacklisted if it goes beyond the limit. So I placed a wait statement inside my java program so it will generate a query every 1205 milliseconds.

- Output:



2. JSON

After dataset was downloaded, it was sorted and categorized. Useful information like champion ID, team composition, champion kills/deaths/assists was filtered out.

- Input: Sample JSON


```
{
    "region":"NA",
    "matchType":"MATCHED_GAME",
    "matchCreation":1449684631617,
    "participants":[ … ],
    "platformId":"NA1",
    "matchMode":"CLASSIC",
    "participantIdentities":[ … ],
    "matchVersion":"5.24.0.251",
    "teams":[ … ],
    "mapId":11,
    "season":"PRESEASON2016",
    "queueType":"RANKED_SOLO_5x5",
    "matchId":2033092160,
    "matchDuration":1904
}
```


- Program: LoL_Predictor.java

```
File dir = new File("input");
File[] files = dir.listFiles((File dir1, String name) -> name.endsWith(".json"));
int j = 0;

for (File file : files) {

    System.out.println("Reading " + file.getName() + " ......");
    try {
        Object object = parser.parse(new FileReader(file));
        JSONObject matchesJsonObject = (JSONObject) object;
        JSONArray matchJsonArray = (JSONArray) matchesJsonObject.get("matches");
        MatchJSON matchJSON = new MatchJSON();
        int i = 1;

        for (Object o : matchJsonArray) {
            JSONObject matchJSONObject = (JSONObject) o;

            matchJSON.setMatchId((long) matchJSONObject.get("matchId"));

            matchJSON.setRegion((String) matchJSONObject.get("region"));

            matchJSON.setPlatformId((String) matchJSONObject.get("platformId"));
            matchJSON.setMatchMode((String) matchJSONObject.get("matchMode"));
            matchJSON.setMatchType((String) matchJSONObject.get("matchType"));
            matchJSON.setMatchCreation((long) matchJSONObject.get("matchCreation"));
            matchJSON.setMatchDuration(Integer.parseInt(matchJSONObject.get("matchDuration").toString()));
            matchJSON.setQueueType((String) matchJSONObject.get("queueType"));
            matchJSON.setMapId(Integer.parseInt(matchJSONObject.get("mapId").toString()));
            matchJSON.setSeason((String) matchJSONObject.get("season"));
```

- Output: champion_name, kills, deaths, assists, result (v as victory, d as defeated)

```
Sejuani,11,5,34,d
Vayne,15,8,10,d
Morgana,3,10,26,d
Warwick,10,14,21,d
Ahri,19,14,18,d
Irelia,11,12,9,v
Akali,28,12,4,v
Tristana,9,13,11,v
Wukong,2,10,17,v
Taric,1,12,24,v
Kassadin,12,12,6,d
…
```

3. HBase

All the above useful data was stored into HBase for further reference and processing.

- Input: result-sample.csv

```
Sejuani,11,5,34,d
Vayne,15,8,10,d
Morgana,3,10,26,d
Warwick,10,14,21,d
Ahri,19,14,18,d
Irelia,11,12,9,v
Akali,28,12,4,v
Tristana,9,13,11,v
Wukong,2,10,17,v
Taric,1,12,24,v
Kassadin,12,12,6,d
…
```

- Program: LoL_Predictor.java

```
Put p = new Put(Bytes.toBytes(String.valueOf(j)));

// adding values using add() method
// accepts column family name, qualifier/row name ,value
p.add(Bytes.toBytes("championId"),
        Bytes.toBytes("Id"), Bytes.toBytes(participant.get("championId").toString()));

p.add(Bytes.toBytes("championInfo"),
        Bytes.toBytes("kills"), Bytes.toBytes(stats.get("kills").toString()));

p.add(Bytes.toBytes("championInfo"),
        Bytes.toBytes("deaths"), Bytes.toBytes(stats.get("deaths").toString()));

p.add(Bytes.toBytes("championInfo"),
        Bytes.toBytes("assits"), Bytes.toBytes(stats.get("assists").toString()));

p.add(Bytes.toBytes("championInfo"),
        Bytes.toBytes("result"), Bytes.toBytes(result));

// Saving the put Instance to the HTable.
hTable.put(p);
```

- Output:

```
99994                                   column=championInfo:kills, timestamp=1449951135696, value=15
99994                                   column=championInfo:result, timestamp=1449951135696, value=d
99995                                   column=championId:Id, timestamp=1449951135696, value=143
99995                                   column=championInfo:assits, timestamp=1449951135696, value=20
99995                                   column=championInfo:deaths, timestamp=1449951135696, value=10
99995                                   column=championInfo:kills, timestamp=1449951135696, value=4
99995                                   column=championInfo:result, timestamp=1449951135696, value=d
99996                                   column=championId:Id, timestamp=1449951135696, value=15
99996                                   column=championInfo:assits, timestamp=1449951135696, value=7
99996                                   column=championInfo:deaths, timestamp=1449951135696, value=10
99996                                   column=championInfo:kills, timestamp=1449951135696, value=21
99996                                   column=championInfo:result, timestamp=1449951135696, value=v
99997                                   column=championId:Id, timestamp=1449951135696, value=7
99997                                   column=championInfo:assits, timestamp=1449951135696, value=9
99997                                   column=championInfo:deaths, timestamp=1449951135696, value=9
99997                                   column=championInfo:kills, timestamp=1449951135696, value=10
99997                                   column=championInfo:result, timestamp=1449951135696, value=v
99998                                   column=championId:Id, timestamp=1449951135696, value=25
99998                                   column=championInfo:assits, timestamp=1449951135696, value=17
99998                                   column=championInfo:deaths, timestamp=1449951135696, value=8
99998                                   column=championInfo:kills, timestamp=1449951135696, value=1
99998                                   column=championInfo:result, timestamp=1449951135696, value=v
```

4. MapReduce (Pig)

MapReduce job written in Pig script ran on the data from HBase to generate summarized results such as averaged win/lose rate as well as averaged kills/deaths/assists.

- Input:

```
99994                                   column=championInfo:kills, timestamp=1449951135696, value=15
99994                                   column=championInfo:result, timestamp=1449951135696, value=d
99995                                   column=championId:Id, timestamp=1449951135696, value=143
99995                                   column=championInfo:assits, timestamp=1449951135696, value=20
99995                                   column=championInfo:deaths, timestamp=1449951135696, value=10
99995                                   column=championInfo:kills, timestamp=1449951135696, value=4
99995                                   column=championInfo:result, timestamp=1449951135696, value=d
99996                                   column=championId:Id, timestamp=1449951135696, value=15
99996                                   column=championInfo:assits, timestamp=1449951135696, value=7
99996                                   column=championInfo:deaths, timestamp=1449951135696, value=10
99996                                   column=championInfo:kills, timestamp=1449951135696, value=21
99996                                   column=championInfo:result, timestamp=1449951135696, value=v
99997                                   column=championId:Id, timestamp=1449951135696, value=7
99997                                   column=championInfo:assits, timestamp=1449951135696, value=9
99997                                   column=championInfo:deaths, timestamp=1449951135696, value=9
99997                                   column=championInfo:kills, timestamp=1449951135696, value=10
99997                                   column=championInfo:result, timestamp=1449951135696, value=v
99998                                   column=championId:Id, timestamp=1449951135696, value=25
99998                                   column=championInfo:assits, timestamp=1449951135696, value=17
99998                                   column=championInfo:deaths, timestamp=1449951135696, value=8
99998                                   column=championInfo:kills, timestamp=1449951135696, value=1
99998                                   column=championInfo:result, timestamp=1449951135696, value=v
```

- Program: LoL_Predictor.pig

```
--1 declare a variable for dataset path
%declare DATASETPATH '/Users/zhaojiyuan/Dropbox/NEU/INFO7374/LoL_Predictor';

--2 load ratings file
champion = LOAD '$DATASETPATH/result-sample.txt' using PigStorage (',')
AS (champion_name:chararray, kills:int, deaths:int, assists:int, result:chararray);

champion = FOREACH champion GENERATE champion_name, kills, deaths, assists, (result == 'v' ? 1 : 0) AS victory, (result == 'd' ? 1 : 0) AS
defeated, result;

--4 group the ratings by movie_id
groupdChampion = GROUP champion BY champion_name;

final_data = FOREACH groupdChampion GENERATE group AS champion_name, ROUND(AVG(champion.kills)*100.00)/100.00 AS avgKills,
ROUND(AVG(champion.deaths)*100.00)/100.00 AS avgDeaths, ROUND(AVG(champion.assists)*100.00)/100.00 AS avgAssits, SUM(champion.victory) AS
sumVictory, SUM(champion.defeated) AS sumDefeated, ROUND(SUM(champion.victory)*10000.00/COUNT(champion.result))/100.00 AS rateVictory,
ROUND(SUM(champion.defeated)*10000.00/COUNT(champion.result))/100.00 AS rateDefeated;

final_data = ORDER final_data BY rateVictory DESC;

--STORE final_data INTO 'mr_result';
STORE final_data INTO 'mr_result' using PigStorage(',');
```

- Output: champion_name, average_kills, average_deaths, average_assists, wins_per_hundred_games, losts_per_hundred_games, win_rate, lose_rate

```
Heimerdinger,6.67,6.65,8.76,33,13,71.74,28.26
Poppy,9.31,7.31,9.31,9,4,69.23,30.77
Ashe,7.51,6.83,10.57,54,27,66.67,33.33
Anivia,6.33,4.57,7.57,13,8,61.9,38.1
Sivir,7.81,5.82,10.34,80,50,61.54,38.46
...
```

5. Machine Learning (Mahout)

Data after MapReduce job was again fed into Mahout for machine learning. Each individual champion and each match's team composition was analyzed. As a result, for every champion, machine learning predicts five different champions that might work best with that particular champion, their predicted win rates are also provided.

- Input: champion1_id, champion2_id, result (1 is victory, 0 as defeated)

```
113,67,0
113,25,0
113,19,0
113,103,0
67,113,0
67,25,0
67,19,0
67,103,0
25,113,0
25,67,0
…
```

- Program: LoL_Mahout

```
DataModel model = new FileDataModel(new File("mahout_input.csv"));

//step 2.find users with similar tastes
UserSimilarity similarity = new TanimotoCoefficientSimilarity(model);

//step 3.find user neignborhood
UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.1, similarity, model);

//step 4.create the recommender engine
UserBasedRecommender recommender = new GenericUserBasedRecommender(model, neighborhood, similarity);
String summonerName1;
String summonerName2;
//step 5.ask the recommender to give recommendations
for (int i = 1; i < 433; i++) {
    try {
        List<RecommendedItem> recommendations = recommender.recommend(i, 5, true);
        for (RecommendedItem recommendation : recommendations) {

            String championRequestString = championInfo + String.valueOf(i) + api_key;
            URL championRequestUrl = new URL(championRequestString);

            String championRequestString1 = championInfo + String.valueOf(recommendation.getItemID()) + api_key;
            URL championRequestUrl1 = new URL(championRequestString1);

            try (BufferedReader in = new BufferedReader(
                    new InputStreamReader(championRequestUrl.openStream()))) {
                String inputLine;
                while ((inputLine = in.readLine()) != null) {
                    JSONObject championJSONObject = (JSONObject) new JSONParser().parse(inputLine);
```

- Output:

```
Champion Syndra might work well with Champion: Kennen (predicted win rate: 0.6592)
Champion Syndra might work well with Champion: Yorick (predicted win rate: 0.6417)
Champion Zyra might work well with Champion: Heimerdinger (predicted win rate: 0.7074)
Champion Zyra might work well with Champion: Poppy (predicted win rate: 0.6991)
Champion Zyra might work well with Champion: Kennen (predicted win rate: 0.6582)
Champion Zyra might work well with Champion: Janna (predicted win rate: 0.6514)
…
```

Conclusion

It is fairly biased to predict who is going to win by just summing the champion-specific information. Looking at champion stats alone will not provide a good predictor, but may be good as an overview and additional attributes in classifiers where the extra dimensionality does not hinder it.

The overall win/rate data might also indicate the current fairness of the champion pool. For example, Heimerdinger has an overall win rate around 70%, which is way over the normal range. This could mean that this champion is currently overpowered and needs some modification and nerf.

This data is limited as they are temporal. Some of the most overpowered champion are new champions. I believe they will be modified later to assure the balance of the game.

Results also suggests that for most games played, individual skill on a champion is the most important aspect in determining a game's outcome over aspects like team compositions and experience.

Further examinations could be looking at other data such as other game statistics of players on a champion, like gold earned in previous games or the kill/death/assist averages. However, the limitation is that this takes more API calls and significantly longer to process. This also potentially requires classifiers that can handle missing attributes, since the API is more

likely to return error response instead of data and therefore just 100,000 champion records is definitely not representative to draw broader and more general conclusion.

Reference

Riot Games Official Developer API: https://developer.riotgames.com/

Source Code

- Final_Project
  - MatchDataAnalyzer.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package final_project;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.URL;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import org.json.JSONArray;
import org.json.JSONObject;

/**
 *
 * @author zhaojiyuan
 */
public class MatchDataAnalyzer extends Configured implements Tool {

    private static String matchList = "https://na.api.pvp.net/api/lol/
na/v2.2/matchlist/by-summoner/";
    private static String summoner = "https://na.api.pvp.net/api/lol/
na/v1.4/summoner/";
```

```
    private static String api_key = "?api_key=60e01f47-cbf0-4fb0-a9a5-
bf89aa642921";
    private static String prefix = "{\"matches\"";

    public static void main(String[] args) throws Exception {
        JSONObject matchListJSON;
        JSONObject summonerJSON;
        JSONObject matchJSON;
        int res = 0;
        PrintWriter writer = new PrintWriter("champion-result.txt",
"UTF-8");

        for (int i = 2; i < 45480241; i++) {
            String matchListQuery = matchList + String.valueOf(i) +
api_key;
            String summonerQuery = summoner + String.valueOf(i) +
api_key;
            URL matchListUrl = new URL(matchListQuery);
            URL summonerUrl = new URL(summonerQuery);

            try (BufferedReader matchListReader = new
BufferedReader(new InputStreamReader(matchListUrl.openStream()))) {
                for (String matchListLine; (matchListLine =
matchListReader.readLine()) != null;) {
                    if (matchListLine.startsWith(prefix)) {
                        try (BufferedReader summonerReader = new
BufferedReader(new InputStreamReader(summonerUrl.openStream()))) {
                            for (String summonerLine; (summonerLine =
summonerReader.readLine()) != null;) {

                                summonerJSON = new
JSONObject(summonerLine);
                                String summonerName =
summonerJSON.getJSONObject(String.valueOf(i)).getString("name");
                                System.out.println("Summoner id: " + i
+ ": " + summonerName);

                            }
                        }

                        matchListJSON = new JSONObject(matchListLine);
                        JSONArray matches =
matchListJSON.getJSONArray("matches");
                        for (int j = 0; j < matches.length(); j++) {
                            MatchTuple matchTuple = new MatchTuple();

                            matchJSON = (JSONObject) matches.get(j);
```

```
matchTuple.setRole(matchJSON.getString("role"));

matchTuple.setSeason(matchJSON.getString("season"));

matchTuple.setPlatformId(matchJSON.getString("platformId"));

matchTuple.setRegion(matchJSON.getString("region"));

matchTuple.setMatchId(matchJSON.getInt("matchId"));

matchTuple.setChampion(matchJSON.getInt("champion"));

matchTuple.setQueue(matchJSON.getString("queue"));

matchTuple.setLane(matchJSON.getString("lane"));

matchTuple.setTimestamp(matchJSON.getInt("timestamp"));

                            System.out.println(matchTuple);

                        }
                        res = ToolRunner.run(new Configuration(), new
MatchDataAnalyzer(), args);

System.out.println("====================================");
                    }

                }
            } catch (IOException exception) {
                continue;
            }

        }

        System.exit(res);
    }

    public static class MatchMapper extends Mapper<IntWritable,
IntWritable, IntWritable, IntWritable> {

        private final static IntWritable ONE = new IntWritable(1);

        @Override
        protected void map(IntWritable key, IntWritable value, Context
context) throws IOException, InterruptedException {
            super.map(key, value, context); //To change body of
generated methods, choose Tools | Templates.
        }
```

```
    }

    public static class MatchReducer extends Reducer<IntWritable,
IntWritable, IntWritable, IntWritable> {

        @Override
        protected void reduce(IntWritable key, Iterable<IntWritable>
values, Context context) throws IOException, InterruptedException {
            super.reduce(key, values, context); //To change body of
generated methods, choose Tools | Templates.
        }

    }

    public int run(String[] args) throws Exception {
        Configuration conf = getConf();
        Job job = new Job(conf, "MatchDataAnalyzer");
        job.setJarByClass(MatchDataAnalyzer.class);
        final File f = new
File(MatchDataAnalyzer.class.getProtectionDomain().getCodeSource().get
Location().getPath());
        //String inFiles = f.getAbsolutePath().replace("/build/
classes", "") + "/src/inFiles/ratings.dat";
        String outFiles = f.getAbsolutePath().replace("/build/
classes", "") + "/src/outFiles/MatchStat";

        //Path in = new Path(inFiles);

        /*Creating Filesystem object with the configuration*/
        FileSystem fs = FileSystem.get(conf);
        /*Check if output path (args[1])exist or not*/
        if (fs.exists(new Path(outFiles))) {
            /*If exist delete the output path*/
            fs.delete(new Path(outFiles), true);
        }

        Path out = new Path(outFiles);
        //FileInputFormat.setInputPaths(job, in);
        FileOutputFormat.setOutputPath(job, out);
        job.setMapperClass(MatchMapper.class);
        //job.setCombinerClass(MovieLensReducer.class);
        job.setReducerClass(MatchReducer.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setOutputKeyClass(MatchTuple.class);
        job.setOutputValueClass(DoubleWritable.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        return 0;
```

```
    }
}
```

- MatchTuple.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package final_project;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.WritableComparable;

/**
 *
 * @author zhaojiyuan
 */
public class MatchTuple implements WritableComparable<MatchTuple> {

    private String role;
    private String season;
    private String platformId;
    private String region;
    private int matchId;
    private int champion;
    private String queue;
    private String lane;
    private int timestamp;

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }

    public String getSeason() {
        return season;
    }

    public void setSeason(String season) {
        this.season = season;
    }

    public String getPlatformId() {
```

```
    return platformId;
}

public void setPlatformId(String platformId) {
    this.platformId = platformId;
}

public String getRegion() {
    return region;
}

public void setRegion(String region) {
    this.region = region;
}

public int getMatchId() {
    return matchId;
}

public void setMatchId(int matchId) {
    this.matchId = matchId;
}

public int getChampion() {
    return champion;
}

public void setChampion(int champion) {
    this.champion = champion;
}

public String getQueue() {
    return queue;
}

public void setQueue(String queue) {
    this.queue = queue;
}

public String getLane() {
    return lane;
}

public void setLane(String lane) {
    this.lane = lane;
}

public int getTimestamp() {
    return timestamp;
```

```
    }

    public void setTimestamp(int timestamp) {
        this.timestamp = timestamp;
    }




    @Override
    public void write(DataOutput out) throws IOException {
        out.writeChars(role);
        out.writeChars(season);
        out.writeChars(platformId);
        out.writeChars(region);
        out.writeInt(matchId);
        out.writeInt(champion);
        out.writeChars(queue);
        out.writeChars(lane);
        out.writeInt(timestamp);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        role = in.readLine();
        season = in.readLine();
        platformId = in.readLine();
        region = in.readLine();
        matchId = in.readInt();
        champion = in.readInt();
        queue = in.readLine();
        lane = in.readLine();
        timestamp = in.readInt();
    }


    @Override
    public int compareTo(MatchTuple o) {
        return (int) (this.getTimestamp()- o.getTimestamp());
    }


    @Override
    public String toString() {
        return "MatchTuple{" + "role=" + role + ", season=" + season +
", platformId=" + platformId + ", region=" + region + ", matchId=" +
matchId + ", champion=" + champion + ", queue=" + queue + ", lane=" +
lane + ", timestamp=" + timestamp + '}';
    }
```

```
}
```

- LoL_Predictor
  - Duo.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package final_project;

/**
 *
 * @author zhaojiyuan
 */
public class Duo {
    int champion1;
    int champion2;
    int result;

    public int getChampion1() {
        return champion1;
    }

    public void setChampion1(int champion1) {
        this.champion1 = champion1;
    }

    public int getChampion2() {
        return champion2;
    }

    public void setChampion2(int champion2) {
        this.champion2 = champion2;
    }

    public int getResult() {
        return result;
    }

    public void setResult(int result) {
        this.result = result;
    }


}
```

  - LoL_Predictor.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package final_project;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.net.MalformedURLException;
import java.util.ArrayList;
import java.util.Iterator;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.util.Bytes;
import static org.apache.hadoop.hbase.util.Bytes.toBytes;
import org.apache.hadoop.mapreduce.Mapper;

/**
 *
 * @author zhaojiyuan
 */
public class LoL_Predictor {

    private static String championInfo = "https://global.api.pvp.net/
api/lol/static-data/na/v1.2/champion/";
    private static String summoner = "https://na.api.pvp.net/api/lol/
na/v1.4/summoner/";
    private static String api_key = "?api_key=60e01f47-cbf0-4fb0-a9a5-
bf89aa642921";
    private static String tableName = "champion";

    public static ArrayList<Match> matchArrayList = new ArrayList<>();
```

```
    public static void main(String[] args) throws
MalformedURLException, FileNotFoundException, IOException {
        importData();
        Mahout();
        //String championSerialNumber = "1111";

        //Result championRecord = retrieveData(championSerialNumber);
        //byte[] championId =
championRecord.getValue(Bytes.toBytes("championId"),
Bytes.toBytes("Id"));
        //byte[] championKills =
championRecord.getValue(Bytes.toBytes("championInfo"),
Bytes.toBytes("kills"));
        //System.out.println(Bytes.toString(championId) + " " +
Bytes.toString(championKills));
    }

    public static void Mahout() throws FileNotFoundException,
UnsupportedEncodingException {
        PrintWriter writer = new PrintWriter("mahout_input.txt",
"UTF-8");
        int i = 1;
        matchArrayList.stream().forEach((next) -> {
            for (Match match1 : matchArrayList) {
                if (next.matchId == match1.matchId && next.teamId ==
match1.teamId && next.championId != match1.championId) {
                    //System.out.println(next.championId + "," +
match1.championId + "," + next.result);
                    writer.println(next.championId + "," +
match1.championId + "," + next.result);
                }
            }
        });
        writer.close();
    }

    public static void importData() throws IOException {
        /*
        Configuration conf = HBaseConfiguration.create();
        HBaseAdmin admin = new HBaseAdmin(conf);

        //creating table descriptor
        HTableDescriptor table = new
HTableDescriptor(toBytes(tableName));

        //creating column family descriptor
        HColumnDescriptor championInfoFamily = new
HColumnDescriptor(toBytes("championInfo"));
```

```
        HColumnDescriptor championIdFamily = new
HColumnDescriptor(toBytes("championId"));

        //adding coloumn family to HTable
        table.addFamily(championInfoFamily);
        table.addFamily(championIdFamily);

        if (admin.tableExists(tableName)) {
            admin.disableTable(tableName);
            admin.deleteTable(tableName);
        }

        admin.createTable(table);
        // Instantiating HTable class
        HTable hTable = new HTable(conf, tableName);
        System.out.println("Table created: " + hTable.getName());
        */
        JSONParser parser = new JSONParser();
        ArrayList<MatchJSON> matchJsonArrayList = new ArrayList<>();

        //PrintWriter writer = new PrintWriter("champion-result.txt",
"UTF-8");
        JSONObject team1;
        JSONObject team2;
        PrintWriter writer = new
PrintWriter("champion_mahout_input.txt", "UTF-8");


        File dir = new File("input");
        File[] files = dir.listFiles((File dir1, String name) ->
name.endsWith(".json"));
        int j = 0;

        for (File file : files) {

            System.out.println("Reading " + file.getName() +
" ......");
            try {
                Object object = parser.parse(new FileReader(file));
                JSONObject matchesJsonObject = (JSONObject) object;
                JSONArray matchJsonArray = (JSONArray)
matchesJsonObject.get("matches");
                MatchJSON matchJSON = new MatchJSON();
                int i = 1;

                for (Object o : matchJsonArray) {
                    JSONObject matchJSONObject = (JSONObject) o;
```

```
                    matchJSON.setMatchId((long)
matchJSONObject.get("matchId"));

                    matchJSON.setRegion((String)
matchJSONObject.get("region"));

                    matchJSON.setPlatformId((String)
matchJSONObject.get("platformId"));
                    matchJSON.setMatchMode((String)
matchJSONObject.get("matchMode"));
                    matchJSON.setMatchType((String)
matchJSONObject.get("matchType"));
                    matchJSON.setMatchCreation((long)
matchJSONObject.get("matchCreation"));

matchJSON.setMatchDuration(Integer.parseInt(matchJSONObject.get("match
Duration").toString()));
                    matchJSON.setQueueType((String)
matchJSONObject.get("queueType"));

matchJSON.setMapId(Integer.parseInt(matchJSONObject.get("mapId").toStr
ing()));
                    matchJSON.setSeason((String)
matchJSONObject.get("season"));
                    matchJSON.setMatchVersion((String)
matchJSONObject.get("matchVersion"));
                    matchJSON.setParticipants((JSONArray)
matchJSONObject.get("participants"));
                    matchJSON.setParticipantIdentities((JSONArray)
matchJSONObject.get("participantIdentities"));
                    matchJSON.setTeams((JSONArray)
matchJSONObject.get("teams"));
                    matchJSON.setTimeline((JSONObject)
matchJSONObject.get("timeline"));

                    matchJsonArrayList.add(matchJSON);
                    //System.out.println("Match " + i++ + " ID: " +
matchJSON.getMatchId());
                    team1 = (JSONObject) matchJSON.getTeams().get(0);
                    team2 = (JSONObject) matchJSON.getTeams().get(1);
                    for (Object object1 : matchJSON.getParticipants())
{
                        JSONObject participant = (JSONObject) object1;
                        JSONObject stats = (JSONObject)
participant.get("stats");
                        int result = ((boolean) stats.get("winner")) ?
1 : 0;
                        j++;
                        /*
```

```
                        // Instantiating Put class
                        // accepts a row name.

                        Put p = new
Put(Bytes.toBytes(String.valueOf(j)));

                        // adding values using add() method
                        // accepts column family name, qualifier/row
name ,value
                        p.add(Bytes.toBytes("championId"),
                                Bytes.toBytes("Id"),
Bytes.toBytes(participant.get("championId").toString()));

                        p.add(Bytes.toBytes("championInfo"),
                                Bytes.toBytes("kills"),
Bytes.toBytes(stats.get("kills").toString()));

                        p.add(Bytes.toBytes("championInfo"),
                                Bytes.toBytes("deaths"),
Bytes.toBytes(stats.get("deaths").toString()));

                        p.add(Bytes.toBytes("championInfo"),
                                Bytes.toBytes("assits"),
Bytes.toBytes(stats.get("assists").toString()));

                        p.add(Bytes.toBytes("championInfo"),
                                Bytes.toBytes("result"),
Bytes.toBytes(result));

                        // Saving the put Instance to the HTable.
                        hTable.put(p);
                        */
                        //writer.println(matchJSON.getMatchId() + "" +
participant.get("teamId") + "," + participant.get("championId") + ","
+ result);
                        //System.out.println(matchJSON.getMatchId() +
"," + participant.get("teamId") + "," + participant.get("championId")
+ "," + result);
                        Match match = new Match();

                        match.setMatchId(matchJSON.getMatchId());
                        match.setResult(result);

match.setChampionId(Integer.valueOf(participant.get("championId").toSt
ring()));

match.setTeamId(Integer.valueOf(participant.get("teamId").toString()))
;
                        //System.out.println(match);
```

```
                        matchArrayList.add(match);

                        //System.out.println(match.getChampionId());


                        /*
                         String championRequestString = championInfo +
String.valueOf(participant.get("championId")) + api_key;
                         URL championRequestUrl = new
URL(championRequestString);
                         try (BufferedReader in = new BufferedReader(
                         new
InputStreamReader(championRequestUrl.openStream()))) {
                         String inputLine;
                         while ((inputLine = in.readLine()) != null) {
                         //System.out.println(inputLine);
                         JSONObject championJSONObject = (JSONObject)
new JSONParser().parse(inputLine);

                         //System.out.println(result + ": " +
championJSONObject.get("name"));
                         //champion name, kills, deaths, assists,
minions killed, victory/defeated
                         //
writer.println(championJSONObject.get("name") + "," +
stats.get("kills") + "," + stats.get("deaths") + "," +
stats.get("assists") + "," + result);

                         }
                         }
                         */
                    }
                    //System.out.println("Team 1: First Dragon: " +
team1.get("firstDragon") + "; First Baron: " + team1.get("firstBaron")
+ "; Winner: " + team1.get("winner"));
                    //System.out.println("Team 2: First Dragon: " +
team2.get("firstDragon") + "; First Baron: " + team2.get("firstBaron")
+ "; Winner: " + team2.get("winner"));
                }

            } catch (Exception e) {
            }
            writer.close();
        }

        //System.out.println("Total memory used: " + (float)
Runtime.getRuntime().totalMemory() / 1024 / 1024 / 1024 + "GB");
        //writer.close();
        //hTable.close();
```

```
    }

    public static Result retrieveData(String championSerialNumber)
throws IOException{
        Configuration conf = HBaseConfiguration.create();
        HTable hTable = new HTable(conf, tableName);
        Get get = new Get(toBytes(championSerialNumber));

        Result result = hTable.get(get);

        return result;
    }

    //public static class LoLMapper extends Mapper<Object, Object,
Object, Object>
}
```

- Match.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package final_project;

/**
 *
 * @author zhaojiyuan
 */
public class Match {
    long matchId;
    int teamId;
    int championId;
    int result;

    public long getMatchId() {
        return matchId;
    }

    public void setMatchId(long matchId) {
        this.matchId = matchId;
    }

    public int getTeamId() {
        return teamId;
    }

    public void setTeamId(int teamId) {
        this.teamId = teamId;
```

```
    }

    public int getChampionId() {
        return championId;
    }

    public void setChampionId(int championId) {
        this.championId = championId;
    }

    public int getResult() {
        return result;
    }

    public void setResult(int result) {
        this.result = result;
    }


}
```

- MatchJSON.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package final_project;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;

/**
 *
 * @author zhaojiyuan
 */
public class MatchJSON {
    long matchId;
    String region;
    String platformId;
    String matchMode;
    String matchType;
    long matchCreation;
    int matchDuration;
    String queueType;
    int mapId;
    String season;
    String matchVersion;
    JSONArray participants;
```

```
JSONArray participantIdentities;
JSONObject timeline;
JSONArray teams;

public long getMatchId() {
    return matchId;
}

public void setMatchId(long matchId) {
    this.matchId = matchId;
}

public String getRegion() {
    return region;
}

public void setRegion(String region) {
    this.region = region;
}

public String getPlatformId() {
    return platformId;
}

public void setPlatformId(String platformId) {
    this.platformId = platformId;
}

public String getMatchMode() {
    return matchMode;
}

public void setMatchMode(String matchMode) {
    this.matchMode = matchMode;
}

public String getMatchType() {
    return matchType;
}

public void setMatchType(String matchType) {
    this.matchType = matchType;
}

public long getMatchCreation() {
    return matchCreation;
}

public void setMatchCreation(long matchCreation) {
```

```
    this.matchCreation = matchCreation;
}

public int getMatchDuration() {
    return matchDuration;
}

public void setMatchDuration(int matchDuration) {
    this.matchDuration = matchDuration;
}

public String getQueueType() {
    return queueType;
}

public void setQueueType(String queueType) {
    this.queueType = queueType;
}

public int getMapId() {
    return mapId;
}

public void setMapId(int mapId) {
    this.mapId = mapId;
}

public String getSeason() {
    return season;
}

public void setSeason(String season) {
    this.season = season;
}

public String getMatchVersion() {
    return matchVersion;
}

public void setMatchVersion(String matchVersion) {
    this.matchVersion = matchVersion;
}

public JSONArray getParticipants() {
    return participants;
}

public void setParticipants(JSONArray participants) {
    this.participants = participants;
```

```
    }

    public JSONArray getParticipantIdentities() {
        return participantIdentities;
    }

    public void setParticipantIdentities(JSONArray
participantIdentities) {
        this.participantIdentities = participantIdentities;
    }

    public JSONObject getTimeline() {
        return timeline;
    }

    public void setTimeline(JSONObject timeline) {
        this.timeline = timeline;
    }

    public JSONArray getTeams() {
        return teams;
    }

    public void setTeams(JSONArray teams) {
        this.teams = teams;
    }

}
```

- LoLDataFetcher
  - MatchDataFetcher.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package final_project;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.URL;
import java.util.concurrent.TimeUnit;


/**
 *
 * @author zhaojiyuan
```

```
 */
public class MatchDataFetcher {

    private static String matchList = "https://na.api.pvp.net/api/lol/
na/v2.2/match/";
    private static String api_key = "?api_key=60e01f47-cbf0-4fb0-a9a5-
bf89aa642921";
    private static String prefix = "{\"matchId\"";

    public static void main(String[] args) throws Exception {

        PrintWriter writer = new PrintWriter("match-1m.json",
"UTF-8");

        for (int i = 1807845889; i < (1807845889+1000000); i++) {
            String matchListQuery = matchList + String.valueOf(i) +
api_key;
            URL matchListUrl = new URL(matchListQuery);

            try (BufferedReader matchListReader = new
BufferedReader(new InputStreamReader(matchListUrl.openStream()))) {
                for (String matchListLine; (matchListLine =
matchListReader.readLine()) != null;) {
                    if (matchListLine.startsWith(prefix) &&
matchListLine.contains("RANKED_SOLO_5x5")) {
                        System.out.println(matchListLine);
                        writer.println(matchListLine);
                        TimeUnit.MILLISECONDS.sleep(1205);
                    }
                }

            } catch (Exception e) {
                continue;
            }

        }

        writer.close();

    }
}
```

- MatchTuple.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package final_project;
```

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.WritableComparable;

/**
 *
 * @author zhaojiyuan
 */
public class MatchTuple implements WritableComparable<MatchTuple> {

    private String role;
    private String season;
    private String platformId;
    private String region;
    private int matchId;
    private int champion;
    private String queue;
    private String lane;
    private int timestamp;

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }

    public String getSeason() {
        return season;
    }

    public void setSeason(String season) {
        this.season = season;
    }

    public String getPlatformId() {
        return platformId;
    }

    public void setPlatformId(String platformId) {
        this.platformId = platformId;
    }

    public String getRegion() {
        return region;
    }
```

```
public void setRegion(String region) {
    this.region = region;
}

public int getMatchId() {
    return matchId;
}

public void setMatchId(int matchId) {
    this.matchId = matchId;
}

public int getChampion() {
    return champion;
}

public void setChampion(int champion) {
    this.champion = champion;
}

public String getQueue() {
    return queue;
}

public void setQueue(String queue) {
    this.queue = queue;
}

public String getLane() {
    return lane;
}

public void setLane(String lane) {
    this.lane = lane;
}

public int getTimestamp() {
    return timestamp;
}

public void setTimestamp(int timestamp) {
    this.timestamp = timestamp;
}



@Override
```

```
    public void write(DataOutput out) throws IOException {
        out.writeChars(role);
        out.writeChars(season);
        out.writeChars(platformId);
        out.writeChars(region);
        out.writeInt(matchId);
        out.writeInt(champion);
        out.writeChars(queue);
        out.writeChars(lane);
        out.writeInt(timestamp);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        role = in.readLine();
        season = in.readLine();
        platformId = in.readLine();
        region = in.readLine();
        matchId = in.readInt();
        champion = in.readInt();
        queue = in.readLine();
        lane = in.readLine();
        timestamp = in.readInt();
    }


    @Override
    public int compareTo(MatchTuple o) {
        return (int) (this.getTimestamp()- o.getTimestamp());
    }


    @Override
    public String toString() {
        return "MatchTuple{" + "role=" + role + ", season=" + season +
", platformId=" + platformId + ", region=" + region + ", matchId=" +
matchId + ", champion=" + champion + ", queue=" + queue + ", lane=" +
lane + ", timestamp=" + timestamp + '}';
    }
}
```

- LoL_Mahout
  - Champion_Predictor.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
```

```
package recommender;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.URL;
import java.util.List;
import org.apache.mahout.cf.taste.common.TasteException;
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import
org.apache.mahout.cf.taste.impl.neighborhood.ThresholdUserNeighborhood
;
import
org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommende
r;
import org.apache.mahout.cf.taste.impl.similarity.*;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;
import org.apache.mahout.cf.taste.recommender.RecommendedItem;
import org.apache.mahout.cf.taste.recommender.UserBasedRecommender;
import org.apache.mahout.cf.taste.similarity.UserSimilarity;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;

/**
 *
 * @author zhaojiyuan
 */
public class Champion_Predictor {

    private static String championInfo = "https://global.api.pvp.net/
api/lol/static-data/na/v1.2/champion/";
    private static String api_key = "?api_key=60e01f47-cbf0-4fb0-a9a5-
bf89aa642921";

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     * @throws org.apache.mahout.cf.taste.common.TasteException
     */
    public static void main(String[] args) throws Exception {

        PrintWriter writer = new PrintWriter("mahout_output.csv",
"UTF-8");

        // TODO code application logic here
        //step 1. create the model
```

```
        DataModel model = new FileDataModel(new
File("mahout_input.csv"));

        //step 2.find users with similar tastes
        UserSimilarity similarity = new
TanimotoCoefficientSimilarity(model);

        //step 3.find user neignborhood
        UserNeighborhood neighborhood = new
ThresholdUserNeighborhood(0.1, similarity, model);

        //step 4.create the recommender engine
        UserBasedRecommender recommender = new
GenericUserBasedRecommender(model, neighborhood, similarity);
        String summonerName1;
        String summonerName2;
        //step 5.ask the recommender to give recommendations
        for (int i = 1; i < 433; i++) {
            try {
                List<RecommendedItem> recommendations =
recommender.recommend(i, 5, true);
                for (RecommendedItem recommendation : recommendations)
{

                    String championRequestString = championInfo +
String.valueOf(i) + api_key;
                    URL championRequestUrl = new
URL(championRequestString);

                    String championRequestString1 = championInfo +
String.valueOf(recommendation.getItemID()) + api_key;
                    URL championRequestUrl1 = new
URL(championRequestString1);

                    try (BufferedReader in = new BufferedReader(
                            new
InputStreamReader(championRequestUrl.openStream()))) {
                        String inputLine;
                        while ((inputLine = in.readLine()) != null) {
                            JSONObject championJSONObject =
(JSONObject) new JSONParser().parse(inputLine);

                            try (BufferedReader in1 = new
BufferedReader(
                                    new
InputStreamReader(championRequestUrl1.openStream()))) {
                                String inputLine1;
                                while ((inputLine1 = in1.readLine()) !
= null) {
```

```
                                    JSONObject championJSONObject1 =
(JSONObject) new JSONParser().parse(inputLine1);

                                    System.out.println("Champion " +
championJSONObject.get("name") + " might work well with Champion: " +
championJSONObject1.get("name") + " (predicted win rate: " +
String.format("%.4f", recommendation.getValue()) + ")");

writer.println(championJSONObject.get("name") + "," +
championJSONObject1.get("name") + "," + String.format("%.4f",
recommendation.getValue()));

                                }
                            }
                        }
                    }
                }
            } catch (Exception e) {
            }

        }
        writer.close();
    }
}
```

- LoL_Predictor.pig

```
--1 declare a variable for dataset path
%declare DATASETPATH '/Users/zhaojiyuan/Dropbox/NEU/INFO7374/
LoL_Predictor';

--2 load ratings file
champion = LOAD '$DATASETPATH/result-sample.txt' using PigStorage
(',')
AS (champion_name:chararray, kills:int, deaths:int, assists:int,
result:chararray);

champion = FOREACH champion GENERATE champion_name, kills, deaths,
assists, (result == 'v' ? 1 : 0) AS victory, (result == 'd' ? 1 : 0)
AS defeated, result;

--4 group the ratings by movie_id
groupdChampion = GROUP champion BY champion_name;

final_data = FOREACH groupdChampion GENERATE group AS champion_name,
ROUND(AVG(champion.kills)*100.00)/100.00 AS avgKills,
ROUND(AVG(champion.deaths)*100.00)/100.00 AS avgDeaths,
ROUND(AVG(champion.assists)*100.00)/100.00 AS avgAssits,
SUM(champion.victory) AS sumVictory, SUM(champion.defeated) AS
sumDefeated, ROUND(SUM(champion.victory)*10000.00/
```

```
COUNT(champion.result))/100.00 AS rateVictory,
ROUND(SUM(champion.defeated)*10000.00/COUNT(champion.result))/100.00
AS rateDefeated;

final_data = ORDER final_data BY rateVictory DESC;

--STORE final_data INTO 'mr_result';
STORE final_data INTO 'mr_result' using PigStorage(',');
```