Social Graph Mining in Yelp dataset

CS 6350.002 - Big Data Management and Analytics

Submitted by:
Dipti Desai (dsd140330)
Himanshu Soni (hss140330)
Shobhit Agarwal (sxa138130)
Sunish Seth (sss140830)

University of Texas at Dallas
Dallas, Texas
December-2015

1. Problem Definition:

The problem statement says “Can you figure out who the trend setters are and who found the best waffle joint before waffles were cool? How much influence does my social circle have on my business choices and my ratings? Social graph mining is a kind of social network analysis where the main purpose of analyzing the data is towards finding out which user is influencing your decisions regarding choosing a business service or how many users are influenced by you in choosing the same. To give an overview of what our project will be doing in terms of social graph mining, we can say that we need to check the user and their friend list (get the friend data from yelp user object) and find all the places that a person’s social circle is interested in. Did they rate the business almost similarly or how much was the difference between their ratings? We need to do a particular set of calculations (form a similarity matrix) between the friends to find similarity between users. If a pair of users are similar, then we can say that the user’s choice has been influenced by a member of his social circle

2. Description of Input Data:

Since we are following all the standards of Yelp dataset challenge and we are working on one of the topic provided by Yelp, we are going to use Yelp dataset for our project. The yelp dataset gives us 5 files to work on:

- yelp_academic_dataset_business.json – Business data
- yelp_academic_dataset_checkin.json – Check-in data
- yelp_academic_dataset_review.json – Review data
- yelp_academic_dataset_tip.json – Tip data
- yelp_academic_dataset_user.json – User data

Our project Social Graph mining will concentrate on yelp_academic_dataset_user.json – User data and yelp_academic_dataset_review.json User and Business data objects representation is given below:

user

```
{
    'type': 'user',
    'user_id': (encrypted user id),
    'name': (first name),
    'review_count': (review count),
    'average_stars': (floating point average, like 4.31),
    'votes': {(vote type): (count)},
    'friends': [(friend user_ids)],
    'elite': [(years_elite)],
    'yelping_since': (date, formatted like '2012-03'),
    'compliments': {
        (compliment_type): (num_compliments_of_this_type),
        ...
    },
    'fans': (num_fans),
}
```

review

```
{
    'type': 'review',
    'business_id': (encrypted business id),
    'user_id': (encrypted user id),
    'stars': (star rating, rounded to half-stars),
    'text': (review text),
    'date': (date, formatted like '2012-03-14'),
    'votes': {(vote type): (count)},
}
```

3. Big Data Strategy:

Before applying any machine learning strategy, it is important to feed the data to the algorithm in specific format so that algorithm can be used efficiently not worrying about the data format. We apply Big Data strategies in two ways. One is to parse data using PIG, HADOOP and then apply machine learning algorithm using Spark machine learning library.

The Json data is loaded into PIG and queried over user and review table to get join table containing user and its review for a business. MapReduce algorithm is used to extract a field of friend list from the huge data of *user.Json*. It is a mapper only function which extracts a user and friend list and emits in term of key value pair respectively. Another MapReduce approach is developed to get data for input user rather than running algorithm over all available data which might slow down the entire process and as the project concentrate on giving output of

similar/influential user to the given User input. Second MapReduce is developed based on Stripe approach which uses associative array to store business –rating as value associated with userID as key

We plan to implement Collaborative filtering using spark. We do not use a predefined collaborative filtering algorithm provided by spark open source. **Reason: Our similarity calculation uses 3 different similarity metrics as follows:**

a) Euclidean Distance
b) Naïve Bayes probability measure
c) Collaborative weight calculation

We then use a weighted aggregation method to calculate our similarity output. All this cannot be configured in the collaborative filtering code given by Spark. So we use Collaborative filtering code specified by Spark as a template and create our code. We use this similarity metrics obtained above to predict a rating.

Why Spark?

Apache Spark is a powerful processing engine providing high speed and ease of use. MapReduce, requires a lot of expensive disk reads and writes for iterative and complex jobs, spark performs these jobs in-memory and reduce the disk I/O overhead. It can process petabytes of data very quickly by partitioning the collection of objects(RDD) across the nodes which can be rebuilt, if a partition is lost. Spark API provides a variety of operations on these distributed data-sets like map, reduce, filter, etc. Along with this, a lot of features like SQL queries, streaming data, machine learning (Mlib) and graph processing (GraphX) are also included in Spark API.

We first randomly remove a rating for a business Id. This later is being predicted and the compared with our expected rating to check accuracy. We use RMSE (Root mean squared Error) to calculate accuracy.

We then pass the remaining data to calculate user similarity. We use three different formulas to calculate similarity which are explained in detail in the section 5

3.1. Data Extraction and Parsing:

Data is available from Yelp website, there is no need to extract is using any API but data needs to be preprocessed before it is used by Machine Learning algorithm. Preprocessing is done using PIG and MapReduce.

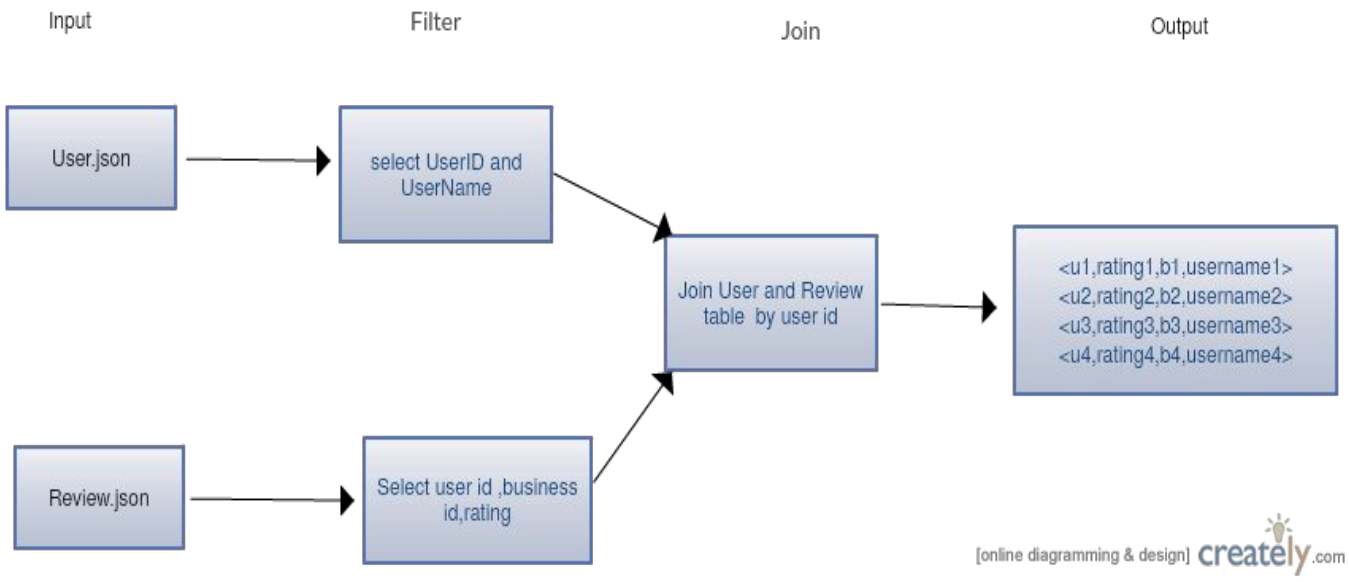
Before applying MapReduce algorithm to get user similarity, it is important to produce associative array with Userid with their corresponding business id and rating values. Yelp provide us data for User.json, Review. json which can be used to get the output of this form.

Preprocessing:

Part 1: Using PIG

To apply Join, We are using PIG as PIG provides optimal join operations. User table is comparatively smaller table and can be replicated over all the nodes.

Below is the data flow diagram for PIG Join operation:



One of the major challenge that was faced is due to json data format. There are fields in the data which has multivalued attributes. External Jar files are used to process the JSON data and load it into PIG. List of Jar files are given below:

- json-simple-1.1.jar;
- elephant-bird-pig-4.1.jar
- elephant-bird-core-4.1.jar
- elephant-bird-hadoop-compat-4.1.jar

These external Libraries make sure data is stored in proper format and can be queried successfully. We do not need multivalued attribute in output but there are multiple errors which can be bypassed with these external libraries. Pig join uses Replicated join which is mapper side join and user table is replicated in all nodes.
Below is the PIG script:

```
Register hdfs://cshadoop1/dsd140330/Yelp/j cson-simple-1.1.jar;
Register hdfs://cshadoop1/dsd140330/Yelp/elephant-bird-pig-4.1.jar
Register hdfs://cshadoop1/dsd140330/Yelp/elephant-bird-core-4.1.jar
Register hdfs://cshadoop1/dsd140330/Yelp/elephant-bird-hadoop-compat-4.1.jar

SET elephantbird.jsonloader.nestedLoad 'true';
User = Load 'user_small.json' using com.twitter.elephantbird.pig.load.JsonLoader()as
json:map[];
UserTable = FOREACH User GENERATE $0#'user_id'AS user_id:chararray, $0#'name' AS
name:chararray;
Review = Load 'small_review.json' using com.twitter.elephantbird.pig.load.JsonLoader()as
json:map[];
ReviewTable = FOREACH Review GENERATE $0#'user_id' AS user_id:chararray , $0#'stars' AS
stars:int , $0#'business_id' AS business_id:chararray;

joinTable = join ReviewTable by user_id,UserTable by user_id using 'replicated';
STORE joinTable into 'joinTable1';
```

Output of the PIG Script:

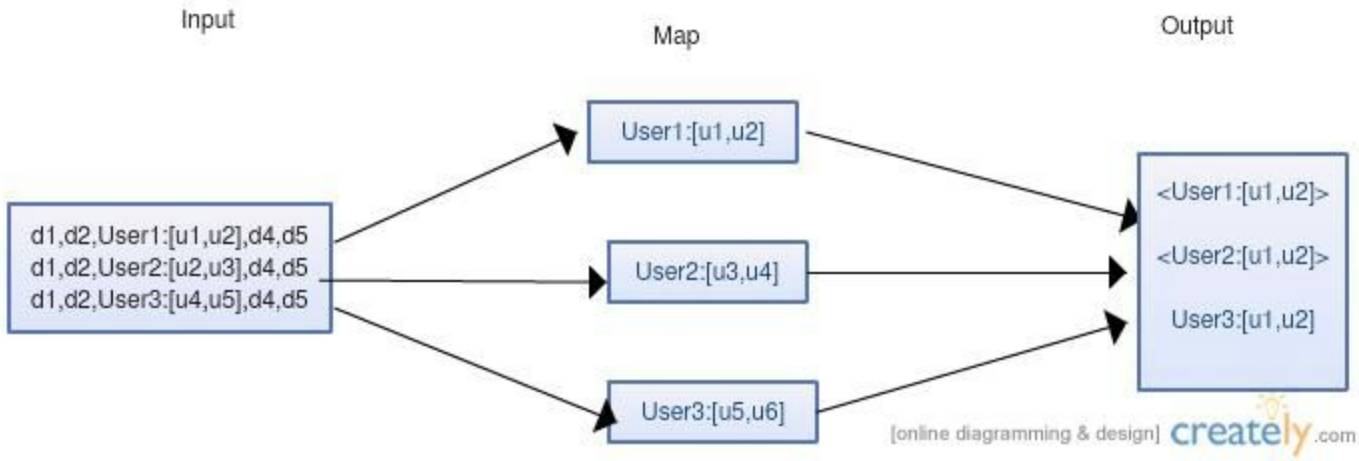
```
Xqd0DzHaiyRqVH3WRG7hzg  5       vcNAWiLM4dR7D2nwwJ7nCA  Xqd0DzHaiyRqVH3WRG7hzg  Lily
H1kH6QZV7Le4zqTRNxoZow  2       vcNAWiLM4dR7D2nwwJ7nCA  H1kH6QZV7Le4zqTRNxoZow  Daniel
zvJCcrpm2yOZrxKffwGQLA  4       vcNAWiLM4dR7D2nwwJ7nCA  zvJCcrpm2yOZrxKffwGQLA  Bill
KBLW4wJA_fwoWmMhiHRVOA  4       vcNAWiLM4dR7D2nwwJ7nCA  KBLW4wJA_fwoWmMhiHRVOA  Sweet
zvJCcrpm2yOZrxKffwGQLA  4       vcNAWiLM4dR7D2nwwJ7nCA  zvJCcrpm2yOZrxKffwGQLA  Bill
Qrs3EICADUKNFoUq2iHStA  1       vcNAWiLM4dR7D2nwwJ7nCA  Qrs3EICADUKNFoUq2iHStA  Carole
jE5xVugujSaskAoh2DRx3Q  5       vcNAWiLM4dR7D2nwwJ7nCA  jE5xVugujSaskAoh2DRx3Q  S
QnhQ8G51XbUpVEyWY2Km-A  5       vcNAWiLM4dR7D2nwwJ7nCA  QnhQ8G51XbUpVEyWY2Km-A  David
tAB7GJpUuaKF4W-3P0d95A  1       vcNAWiLM4dR7D2nwwJ7nCA  tAB7GJpUuaKF4W-3P0d95A  Kristi
GP-h9colXgkT79BW7aDJeg  5       vcNAWiLM4dR7D2nwwJ7nCA  GP-h9colXgkT79BW7aDJeg  Shaq
uK8tzraOp4M5u3uYrqIBXg  5       UsFtqoBl7naz8AVUBZMjQQ  uK8tzraOp4M5u3uYrqIBXg  Robin
I_47G-R2_egp7ME5u_ltew  3       UsFtqoBl7naz8AVUBZMjQQ  I_47G-R2_egp7ME5u_ltew  Kevin
PP_xoMSYlGr2pb67BbqBdA  2       UsFtqoBl7naz8AVUBZMjQQ  PP_xoMSYlGr2pb67BbqBdA  Mike
JPPhyFE-UE453zA6K0TVgw  4       UsFtqoBl7naz8AVUBZMjQQ  JPPhyFE-UE453zA6K0TVgw  Molly
fhNxoMwwTipzjO8A9LFe8Q  3       cE27W9VPgO88Qxe4ol6y_g  fhNxoMwwTipzjO8A9LFe8Q  Kavi
-6rEfobYjMxpUWLNxszaxQ  1       cE27W9VPgO88Qxe4ol6y_g  -6rEfobYjMxpUWLNxszaxQ  Andrew
KZuaJtFindQM9x2ZoMBxcQ  1       cE27W9VPgO88Qxe4ol6y_g  KZuaJtFindQM9x2ZoMBxcQ  Anonymous
H9E5VejGEsRhwcbOMFknmQ  4       cE27W9VPgO88Qxe4ol6y_g  H9E5VejGEsRhwcbOMFknmQ  Pierce
ljwgUJowB69klaR8Au-H7g  4       cE27W9VPgO88Qxe4ol6y_g  ljwgUJowB69klaR8Au-H7g  Brian
JbAeIYc89Sk8SWmrBCJs9g  5       HZdLhv6COCleJMo7nPl-RA  JbAeIYc89Sk8SWmrBCJs9g  Leticia
l_szjd-ken3ma6oHDkTYXg  2       HZdLhv6COCleJMo7nPl-RA  l_szjd-ken3ma6oHDkTYXg  Casey
zo_soThZw8eVglPbCRNC9A  4       HZdLhv6COCleJMo7nPl-RA  zo_soThZw8eVglPbCRNC9A  March
```

(Fourth column is again the user id, it is similar to column 1, It can be ignored in the next Map Reduce Job)**

Part 2: Using MapReduce

Similarity will be calculated for particular user based on its Friend list therefore part 2 will concentrate on MapReduce Job to extract user list for each user and save them into a file named user list. This File can be used for *Part 3* MapReduce process which gives user list along with

business and rating key-value pair in associative array. Current MapReduce is MAP only functionality, it acts like filtering. One of the major concern here is again the format of data which is Json with multivalued attribute values and In this case we need to use these multivalued attribute e.g. Friend List.



Map Reduce Code for Mapper:

```
public void map(LongWritable key, Text value, Context context)
{
    String UserID=" ";
    JSONArray UserList=null;
    String line = value.toString();
    String[] tuple = line.split("\\n");

    try{
        for(int i=0;i<tuple.length; i++){
            JSONObject obj = new JSONObject(tuple[i]);
            UserID = obj.getString("user_id");
            //UserList = obj.getString("friends");
            UserList =  obj.getJSONArray("friends");
           }
        context.write(new Text(UserID), new Text(UserList.toString()));
    }catch(Exception e){System.out.println(e.getMessage());}


}
```

Output of the Hadoop Job 1 key value pair : **<USERID ,[Friend List]>**

Key Value Pairs for Mapper: (Map only function)

Input Key - Value Pair	<Offset Address,User Record>
Output Key - Value pair	<User id , [User Friend List]>

Single User Record is shown below which will act as a value for mapper:

{"yelping_since": "2004-10", "votes": {"funny": 1, "useful": 11, "cool": 5}, "review_count": 11, "name": "Ken", "**user_id": "fHtTaujcyKvXglE33Z5yIw**", "**friends": ["18kPq7GPye-YQ3LyKyAZPw", "rpOyqD_893cqmDAtJLbdog", "i63u3SdbrLsP4FxiSKP0Zw", "uKgbjPhcrS2pi9hS39Az6A"],** "fans": 2, "average_stars": 4.6399999999999997, "type": "user", "compliments": {"cute": 2}, "elite": []}

Output key/value pair :

<fHtTaujcyKvXglE33Z5yIw, ["18kPq7GPye-YQ3LyKyAZPw", "rpOyqD_893cqmDAtJLbdog", "i63u3SdbrLsP4FxiSKP0Zw", "uKgbjPhcrS2pi9hS39Az6A"]>

Part 3: Using MapReduce

Input of this method is output of files that are generated at Part 1 and Part 2. Join Table and User list that is obtained from last part is given as input. User list is used in setup part and Hash Set is created to filter the required data in Map stage. Along with these files, USERID is also provided in the input .User id is unique used id of a person for whom similarities/influence is calculated. Code for the setup stage is as follow:

```
private Set<String> Users = new HashSet<String>();

//initial set up to fill the hash set with user list
protected void setup(Context context) throws IOException, InterruptedException {
    Configuration connection = context.getConfiguration();
    try {
        FileSystem file = FileSystem.get(new Configuration());
        //initialize buffer reader to read the file from path given in the arguments
        //arg[2] userlist file path
        BufferedReader r = new BufferedReader(new InputStreamReader(file.open(new Path(connection.get("filepath")))));
        String user = connection.get("UserId");
        Users.add(user);
        String line=r.readLine();
         String[] temp = line.split("\\s+");
         if(temp[0].equals(user))
         {
            String friendList = temp[1];
            StringTokenizer st = new StringTokenizer(friendList, ",\"");

            while (st.hasMoreTokens()) {
                String temp3 = st.nextToken();
                if (!temp3.equals("[") && !temp3.equals("]")) {
                Users.add(temp3); // add users in to hash set
                }}
```

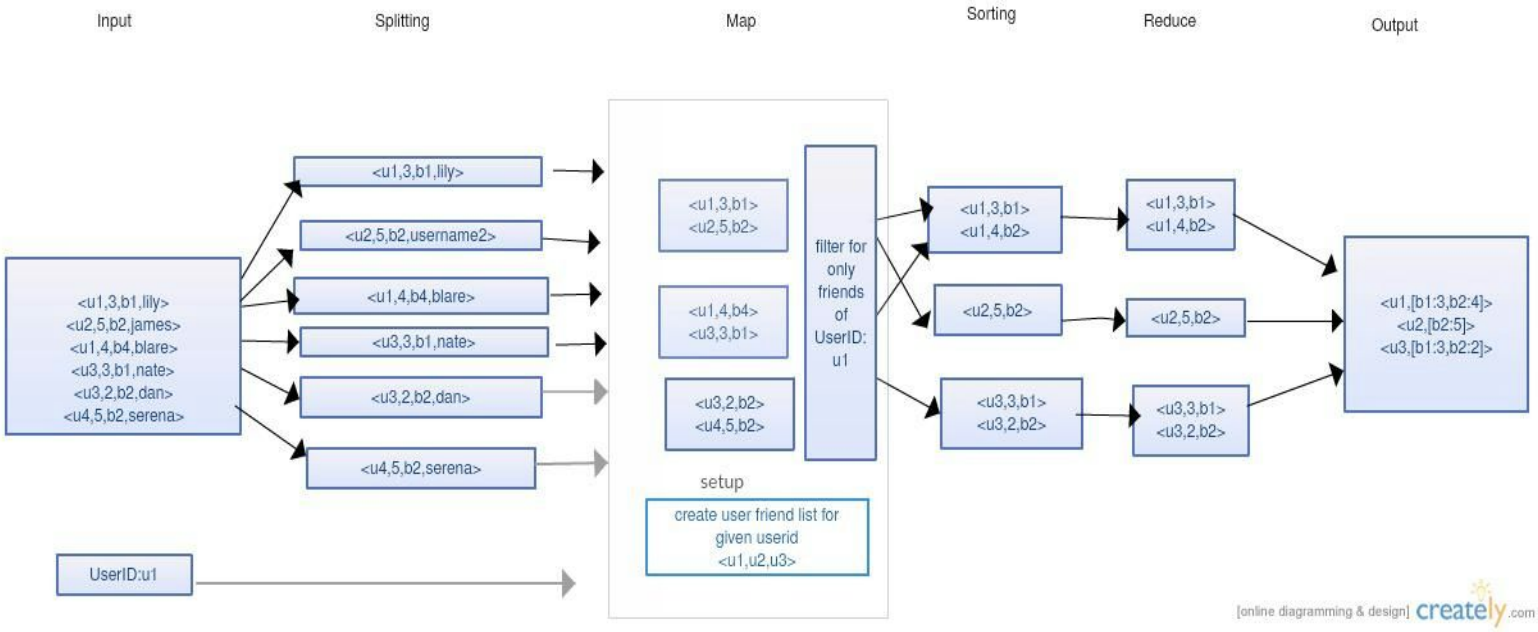
Map stage will emit records of only those users that are available in friend list of given input user (i.e. person for whose similarity/influence) that needs to be calculated. Record contains User ID as key and {BusinessID: Rating} as value. We make use of Stripe approach. Output of this phase : <UserID ,[BusinessID:Rating]>

```
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    String[]line = value.toString().split("\\s+");
    String user= line[0].trim();
    String rating = line[1].trim();
    String Bid = line[2].trim();

    if(Users.size()!=0 && Users.contains(user))
    {
        MapWritable mapWritable = new MapWritable();
        String keyUser = user;

        mapWritable.put(new Text(Bid),new Text(rating));
        context.write(new Text(keyUser),mapWritable);
    }
}
```

Hadoop Framework will make sure to combine all similar keys and provide it to Reducer in sorted order. Reducer will combine all HashMaps of business ids and ratings related to one user and emit the collective MAP output. Reducer code is shown below:

```
public static class UserBusinessRatingReducer extends Reducer<Text, MapWritable, Text, Text> {

    public void reduce(Text key, Iterable<MapWritable> values, Context context) throws IOException, InterruptedExcepti

        HashMap<String,String> Resultmap = new HashMap<String,String>();
        // value is list if HAshmap. To create new result value hashmap,
        //we need to traverse list of hashmaps and save it into result hashmap

        for(MapWritable myMap : values){
            for (Entry<Writable, Writable> ValueKey: myMap.entrySet()) {

                String bid = ValueKey.getKey().toString();
                String rating = ValueKey.getValue().toString();

                Resultmap.put(bid,rating);}     }

        Text Resultvalue = new Text();
        String tempval = "";
        for (Map.Entry<String,String> e : Resultmap.entrySet()) {
            if(null != e.getKey() && null != e.getValue()){
                tempval+= "{"+e.getKey() + ", ::" + e.getValue() + "},";    }
        }
        if(tempval.length() > 0)
            tempval = tempval.substring(0, tempval.length()-1);

        Resultvalue.set(tempval);
        context.write(key,Resultvalue);
    }
}
```

Entire Data flow of MapReduce is shown in below diagram:



Output of the MapReduce Job 2 key value pair : **<UserID , {BusinessID:Rating}>**

Key Value Pair for Hadoop part 2

Mapper	
Input Key-Value Pair	<Offset Address ,Join Table Record> *Join table obtained from Pig

Output Key-Value pair	<User id , [Business Id :Rating]>

Reducer	
Input Key-Value Pair	<User id , [Business Id :Rating]>
Output Key-Value pair	<User id , {[Business ID: Rating], {[Business ID: Rating].....}>

Part 4: Analysis Using Spark

The following is an example of the table which will be formed when a user table is joined with business and review/star tables. Item here represents business and each value is the rating that a particular user gave to a particular business.

Calculating Result 1:

	Item 101	Item 102	Item 103	Distance	Similarity to user 1
User 1	5.0	3.0	2.5	0.000	1.000
User 2	2.0	2.5	5.0	3.937	0.203
User 3	2.5	-	-	2.500	0.286
User 4	5.0	-	3.0	0.500	0.667
User 5	4.0	3.0	2.0	1.118	0.472

Now, to calculate the similarity, we can use Euclidean distance and then normalize the calculation using 1/(1+distance).

Calculating Result 2:

User 1 with User 2 will be (3/3 = 1)
User 1 with User 3 will be (1/3 = 0.333)
User 1 with User 4 will be (2/3 = 0.666)
User 1 with User 5 will be (3/3 = 1)

Calculating Result 3:

$$w(a,i) = \frac{\sum_j (v_{a,j} - \overline{v}_a)(v_{i,j} - \overline{v}_i)}{\sqrt{\sum_j (v_{a,j} - \overline{v}_a)^2 \sum_j (v_{i,j} - \overline{v}_i)^2}}$$

Example to calculate the similarity matrix:
So, to calculate the similarity matrix we need the following calculations:
So now we take 30%(result1) + 10%(result2) + 10%(result3) = similarity matrix

Part 5: Validation

Assume ITEM 104 is rated by the user but randomly chose to be the business for which we want to predict the rating.
Let's assume, that the user1 rated ITEM 104 as 4.2.
Once we know the similarity between user 1 and all the users we can find the rating for ITEM 104 for user 1 and check our accuracy to be near to 4.2

To find calculated rating using following formula:

$$p_{a,j} = \overline{v}_a + \kappa \sum_{i=1}^{n} w(a,i)(v_{i,j} - \overline{v}_i)$$

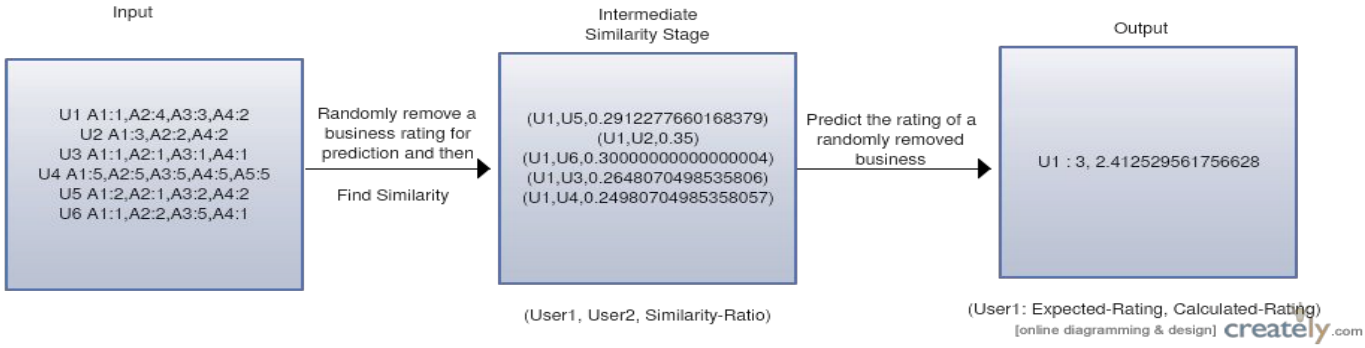
If it is accurate enough, then we can say that, user 1 is influenced by his social circle i.e. his friends.

Input:
U1 A1:1,A2:4,A3:3,A4:2
U2 A1:3,A2:2,A4:2
U3 A1:1,A2:1,A3:1,A4:1
U4 A1:5,A2:5,A3:5,A4:5,A5:5
U5 A1:2,A2:1,A3:2,A4:2

U6 A1:1,A2:2,A3:5,A4:1

Similarity Matrix:
(U1,U5,0.2912277660168379)
(U1,U2,0.35)
(U1,U6,0.30000000000000004)
(U1,U3,0.2648070498535806)
(U1,U4,0.24980704985358057)

Output:
U1 : 3, 2.412529561756628
RMSE : 0.588



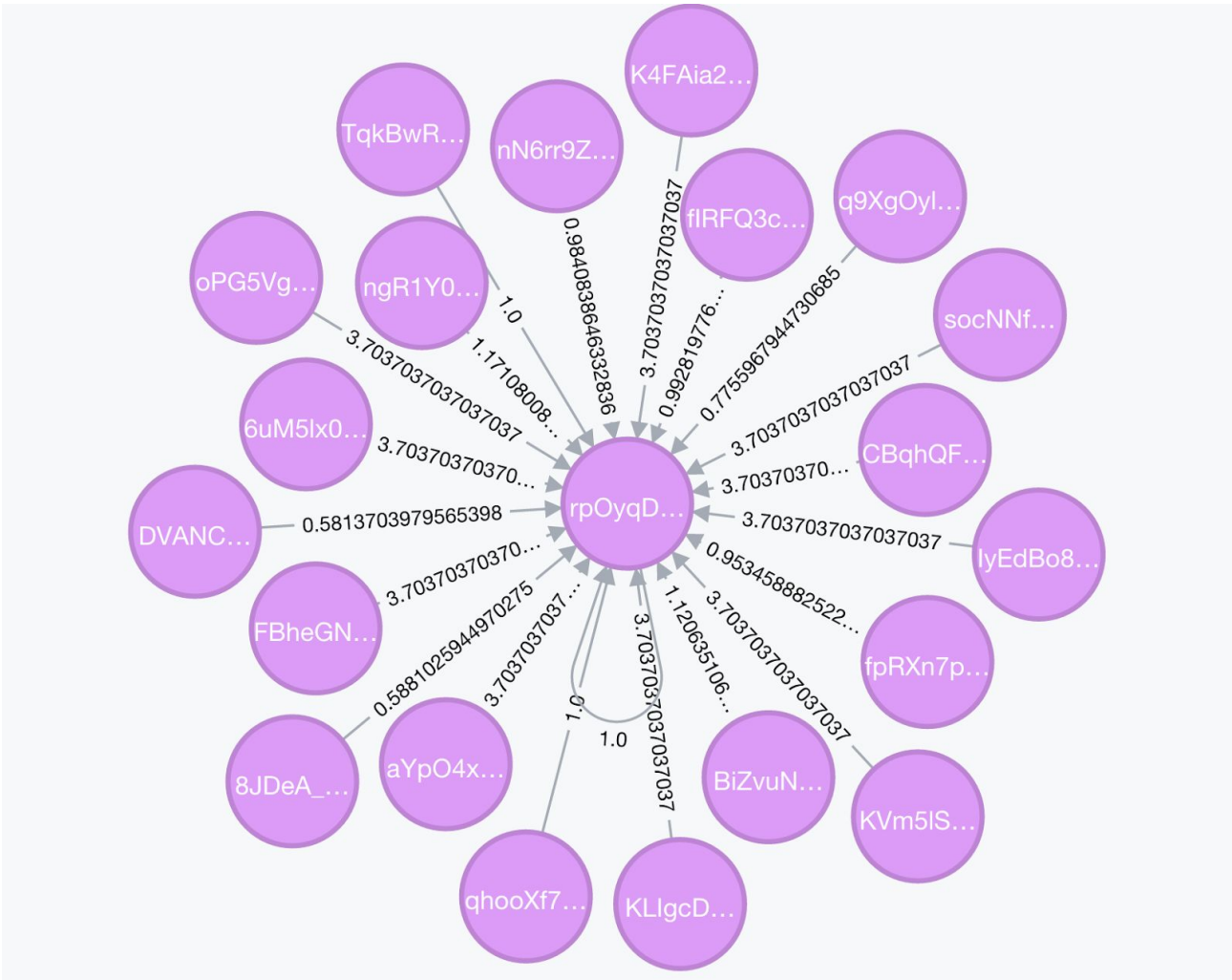
Analysis of Result

The output of the Spark job which generates a similarity measure between the queried user and his friends also performs validation on the data using a RMSE measure as explained above. Now we can leverage the results to identify key friends of users. For Example, we want to concentrate on the '*top 10 friends of the user*' from whom he is mostly influenced. We can execute such queries in the db inside neo4j.

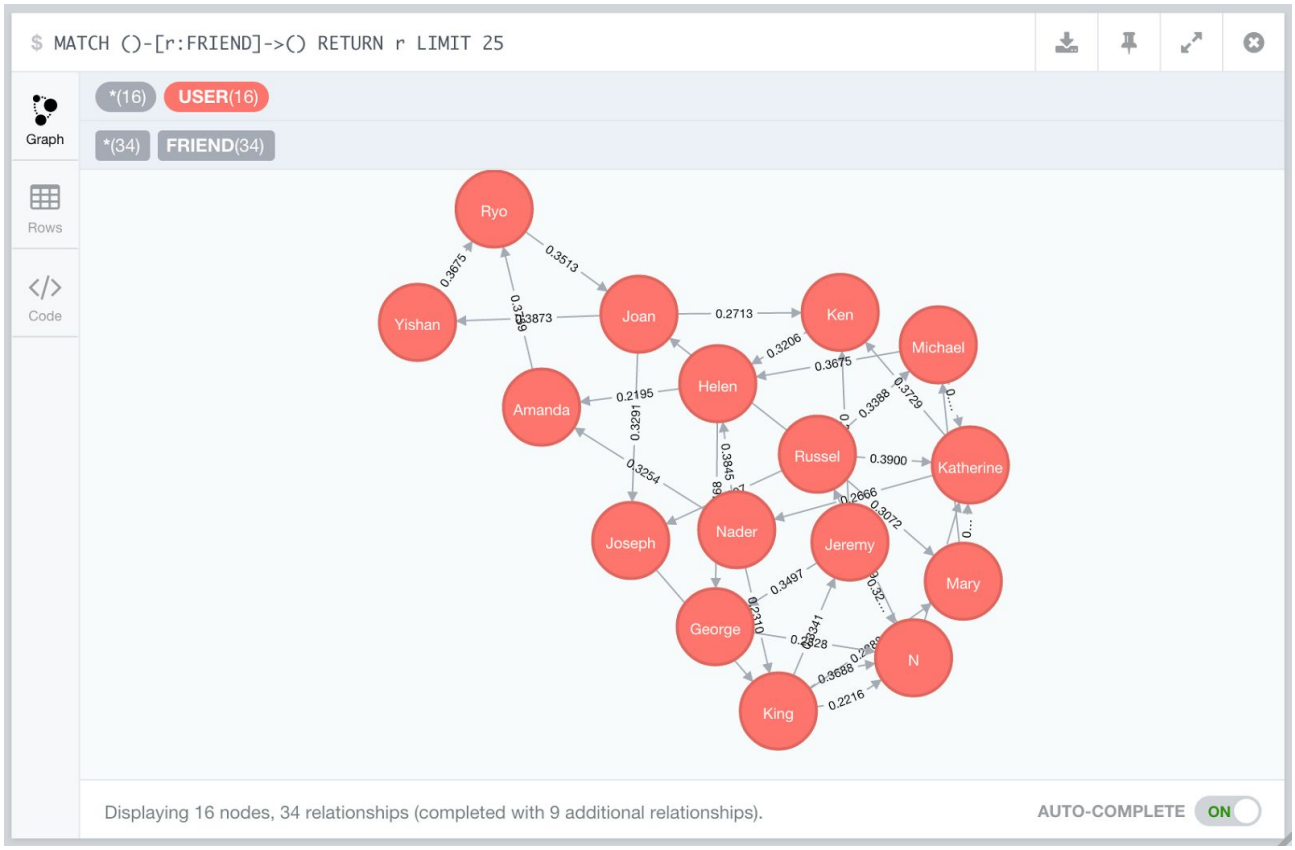
The following query will perform our operation :

```
MATCH ()-[r:FRIEND]->() WHERE r.similarity > '0.35' RETURN r
```

The output is as below :



Performing a regression on all the user to calculate the similarity between him and his friends, we will have a final output with the similarity measure for all the friends. Dumping this new result inside neo4j and updating the existing dataset will allow us to query and utilize the yelp data. One such example output is as below.



Handling bad or missing data:

Our data is bound to have missing data. We do not expect that all the users to rate all the business. Collaborative filtering algorithm is designed to ignore them and thus handling missing data. Bad data will impact the accuracy a lot. There is no way to handle bad data or fake data using our algorithm.

Another issue faced by us was the irregularity amongst the friends list of a user. To perform analysis we expect that a user should have a very dense network of friends(i.e. should have a big friends list). But yelp users have a very limited or in most cases very few friends. Such sparseness in the data can be bad for our machine learning task.

Role:

Data Cleaning using PIG – Dipti and Himanshu
Map Reduce Part 1- Dipti and Shobhit
Map Reduce Part 2- Dipti and Shobhit
Spark Setup : Sunish & Himanshu
Spark Implementation – Sunish
Representation of Graph using Neo4j - Himanshu
Shell Script for Batch - Himanshu

Conclusion:

Project can be improved by taking into consideration the reviews to verify the ratings. We can also consider the categories of the business to find similarity. Social networks provide an important source of information regarding users and their interactions. We evaluated the likelihood of a user making choices regarding places to visit based on his friends. This consideration involved limited parameters such as the user rating. Further we can increase the span and include other factors such as user geography, interests, stars, etc.