FINAL PROJECT REPORT

Recommender System for Google+ users using Ego Networks

Big Data Management and Analytics **(**CS 6350.002**)**

Under the guidance of Prof. Anurag Nagar

TEAM MEMBERS

Lahari Ganesha (lxg150730)

Mahima Jayaprakash (mxj151430)

Panchami Rudrakshi (pgr150030)

Vaishnavi Bhat (vxv150130)

Kavyashree Anantha Raman (kxa152030)

1. PROBLEM STATEMENT

This project aims to automatically discover the user's social circles i.e. for a given user and his personal social network, the project aims to identify his circles, each of which is a subset of his friends.

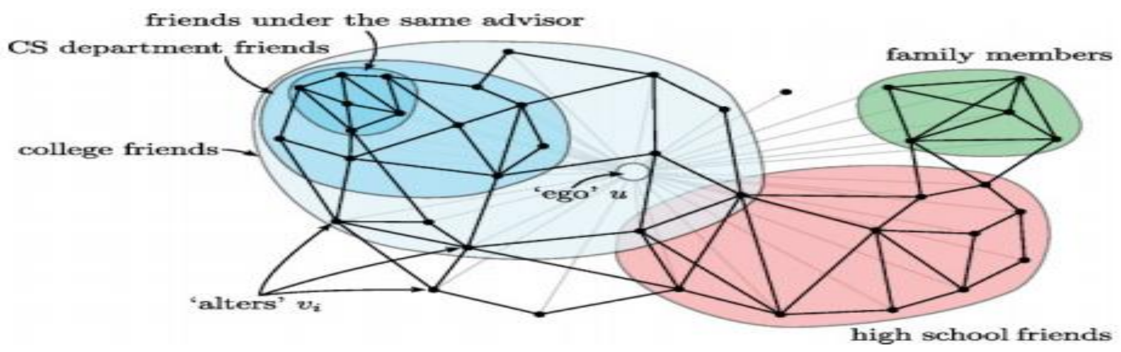
2. BACKGROUND ANALYSIS

2.1 GOOGLE+

Google+ users identify their circles manually. This burdens the user to take time to think about every contact and categorize them. Users have to create new Circles, and that requires even more thought. Users are unsure which Circles to put certain people in and which to leave them out of. And what if there is a need to create a new Circle that should include some of the people in other Circles one already has.

In this project, we aim to formulate the problem of circle detection as a clustering problem on user's ego networks, the network of friendship between his friends.

2.2 EGO NETWORKS



Ego networks consist of a focal node ("ego"- the user) and the nodes to whom ego is directly connected to ("alters"- connections, friends) plus the ties, if any, among the alters. Further, each alter in an ego network has his/her own ego network, and all ego networks together form a human social network. People have the strongest ties with people who are similar to themselves on key attributes, such as social class, age, sex, race, political views, etc. Also, the people with heterogeneous networks are "better off". The greater the diversity of their network, the more chance that someone in the network has something ego needs. The input is an ego-network G = (V, E), along with 'profiles' for each user v ∈ V. The 'ego' node u of the ego-network is not included in G, instead G consists only of u's friends ('alters') so that the creators of circles do not themselves appear in their own circles. For each ego-network, our goal is to predict a set of circles C = {C1 . . . CK}, Ck ⊆ V , and associated parameter vectors θk that encode how each circle emerged.

2.3 RECOMMENDATION SYSTEM

A computer system that makes suggestions for its users is called a recommender system. Recommender system is a subclass of information filtering system that seek to predict the 'rating' or 'preference' that a user would give to an item[7].

2.3.1 Types of Recommendation System

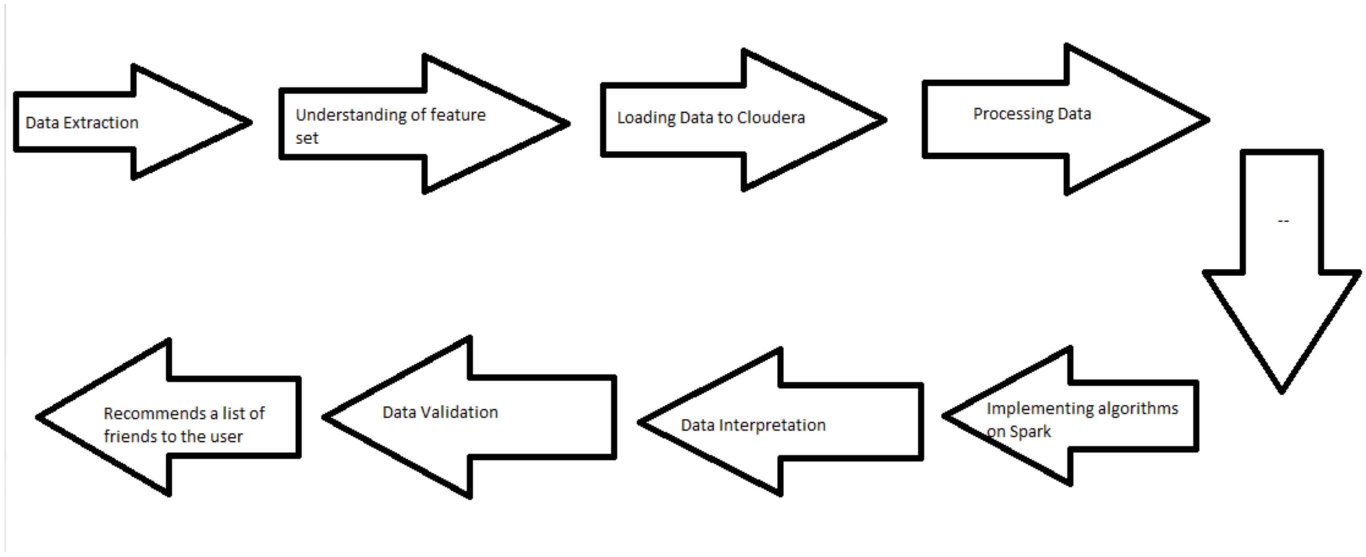
The recommendation systems can be categorized as:

- **Content-based systems:** It utilizes a series of discrete characteristics of an item in order to recommend additional items with similar properties. It considers the characteristics of the things you like, and it recommends similar sorts of things. For instance, if you like “Billie Jean”, “Crazy Train”, and “Don't Stop the Music”, then you might like other songs in the key of F-sharp minor, such as Rachmaninoff's “Piano Concerto No. 1”, even if no one else has ever had that particular set of favourite songs before.
- **Collaborative filtering systems:** These systems recommend for a user based on what things another user who is similar to the user being considered likes. That is, if user A likes orange Mango and Banana and User B like Orange and Mango, there is a good chance that user B would like Banana. Hence, the system suggests Banana to user B. The system here has no idea about what exactly an orange or Banana or mango is unlike content based filtering. Collaborative filtering is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past. The recommender system compares the collected data to similar and dissimilar data collected from others and calculates a list of recommended items for the user. Several commercial and non-commercial examples are listed in the article on collaborative filtering systems.
- In this Project we are mainly sticking with the Collaborative filtering Technique.

2.3.2 **Recommendation support systems:** These are the tools to support people in sharing their recommendations, helping both those who produce recommendations and those who seek for recommendation.

2.3.3 **Social data mining systems:** Here systems try to extract the implicit preferences from computational records of social activity, such as Usenet messages, system usage history, citations or hyperlinks.

3. DATA FLOW DIAGRAM


Data Extraction
Understanding of feature set
Loading Data to Cloudera
Processing Data
--
Implementing algorithms on Spark
Data Interpretation
Data Validation
Recommends a list of friends to the user

The data is extracted from http://snap.stanford.edu/index.html. The working of circles feature in Google+ is analysed. The data extracted is uploaded to Cloudera and processed. The algorithms to suggest most common friends are implemented in Scala. The input/output data is interpreted and validated. Finally, the user is recommended with a list of suggested most common friends.

Dataset : https://snap.stanford.edu/data/egonets-Gplus.html.

3.1 DATASET DESCRIPTION

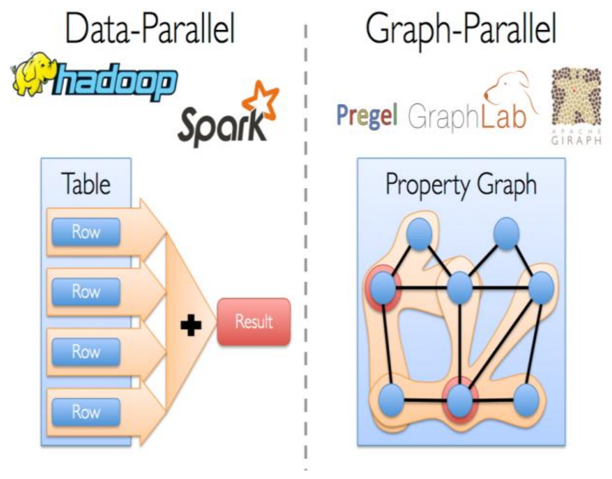
The datasets consists of the following files:

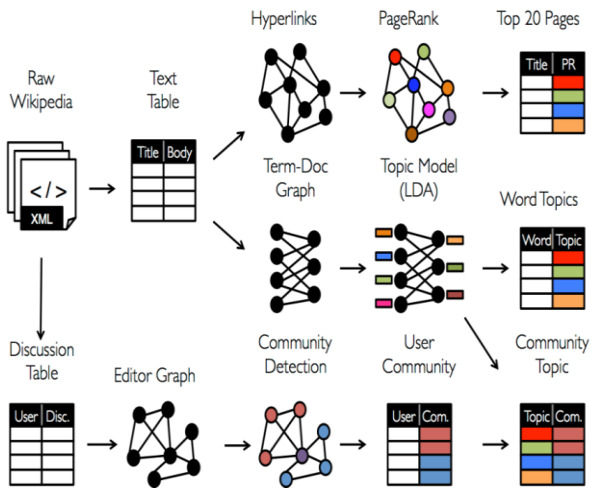
1. A circles file
2. A feature names file
3. An Edges file
4. An User's Features file
5. All features of a user's network
6. Followers files
7. A combined massive dataset containing all items

4. TECHNOLOGIES USED

4.1 Graph-X

Spark uses GraphX as a distributed graph processing framework on top of it. It has an API that models Pregel abstraction using graph computation. These can execute graph algorithms more efficiently than data parallel systems by restricting the types of computation that can be expressed and introducing new techniques to partition and distribute graphs. The GraphX API enables users to view data both as a graph and as collections (i.e., RDDs) without data movement or duplication[1].





4.2 Spark

Spark provides an application programming interface to the users centred on resilient distributed dataset (RDD), a read-only multi-set of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way[2]. For development and testing, Spark supports a pseudo-distributed local mode, where distributed storage is not required and the local file system can be used instead. Spark is run on a single machine with one executor per CPU core.

4.3 Scala

Scala is a general purpose programming language. Scala source code is compiled to Java byte code, to make it compatible to run on JVM. Scala is object-oriented, and features functional programming languages like Scheme, Standard ML and Haskell, including currying, type inference, immutability, lazy evaluation, and pattern matching. It also has an advanced type system supporting algebraic data types, covariance and contra-variance, higher-order types, and anonymous types. Scala supports operator overloading, optional parameters, named parameters, raw strings, and no checked exceptions.

5. FEATURE ANALYSIS

We use the below features to analyse and recommend friends

1) Top 10 User pairs with most number of Common Friends.

   We list top 10 pairs of users who have the most number of mutual friends in a given network (i.e., for a given user).

   Flow:

   - The file gplus_combined.txt consisting of edges from all ego networks combined is used to create edges and vertices RDDs.
     (Edges RDD consists of list of edges with vertex IDs replaced by hash keys; vertices RDD consists of a list of all the vertex IDs and the corresponding hash keys)
   - A graph is constructed from the edges RDD using the Graph-X API for Spark.
   - From the graph just constructed, all the vertices connected to one particular user U(whose corresponding hash key is fetched from the vertices RDD) is saved into a list(say, list A).
   - The edges not starting from the chosen user/vertex are filtered out from the graph. For the filtered source vertices, a list of (Source Vertex ID, Number of vertices which can be reached from the source, Number of common vertices than can be reached from the vertex U and the current vertex) is created(say List B).
   - This list is joined with the vertices RDD (joined based on the hash keys) and sorted in descending order.
   - Taking first 10 from the result gives the top 10 users with most number of common friends with the given user.

2) Suggesting Friends based on Jaccard similarity and cosine similarity.

   Flow:

   - The file gplus_combined.txt consisting of edges from all ego networks combined is used to create edges and vertices RDDs(Edges RDD consists of list of edges with vertex IDs replaced by hash keys; vertices RDD consists of a list of all the vertex IDs and the corresponding hash keys).
   - A graph is constructed from the edges RDD using the Graph-X API for Spark
   - From the graph just constructed, all the vertices connected to one particular user U(whose corresponding hash key is fetched from the vertices RDD) is saved into a list(say, list A).
   - The edges not starting from the chosen user/vertex are filtered out from the graph. For the filtered source vertices, a list of (Source Vertex ID, Number of vertices which can be reached

from the source, Number of common vertices than can be reached from the vertex U and the current vertex) is created(say List B).

- Jaccard similarity between U is calculated for every vertex in list B. This list is joined with the vertices RDD(joined based on the hash keys) and sorted in descending order.
- Taking first 5 from the result gives the top 5 recommendations for user U.
- Similar operations can be done using Cosine similarity.

3) Similar users with identical interests using cosine similarity.

Mathematically, cosine similarity calculates the similarity between 2 vectors by measuring the cosine of angle between them. It is used to measure cohesion within clusters.

Flow:

- For a particular user (say U), its featnames(feature names),feat and egofeat files are considered.
- The feat file for the user U is mapped to (VertexId, Array of feature values).
- Similar operation is performed on the egofeat file.
- The user U's feature set(Array of feature values) is mapped with every other user's(from the feat file) feature set.
- The sum of features of user U(Magnitude of Vector 1), sum of features of every other user(Magnitude of Vector 2) and the dot product of the two feature sets(count of the number of 1's in the two feature sets) are sent as parameters to the method cosineSimilarity.
- The users are then sorted in decreasing order based on the values of cosine similarity. Choosing top 5 gives the similar users with identical interest.

4) Count friends by gender.

For a given user, this section outputs the split of count of friends based on their gender.

Flow :

- The feat file of the chosen user U(containing a set of users in the ego's network) consists of a list of the set of users(connected to user U) and the feature set of each of the users.
- From every user's feature set, mapping is done to get the first three columns(which represent gender).
- Reduce operation is performed to count the number of male, female and gender-not-specified users for the chosen user U.

5) Get people in the user's network by place/University or any other Filter.

   For a specific user, we fetch the number of friends along with their university. We repeat the same process to fetch the details about the place where they live. These can be used as parameters to recommend friends.

   Flow :

- All features of a given user U filtered based on place/university are considered by filtering the feat file.
- For rest of the users in user U's network, the number of users for each feature is calculated by taking the sum(reduce phase).

  (Note: 1 represents that the particular feature applies to the user; hence taking the sum gives the number of users)
- For every feature considered specific to the user U, the corresponding number of its alters belonging to that feature are finally determined.

6) Find the top 5 users with most number of followers in the complete dataset.

We use the gplus_combined.txt file to determine top 5 users who have most number of followers.

Flow :

- All the .followers files corresponding to all the 130 ego networks are considered and the number of followers(ie., users) in each of the files is determined by taking length.
- The 130 ego network IDs along with their lengths(which is equal to the number of followers) are sorted in decreasing order and first 5 are considered. This gives the top 5 users with most number of followers in the complete dataset.

6. SCREENSHOTS

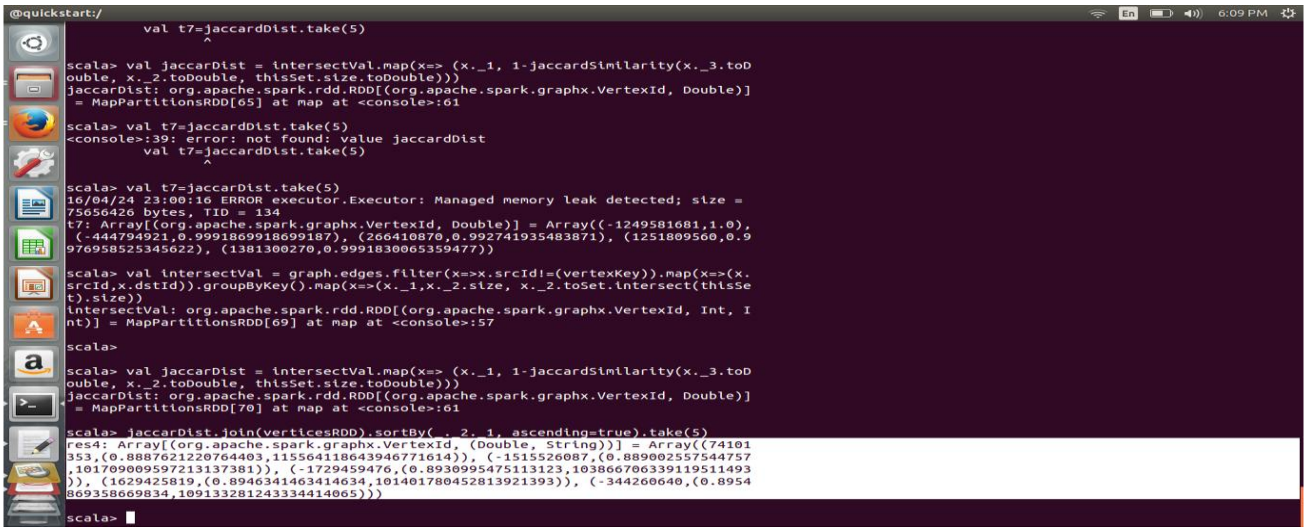
Some of the sample outputs for the below features are shown:

i) Suggest friends using Jaccard similarity

ii) Suggest friends based on Cosine similarity

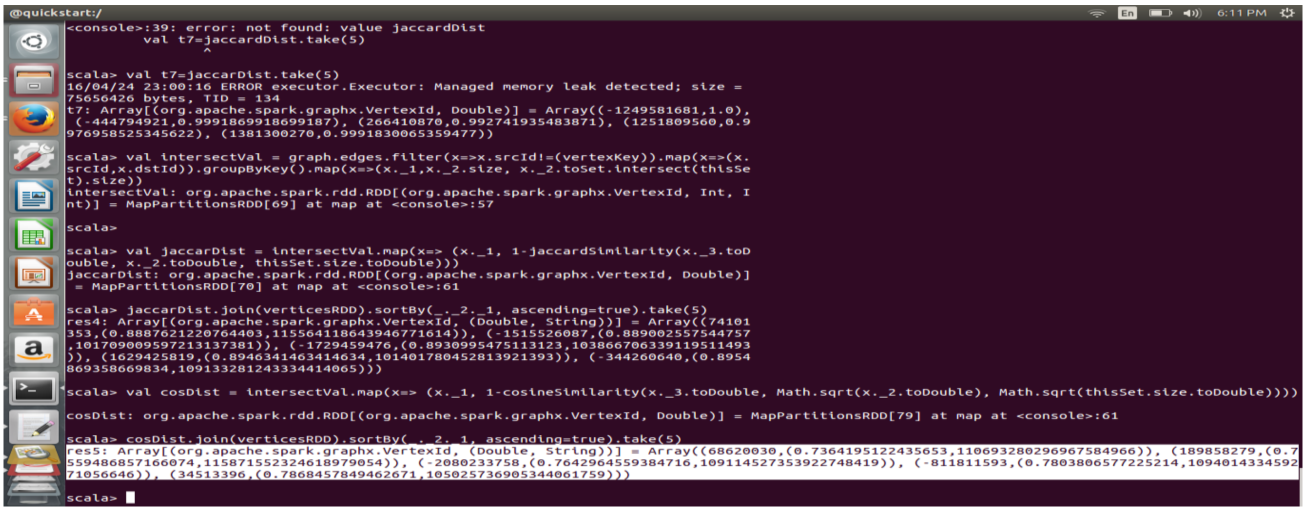
iii) Fetch people in users by University

i) **Suggest friends using Jaccard similarity:**



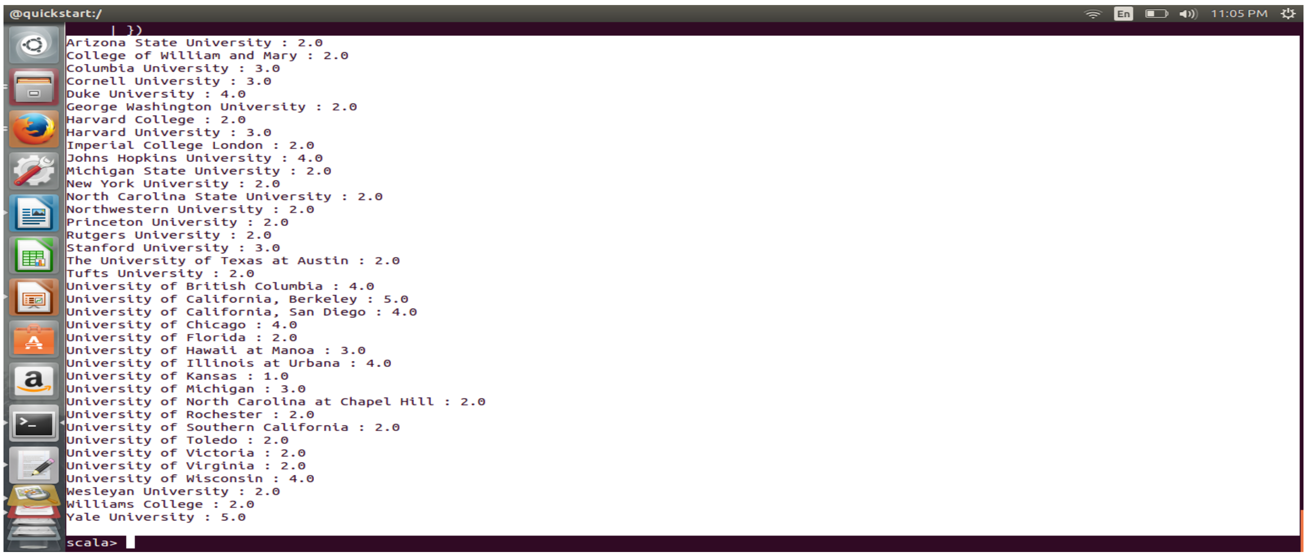
```
@quickstart:/
           val t7=jaccardDist.take(5)
                  ^

scala> val jaccarDist = intersectVal.map(x=> (x._1, 1-jaccardSimilarity(x._3.toD
ouble, x._2.toDouble, thisSet.size.toDouble)))
jaccarDist: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, Double)]
 = MapPartitionsRDD[65] at map at <console>:61

scala> val t7=jaccardDist.take(5)
<console>:39: error: not found: value jaccardDist
           val t7=jaccardDist.take(5)
                  ^

scala> val t7=jaccarDist.take(5)
16/04/24 23:00:16 ERROR executor.Executor: Managed memory leak detected; size =
75656426 bytes, TID = 134
t7: Array[(org.apache.spark.graphx.VertexId, Double)] = Array((-1249581681,1.0),
 (-444794921,0.9991869918699187), (266410870,0.992741935483871), (1251809560,0.9
976958525345622), (1381300270,0.9991830065359477))

scala> val intersectVal = graph.edges.filter(x=>x.srcId!=(vertexKey)).map(x=>(x.
srcId,x.dstId)).groupByKey().map(x=>(x._1,x._2.size, x._2.toSet.intersect(thisSe
t).size))
intersectVal: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, Int, I
nt)] = MapPartitionsRDD[69] at map at <console>:57

scala>

scala> val jaccarDist = intersectVal.map(x=> (x._1, 1-jaccardSimilarity(x._3.toD
ouble, x._2.toDouble, thisSet.size.toDouble)))
jaccarDist: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, Double)]
 = MapPartitionsRDD[70] at map at <console>:61

scala> jaccarDist.join(verticesRDD).sortBy(_._2._1, ascending=true).take(5)
res4: Array[(org.apache.spark.graphx.VertexId, (Double, String))] = Array((74101
353,(0.8887621220764403,115564118643946771614)), (-1515526087,(0.889002557544757
,101709009597213137381)), (-1729459476,(0.8930995475113123,103866706339119511493
)), (1629425819,(0.8946341463414634,101401780452813921393)), (-344260640,(0.8954
869358669834,109133281243334414065)))

scala> 
```

ii) **Suggest friends based on Cosine similarity:**



```
@quickstart:/
<console>:39: error: not found: value jaccardDist
           val t7=jaccardDist.take(5)
                  ^

scala> val t7=jaccarDist.take(5)
16/04/24 23:00:16 ERROR executor.Executor: Managed memory leak detected; size =
75656426 bytes, TID = 134
t7: Array[(org.apache.spark.graphx.VertexId, Double)] = Array((-1249581681,1.0),
 (-444794921,0.9991869918699187), (266410870,0.992741935483871), (1251809560,0.9
976958525345622), (1381300270,0.9991830065359477))

scala> val intersectVal = graph.edges.filter(x=>x.srcId!=(vertexKey)).map(x=>(x.
srcId,x.dstId)).groupByKey().map(x=>(x._1,x._2.size, x._2.toSet.intersect(thisSe
t).size))
intersectVal: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, Int, I
nt)] = MapPartitionsRDD[69] at map at <console>:57

scala>

scala> val jaccarDist = intersectVal.map(x=> (x._1, 1-jaccardSimilarity(x._3.toD
ouble, x._2.toDouble, thisSet.size.toDouble)))
jaccarDist: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, Double)]
 = MapPartitionsRDD[70] at map at <console>:61

scala> jaccarDist.join(verticesRDD).sortBy(_._2._1, ascending=true).take(5)
res4: Array[(org.apache.spark.graphx.VertexId, (Double, String))] = Array((74101
353,(0.8887621220764403,115564118643946771614)), (-1515526087,(0.889002557544757
,101709009597213137381)), (-1729459476,(0.8930995475113123,103866706339119511493
)), (1629425819,(0.8946341463414634,101401780452813921393)), (-344260640,(0.8954
869358669834,109133281243334414065)))

scala> val cosDist = intersectVal.map(x=> (x._1, 1-cosineSimilarity(x._3.toDouble, Math.sqrt(x._2.toDouble), Math.sqrt(thisSet.size.toDouble))))

cosDist: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, Double)] = MapPartitionsRDD[79] at map at <console>:61

scala> cosDist.join(verticesRDD).sortBy(_._2._1, ascending=true).take(5)
res5: Array[(org.apache.spark.graphx.VertexId, (Double, String))] = Array((68620030,(0.7364195122435653,110693280296967584966)), (189858279,(0.7
559486857166074,115871552324618979054)), (-2080233758,(0.7642964559384716,109114527353922748419)), (-811811593,(0.7803806577225214,1094014334592
71056646)), (34513396,(0.7868457849462671,105025736905344061759)))

scala> 
```

iii) **Fetch people in users by University:**



```
@quickstart:/
           | })
Arizona State University : 2.0
College of William and Mary : 2.0
Columbia University : 3.0
Cornell University : 3.0
Duke University : 4.0
George Washington University : 2.0
Harvard College : 2.0
Harvard University : 3.0
Imperial College London : 2.0
Johns Hopkins University : 4.0
Michigan State University : 2.0
New York University : 2.0
North Carolina State University : 2.0
Northwestern University : 2.0
Princeton University : 2.0
Rutgers University : 2.0
Stanford University : 3.0
The University of Texas at Austin : 2.0
Tufts University : 2.0
University of British Columbia : 4.0
University of California, Berkeley : 5.0
University of California, San Diego : 4.0
University of Chicago : 4.0
University of Florida : 2.0
University of Hawaii at Manoa : 3.0
University of Illinois at Urbana : 4.0
University of Kansas : 1.0
University of Michigan : 3.0
University of North Carolina at Chapel Hill : 2.0
University of Rochester : 2.0
University of Southern California : 2.0
University of Toledo : 2.0
University of Victoria : 2.0
University of Virginia : 2.0
University of Wisconsin : 4.0
Wesleyan University : 2.0
Williams College : 2.0
Yale University : 5.0

scala> 
```

7. CONCLUSION

Social networks provide an important source of information regarding users and their interactions. This is especially valuable to recommender systems. In this project we presented a social network-based recommender system (SNRS) which makes recommendations by considering a user's own preference, an item's general acceptance and influence from friends. In particular, we implemented a model for the correlations between immediate friends with the histogram of friend's rating differences. With the exposure of GraphX and Ego Network analysis, we were able to implement simple recommender system using Google+ data.

8. REFERENCES

1. http://ampcamp.berkeley.edu/big-data-mini-course/graph-analytics-with-graphx.html
2. https://en.wikipedia.org/wiki/Scala_(programming_language)
3. https://en.wikipedia.org/wiki/Spark_(software)
4. https://cs.stanford.edu/people/jure/pubs/circles-tkdd14.pdf
5. http://www.lifewithalacrity.com/2011/07/managing-your-social-graph-with-google-plus.html
6. http://www.faculty.ucr.edu/~hanneman/nettext/C9_Ego_networks.html
7. http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html
8. http://i.stanford.edu/~julian/pdfs/nips2012.pdf
9. http://files.grouplens.org/papers/rec-sys-overview.pdf