***PROJECT REPORT***

***QUESTION1)***

In this part of the project report, I choose *“extract\_email\_network”* function. The implementation of this function is converting the input RDD into such an RDD that when we materialize the latter RDD we have a list of tuples in the form such that *(sender,receiver,datetime)* where each sender and receiver are just one email address of which type is string and datetime is a datetime object.

Our dataset is a big string with the first and the second transformations

***map(lambda item:Parser().parsestr(item))***and

***map(lambda item:(item.get('From'),[item.get('To'),item.get('Cc'),item.get('Bcc')],date\_to\_dt(item.get('Date'))))***

first I structed the message objects from dataset then created the tuples, respectively, where I obtained “datetime” object by applying *date\_to\_dt()* function to the date string of the tuple. Hence the resulting tuples are of the form:

*(from\_email, [list of recipient emails], datetime)*

In this form “[list of recipient emails]” contains three sections namely, “From”, “Cc” and “BCc”. The problems with this version are for some tuples “From” part is a string which made up of some email addresses or for many of the tuples “Cc” and “BCc” parts are just “None” type, for example:

***('phillip.allen@enron.com',***

***['tim.belden@enron.com', None, None],***

***datetime.datetime(2001, 5, 14, 16, 39, tzinfo=datetime.timezone(datetime.timedelta(-1, 61200))) )***

or

***('phillip.allen@enron.com',***

***['frank.davis@enron.com, niamh.clarke@enron.com, sonya.clarke@enron.com', None,None],***

***datetime.datetime(2000, 7, 14, 7, 1, tzinfo=datetime.timezone(datetime.timedelta(-1, 61200))))***

To fix these problem, first I applied the transformation:

***flatMap(lambda x:val\_by\_vec(x[0],x[1],x[2])).***

In this way, I divided the [list of recipient emails] in such a way that every part of the list now became the second element of a tuple of which first element is the email address of sender and the third is the datetime object. Since I used flatmap() now all of those tuples, of which second elements was previously in the same list, are listed in the same list. Then to get rid of “None” filter transformation: *filter(lambda x: x[1] != None)*

At this point, when I materialize my RDD, some tuples appear like:

***('phillip.allen@enron.com', 'joyce.teixeira@enron.com', datetime.datetime(2000, 7, 14, 6, 59, tzinfo=datetime.timezone(datetime.timedelta(-1, 61200))))***

Whereas some others appear like:

***('stephanie.miller@enron.com',***

***'sarah.novosel@enron.com, christi.nicolay@enron.com, james.steffes@enron.com, pallen@enron.com, \r\n\tpkaufma@enron.com, richard.sanders@enron.com, \r\n\trichard.shapiro@enron.com, \r\n\tsusan.mara@enron.com',***

***datetime.datetime(2000, 12, 13, 5, 26, tzinfo=datetime.timezone(datetime.timedelta(-1, 57600)))).***

In the second type of tuples, as can be seen above, we still observe a whole string which is made up of many receiver email addresses that are separated by comma(,) followed by many white space characters. To split such strings into a list where each email address is a list item, I run two transformations one after another

***map(lambda x:(x[0],re.sub('\,\*\s+',',',x[1]),x[2]))*** and

***map(lambda x:(x[0],x[1].split(','),x[2]))***

Hence, first, by employing “regex” module, I replaced all the white space characters coming after a comma with a single comma(,) then split the string from the point where a comma is observed. Once again, the receiver part of some tuples is a list that contains separate email addresses of receivers in the form of string such as (transformed version of previous tuple):

***('stephanie.miller@enron.com',***

***['sarah.novosel@enron.com’, ‘christi.nicolay@enron.com’, ‘james.steffes@enron.com’, ‘pallen@enron.com’, ‘pkaufma@enron.com’, ‘richard.sanders@enron.com’, ‘richard.shapiro@enron.com’, ‘susan.mara@enron.com',],***

***datetime.datetime(2000, 12, 13, 5, 26, tzinfo=datetime.timezone(datetime.timedelta(-1, 57600)))).***

Again to deal with this problem I used the same code as I used before

***flatMap(lambda x:val\_by\_vec(x[0],x[1],x[2]))***

As a result of this transformation, I transformed tuples into the form (sender,receiver,time object) where both sender and receiver are represented by one email address which is in the form of string. Then I run two consequitive filter transformation

***filter(lambda x:valid\_email(x[0]) and valid\_email(x[1]))*** and

***.filter(lambda x:x[0]!=x[1])***

by which I selected the tuples where both the sender and the recipient Email addresses are both valid and are ending with enron.com and removed self-loops, respectively. And finally, to eliminate all the duplicates, I employed distinct transformation where I run on previous RDD and named it as “distinct” triples which is return as the output of the function

***rdd\_email\_triples\_enron.distinct()***

***return(distinct\_triples)***

The entire code is below the schema

|  |
| --- |
|  |
| ***Codes***  def extract\_email\_network(rdd):  import re  val\_by\_vec=lambda x,t,p:((x,item,p) for item in t)  email\_regex = "[\w.!#$%&'\*+-/=?^\_`{|}~]+@([a-zA-Z0-9]\*\.+(enron.com)$|(enron.com)$)"  valid\_email=(lambda s: True if re.compile(email\_regex).fullmatch(s)!=None else False)    rdd\_mail=rdd.map(lambda item:Parser().parsestr(item))  rdd\_full\_email\_tuples=rdd\_mail.map(lambda item:(item.get('From'),[item.get('To'), item.get('Cc'), item.get('Bcc')],date\_to\_dt(item.get('Date'))))  rdd\_email\_triples=rdd\_full\_email\_tuples.flatMap(lambda x:val\_by\_vec(x[0],x[1],x[2]))  rdd\_email\_triples\_enron=rdd\_email\_triples.filter(lambda x: x[1]!=None)\  .map(lambda x:(x[0],re.sub('\,\*\s+',',',x[1]),x[2]))\  .map(lambda x:(x[0],x[1].split(','),x[2]))\  .flatMap(lambda x:val\_by\_vec(x[0],x[1],x[2]))\  .filter(lambda x:valid\_email(x[0]) and valid\_email(x[1]))\  .filter(lambda x:x[0]!=x[1])  distinct\_triples = rdd\_email\_triples\_enron.distinct()    return(distinct\_triples) |

***QUESTION2)***

***2.1)***

In this question I selected the range from 1st of January 2000 to 1st of January 2001. employed *“get\_out\_degrees”* function as this function gives me the nodes in descending order that shows the (number of email which was out from that sender, sender).

Then first thing I did is finding the first 20% percentage of the output of get\_out\_degrees function with the given base RDD[[1]](#footnote-1) and then materializing it by using take() function.

***p=int((get\_out\_degrees(rdd).count())\*(0.20))***

***L1=get\_out\_degrees(rdd).take(p)***

Then I parallelized L1 again to create an RDD, which I called first20\_sender\_rdd, that contains data about the 20% of the most frequent email senders in the network. I run, first, map() transformation on first20\_sender\_rdd to pick the first elements, which is the number of email was sent by the corresponding sender, and then applied reduce() on transformed RDD. Therefore, the output of *“first20\_sender”,* gives me the total number of email that were sent by the 20% of the most frequent email senders in the network.

***first20\_sender\_rdd=sc.parallelize(L1)***

***first20\_sender=first20\_sender\_rdd.map(lambda x:x[0]).reduce(lambda a,x:a+x)***

I also apply the same series of functions on get\_out\_degrees(rdd) :

***total=get\_out\_degrees(rdd).map(lambda x:x[0]).reduce(lambda a,x:a+x)***

so that I found the total number of email sent in the whole network. Then I used the code ***ratio\_out=(first20\_sender/total)\*100*** to find the result . As a result, for the given period of time the total number of email that were sent by the 20% of the most frequent email senders in the network accounted for 99.6% of the total email sent in the network which does not satisfy 80/20 rule. I applied the same process to highest in degree nodes and ended up with 85.07 for the same period of time.

|  |
| --- |
| ***Complete Codes:***  p=int((get\_out\_degrees(rdd).count())\*(0.20))  L1=get\_out\_degrees(rdd).take(p)  first20\_sender\_rdd=sc.parallelize(L1)  first20\_sender=first20\_sender\_rdd.map(lambda x:x[0]).reduce(lambda a,x:a+x)  total=get\_out\_degrees(rdd).map(lambda x:x[0]).reduce(lambda a,x:a+x)  L2=get\_in\_degrees(rdd).take(p)  first20\_receiver\_rdd=sc.parallelize(L2)  first20\_receiver=first20\_receiver\_rdd.map(lambda x:x[0]).reduce(lambda a,x:a+x)  ratio\_out=(first20\_sender/total)\*100  ratio\_in=(first20\_receiver/total)\*100 |

***2.1)***

As the answer of this question I used “get\_out\_degrees” and “get\_in\_degrees” functions. Again, I used the same time period of time 1st of January 2000 to 1st of January 2001. Then I created a loop, the design of my loop is as follow:

In each iteration of the loop my code calculates the total nodes, number of email sent and received by the most frequent sender and receiver by using:

(get\_out\_degrees(rdd).count())

((get\_out\_degrees(rdd).take(1))[0][0])

((get\_in\_degrees(rdd).take(1))[0][0])

respectively, for that given month. And then replace these values to the corresponding lists. To calculate the last month which is 1st of December 2000 to 1st of January 2001, I created a separate code which functions exactly the same as the loop though just for one month.

I draw the graphs in R. the x-axis shows the number of emails and the y-axis shows the number of nodes in the network. As can be seen, although for some months we observe a opposite behaviour between the number of nodes and the number of emails which are sent(received) by the highest sender(receiver) (for example 8th and 9th months for Kmax out degree and 11th month for Kmax in), we can say that overall there is a positive correlation between them.

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
| ***Complete Codes:*** |
| L\_T=[]  L\_R=[]  L\_S=[]  for i in range(11):  rdd=convert\_to\_weighted\_network(extract\_email\_network(  utf8\_decode\_and\_filter(sc.sequenceFile('/user/mfac091/project/enron-full'))),  (datetime(2000, i+1, 1, tzinfo = timezone.utc),  datetime(2000, i+2, 1, tzinfo = timezone.utc)))  L\_T.append(get\_out\_degrees(rdd).count())  L\_S.append((get\_out\_degrees(rdd).take(1))[0][0])  L\_R.append((get\_in\_degrees(rdd).take(1))[0][0])  rdd\_rest=convert\_to\_weighted\_network(  extract\_email\_network(  utf8\_decode\_and\_filter(sc.sequenceFile(  '/user/mfac091/project/enron-full'))),  (datetime(2000, 12, 1, tzinfo = timezone.utc),  datetime(2001, 1, 1, tzinfo = timezone.utc)))  L\_T.append(get\_out\_degrees(rdd\_rest).count())  L\_S.append((get\_out\_degrees(rdd\_rest).take(1))[0][0])  L\_R.append((get\_in\_degrees(rdd\_rest).take(1))[0][0]) |

|  |  |  |
| --- | --- | --- |
| ***Dataset*** | | |
| Total\_email\_transaction | Out | In |
| 1977 | 547 | 144 |
| 2322 | 745 | 190 |
| 2474 | 684 | 227 |
| 2373 | 823 | 184 |
| 2163 | 681 | 254 |
| 3974 | 1138 | 290 |
| 3204 | 1045 | 279 |
| 4039 | 1778 | 318 |
| 3800 | 1586 | 399 |
| 4749 | 1535 | 405 |
| 5055 | 2232 | 391 |
| 4637 | 1698 | 326 |

1. rdd=convert\_to\_weighted\_network(extract\_email\_network(utf8\_decode\_and\_filter(sc.sequenceFile('/user/mfac091/project/enron-full'))),(datetime(2000, 1, 1, tzinfo = timezone.utc),datetime(2001, 1, 1, tzinfo = timezone.utc))) [↑](#footnote-ref-1)