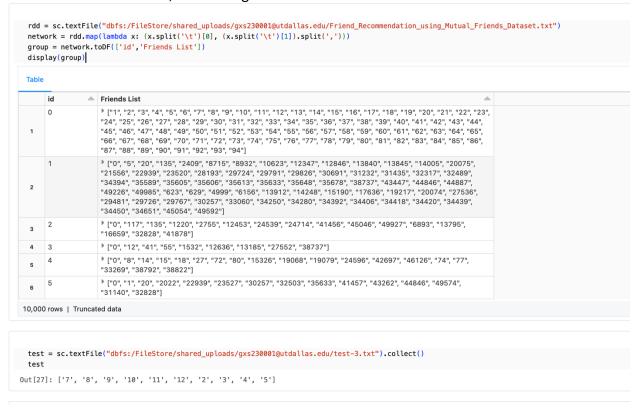
## **REPORT FOR ASSIGNMENT 2**

## Question 1:-

- 1. Read the input data from the dataset and map each friend list and user ID to a network RDD.
- 2. Uploading test data, using a user-defined function to filter out the matched data from the network to TestCases, and storing the outcome in an RDD named User.



3. In the next step we started calculating the strength value based on the network users' IDs' mutual connection.

```
def connected(userlst,peoplelst):
    result = []
    for uid in peoplelst:
        if uid not in userlst:
            result.append(uid)
    return result
```

4. After that we started assembling a list of additional userIDs associated with high matches.

```
def checker(userlst,peoplelst):
    mutual = len(set(userlst).intersection(peoplelst))
    return mutual

def connected(userlst,peoplelst):
    result = []
    for uid in peoplelst:
        if uid not in userlst:
            result.append(uid)
    return result
```

5. In the next step we started combining the lists of two connections by first retaining the most often occurring IDs and then adding items from each parent list that are not already in the new combined list.

```
from collections import Counter

def merger(list1, list2):
    final = list1 + list2
    element = Counter(final)
    most_common = element.most_common(10)
    new_list = [element for element, count in most_common]
    r = 10 - len(new_list)
    if r > 0:
        re = [element for element in list1 + list2 if element not in new_list]
        new_list += re[:r]

return new_list
```

6. Define the function responsible for generating suggestions for every test case.

7. Since the result cannot be mapped by applying transformation over transformation, it is saved in a dictionary called dict.

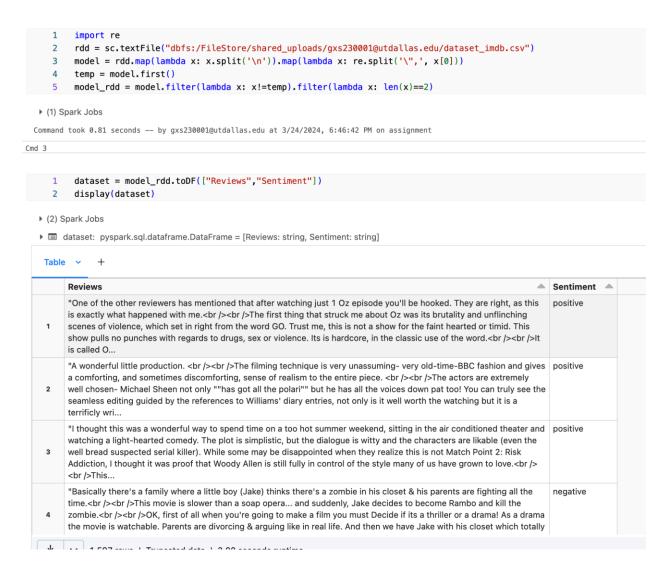
```
dic = {}
TestData = User.collect()
for chunk in TestData:
    dic[chunk[0]] = result(chunk[0], chunk[1])
```

Table			
	UserID 📤	Suggested Friends ID	
1	2	[20, 22939, 35633, 44846, 30257, 5, 2409, 8715, 8932, 10623]	
2	3	[3, 8, 5, 20, 1, 2, 4, 6, 7, 9]	
3	4	[46, 83, 4, 38, 85, 1, 2, 3, 5, 6]	
4	5	[5, 2, 3, 4, 6, 7, 8, 9, 10, 11]	
5	7	[20, 135, 5, 2409, 8715, 8932, 10623, 12347, 12846, 13840]	
6	8	[18, 27, 80, 8, 14, 15, 74, 77, 1, 2]	

## Question 2:-

10 rows

1. Perusing we enter information from the dataset and map the sentiment of each review into the IMDB\_Dataset RDD.



2. The data was cleaned using a variety of libraries, and the total number of positive and negative reviews was computed. From there, the prior probabilities were counted.

```
sw = stopwords.words('english')
     ltizer = WordNetLemmatizer()
tizer = RegexpTokenizer(r'\w+')
     def clean(lst):
         tlst = []
         for wrd in lst:
6
           if wrd not in sw:
                  tlst.append(wrd)
9
         return tlst
10
     def vb(rev,sval):
11
         wcount = {}
13
          for token in rev:
             if sval == 'positive':
14
                 wcount.setdefault(token,[0,0])
15
16
                  wcount[token][1] += 1
                 global TPW
TPW += 1
17
18
              elif sval == 'negative':
               wcount.setdefault(token,[0,0])
20
21
                  wcount[token][0] += 1
22
                  global TNW
                 TNW += 1
24
         return wcount
25
    tpr = model_rdd.filter(lambda x: x[1]=='positive').count()
27
     tnr = model_rdd.filter(lambda x: x[1] == 'negative').count()
28
     TPW = 0
    TNW = 0
29
    c1 = model_rdd.map(lambda x: (tizer.tokenize(x[0]), x[1]))
     c2 = c1.map(lambda x: ([wrd.lower() for wrd in x[0]], x[1]))
    c3 = c2.map(lambda x: ([ltizer.lemmatize(chunk) for chunk in x[0]], x[1]))
32
     c4 = c3.map(lambda x: (clean(x[0]), x[1]))
33
     Vrdd = c4.map(lambda x: (vb(x[0],x[1]), x[1]))
    Vrdd.collect()
'flex': [1, 0],
'posterior': [1, 0],
'muscle': [1, 0],
'even': [1, 0],
'ridiculously': [1, 0],
'campy': [1, 0],
'gay': [1, 0],
```

3. To avoid the divide by zero error, conditional probabilities were computed using Laplace Smoothing, and the result was saved in a dictionary with words serving as the key and conditional probabilities serving as the values.

```
pp = tpr / ttr
pn = tnr / ttr
AllWords = Vrdd.flatMap(lambda x: x[0].keys()).distinct().collect()
uw = len(AllWords)
ppc = sc.broadcast(pp)
pnc = sc.broadcast(pn)
uwc = sc.broadcast(uw)
AllWords_bc = sc.broadcast(AllWords)
def ccp(record):
    wcount, sentiment = record
    cps = \{\}
    for word, counts in wcount.items():
        positive_count = counts[1]
        negative_count = counts[0]
        total_count = positive_count + negative_count
        positive_prob = (positive_count + 1) / (TPW + uwc.value)
        negative_prob = (negative_count + 1) / (TNW + uwc.value)
        cps[word] = (positive_prob, negative_prob)
    return cps, sentiment
cps_rdd = Vrdd.map(ccp)
cps_rdd.collect()
```

4. Determined the likelihoods of a review and used previously determined prior probabilities to classify it.

```
def cv(review):
           t_1 = tizer.tokenize(review)
   3
           t_2 = [wrd.lower() for wrd in t_1]
   4
           t_l = [ltizer.lemmatize(chunk) for chunk in t_2]
   5
           c_t = clean(t_l)
   6
           llp = 0
   7
           lln = 0
   9
  10
           for word in c t:
  11
               if word in AllWords:
                  p, n = cps_rdd.map(lambda x: x[0]).map(lambda x: (VE(x,word))).reduce(lambda x,y: ct(x,y))
  12
  13
                   llp += math.log(p)
  14
                   lln += math.log(n)
  15
  16
           llp += math.log(ppc.value)
  17
           lln += math.log(pnc.value)
           if llp > lln:
  19
  20
              return 'positive'
  21
           else:
  22
              return 'negative'
Command took 0.07 seconds -- by gxs230001@utdallas.edu at 3/24/2024, 8:47:11 PM on assignment
1 11
```

5. The next step was to text the Naïve Bayes Classifier.

```
test_data = ['This is a nice little movie with a nice story', 'So many good actors and they are all acting so badly!','The whole movie was negative.']

val = ['positive', 'negative', 'negative']

cnr = []

for nr in test_data:

class_nr = cv(nr)

print("The review: {} is classified as: {}".format(nr,class_nr))

cnr.append(class_nr)
```

The review: This is a nice little movie with a nice story is classified as: positive
The review: So many good actors and they are all acting so badly! is classified as: negative

The review: The whole movie was negative. is classified as: negative

Command took 15.78 minutes -- by gxs230001@utdallas.edu at 3/24/2024, 9:08:51 PM on assignment

6. At the end after successful testing we got the 75% accuracy using the dataset and the above-described Naïve Bayes classifier.

```
test_data = ['This is a nice little movie with a nice story', 'So many good actors and they are all acting so badly!','The whole movie was negative.']
val = ['positive', 'negative', 'negative']
cnr = []
for nr in test_data:
class_nr = cv(nr)
print("The review: {} is classified as: {}".format(nr,class_nr))
cnr.append(class_nr)
```

## ▶ (14) Spark Jobs

The review: This is a nice little movie with a nice story is classified as: positive
The review: So many good actors and they are all acting so badly! is classified as: negative
The review: The whole movie was negative. is classified as: negative

Command took 15.78 minutes -- by gxs230001@utdallas.edu at 3/24/2024, 9:08:51 PM on assignment