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.

```
rdd = sc.textFile("dbfs:/FileStore/shared_uploads/gxs230001@utdallas.edu/Friend_Recommendation_using_Mutual_Friends_Dataset.txt")
network = rdd.map(lambda x: (x.split('\t')[0], (x.split('\t')[1]).split(',')))
group = network.toDF(['id','Friends List'])
display(group)
```

Table

	id	Friends List
1	0	["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47", "48", "49", "50", "51", "52", "53", "54", "55", "56", "57", "58", "59", "60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "70", "71", "72", "73", "74", "75", "76", "77", "78", "79", "80", "81", "82", "83", "84", "85", "86", "87", "88", "89", "90", "91", "92", "93", "94"]
2	1	["0", "5", "20", "135", "2409", "8715", "8932", "10623", "12347", "12846", "13840", "13845", "14005", "20075", "21556", "22939", "23520", "28193", "29724", "29791", "29826", "30691", "31232", "31435", "32317", "32489", "34394", "35589", "35605", "35606", "35613", "35633", "35648", "35678", "38737", "43447", "44846", "44887", "49226", "49985", "623", "629", "4999", "6156", "13912", "14248", "15190", "17636", "19217", "20074", "27536", "29481", "29726", "29767", "30257", "33060", "34250", "34280", "34392", "34406", "34418", "34420", "34439", "34450", "34651", "45054", "49592"]
3	2	["0", "117", "135", "1220", "2755", "12453", "24539", "24714", "41456", "45046", "49927", "6893", "13795", "16659", "32828", "41878"]
4	3	["0", "12", "41", "55", "1532", "12636", "13185", "27552", "38737"]
5	4	["0", "8", "14", "15", "18", "27", "72", "80", "15326", "19068", "19079", "24596", "42697", "46126", "74", "77", "33269", "38792", "38822"]
6	5	["0", "1", "20", "2022", "22939", "23527", "30257", "32503", "35633", "41457", "43262", "44846", "49574", "31140", "32828"]

10,000 rows | Truncated data

```
test = sc.textFile("dbfs:/FileStore/shared_uploads/gxs230001@utdallas.edu/test-3.txt").collect()
test
```

```
Out[27]: ['7', '8', '9', '10', '11', '12', '2', '3', '4', '5']
```

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])
```

```
from pyspark.sql.types import StructType, StructField, StringType
df = spark.createDataFrame(RecommenderDict.items(),
                           schema=StructType(fields=[
                               StructField("UserID", StringType()),
                               StructField("Suggested Friends ID", StringType())]))

display(df)
```

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]

10 rows

Question 2:-

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

```
1  import re
2  rdd = sc.textFile("dbfs:/FileStore/shared_uploads/gxs230001@utdallas.edu/dataset_imdb.csv")
3  model = rdd.map(lambda x: x.split('\n')).map(lambda x: re.split('\",', x[0]))
4  temp = model.first()
5  model_rdd = model.filter(lambda x: x!=temp).filter(lambda x: len(x)==2)
```

▸ (1) Spark Jobs

```
Command took 0.81 seconds -- by gxs230001@utdallas.edu at 3/24/2024, 6:46:42 PM on assignment
```

```
Cmd 3
```

```
1  dataset = model_rdd.toDF(["Reviews","Sentiment"])
2  display(dataset)
```

▸ (2) Spark Jobs

▸ dataset: pyspark.sql.dataframe.DataFrame = [Reviews: string, Sentiment: string]

Table +

	Reviews	Sentiment
1	"One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me. The first thing that struck me about Oz was its brutality and unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a show for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the classic use of the word. It is called O...	positive
2	"A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece. The actors are extremely well chosen- Michael Sheen not only ""has got all the polari"" but he has all the voices down pat too! You can truly see the seamless editing guided by the references to Williams' diary entries, not only is it well worth the watching but it is a terrificly wri...	positive
3	"I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dialogue is witty and the characters are likable (even the well bread suspected serial killer). While some may be disappointed when they realize this is not Match Point 2: Risk Addiction, I thought it was proof that Woody Allen is still fully in control of the style many of us have grown to love. This...	positive
4	"Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time. This movie is slower than a soap opera... and suddenly, Jake decides to become Rambo and kill the zombie. OK, first of all when you're going to make a film you must Decide if its a thriller or a drama! As a drama the movie is watchable. Parents are divorcing & arguing like in real life. And then we have Jake with his closet which totally	negative

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:
        if wrd not in sw:
            tlst.append(wrd)
    return tlst

def vb(rev,sval):
    wcount = {}
    for token in rev:
        if sval == 'positive':
            wcount.setdefault(token,[0,0])
            wcount[token][1] += 1
            global TPW
            TPW += 1
        elif sval == 'negative':
            wcount.setdefault(token,[0,0])
            wcount[token][0] += 1
            global TNW
            TNW += 1
    return wcount

tpr = model_rdd.filter(lambda x: x[1]=='positive').count()
tnr = model_rdd.filter(lambda x: x[1]=='negative').count()
TPW = 0
TNW = 0
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]))
c4 = c3.map(lambda x: (clean(x[0]), x[1]))
Vrdd = c4.map(lambda x: (vb(x[0],x[1]), x[1]))
Vrdd.collect()
```

▸ (3) Spark Jobs

```
'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)
    t_2 = [wrd.lower() for wrd in t_1]
    t_l = [ltizer.lemmatize(chunk) for chunk in t_2]
    c_t = clean(t_l)

    llp = 0
    lln = 0

    for word in c_t:
        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))
            llp += math.log(p)
            lln += math.log(n)

    llp += math.log(ppc.value)
    lln += math.log(pnc.value)

    if llp > lln:
        return 'positive'
    else:
        return 'negative'

```

Command took 0.07 seconds -- by gxs230001@utdallas.edu at 3/24/2024, 8:47:11 PM on assignment

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)

```

▸ (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

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