Part II: Problem Solving

Submitted by- Himanshu Agrawal

Question 1:

Algorithm:

```
from pprint import pprint
def mapper(row):
    returnList= []
    if row[0] == 'N':
        for i in range(Mrows):
            returnList.append(((i, row[1][0][1]), [('N', row[1][0][0], row[1][1])]))
    else:
        for k in range(Ncolumns):
            returnList.append(((row[1][0][0], k), [('M', row[1][0][1], row[1][1])]))
    return returnList

def reducer(collection):
    listofAll = []
    listofAll = sorted(list(collection[1]),key=lambda x: [x[0][0],x[0][1]])
    summation = 0
    for i in range(len(listofAll)//2):
        summation += listofAll[i][0][2] * listofAll[i + len(listofAll)//2][0][2]
    return (collection[0],summation)

rdd = sc.parallelize([('M', ((0,0), 2)), ('M', ((0,1), 1)), ('M', ((1,0), 6)), ('M', ((1,1), 3)), ('N', ((0,0), 5)), ('N', ((0,1),
11)), ('N', ((1,0), 0)), ('N', ((1,1), 1))])
mappedRdd = rdd.flatMap(mapper)
output_rdd = mappedRdd.groupByKey().map(reducer)
pprint(output_rdd.take(100))
```

Assumptions:

1. Rdd is the data which has records in the form (matrix_name, ((i, j), v)) where matrix_name is “M” or “N”, i and j correspond to row and column respectively, and v is the value.
2. For reference I have taken a matrix records as shown above to Test.
3. Value of Mrows and Ncolumns is globally accessible. Where, Mrows is the number of rows in Matrix ‘M’ and Ncolumns is the number of columns in Matrix ‘N’.
4. Final output is in the output_rdd, with format as ((i,k),Value), where i,k represents the output matrix coordinates and value is its value in matrix.

Question 2:

Insert below code after

#add in gradient calculation
grads = tf.gradients(penalizedCost, [beta])[0] in the demo code.

Code:

```
        momentum = tf.Variable(tf.zeros([featuresZ_pBias.shape[1], 1]),name = "momentum")
        gamma = 0.9  # typical momentum value
        momentum = tf.multiply(momentum, gamma) + learning_rate * grads
        training_op = tf.assign(beta, beta - momentum)
```

Also, remove the original training_op.

Question 3:
Part A)

Suppose we have a characteristic matrix as discussed in the class as below:

Element	S1	S2
A	1	0
B	0	0
C	1	1
D	1	0
E	0	1

S1,S2 are 2 sets and the 1 in cell represent if that element is present in that particular set otherwise its 0.

Now, to find the minhash(represented by h(S1)) of a column, we perform a random permutation of the rows of the matrix and then for a Set column we pick the row number where we find the first 1 going from top to bottom.

Note: There are three types of rows in a characteristic matrix as can be seen in table above:

1- Type X rows that has 1 in both columns.
2- Type Y rows that have 1 in one of the columns and 0 in the other.
3- Type Z rows that have 0 in both columns.

To find the Jaccard similarity for sets S1 and S2 we use : $sim(S1, S2) = \frac{X}{X+Y}$ where X is the size of S1 ∩ S2 and X + Y is the size of S1 ∪ S2.
Now, if Jaccard similarity of two sets is zero, it implies that X=0 i.e. there is no element for which value of both S1 and S2 is 1. So, for any permutations of rows we take(h), there will never be a minhash function that yields $h(S1) = h(S2)$.

So the estimate we would get from minhashing on such a characteristic matrix where Jaccard Similarity is 0 would always yield a 0.

Example characteristic matrix:

Element	S1	S2
A	0	1
B	1	0
C	0	1
D	1	0
E	0	0

Here, $sim(S1, S2) = 0$ and it proves the above explanation.

Question 3:
Part B)

Algorithm:

1. rdd = sc.parallelize([('S1', ['a', 'e']), ('S2', ['c', 'd']), ('S3', ['b']), ('S4', ['a', 'd', 'e'])])
2. outputRdd = rdd.map(lambda row: (row[0], [min([getHash(hash_name, i) for i in row[1]]) for hash_name in hashes]))

Assumptions:

1) The rdd has records of the form (set_name,(elem1,elem2 …)) like (S1,(‘a’,’b’)). An example is used as shown above in line 1.
2) We assume that we have a hash function generator which gives us 500 hash functions stored in a variable name **hashes (Declaration as hashes = [getHfunc(i) for i in rand(1, num=500)]**
3) outputRdd is of the form: (set_name1,[columns_values_of_set1]),(…)…
4) This approach to find the signature matrix in line 2 is efficient as follows:
   a) For all the hashes ***h_i***, I calculate the hash value for the set elements and find the minimum hash value return for a particular ***h_i***
   b) This approach gives the value of the first 1 we would find a permuted sequence of rows which is very efficient as we do not have to travers a column to find the first 1.
   c) The getHash function takes input a hash function ***h_i*** and a value. It returns the hashed value on applying hash function ***h_i*** to the value.
5) According to Piazza post @117
   Each set is a record:
   (set_name, (elem1, elem2, ...))
   HDFS always partitions chunks of records. So here, each chunk (i.e. partition of records) will be comprised of many sets.

Question 4:

Part A)

Let :

r - rows in each band b = 5

b - number of bands = 100(Since total rows = 500)

s - Jaccard similarity of two documents.

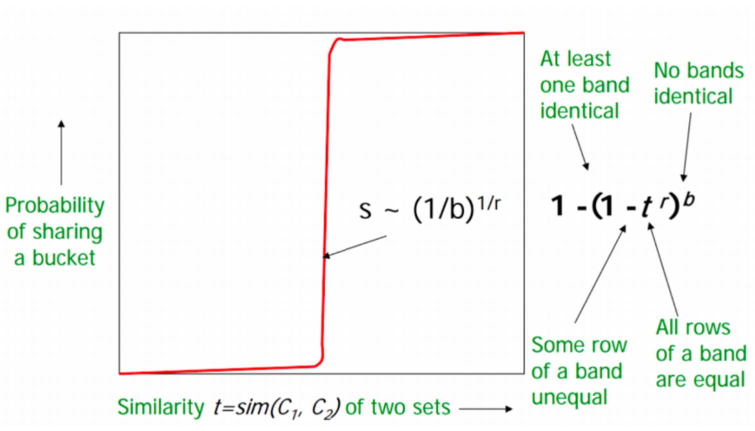
Probability(minhash sig match in 1 band) $= s^r$
Probability(no minhash sig match in a band) $= 1 - s^r$
Probability(no minhash sig match in all bands)= $(1 - s^r)^b$
Probability(Two sets are matched in at least 1 band) = $1 - (1 - s^r)^b$
$= 1 - (1 - 0.75^5)^{100}$
$= 0.9999$

Question 4)

Part B)

We want the Jaccard Similarity to be atleast 90% on over 99% of document pairs. At the same time we also want to keep our false positives low.

Now, we know that $b * n = 500$, and $t = (1/b)^{1/r}$, where t is the threshold i.e. the jaccard similarity we want(90%).
Using the above 2 equations, we can calculate value of r = 18 and b = 28.
The formula $1 - (1 - s^r)^b$ gives the probability of finding a candidate pair which has similarity s with r and b values.
For r,b,and s values as above in above formula gives a Probability >0.99 which is required in this question.


At least one band identical
No bands identical
Probability of sharing a bucket
s ~ (1/b)^{1/r}
1 -(1 -t^r)^b
Some row of a band unequal
All rows of a band are equal
Similarity t=sim(C1, C2) of two sets

(Source: [LINK])

The above graph shows the general trend of probability with the Similarity for various values of r and b.
I tested for some other values of r and b as follows:

r	b	Prob(%)
17	29	99.5
16	31	99.83
19	27	97
25	20	77.46
18	28	99

From above we can see that varying the values of r and b leads to different probability percentages of number of pairs of candidates which satisfy atleast 90% similarity. Higher b implies lower similarity threshold (higher false positives) and lower b implies higher similarity threshold (higher false negatives). So we choose b and r to get the best S-curve i.e minimum false negative and false positive rate.

Question 4)

Part C)

Using the Speedup technique as discussed in **MMDSv3 3.3.6.**

As mentioned in the algorithm of that section I will use the same procedure.

Now, we can speed up the LSH process indirectly by increasing the speed of the creation of the signature matrix which is the preceding step of LSH.
Signature matrix creation can be speedup using **MMDSv3 3.3.6 as discussed below:**

1- Suppose we have k-rows in the characteristic matrix, we will only consider the first **m** rows. Here, m is a small value as compared to k.
2- The reason is that we do not need to construct a full permutation of k elements, but only pick a small number m out of the k rows and then pick a random permutation of those rows.
3- Now we choose a hash function that hashes row numbers, and compute hash values for only the first m rows
4- As long as each column has at least one 1 in the first m rows in permuted order, the rows after the mth have no effect on any minhash value. But if some columns are all-0's in the first m rows, we have no minhash value for those columns, and ∞.
5- Now, when we examine the minhash signature for a pair of set to compute Jaccard Similarity we get 3 types of conditions:
   A) Both columns have valid value. We can treat as normal values as X and Y as mentioned in Q3-PartA
   B) One column has ∞. In this case we treat it as type Y since the ∞ value would eventually have some value if considered all k-rows but would have no effect on Similarity as their minhash would be unequal.
   C) Both ∞, in this case we simply discard the value and do not use it in calculation of the similarity of the sets.

If we can limit the number of Type C conditions, then we get almost as many examples to average as there are rows in the signature matrix. This will decrease the Jaccard similarity but not much if we limit cases C. This way we have increased the speed of minhash at the expense of decreasing the similarity a bit.

As discussed in Q4 PartB, the false positive rate depends upon the probability that a pair of candidates(which does not belong to a particular bucket) is likely to be hashed into that bucket, depends on the threshold value (Similarity threshold). Also, by having the Type C, we can now have some ∞ values in a band which will result in less number of row (r) in the formula $1-(1-s^{r-1})^{b}$. We see a r-1 for every Type C row discarded which also results in the lower probability hence higher false positives. Thus, by the possibility of lowering the Jaccard Similarity by speeding up the Minhash we have increased the chances of the False Positives.

I will use this above modification in the algorithm of Part 3B, and choose m values instead of all 500 values.