DS 702 – Assignment 1

Map Reduce

1. Assuming that no combiner is used at the map task, there will be significant skew in the time taken for reducers to process their value list. A reducer is defined to perform the reduce function on a single key. Furthermore, we know that due to the diverse nature of data, the length of the value list will vary from key to key (i.e., Some keys might have a longer value list than other keys). Thus, causing large variation of times to process different reducers, leading to a significant skew.

2. Answer

    a. If we combine the reducers into approximately 10 reduce tasks we will expect the skew to not be significant as multiple keys with varying value list will be in a single reduce task, thus through averaging we can assume that the combined length of value lists across the 10 reduce tasks will be similar leading to insignificant skew.
    b. Since the total number of map tasks are 100 combining the reducers into a 10,000 reduce tasks could possibly mean that the number of reduce tasks are more than the number than the number of keys and in turn the number of reducers. This could cause a significant skew firstly because each reduce task has an overhead associated with it and secondly if the number of reduce tasks is greater than the number of reducers, essentially that means that there would be one reducer on one reduce task causing the same significant skew as discusses in question 1.

Algorithms Using MapReduce

(a) Bag Union, defined to be the bag of tuples in which tuple t appears the sum of the numbers of times it appears in R and S.

```
# UNION (R OR S)

def mapFunction(input_Tuple):
    key, value = input_Tuple
    yield (key, 1)

def reduceFunction(key, list_of_values): # (tupleKey: [1,1,1,1,1,1,1,1,1])
    yield (key, sum(list_of_values)) # sum = 9
```

(b) Bag Intersection, defined to be the bag of tuples in which tuple t appears the minimum of the numbers of times it appears in R and S.

```
# INTERSECTION (R AND S)

def mapFunction(input_Tuple):

    key, value = input_Tuple
    if input_Tuple in relation_R:
        yield (key, 1)
    elif input_Tuple in relation_S:
        yield (key, 2)

def reduceFunction(key, list_of_values): # (tupleKey: [1,2,1,2,2,1,2,1,1])
    list_of_ones = [v for v in list_of_values if v == 1] # [1,1,1,1,1]  sum = 5
    list_of_twos = [v for v in list_of_values if v == 2] # [2,2,2,2]    sum/2 = 8/2 = 4
    yield (key, min( sum(list_of_ones), (sum(list_of_twos)/2) )) # min(5,4) = 4
```

(c) Bag Difference, defined to be the bag of tuples in which the number of times a tuple t appears is equal to the number of times it appears in R minus the number of times it appears in S. A tuple that appears more times in S than in R does not appear in the difference

```python
# DIFFERENCE (R - S)

def mapFunction(input_Tuple):

    key, value = input_Tuple
    if input_Tuple in relation_R:
        yield (key, 1)
    elif input_Tuple in relation_S:
        yield (key, 2)

def reduceFunction(key, list_of_values): # (tupleKey: [1,2,1,2,2,1,2,1,1])
    list_of_Rs = [v for v in list_of_values if v == 1] # [1,1,1,1,1]    sum = 5
    list_of_Ss = [v for v in list_of_values if v == 2] # [2,2,2,2]  sum/2 = 8/2 = 4

    if sum(list_of_Rs) > (sum(list_of_Ss)/2:
        yield (key, (sum(list_of_Rs) - (sum(list_of_Ss)/2))) # 5 - 4 = 1
    else:
        yield (key, None)
```

Finding Similar Items

1. Jaccard Similarity between 2 sets is calculated as follows:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

- Set 1 & 2

  | {2,3} | / | {1,2,3,4,5,7} | = 2 / 6

- Set 2 & 3

  | {2} | / | {2,3,4,5,6,7} | = 1 / 6

- Set 1 & 3

  | {2,4} | / | {1,2,3,4,6} | = 2 / 5

2. Jaccard Bag Similarity

   - Bag 1 & 2

     | {1,1,2} | / | {1,1,1,1,1,2,2,2,3} | = 3 / 9

   - Bag 2 & 3

     | {1,2,3} | / | {1,1,1,2,2,2,3,3,4} | = 3 / 9

   - Bag 1 & 3

     | {1,2} | / | {1,1,1,1,2,2,3,4} | = 2 / 8

Spark Code

Files attached:

Friend_Recommender.ipynb

db.txt

<Ran on Google Colabs>

<Solutions to specific questions are on the notebook>