DATASCI W261: Machine Learning at Scale

- **Sayantan Satpati**
- **sayantan.satpati@ischool.berkeley.edu**

- **W261**
- **Week-3**
- **Assignment-4**
- **Date of Submission: 22-SEP-2015**

=== Week 3 Hadoop & Apriori ===

What is a merge sort? Where is it used in Hadoop?

Merge sort is a comparison based sorting algorithm [O(nlogn)] which works by recursively dividing an unsorted array till the individual lists are of size 1, and then merging the sublists to produce new sorted sublists until there is only 1 sublist remaining.

In Hadoop Merge Sort is used in the following places: Each mapper produces a sorted spill file (if it exceeds memory limitations) for each Reducer. Once the mappers are done processing on a single m/c, the spill files for each reducer are merged together by the combiner. Similarly, before the reduce phase, the sorted files from different mapper m/c(s) are merged together by a combiner before streaming the data to the reducer.

How is a combiner function in the context of Hadoop?

Combiner is used in 1) Combine outputs from several mappers running on the same m/c, so that less amount of data is transferred over network & 2) Combine the outputs from different mapper m/c(s), before the data is streamed to the reducer.

Give an example where it can be used and justify why it should be used in the context of this problem.

For example in the Word Count problem, stop words ("the", "for", "an" etc) might occur very frequently. Instead of passing the tuple ("the", 1) 1 million times over network, the combiner acts an in-mapper mini reducer & combines the output of all the mappers running on the same m/c passing a single tuple ("the", 1000000), which is much more efficient. However, there is a restriction - combiner can only be used when the function you want to apply to is both commutative (Ex: x + y = y + x), and associative (Ex: x + (y + z) = (x + y) + z). It can be used in Word Count problem since sum of the whole is equal to sum of its parts. But, it cannot be used to calculate something like a Median of a list of numbers.

What is the Hadoop shuffle?

Hadoop shuffle is the process of transferring data from mappers to reducers. It essentially involves **partition, sort, and combine** phases. Each mapper processes a chunk of the input data depending on the block size (64MB, 128 MB etc) and creates records for each Reducer (taking Partition Logic into consideration). These in-memory records might get written into disk as sorted spill files. Once all the mappers are done processing on a single m/c, the combiner may kick in and combine the data (multiple spill files) for each reducer using Merge Sort. Once all the mappers are done processing, the combined spill files (meant for each reducer) are transferred over network to the reducer node, where they are all Merge Sorted by the Combiner before streaming the sorted file to the Reducer.

What is the Apriori algorithm? Describe an example use in your domain of expertise. Define confidence and lift.

Per wikipedia, Apriori is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis.

I work for ebay in the Search and Information Retrieval Space. Apriori Algorithm is used to power the related searches functionality in ebay. It shows related searches (based on keywords) to an user by mining the data on what other users might have searched when the searched using the same keyword that the current user has used.

- **Confidence(X => Y)** : is the proportion the transactions that contains X, which also contains Y.
- **Lift (X => Y)** : In general, if we have a high Support and a high Confidence, this means we have a strong association. However, this can be deceptive in some cases because if Item X and Item Y have a high support, we can have a high value for confidence even when they are independent. A better measure to judge the strength of an association rule is the Lift. The Lift ratio is the confidence of the rule divided by the confidence assuming the consequent (Item Y) and the antecedent (Item X) are independent

HW3.1.

Product Recommendations: The action or practice of selling additional products or services to existing customers is called cross-selling. Giving product recommendation is one of the examples of cross-selling that are frequently used by online retailers. One simple method to give product recommendations is to recommend products that are frequently browsed together by the customers.

Suppose we want to recommend new products to the customer based on the products they have already browsed on the online website. Write a program using the A-priori algorithm to find products which are frequently browsed together. Fix the support to s = 100 (i.e. product pairs need to occur together at least 100 times to be considered frequent) and find itemsets of size 2 and 3.

Use the online browsing behavior dataset at:

https://www.dropbox.com/s/zlfyiwa70poqg74/ProductPurchaseData.txt?dl=0 (https://www.dropbox.com/s/zlfyiwa70poqg74/ProductPurchaseData.txt?dl=0)

Each line in this dataset represents a browsing session of a customer. On each line, each string of 8 characters represents the id of an item browsed during that session. The items are separated by spaces.

Do some exploratory data analysis of this dataset. Report your findings such as number of unique products; largest basket, etc. using Hadoop Map-Reduce.

Environment

Hadoop

- For the purpose of this assignment, Cloudera VM (in Mac) has been used which comes pre-packaged with CDH 4.7.
- Folder structure for input and output for HW3.1 and HW3.2

```
[cloudera@localhost wk3]$ hadoop fs -ls /user/cloudera/w261/wk3/*/input
Found 1 items
-rw-r--r--   3 cloudera cloudera    3458517 2015-09-21 13:15 /user/cloudera/w261/wk3/hw31/input/Pr
oductPurchaseData.txt
Found 1 items
-rw-r--r--   3 cloudera cloudera    3458517 2015-09-21 13:15 /user/cloudera/w261/wk3/hw32/input/Pr
oductPurchaseData.txt
[cloudera@localhost wk3]$ hadoop fs -ls /user/cloudera/w261/wk3/*/output
Found 3 items
-rw-r--r--   3 cloudera cloudera          0 2015-09-21 13:16 /user/cloudera/w261/wk3/hw31/outpu
t/_SUCCESS
drwxr-xr-x   - cloudera cloudera          0 2015-09-21 13:15 /user/cloudera/w261/wk3/hw31/outpu
t/_logs
-rw-r--r--   3 cloudera cloudera        116 2015-09-21 13:16 /user/cloudera/w261/wk3/hw31/output/p
art-00000
Found 3 items
-rw-r--r--   3 cloudera cloudera          0 2015-09-21 13:18 /user/cloudera/w261/wk3/hw32/outpu
t/_SUCCESS
drwxr-xr-x   - cloudera cloudera          0 2015-09-21 13:16 /user/cloudera/w261/wk3/hw32/outpu
t/_logs
-rw-r--r--   3 cloudera cloudera        161 2015-09-21 13:18 /user/cloudera/w261/wk3/hw32/output/p
art-00000
```

Amazon EC2

- For HW3.3, Ubuntu 1404 (Micro) VM was spinned up in Amazon EC2 Cluster (More details in that section) in order to run PyFim on

the Product Data.

In [1]:
```
%%writefile mapper_hw31.py

#!/usr/bin/env python

import sys
import re
import itertools

for line in sys.stdin:
    try:
        # Remove leading & trailing chars
        line = line.strip()
        # Split the line by <TAB> delimeter
        items = re.split(r'\s', line)

        for i in items:
            print '%s\t%d\t%d' %(i, 1, len(items))

        ''' Commenting out
        #Sort the list
        items.sort()

        for c in itertools.combinations(items, 1):
            print '%s,%s\t%d\t%d' %(c[0], '*', 1, len(items))

        for c in itertools.combinations(items, 2):
            print '%s,%s\t%d' %(c[0], c[1], 1)

        for c in itertools.combinations(items, 3):
            print '%s,%s,%s\t%d' %(c[0], c[1], c[2], 1)
        '''
    except Exception as e:
        print e
```

```
Overwriting mapper_hw31.py
```

In [2]:
```
!chmod a+x mapper_hw31.py
```

In [3]:
```
%%writefile reducer_hw31.py
#!/usr/bin/python
import sys
import re
from sets import Set

'''
a1 1 5
a1 1 5
a1 1 6
a2 1 18
'''

itemset_1_cnt = 0
item_last = None

THRESHOLD = 100

# Statistics
# Unique Items
uniq = Set()
# Max Basket Length
max_basket_len = 0
# Total Itemset Counts for Sizes: 1
total_itemset_1 = 0

for line in sys.stdin:
    # Remove leading & trailing chars
        line = line.strip()
        # Split the line by <TAB> delimeter
        tokens = re.split(r'\s', line)

        item = tokens[0]
        cnt = int(tokens[1])
        basket_len = int(tokens[2])

        if not item_last:
            item_last = item

        # Basket Length
```

```
        if basket_len > max_basket_len:
            max_basket_len = basket_len

        # Unique Items
        uniq.add(item)

        if item_last != item:
            # Check whether itemset 1 exceeds the support of 100
            if itemset_1_cnt >= THRESHOLD:
                total_itemset_1 += 1

            item_last = item
            itemset_1_cnt = cnt
        else:
            itemset_1_cnt += cnt

# Last Record
if itemset_1_cnt >= THRESHOLD:
    total_itemset_1 += 1

print '=== Statistics ==='
print 'Total Unique Items: %d' %(len(uniq))
print 'Maximum Basket Length: %d' %(max_basket_len)
print 'Total # frequent itemsets of size 1: %d' %(total_itemset_1)
```

```
Overwriting reducer_hw31.py
```

In [4]:
```
!chmod a+x reducer_hw31.py
```

In [8]:
```
'''
HW3.1. Product Recommendations
'''

# Delete existing Output Dirs if available
!hadoop fs -rm -r -skipTrash /user/cloudera/w261/wk3/hw31/output

# Run the Hadoop Streaming Command
!hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.0.0-mr1-cdh
4.7.0.jar \
-input /user/cloudera/w261/wk3/hw31/input/ProductPurchaseData.txt \
-output /user/cloudera/w261/wk3/hw31/output \
-file ./mapper_hw31.py \
-mapper 'python mapper_hw31.py' \
-file ./reducer_hw31.py \
-reducer 'python reducer_hw31.py'

# Show Output
!hadoop fs -cat /user/cloudera/w261/wk3/hw31/output/part-00000
```

```
Deleted /user/cloudera/w261/wk3/hw31/output
packageJobJar: [./mapper_hw31.py, ./reducer_hw31.py, /tmp/hadoop-cloudera/hadoop-unjar127859
6197593495009/] [] /tmp/streamjob4204464395455964255.jar tmpDir=null
15/09/21 13:15:49 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
15/09/21 13:15:49 INFO mapred.FileInputFormat: Total input paths to process : 1
15/09/21 13:15:50 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-cloudera/mapred/loc
al]
15/09/21 13:15:50 INFO streaming.StreamJob: Running job: job_201509211304_0004
15/09/21 13:15:50 INFO streaming.StreamJob: To kill this job, run:
15/09/21 13:15:50 INFO streaming.StreamJob: UNDEF/bin/hadoop job  -Dmapred.job.tracker=local
host.localdomain:8021 -kill job_201509211304_0004
15/09/21 13:15:50 INFO streaming.StreamJob: Tracking URL: http://0.0.0.0:50030/jobdetails.js
p?jobid=job_201509211304_0004
15/09/21 13:15:51 INFO streaming.StreamJob:  map 0%  reduce 0%
15/09/21 13:16:10 INFO streaming.StreamJob:  map 100%  reduce 0%
15/09/21 13:16:23 INFO streaming.StreamJob:  map 100%  reduce 100%
15/09/21 13:16:26 INFO streaming.StreamJob: Job complete: job_201509211304_0004
15/09/21 13:16:26 INFO streaming.StreamJob: Output: /user/cloudera/w261/wk3/hw31/output
=== Statistics ===
Total Unique Items: 12592
Maximum Basket Length: 37
Total # frequent itemsets of size 1: 647
```

HW3.2. (Computationally prohibitive but then again Hadoop can handle this)

Note: for this part the writeup will require a specific rule ordering but the program need not sort the output.

List the top 5 rules with corresponding confidence scores in decreasing order of confidence score for frequent (100>count) itemsets of size 2. A rule is of the form:

(item1) ⇒ item2.

Fix the ordering of the rule lexicographically (left to right), and break ties in confidence (between rules, if any exist) by taking the first ones in lexicographically increasing order. Use Hadoop MapReduce to complete this part of the assignment; use a single mapper and single reducer; use a combiner if you think it will help and justify.

In [11]:
```
%%writefile mapper_hw32.py

#!/usr/bin/env python

import sys
import re
import itertools

for line in sys.stdin:
    try:
        # Remove leading & trailing chars
        line = line.strip()
        # Split the line by <TAB> delimeter
        items = re.split(r'\s+', line)
        #Sort the list
        items.sort()

        l = len(items)

        for i in xrange(l):
            print '%s,*\t%d' %(items[i], 1)
            for j in xrange(i+1, l):
               print '%s,%s\t%d' %(items[i], items[j], 1)
               print '%s,%s\t%d' %(items[j], items[i], 1)
    except Exception as e:
        print e
```

```
Overwriting mapper_hw32.py
```

In [12]:
```
!chmod a+x mapper_hw32.py
```

In [13]:
```
%%writefile reducer_hw32.py
#!/usr/bin/python
import sys
import re
import heapq

itemset_1_cnt = 0
itemset_2_cnt = 0

itemset_1_last = None
itemset_2_last = None

'''
a1,* 1
a1,* 1
a1,b1 1
a1,b1 1
a1,b2 1
a1,b2 1
a2,* 1
'''

THRESHOLD = 100
# Store Itemsets 2
dict = {}

for line in sys.stdin:
    # Remove leading & trailing chars
        line = line.strip()
        # Split the line by <TAB> delimeter
        tokens = re.split(r'\s+', line)

        # Split the key by <COMMA> delimeter
        items = tokens[0].split(",")
        i1 = items[0]
        i2 = items[1]

        if not itemset_1_last:
            itemset_1_last = i1
```

```
        if itemset_1_last != i1:
            '''
            if itemset_1_cnt >= THRESHOLD:
                confidence = (itemset_2_cnt * 1.0) / itemset_1_cnt
                print '[%d,%d]%s\t%f' %(itemset_1_cnt, itemset_2_cnt, tokens[0], confidence)
                dict[tokens[0]] = confidence
            '''

            # Reset
            itemset_1_last = i1
            itemset_1_cnt = int(tokens[1])
            itemset_2_last = None
            itemset_2_cnt = 0
        else:
            if i2 == '*':
                itemset_1_cnt += int(tokens[1])
            else:
                if itemset_2_last != tokens[0]:
                    if itemset_1_cnt >= THRESHOLD and itemset_2_cnt >= THRESHOLD:
                        confidence = (itemset_2_cnt * 1.0) / itemset_1_cnt
                        #print '[%d,%d]%s\t%f' %(itemset_1_cnt, itemset_2_cnt, itemset_2_last,
confidence)
                        dict[itemset_2_last] = confidence
                    itemset_2_last = tokens[0]
                    itemset_2_cnt = int(tokens[1])
                else:
                    itemset_2_cnt += int(tokens[1])

# Last Set of Counts
if itemset_1_cnt >= THRESHOLD and itemset_2_cnt >= THRESHOLD:
    confidence = (itemset_2_cnt * 1.0) / itemset_1_cnt
    #print '[%d,%d]%s\t%f' %(itemset_1_cnt, itemset_2_cnt, itemset_2_last, confidence)
    dict[itemset_2_last] = confidence

print '=== Top 5 Confidence ==='
sorted_dict = sorted(dict.items(), key=lambda x:(-x[1], x[0]))
for j,k in sorted_dict[:5]:
    print '%s\t%f' %(j,k)
```

```
```

```
Overwriting reducer_hw32.py
```

In [14]:
```
!chmod a+x reducer_hw32.py
```

In [15]:
```
'''
HW3.2. Confidence Caculations
List the top 5 rules with corresponding confidence scores in decreasing order of confidence sc
ore
for frequent (100>count) itemsets of size 2
'''

# Delete existing Output Dirs if available
!hadoop fs -rm -r -skipTrash /user/cloudera/w261/wk3/hw32/output

# Run the Hadoop Streaming Command
!hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.0.0-mr1-cdh
4.7.0.jar \
-D map.output.key.field.separator=, \
-D mapred.text.key.partitioner.options=-k1,1 \
-input /user/cloudera/w261/wk3/hw32/input/ProductPurchaseData.txt \
-output /user/cloudera/w261/wk3/hw32/output \
-file ./mapper_hw32.py \
-mapper 'python mapper_hw32.py' \
-file ./reducer_hw32.py \
-reducer 'python reducer_hw32.py'

# Show Output
!hadoop fs -cat /user/cloudera/w261/wk3/hw32/output/part-00000
```

```
Deleted /user/cloudera/w261/wk3/hw32/output
packageJobJar: [./mapper_hw32.py, ./reducer_hw32.py, /tmp/hadoop-cloudera/hadoop-unjar186729
1528774777589/] [] /tmp/streamjob8875451858342305964.jar tmpDir=null
15/09/21 13:16:53 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
15/09/21 13:16:53 INFO mapred.FileInputFormat: Total input paths to process : 1
15/09/21 13:16:54 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-cloudera/mapred/loc
al]
15/09/21 13:16:54 INFO streaming.StreamJob: Running job: job_201509211304_0005
15/09/21 13:16:54 INFO streaming.StreamJob: To kill this job, run:
15/09/21 13:16:54 INFO streaming.StreamJob: UNDEF/bin/hadoop job  -Dmapred.job.tracker=local
host.localdomain:8021 -kill job_201509211304_0005
15/09/21 13:16:54 INFO streaming.StreamJob: Tracking URL: http://0.0.0.0:50030/jobdetails.js
p?jobid=job_201509211304_0005
15/09/21 13:16:55 INFO streaming.StreamJob:  map 0%  reduce 0%
15/09/21 13:17:11 INFO streaming.StreamJob:  map 19%  reduce 0%
15/09/21 13:17:12 INFO streaming.StreamJob:  map 34%  reduce 0%
15/09/21 13:17:14 INFO streaming.StreamJob:  map 53%  reduce 0%
15/09/21 13:17:18 INFO streaming.StreamJob:  map 72%  reduce 0%
15/09/21 13:17:21 INFO streaming.StreamJob:  map 91%  reduce 0%
15/09/21 13:17:23 INFO streaming.StreamJob:  map 95%  reduce 0%
15/09/21 13:17:24 INFO streaming.StreamJob:  map 100%  reduce 0%
15/09/21 13:17:37 INFO streaming.StreamJob:  map 100%  reduce 68%
15/09/21 13:17:44 INFO streaming.StreamJob:  map 100%  reduce 69%
15/09/21 13:17:50 INFO streaming.StreamJob:  map 100%  reduce 70%
15/09/21 13:17:56 INFO streaming.StreamJob:  map 100%  reduce 71%
15/09/21 13:18:02 INFO streaming.StreamJob:  map 100%  reduce 72%
15/09/21 13:18:04 INFO streaming.StreamJob:  map 100%  reduce 100%
15/09/21 13:18:07 INFO streaming.StreamJob: Job complete: job_201509211304_0005
15/09/21 13:18:07 INFO streaming.StreamJob: Output: /user/cloudera/w261/wk3/hw32/output
=== Top 5 Confidence ===
DAI93865,FRO40251       1.000000
GRO85051,FRO40251       0.999176
GRO38636,FRO40251       0.990654
ELE12951,FRO40251       0.990566
DAI88079,FRO40251       0.986726
```

HW3.3

Benchmark your results using the pyFIM implementation of the Apriori algorithm (Apriori - Association Rule Induction / Frequent Item Set Mining implemented by Christian Borgelt). You can download pyFIM from here:*

http://www.borgelt.net/pyfim.html (http://www.borgelt.net/pyfim.html)

Comment on the results from both implementations (your Hadoop MapReduce of apriori versus pyFIM) in terms of results and execution times.

For this part, the following steps were performed:

1. Since I was using a Mac, I spinned up a Ubuntu 1404 (Micro) VM in Amazon EC2 Cluster
2. Installed all required libraries: pip, git, ipython
3. Downloaded the fim.so file from the link & set PYTHONPATH & LD_LIBRARY_PATH
4. Ran the following command in the VM

```
In [16]: !python top5pyfim.py | sort -n -r -k 3
```

```
FRO40251    ('DAI93865',)    1.0
FRO40251    ('GRO85051',)    0.999176276771
FRO40251    ('GRO38636',)    0.990654205607
FRO40251    ('ELE12951',)    0.990566037736
FRO40251    ('DAI88079',)    0.986725663717
FRO40251    ('FRO92469',)    0.983510011779
SNA82528    ('DAI43868',)    0.972972972973
DAI62779    ('DAI23334',)    0.954545454545
```

Top 5 Comparison from HW3.2 (Map/Reduce) and HW3.3 (PyFim Apriori)

Map/Reduce

```
DAI93865,FRO40251    1.000000
GRO85051,FRO40251    0.999176
GRO38636,FRO40251    0.990654
ELE12951,FRO40251    0.990566
DAI88079,FRO40251    0.986726
```

HW3.3 (PyFim Apriori)

```
FRO40251    ('DAI93865',)    1.0
FRO40251    ('GRO85051',)    0.999176276771
FRO40251    ('GRO38636',)    0.990654205607
FRO40251    ('ELE12951',)    0.990566037736
FRO40251    ('DAI88079',)    0.986725663717
```

As it can be seen from above, both the results are identical with each other

- The Map/Reduce approach is much more scalable, even though it took longer to execute since there is a start up cost for Hadoop Map Reduce Jobs. As the file size gets bigger, the Map Reduce would perform much better than using PyFim.

HW3.4 (Conceptual Exercise)

Suppose that you wished to perform the Apriori algorithm once again, though this time now with the goal of listing the top 5 rules with corresponding confidence scores in decreasing order of confidence score for itemsets of size 3 using Hadoop MapReduce. A rule is now of the form:

(item1, item2) ⇒ item3

Recall that the Apriori algorithm is iterative for increasing itemset size, working off of the frequent itemsets of the previous size to explore ONLY the NECESSARY subset of a large combinatorial space. Describe how you might design a framework to perform this exercise.

In particular, focus on the following: — map-reduce steps required

- enumeration of item sets and filtering for frequent candidates

- In order to solve this problem a chain of Hadoop Map Reduce Jobs need to be used. The design would be something as follows:

Input Document: D

```
item1 item2 item3 item4 item5
item1 item2 item4 item5
item3 item4 item5
.
.
.
```

1st Pass:

- **Mapper**: Read Document line by line and emit items as (item, 1)
- **Reducer**: Sum the items and emit only those that are > support count (Ex: s=100). At the end of the first pass the reducer file (L1) would look like the following. Since the output from Reducer is not supposed to be a big file, the number of reducer would be passed as 1 to the Hadoop Job, so that only one output file is generated from this pass.

```
item1, 102
item3, 105
```

2nd Pass:

- **Mapper**: The output file from 1st pass would be passed to the Mapper in the form of a Distributed Cache. Assuming this file is small (Typical Super Store doesn't have more than 10,000 or so frequently sold items) it is loaded into the memory of the m/c(s) runnning mappers. The mapper would load L1 into a Dictionary (Ex: key=item1, value=102 etc). It would also create a candidate generation C2 by taking all combination pairs from L1 and create another Dictionary where keys are candidate pairs;values can be anyting like '-'. C2 would look like the following:

```
(item1, item2) -
(item1, item3) -
```

  The mapper would then read portion of the Document D passed to it, and emit only those pairs which are present in C2 (Candidates). Output of mapper would have key like "item1,item3" and value as "1".
- **Reducer**: Hadoop framework would make sure Reducer receives the values for the same keys. The job of the reducer would be to

sum the values for same keys, and emit only those that are > support count (Ex: s=100), just like in the 1st pass. Since the output from Reducer is not supposed to be a very big file (Though it will be much bigger than the file from 1st pass), the number of reducer would be passed as 1 to the Hadoop Job, so that only one output file is generated from this pass. The output of the reducer (L2) would look like the following.

```
item1,item2 150
item1,item3 240
```

3rd Pass:

- **Mapper**: The output file from 2nd pass would be passed to the Mapper in the form of a Distributed Cache. Assuming this file is small it is loaded into the memory, as a Dictionary, of the m/c(s) runnning mappers. The mapper would read portion of the Document D passed to it, read line by line, and emit the following as long as the line has the candidate pairs. The mapper would emit something like the following if a candidate pair is present in the Document D:

```
item1,item2,* 150
item1,item2,item4 1
```

- **Reducer**: Paritioning has to be done based on the leftmost item pair (item1,item2), which would ensure all records with the same item pair would be passed to the same reducer. And secondary sort would ensure that the reducer would get to read the total (item1,item2,* 150) before it starts calculating the confidence scores. Each reducer would need to store the confidence scores in memory to be able to spit out the top 5 once the processing is done. If the number of records exceeds the memory of the reducer, then top 5 has to be recomputed every time a confidence score is calculated by the reducer.