DATASCI W261, Machine Learning at Scale

Assignement: week #3

Lei Yang (mailto:leiyang@berkeley.edu) | Michael Kennedy (mailto:mkennedy@ischool.berkeley.edu) | Natarajan Krishnaswami (mailto:natarajan@krishnaswami.org)

Due: 2016-02-02, 8AM PST

HW3.0. Q&A

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

Merge sort is a sorting algorithm which quickly combines two sorted lists into a single list of items. Merge sort benefits from distributable in its least efficient step, which is the sorting of the child lists. The merging of child lists into a single sorted list is done in linear time. Merge sorting is used in the shuffle stage of Hadoop to rearrange keys prior to sending them to the reducer. Key-value pairs from different mappers are sorted at their mappers, and then distributed across the reducers in a sorted form.

How is a combiner function used in the context of Hadoop?

Combiners are used for local aggregation during the mapper processes of Hadoop. They are run when the incomplete output from the mapper becomes too large to fit within memory and "spills over" onto disk. The combiner is responsible for shrinking the data back down so that the mapper can run faster by keeping data in memory and so that the network operations in the partitioner are kept to a bare minimum. Depending on the size and scope of the problem, Hadoop will run combiners any number of times including zero with no input from the user. For this reason, it is critical that the combiner is able to receive records in the format of the mapper's output and emit data in the same format. The combining operation must also be associative and commutative so that the variable number of runs will not affect the result.

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

Combiners can be used in long word-count operations. A typical mapper output for a word-count problem will be greater than the size of the document since it emits each individual word and the number associated with it. Transferring this data across the network can drastically reduce the performance of this operation, as well as making the subsequent sorting operation take much longer. Adding a combiner can reduce the size of the mapper output from being tied to the size of the document to being tied to the size of the vocabulary.

What is the Hadoop shuffle?

Shuffle happens after all mapper processes complete and before reducer starts, all key-value pairs are sorted by key, and the same key is guaranteed to be delivered to the same reducer.

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

- Aprior algorithm is used to find frequent itemsets, each iteration has two scans of data and a filtering in between:
  1. generate a condidate set C_k for itemsets of size k, based on the output of previous iteration L_{k-1}
  2. remove all members from the set whose support is less than the user specified threshold s_i
  3. generate the final set L_k for frequent itemset of size k, based on output after filtering
- For example, to find itemsets of size k from a basket set, the process is:
  1. count all single words from all baskets, output C_1
  2. remove all words with support below threshold, output L_1
  3. using L_1, generate candidate set for frequent pair set C_2
  4. remove all pairs with support below threshold, get L_2
  5. using L_2, generate candidate set for frequent triple set C_3
  6. remove all triples with support below threshold, get L_3
- Confidence is the relative frequency of an association rule, for example:
  - if the count of triple (a,b,c) is n
  - and the count of pair (a,b) is m
  - the confidence of rule (a,b) => c is $\frac{n}{m}$

start yarn, hdfs, and job history

In [ ]:
```
!/usr/local/Cellar/hadoop/2*/sbin/start-yarn.sh
```

```
!/usr/local/Cellar/hadoop/2*/sbin/start-dfs.sh
!/usr/local/Cellar/hadoop/2*/sbin/mr-jobhistory-daemon.sh --config /usr/local/Cellar/hadoop/2*/lib
exec/etc/hadoop/ start historyserver
```

HW3.1. Use Counters to do EDA (exploratory data analysis and to monitor progress)

Counters are lightweight objects in Hadoop that allow you to keep track of system progress in both the map and reduce stages of processing. By default, Hadoop defines a number of standard counters in "groups"; these show up in the jobtracker webapp, giving you information such as "Map input records", "Map output records", etc.

While processing information/data using MapReduce job, it is a challenge to monitor the progress of parallel threads running across nodes of distributed clusters. Moreover, it is also complicated to distinguish between the data that has been processed and the data which is yet to be processed. The MapReduce Framework offers a provision of user-defined Counters, which can be effectively utilized to monitor the progress of data across nodes of distributed clusters.

Use the Consumer Complaints Dataset provide here (https://www.dropbox.com/s/vbalm3yva2rr86m/Consumer_Complaints.csv?dl=0) to complete this question:

- The consumer complaints dataset consists of diverse consumer complaints, which have been reported across the United States regarding various types of loans. The dataset consists of records of the form:
  - Complaint ID,Product,Sub-product,Issue,Sub-issue,State,ZIP code,Submitted via,Date received,Date sent to company,Company,Company response,Timely response?,Consumer disputed?

Here's is the first few lines of the of the Consumer Complaints Dataset:

- Complaint ID,Product,Sub-product,Issue,Sub-issue,State,ZIP code,Submitted via,Date received,Date sent to company,Company,Company response,Timely response?,Consumer disputed?
- 1114245,Debt collection,Medical,Disclosure verification of debt,Not given enough info to verify debt,FL,32219,Web,11/13/2014,11/13/2014,"Choice Recovery, Inc.",Closed with explanation,Yes,
- 1114488,Debt collection,Medical,Disclosure verification of debt,Right to dispute notice not received,TX,75006,Web,11/13/2014,11/13/2014,"Expert Global Solutions, Inc.",In progress,Yes,
- 1114255,Bank account or service,Checking account,Deposits and withdrawals,,NY,11102,Web,11/13/2014,11/13/2014,"FNIS (Fidelity National Information Services, Inc.)",In progress,Yes,
- 1115106,Debt collection,"Other (phone, health club, etc.)",Communication tactics,Frequent or repeated calls,GA,31721,Web,11/13/2014,11/13/2014,"Expert Global Solutions, Inc.",In progress,Yes,

User-defined Counters

- Now, let's use Hadoop Counters to identify the number of complaints pertaining to *debt collection*, *mortgage* and *other* categories (all other categories get lumped into this one) in the consumer complaints dataset. Basically produce the distribution of the Product column in this dataset using counters (limited to 3 counters here).
- Hadoop offers Job Tracker, an UI tool to determine the status and statistics of all jobs. Using the job tracker UI, developers can view the Counters that have been created. Screenshot your job tracker UI as your job completes and include it here. Make sure that your user defined counters are visible.

HW3.1 Answer:

Mapper

- as the shuffler will do the sorting, mapper just need to emit word with integer as the key

In [2]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
for line in sys.stdin:
    # extract the column values
    parts = line.strip().split(',')
    # product is in second column
    prod = parts[1].strip().lower()
    # emit product name as key, no need for value as we are only count product name
    print "%s\t%s" %(prod, 'na')
```

```
Overwriting mapper.py
```

Reducer

In [3]:
```
%%writefile reducer.py
```

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    # product name
    prod = line.split('\t')[0].strip()

    # compare with what we want to count and adjust the counter
    if prod == 'debt collection':
        sys.stderr.write("reporter:counter:HW3_1,debt,1\n")
    elif prod == 'mortgage':
        sys.stderr.write("reporter:counter:HW3_1,mortgage,1\n")
    else:
        sys.stderr.write("reporter:counter:HW3_1,others,1\n")
```

```
Overwriting reducer.py
```

Run the job with Hadoop Streaming

- add parameter *-D mapred.reduce.tasks=2* to specify number of reducers

In [4]:
```
!hdfs dfs -rm -r results
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.reduce.tasks=2 \
-files mapper.py,reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-input /user/lei/Consumer_Complaints.csv \
-output results
```

```
Deleted results
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar3709839099986008331/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob3447758072235333814.jar tmpDir=null
```

Check counter value

	Name	Map	Reduce	Total
HW3_1	debt	0	44,372	44,372
	mortgage	0	125,752	125,752
	others	0	142,789	142,789

HW3.2. Analyze the performance of your Mappers, Combiners and Reducers using Counters

For this brief study the Input file will be one record (the next line only):

foo foo quux labs foo bar quux

- Perform a word count analysis of this single record dataset using a Mapper and Reducer based WordCount (i.e., no combiners are used here) using user defined Counters to count up how many time the mapper and reducer are called. What is the value of your user defined Mapper Counter, and Reducer Counter after completing this word count job. The answer should be 1 and 4 respectively. Please explain.

Mapper

In [5]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys, re, string

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_2,Mapper_cnt,1\n")

# input comes from STDIN (standard input)
for line in sys.stdin:

    # split the line into words
    words = line.split()

    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
```

```
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

```
Overwriting mapper.py
```

Reducer

In [6]:
```
%%writefile reducer.py
#!/usr/bin/python
from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# increase counter for reducer being called
sys.stderr.write("reporter:counter:HW3_2,Reducer_cnt,1\n")

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # print out count
            print '%s\t%s' %(current_word, current_count)
        current_count = count
        current_word = word

# do not forget to print the last word count if needed!
if current_word == word:
    print '%s\t%s' %(current_word, current_count)
```

```
Overwriting reducer.py
```

Write the file and put on HDFS

In [85]:
```
%%writefile wordcount.txt
foo foo quux labs foo bar quux
```

```
Overwriting wordcount.txt
```

In [86]:
```
!hdfs dfs -rm /user/lei/wordcount.txt
!hdfs dfs -put wordcount.txt /user/lei
```

```
Deleted /user/lei/wordcount.txt
```

Run the job with Hadoop streaming

In [7]:
```
!hdfs dfs -rm -r results
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.map.tasks=1 \
-D mapred.reduce.tasks=4 \
-files mapper.py,reducer.py \
-mapper mapper.py \
-reducer reducer.py \
```

```
-reducer reducer.py \
-input '/user/lei/wordcount.txt' \
-output results
```

```
Deleted results
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar7556678054711269180/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob3086524328830560864.jar tmpDir=null
```

HW3.2 Results:

	Name	Map	Reduce	Total
HW3_2	Mapper_cnt	1	0	1
	Reducer_cnt	0	4	4

HW3.2 Exploratory analysis on consumer complaint data

Please use mulitple mappers and reducers for these jobs (at least 2 mappers and 2 reducers).

- Perform a word count analysis of the Issue column of the Consumer Complaints Dataset using a Mapper and Reducer based WordCount (i.e., no combiners used anywhere) using user defined Counters to count up how many time the mapper and reducer are called. What is the value of your user defined Mapper Counter, and Reducer Counter after completing your word count job.

Mapper

In [8]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys, re, string
# define regex for punctuation removal
regex = re.compile('[%s]' % re.escape(string.punctuation))

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_2,Mapper_cnt,1\n")

for line in sys.stdin:
    # extract the column values
    parts = line.strip().split(',')
    # issue is in 4th column
    issue = parts[3].strip().lower()
    # emit issue as key, and 1 as count
    print "%s,%s" %(regex.sub('', issue), '1')
```

```
Overwriting mapper.py
```

Reducer

In [9]:
```
%%writefile reducer.py
#!/usr/bin/python
from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# increase counter for reducer being called
sys.stderr.write("reporter:counter:HW3_2,Reducer_cnt,1\n")

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split(',', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue
```

```
    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # print out count
            print '%s,%s' %(current_word, current_count)
        current_count = count
        current_word = word

# do not forget to print the last word count if needed!
if current_word == word:
    print '%s,%s' %(current_word, current_count)
```

```
Overwriting reducer.py
```

Run the job with Hadoop streaming

In [10]:
```
!hdfs dfs -rm -r results
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.map.tasks=4 \
-D mapred.reduce.tasks=2 \
-files mapper.py,reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-input '/user/lei/Consumer_Complaints.csv' \
-output results
```

```
Deleted results
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar5125320870294825413/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob2595898526152944027.jar tmpDir=null
```

HW3.2 Results:

we can see that the counter values are consistent with our specification of times for mapper and reducer to be called.

	Name	Map	Reduce	Total
HW3_2	Mapper_cnt	4	0	4
	Reducer_cnt	0	2	2

And the issue counts are below:

In [11]:
```
!hdfs dfs -cat /user/leiyang/results/part-0000*
```

```
account terms and changes,350
application processing delay,243
application,8625
apr or interest rate,3431
billing disputes,6938
billing statement,1220
cant contact lender,221
closingcancelling account,2795
collection practices,1003
convenience checks,75
credit card protection  debt protection,1343
credit determination,1490
customer service  customer relations,1367
dealing with my lender or servicer,1944
delinquent account,1061
deposits and withdrawals,10555
disclosure verification of debt,5214
health club,12545
improper contact or sharing of info,2832
incorrectmissing disclosures or info,64
late fee,1797
loan modification,70487
loan servicing,36767
makingreceiving payments,3226
managing the loan or lease,4560
money was not available when promised,274
other fee,1075
other transaction issues,387
other,6273
payoff process,1155
```

```
privacy,240
repaying your loan,3844
rewards,1002
shopping for a line of credit,137
taking out the loan or lease,1242
takingthreatening an illegal action,2505
unable to get credit reportcredit score,4357
unsolicited issuance of credit card,640
using a debit or atm card,2422
account opening,16205
advertising and marketing,1193
applied for loandid not receive money,139
arbitration,168
balance transfer fee,95
balance transfer,502
bankruptcy,222
cant repay my loan,1647
cant stop charges to bank account,131
cash advance fee,104
cash advance,136
charged bank acct wrong day or amt,71
charged fees or interest i didnt expect,807
collection debt dispute,904
communication tactics,6920
contd attempts collect debt not owed,11848
credit decision  underwriting,2774
credit line increasedecrease,1149
credit monitoring or identity protection,1453
credit reporting companys investigation,4858
credit reporting,1701
false statements or representation,2508
forbearance  workout plans,350
fraud or scam,566
getting a loan,291
identity theft  fraud  embezzlement,3276
improper use of my credit report,1477
incorrect information on credit report,29069
issue,1
managing the line of credit,446
other service issues,151
overlimit fee,127
payment to acct not credited,92
problems caused by my funds being low,5663
problems when you are unable to pay,3821
received a loan i didnt apply for,118
sale of account,139
settlement process and costs,4350
shopping for a loan or lease,535
transaction issue,1098
wrong amount charged or received,98
```

HW3.2 Exploratory analysis on consumer complaint data

Please use mulitple mappers and reducers for these jobs (at least 2 mappers and 2 reducers).

- Perform a word count analysis of the Issue column of the Consumer Complaints Dataset using a Mapper, Reducer, and standalone combiner (i.e., not an in-memory combiner) based WordCount using user defined Counters to count up how many time the mapper, combiner, reducer are called. What is the value of your user defined Mapper Counter, and Reducer Counter after completing your word count job.

The definitions of mapper and reducer don't need to change in this case, we can just use the reducer as a standalone combiner, specified by Hadoop-streaming parameter (-combiner)

In [12]:
```
!hdfs dfs -rm -r results
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.map.tasks=4 \
-D mapred.reduce.tasks=2 \
-files mapper.py,reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-combiner reducer.py \
-input '/user/lei/Consumer_Complaints.csv' \
-output results
```

```
Deleted results
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar2110887529466242773/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob4709300790161981780.jar tmpDir=null
```

HW3.2 Results:

We can see that the reducer.py was called 8 times during map step as **combiner**, and 2 times during reduce step as **reducer**.

	Name	Map	Reduce	Total
HW3_2	Mapper_cnt	4	0	4
	Reducer_cnt	8	2	10

And the issue counts from two reducers are below:

```
In [13]: !hdfs dfs -cat /user/leiyang/results/part-0000*
```

```
account terms and changes,350
application processing delay,243
application,8625
apr or interest rate,3431
billing disputes,6938
billing statement,1220
cant contact lender,221
closingcancelling account,2795
collection practices,1003
convenience checks,75
credit card protection  debt protection,1343
credit determination,1490
customer service  customer relations,1367
dealing with my lender or servicer,1944
delinquent account,1061
deposits and withdrawals,10555
disclosure verification of debt,5214
health club,12545
improper contact or sharing of info,2832
incorrectmissing disclosures or info,64
late fee,1797
loan modification,70487
loan servicing,36767
makingreceiving payments,3226
managing the loan or lease,4560
money was not available when promised,274
other fee,1075
other transaction issues,387
other,6273
payoff process,1155
privacy,240
repaying your loan,3844
rewards,1002
shopping for a line of credit,137
taking out the loan or lease,1242
takingthreatening an illegal action,2505
unable to get credit reportcredit score,4357
unsolicited issuance of credit card,640
using a debit or atm card,2422
account opening,16205
advertising and marketing,1193
applied for loandid not receive money,139
arbitration,168
balance transfer fee,95
balance transfer,502
bankruptcy,222
cant repay my loan,1647
cant stop charges to bank account,131
cash advance fee,104
cash advance,136
charged bank acct wrong day or amt,71
charged fees or interest i didnt expect,807
collection debt dispute,904
communication tactics,6920
contd attempts collect debt not owed,11848
credit decision  underwriting,2774
credit line increasedecrease,1149
credit monitoring or identity protection,1453
credit reporting companys investigation,4858
credit reporting,1701
false statements or representation,2508
forbearance  workout plans,350
```

```
fraud or scam,566
getting a loan,291
identity theft  fraud  embezzlement,3276
improper use of my credit report,1477
incorrect information on credit report,29069
issue,1
managing the line of credit,446
other service issues,151
overlimit fee,127
payment to acct not credited,92
problems caused by my funds being low,5663
problems when you are unable to pay,3821
received a loan i didnt apply for,118
sale of account,139
settlement process and costs,4350
shopping for a loan or lease,535
transaction issue,1098
wrong amount charged or received,98
```

HW3.2 Exploratory analysis on consumer complaint data

- Using a single reducer: What are the top 50 most frequent terms in your word count analysis?
- Present the top 50 terms and their frequency and their relative frequency.
- If there are ties please sort the tokens in alphanumeric/string order. Present bottom 10 tokens (least frequent items).

Notes:

- for a single reducer (job) to get list of relative frequencies, we need to implement **order inversion** to get total count first.
- **mapper** will emit **'dummy_sort_key, issue_name / *, count'**, as it is impossible to sort count with secondary sorting if we use the issue name as partitioner option.
- we need to sort numerically of the count, and in the mean time guarantee the emits for total calculation **(key, *, count)** arrive first, thus we define *-inf* as the dummy sort key for those emits, as other counts are always postive.
- **reducer** will get total count first, then joint count for each word, and finally relative frequency. It needs to be a generic process such that if the combiner is not called, the final results would still be correct.
- specify secondary sort on issue name

Mapper

In [14]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys, re, string
# define regex for punctuation removal
regex = re.compile('[%s]' % re.escape(string.punctuation))

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_2,Mapper_cnt,1\n")

for line in sys.stdin:
    # get issue count and name
    issue, count = line.strip().split(',')
    # emit issue as key, and 1 as count
    print "%s,%s,%s" %(count, issue, count)
    # for order inversion, to calculate total count
    print '%s,%s,%s' %('-3', '*', count)

# test for tie-break
#print '%s,%s,%s' %(1, 'zzz', 3)
#print '%s,%s,%s' %(1, 'oko', 3)
#print '%s,%s,%s' %(1, 'ccc', 3)
```

```
Overwriting mapper.py
```

Reducer

In [15]:
```
%%writefile reducer.py
#!/usr/bin/python
from operator import itemgetter
import sys


# buffer for top and bottom
n_bottom, n_top = 10, 50
bottom, top = [], []
```

```
bottom, top = [], []
n_total = 0

# input comes from STDIN
for line in sys.stdin:
    dummy, issue, count = line.strip().split(',', 2)

    # skip bad count
    try:
        count = int(count)
    except ValueError:
        continue

    # get total count
    if '*' == issue:
        n_total += count
        continue

    # calculate relative frequency
    rf = 1.0*count/n_total

    # buffer top and bottom
    if len(bottom) < n_bottom:
        bottom.append([issue, count, rf])

    if len(top) < n_top:
        top.append([issue, count, rf])
    else:
        top = top[1:] + [[issue, count, rf]]

# print results:
top.reverse()
print '\ntop %d issues:' %n_top
for rec in top:
    print '%.2f%%\t%d\t%s' %(100*rec[2], rec[1], rec[0])

print '\nbottom %d issues:' %n_bottom
for rec in bottom:
    print '%.2f%%\t%d\t%s' %(100*rec[2], rec[1], rec[0])
```

```
Overwriting reducer.py
```

Run the job with Hadoop streaming

In [16]:
```
# assuming count results are available
!hdfs dfs -rm -r results2
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
-D map.output.key.field.separator=, \
-D map.output.key.value.fields.spec=0-1:0- \
-D mapred.text.key.comparator.options='-k1,1n -k2,2' \
-D mapred.map.tasks=2 \
-D mapred.reduce.tasks=1 \
-files mapper.py,reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-input /user/leiyang/results/part-0000* \
-output results2
```

```
Deleted results2
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar1115560675774446028/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob3811758649004660161.jar tmpDir=null
```

The sorted top and bottom issues are:

In [17]:
```
!hdfs dfs -cat /user/leiyang/results2/part-0000*
```

```
top 50 issues:
22.53%  70487   loan modification
11.75%  36767   loan servicing
9.29%   29069   incorrect information on credit report
5.18%   16205   account opening
4.01%   12545   health club
3.79%   11848   contd attempts collect debt not owed
```

```
3.37%   10555   deposits and withdrawals
2.76%   8625    application
2.22%   6938    billing disputes
2.21%   6920    communication tactics
2.00%   6273    other
1.81%   5663    problems caused by my funds being low
1.67%   5214    disclosure verification of debt
1.55%   4858    credit reporting companys investigation
1.46%   4560    managing the loan or lease
1.39%   4357    unable to get credit reportcredit score
1.39%   4350    settlement process and costs
1.23%   3844    repaying your loan
1.22%   3821    problems when you are unable to pay
1.10%   3431    apr or interest rate
1.05%   3276    identity theft  fraud  embezzlement
1.03%   3226    makingreceiving payments
0.91%   2832    improper contact or sharing of info
0.89%   2795    closingcancelling account
0.89%   2774    credit decision  underwriting
0.80%   2508    false statements or representation
0.80%   2505    takingthreatening an illegal action
0.77%   2422    using a debit or atm card
0.62%   1944    dealing with my lender or servicer
0.57%   1797    late fee
0.54%   1701    credit reporting
0.53%   1647    cant repay my loan
0.48%   1490    credit determination
0.47%   1477    improper use of my credit report
0.46%   1453    credit monitoring or identity protection
0.44%   1367    customer service  customer relations
0.43%   1343    credit card protection  debt protection
0.40%   1242    taking out the loan or lease
0.39%   1220    billing statement
0.38%   1193    advertising and marketing
0.37%   1155    payoff process
0.37%   1149    credit line increasedecrease
0.35%   1098    transaction issue
0.34%   1075    other fee
0.34%   1061    delinquent account
0.32%   1003    collection practices
0.32%   1002    rewards
0.29%   904     collection debt dispute
0.26%   807     charged fees or interest i didnt expect
0.20%   640     unsolicited issuance of credit card

bottom 10 issues:
0.00%   1       issue
0.02%   64      incorrectmissing disclosures or info
0.02%   71      charged bank acct wrong day or amt
0.02%   75      convenience checks
0.03%   92      payment to acct not credited
0.03%   95      balance transfer fee
0.03%   98      wrong amount charged or received
0.03%   104     cash advance fee
0.04%   118     received a loan i didnt apply for
0.04%   127     overlimit fee
```

3.2.1 OPTIONAL - Using 2 reducers:

- What are the top 50 most frequent terms in your word count analysis?
- Present the top 50 terms and their frequency and their relative frequency.
- If there are ties please sort the tokens in alphanumeric/string order. Present bottom 10 tokens (least frequent items).

In [ ]:

HW3.3. Shopping Cart Analysis

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.

For this homework use the online browsing behavior dataset here (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.
- Here are the first few lines of the ProductPurchaseData
  - FRO11987 ELE17451 ELE89019 SNA90258 GRO99222
  - GRO99222 GRO12298 FRO12685 ELE91550 SNA11465 ELE26917 ELE52966 FRO90334 SNA30755 ELE17451 FRO84225 SNA80192
  - ELE17451 GRO73461 DAI22896 SNA99873 FRO86643
  - ELE17451 ELE37798 FRO86643 GRO56989 ELE23393 SNA11465
  - ELE17451 SNA69641 FRO86643 FRO78087 SNA11465 GRO39357 ELE28573 ELE11375 DAI54444

Do some exploratory data analysis of this dataset.

- How many unique items are available from this supplier?
- **Using a single reducer:**
  - Report your findings such as number of unique products; largest basket;
  - Report the top 50 most frequently purchased items, their frequency, and their relative frequency (break ties by sorting the products alphabetical order) etc. using Hadoop Map-Reduce.

Mapper - pair count

- where $size$ is the basket size for each new session, otherwise zero to minimize data transfer

In [21]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_3,Mapper_cnt,1\n")

for line in sys.stdin:
    # get all products
    products = line.strip().split(' ')
    size = len(products)
    if size==0:
        continue
    for i in range(size):
        # emit word key
        print '%s,%s,%s' %(products[i], 1, size if i==0 else 0)
```

```
Overwriting mapper.py
```

Reducer - pair count

In [106]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import numpy as np

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_3,Reducer_cnt,1\n")

max_size = 0
current_prod = None
current_count = 0

for line in sys.stdin:
    # get mapper output
    prod, count, size = line.strip().split(',', 2)

    # skip bad counts
    try:
        count = int(count)
        size = int(size)
    except ValueError:
        continue

    # handle basket size
    max_size = max(max_size, size)

    # count unique and get frequency
    if current_prod == prod:
```

```
        current_count += count
    else:
        # one product just finishes streaming
        if current_prod:
            # emit product count
            print '%s,%s' %(current_prod, current_count)

        # reset for new prod
        current_prod = prod
        current_count = count

#print 'max basket size: %d' %max_size
```

```
Overwriting reducer.py
```

Mapper - relative frequency & sort

- use **order inversion** for reletive frequency, for each word emit $(dummy\_sort, *, count)$ and $(dummy\_sort, product, count)$

In [107]:
```
%%writefile mapper_s.py
#!/usr/bin/python
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_3s,Mapper_cnt,1\n")

for line in sys.stdin:
    # get product and count
    prod, count = line.strip().split(',')
    # emit prod as key, and count
    print "%s,%s,%s" %(count, prod, count)
    # for order inversion, to calculate total count
    print '%d,%s,%s' %(1e+10, '*', count)
```

```
Overwriting mapper_s.py
```

Reducer - relative frequency & sort

- get the top 50 pairs with most count
- obtain unique products

In [108]:
```
%%writefile reducer_s.py
#!/usr/bin/python
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_3,Reducer_cnt,1\n")

n_total = 0
n_top = 50
n_unique = 0

for line in sys.stdin:

    dummy, product, count = line.strip().split(',', 2)

    try:
        count = int(count)
    except ValueError:
        continue

    # handle total
    if product == '*':
        n_total += count
        continue

    # get relative frequency
    n_unique += 1
    if n_unique <= n_top:
        print '%s\t%s\t%.4f%%' %(product, count, 100.0*count/n_total)

print 'total browsing items: %d' %n_total
```

```
print 'unique product: %d' %n_unique
```

```
Overwriting reducer_s.py
```

MapReducing

In [109]:
```
# job 1 - pair count
!hdfs dfs -rm -r results
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.ja
r \
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
-D map.output.key.field.separator=, \
-D map.output.key.value.fields.spec=0:1- \
-D mapred.text.key.comparator.options='-k1,1' \
-D mapred.map.tasks=3 \
-D mapred.reduce.tasks=1 \
-files mapper.py,reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-input /user/leiyang/ProductPurchaseData.txt \
-output results

# job 2 - relative frequency & sort with order inversion
!hdfs dfs -rm -r results2
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.ja
r \
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
-D map.output.key.field.separator=, \
-D map.output.key.value.fields.spec=0-1:0- \
-D mapred.text.key.comparator.options='-k1,1nr -k2,2' \
-D mapred.map.tasks=3 \
-D mapred.reduce.tasks=1 \
-files mapper_s.py,reducer_s.py \
-mapper mapper_s.py \
-reducer reducer_s.py \
-input /user/leiyang/results/part-0000* \
-output results2
```

```
Deleted results
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar6113682566909254738/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob2500948326253885932.jar tmpDir=null
Deleted results2
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar2055398774848871732/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob4358016637415128494.jar tmpDir=null
```

In [110]:
```
!hdfs dfs -cat results2/part-0*
```

```
DAI62779        6667    1.7507%
FRO40251        3881    1.0191%
ELE17451        3875    1.0175%
GRO73461        3602    0.9458%
SNA80324        3044    0.7993%
ELE32164        2851    0.7486%
DAI75645        2736    0.7184%
SNA45677        2455    0.6447%
FRO31317        2330    0.6118%
DAI85309        2293    0.6021%
ELE26917        2292    0.6019%
FRO80039        2233    0.5864%
GRO21487        2115    0.5554%
SNA99873        2083    0.5470%
GRO59710        2004    0.5262%
GRO71621        1920    0.5042%
FRO85978        1918    0.5036%
GRO30386        1840    0.4832%
ELE74009        1816    0.4769%
GRO56726        1784    0.4685%
DAI63921        1773    0.4656%
GRO46854        1756    0.4611%
ELE66600        1713    0.4498%
DAI83733        1712    0.4496%
FRO32293        1702    0.4469%
ELE66810        1697    0.4456%
SNA55762        1646    0.4322%
DAI22177        1627    0.4272%
FRO78087        1531    0.4020%
ELE99737        1516    0.3981%
```

```
ELE34057        1489    0.3910%
GRO94758        1489    0.3910%
FRO35904        1436    0.3771%
FRO53271        1420    0.3729%
SNA93860        1407    0.3695%
SNA90094        1390    0.3650%
GRO38814        1352    0.3550%
ELE56788        1345    0.3532%
GRO61133        1321    0.3469%
DAI88807        1316    0.3456%
ELE74482        1316    0.3456%
ELE59935        1311    0.3443%
SNA96271        1295    0.3401%
DAI43223        1290    0.3387%
ELE91337        1289    0.3385%
GRO15017        1275    0.3348%
DAI31081        1261    0.3311%
GRO81087        1220    0.3204%
DAI22896        1219    0.3201%
GRO85051        1214    0.3188%
total browsing items: 380823
unique product: 12591
```

3.3.1 OPTIONAL - Using 2 reducers:

- Report your findings such as number of unique products; largest basket;
- Report the top 50 most frequently purchased items, their frequency, and their relative frequency (break ties by sorting the products alphabetical order) etc. using Hadoop Map-Reduce.

Notes:

- the challenge is from total calculation since we have multiple reducers, as only one will get the * key and be able to calculate the marginal.
- possible solution:
  - write a customer partitioner, emit two dummy pairs, dispatch one for each reducer

HW3.4. (Computationally prohibitive but then again Hadoop can handle this) Pairs

- Suppose we want to recommend new products to the customer based on the products they have already browsed on the online website.
- Write a map-reduce program to find products which are frequently browsed together.
- Fix the support count (cooccurence count) to s = 100 (i.e. product pairs need to occur together at least 100 times to be considered frequent), and find pairs of items (sometimes referred to itemsets of size 2 in association rule mining) that have a support count of 100 or more.

List the top 50 product pairs with corresponding support count (aka frequency), and relative frequency or support (number of records where they coccur, the number of records where they coccur/the number of baskets in the dataset) in decreasing order of support for frequent (100>count) itemsets of size 2.

Use the Pairs pattern (lecture 3) to extract these frequent itemsets of size 2. Free free to use combiners if they bring value. Instrument your code with counters for count the number of times your mapper, combiner and reducers are called.

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term w ∈ doc d do
4:             for all term u ∈ Neighbors(w) do
5:                 Emit(pair (w, u), count 1)        ▷ Emit count for each co-occurrence
1: class Reducer
2:     method Reduce(pair p, counts [c1, c2, ...])
3:         s ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             s ← s + c                              ▷ Sum co-occurrence counts
6:         Emit(pair p, count s)
```

Figure 3.8: Pseudo-code for the "pairs" approach for computing word co-occurrence matrices from large corpora.

Please output records of the following form for the top 50 pairs (itemsets of size 2):

```
item1, item2, support count, support
```

Fix the ordering of the pairs lexicographically (left to right), and break ties in support (between pairs, if any exist) by taking the first ones in lexicographically increasing order.

Report the compute time for the Pairs job. Describe the computational setup used (E.g., single computer; dual core; linux, number of mappers, number of reducers)

Spec	Value
Computer	single
OS	OS X El Capitan
Processor	2.2 GHz Intel Core i7
Memory	16 GB 1600 MHz DDR3

Instrument your mapper, combiner, and reducer to count how many times each is called using Counters and report these counts.

Mapper

- for each session (row), use pair pattern with **order inversion**, emit $((w_i\_w_j), 1)$ for all pairs, and one $(*, 1)$ for the session (for total session count).
- the fourth field of every emit is used to indicate basket size for every session (row), size is postive for *only* one emit, and zero for rest of the emit, so we minimize data transfer

In [59]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_4,Mapper_cnt,1\n")

for line in sys.stdin:
    # get all products from the session
    products = line.strip().split(' ')
    size = len(products)
    if size==0:
        continue

    # sort products the pair is lexicographically sound
    products.sort()

    # get pairs of products
    pairs = [[products[i], products[j]] for i in range(size) for j in range(i+1, size)]

    # emit dummy record
    print '%s,%s' %('*', 1)

    # emit product pairs
    for pair in pairs:
        print '%s_%s,%s' %(pair[0], pair[1], 1)
```

```
Overwriting mapper.py
```

Combiner

- local aggregation for count

In [60]:
```
%%writefile combiner.py
#!/usr/bin/python
import sys

# increase counter for reducer being called
sys.stderr.write("reporter:counter:HW3_4,Combiner_cnt,1\n")

current_pair = None
current_count = 0

for line in sys.stdin:
    # get all products from the session
    pair, count = line.strip().split(',', 1)
```

```
    pair, count = line.strip().split(',', 1)

    # skip bad count
    try:
        count = int(count)
    except ValueError:
        continue

    # accumulate counts for whatever keys it receives
    if current_pair == pair:
        current_count += count
    else:
        # previous pair finishes streaming, emit results
        if current_pair:
            print '%s,%s' %(current_pair, current_count)
        # reset new pair
        current_pair = pair
        current_count = count
```

```
Overwriting combiner.py
```

Reducer

- count number of basket based on $(*, 1)$ emits
- get suport and relative frequency for each pair in the stream

In [61]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys

# increase counter for reducer being called
sys.stderr.write("reporter:counter:HW3_4,Reducer_cnt,1\n")

n_basket = 0
min_support = 100
current_pair = None
current_count = 0

for line in sys.stdin:
    # get all products from the session
    pair, count = line.strip().split(',', 1)

    # skip bad count
    try:
        count = int(count)
    except ValueError:
        continue

    # get total sessions/baskets
    if pair == '*':
        n_basket += count
        continue

    # get pair count
    if current_pair == pair:
        current_count += count
    else:
        # previous pair finishes streaming
        if current_pair and current_count > min_support:
            # get relative freq
            rf = 100.0*current_count/n_basket
            # emit
            print '%s,%s,%.4f%%' %(current_pair, current_count, rf)
        # reset new pair
        current_pair = pair
        current_count = count

#print '\ntotal basket: %d' %n_basket
```

```
Overwriting reducer.py
```

Mapper for sort (or use identity mapper)

In [62]:
```
%%writefile mapper_s.py
#!/usr/bin/python
import sys
```

```
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_4,Mapper_s_cnt,1\n")

for line in sys.stdin:
    # just emit
    print line.strip()
```

```
Overwriting mapper_s.py
```

Reducer for sort

In [63]:
```
%%writefile reducer_s.py
#!/usr/bin/python
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_4,Reducer_s_cnt,1\n")

n_out = 0
n_top = 50

print 'top %d pairs: ' %n_top

for line in sys.stdin:
    # parse mapper output
    pair, count, rf = line.strip().split(',', 2)
    n_out += 1
    if n_out <= n_top:
        w1, w2 = pair.split('_')
        print '%s\t%s\t%s\t%s' %(w1, w2, count, rf)
```

```
Overwriting reducer_s.py
```

MapReducing without combiner

In [64]:
```
# job 1 - count
!hdfs dfs -rm -r results
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
-D map.output.key.field.separator=, \
-D map.output.key.value.fields.spec=0:1- \
-D mapred.text.key.comparator.options='-k1,1' \
-D mapred.map.tasks=3 \
-D mapred.reduce.tasks=1 \
-files mapper.py,reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-input /user/leiyang/ProductPurchaseData.txt \
-output results

# job 2 - sort relative frequency
!hdfs dfs -rm -r results2
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
-D map.output.key.field.separator=',' \
-D map.output.key.value.fields.spec=0-1:2- \
-D mapred.text.key.comparator.options='-k2,2nr -k1,1' \
-D mapred.map.tasks=2 \
-D mapred.reduce.tasks=1 \
-files mapper_s.py,reducer_s.py \
-mapper mapper_s.py \
-reducer reducer_s.py \
-input /user/leiyang/results/part-0000* \
-output results2
```

```
Deleted results
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar6313500269707918499/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob6247307708596164046.jar tmpDir=null
Deleted results2
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar2314263346417839441/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob2015921523134373354.jar tmpDir=null
```

HW3.4 results without combiner

- 3 mappers, 1 reducer

	Name	Map	Reduce	Total
HW3_4	Mapper_cnt	3	0	3
	Reducer_cnt	0	1	1

Job Name:	streamjob5914859139307550 70
User Name:	leiyang
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sun Jan 31 14:55:30 EST 2016
Started:	Sun Jan 31 14:55:34 EST 2016
Finished:	Sun Jan 31 14:55:58 EST 2016
Elapsed:	23sec
Diagnostics:	
Average Map Time	8sec
Average Shuffle Time	2sec
Average Merge Time	3sec
Average Reduce Time	4sec

In [65]:
```
!hdfs dfs -cat results2/part-0*
```

```
top 50 pairs:
DAI62779        ELE17451        1592    5.1188%
FRO40251        SNA80324        1412    4.5400%
DAI75645        FRO40251        1254    4.0320%
FRO40251        GRO85051        1213    3.9002%
DAI62779        GRO73461        1139    3.6623%
DAI75645        SNA80324        1130    3.6333%
DAI62779        FRO40251        1070    3.4404%
DAI62779        SNA80324        923     2.9678%
DAI62779        DAI85309        918     2.9517%
ELE32164        GRO59710        911     2.9292%
DAI62779        DAI75645        882     2.8359%
FRO40251        GRO73461        882     2.8359%
DAI62779        ELE92920        877     2.8198%
FRO40251        FRO92469        835     2.6848%
DAI62779        ELE32164        832     2.6752%
DAI75645        GRO73461        712     2.2893%
DAI43223        ELE32164        711     2.2861%
DAI62779        GRO30386        709     2.2797%
ELE17451        FRO40251        697     2.2411%
DAI85309        ELE99737        659     2.1189%
DAI62779        ELE26917        650     2.0900%
GRO21487        GRO73461        631     2.0289%
DAI62779        SNA45677        604     1.9421%
ELE17451        SNA80324        597     1.9196%
DAI62779        GRO71621        595     1.9131%
DAI62779        SNA55762        593     1.9067%
DAI62779        DAI83733        586     1.8842%
ELE17451        GRO73461        580     1.8649%
GRO73461        SNA80324        562     1.8070%
DAI62779        GRO59710        561     1.8038%
DAI62779        FRO80039        550     1.7684%
DAI75645        ELE17451        547     1.7588%
DAI62779        SNA93860        537     1.7266%
DAI55148        DAI62779        526     1.6913%
DAI43223        GRO59710        512     1.6462%
ELE17451        ELE32164        511     1.6430%
DAI62779        SNA18336        506     1.6270%
ELE32164        GRO73461        486     1.5627%
DAI62779        FRO78087        482     1.5498%
DAI85309        ELE17451        482     1.5498%
DAI62779        GRO94758        479     1.5401%
DAI62779        GRO21487        471     1.5144%
GRO85051        SNA80324        471     1.5144%
ELE17451        GRO30386        468     1.5048%
FRO85978        SNA95666        463     1.4887%
DAI62779        FRO19221        462     1.4855%
DAI62779        GRO46854        461     1.4823%
DAI43223        DAI62779        459     1.4758%
```

```
ELE92920        SNA18336        455     1.4630%
DAI88079        FRO40251        446     1.4340%
```

MapReducing with combiner

In [66]:
```
# job 1 - add combiner below
!hdfs dfs -rm -r results
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
-D map.output.key.field.separator=, \
-D map.output.key.value.fields.spec=0:1- \
-D mapred.text.key.comparator.options='-k1,1' \
-D mapred.map.tasks=3 \
-D mapred.reduce.tasks=1 \
-files mapper.py,reducer.py,combiner.py \
-mapper mapper.py \
-reducer reducer.py \
-combiner combiner.py \
-input /user/leiyang/ProductPurchaseData.txt \
-output results

# job 2 - sort relative frequency
!hdfs dfs -rm -r results2
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
-D map.output.key.field.separator=',' \
-D map.output.key.value.fields.spec=0-1:2- \
-D mapred.text.key.comparator.options='-k2,2nr -k1,1' \
-D mapred.map.tasks=2 \
-D mapred.reduce.tasks=1 \
-files mapper_s.py,reducer_s.py \
-mapper mapper_s.py \
-reducer reducer_s.py \
-input /user/leiyang/results/part-0000* \
-output results2
```

```
Deleted results
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar3295224449230340302/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob7434701589508937802.jar tmpDir=null
Deleted results2
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar9434479643349908827/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob5402224175670776010.jar tmpDir=null
```

HW3.4 Results with combiner

- 3 mappers, 1 reducer
- the combiner was called 1 time by each map process, total 3 times

	Name	Map	Reduce	Total
HW3_4	Combiner_cnt	3	0	3
	Mapper_cnt	3	0	3
	Reducer_cnt	0	1	1

Job Name:	streamjob7465990197884943O2
User Name:	leiyang
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sun Jan 31 15:08:04 EST 2016
Started:	Sun Jan 31 15:08:08 EST 2016
Finished:	Sun Jan 31 15:08:28 EST 2016
Elapsed:	20sec
Diagnostics:	
Average Map Time	9sec
Average Shuffle Time	2sec
Average Merge Time	1sec
Average Reduce Time	2sec

In [68]:
```
!hdfs dfs -cat results2/part-0*
```

```
top 50 pairs:
```

DAI62779	ELE17451	1592	5.1188%
FRO40251	SNA80324	1412	4.5400%
DAI75645	FRO40251	1254	4.0320%
FRO40251	GRO85051	1213	3.9002%
DAI62779	GRO73461	1139	3.6623%
DAI75645	SNA80324	1130	3.6333%
DAI62779	FRO40251	1070	3.4404%
DAI62779	SNA80324	923	2.9678%
DAI62779	DAI85309	918	2.9517%
ELE32164	GRO59710	911	2.9292%
DAI62779	DAI75645	882	2.8359%
FRO40251	GRO73461	882	2.8359%
DAI62779	ELE92920	877	2.8198%
FRO40251	FRO92469	835	2.6848%
DAI62779	ELE32164	832	2.6752%
DAI75645	GRO73461	712	2.2893%
DAI43223	ELE32164	711	2.2861%
DAI62779	GRO30386	709	2.2797%
ELE17451	FRO40251	697	2.2411%
DAI85309	ELE99737	659	2.1189%
DAI62779	ELE26917	650	2.0900%
GRO21487	GRO73461	631	2.0289%
DAI62779	SNA45677	604	1.9421%
ELE17451	SNA80324	597	1.9196%
DAI62779	GRO71621	595	1.9131%
DAI62779	SNA55762	593	1.9067%
DAI62779	DAI83733	586	1.8842%
ELE17451	GRO73461	580	1.8649%
GRO73461	SNA80324	562	1.8070%
DAI62779	GRO59710	561	1.8038%
DAI62779	FRO80039	550	1.7684%
DAI75645	ELE17451	547	1.7588%
DAI62779	SNA93860	537	1.7266%
DAI55148	DAI62779	526	1.6913%
DAI43223	GRO59710	512	1.6462%
ELE17451	ELE32164	511	1.6430%
DAI62779	SNA18336	506	1.6270%
ELE32164	GRO73461	486	1.5627%
DAI62779	FRO78087	482	1.5498%
DAI85309	ELE17451	482	1.5498%
DAI62779	GRO94758	479	1.5401%
DAI62779	GRO21487	471	1.5144%
GRO85051	SNA80324	471	1.5144%
ELE17451	GRO30386	468	1.5048%
FRO85978	SNA95666	463	1.4887%
DAI62779	FRO19221	462	1.4855%
DAI62779	GRO46854	461	1.4823%
DAI43223	DAI62779	459	1.4758%
ELE92920	SNA18336	455	1.4630%
DAI88079	FRO40251	446	1.4340%

HW3.5: Stripes

- Repeat 3.4 using the stripes design pattern for finding cooccuring pairs.
- Report the compute times for stripes job versus the Pairs job.
- Describe the computational setup used (E.g., single computer; dual core; linux, number of mappers, number of reducers)
- Instrument your mapper, combiner, and reducer to count how many times each is called using Counters and report these counts.
- Discuss the differences in these counts between the Pairs and Stripes jobs

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term w ∈ doc d do
4:             H ← new AssociativeArray
5:             for all term u ∈ Neighbors(w) do
6:                 H{u} ← H{u} + 1                ▷ Tally words co-occurring with w
7:             Emit(Term w, Stripe H)

1: class Reducer
2:     method Reduce(term w, stripes [H_1, H_2, H_3, ...])
3:         H_f ← new AssociativeArray
4:         for all stripe H ∈ stripes [H_1, H_2, H_3, ...] do
5:             Sum(H_f, H)                        ▷ Element-wise sum
6:         Emit(term w, stripe H_f)
```

Figure 3.9: Pseudo-code for the "stripes" approach for computing word co-occurrence matrices from large corpora.

Mapper

- build associative array for each session, and do local in-memory aggregation
- for the associative array, we implement the rule that *any key will only have words that alphabetically behind it in the associative array*, to have unique pairs

In [71]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_5,Mapper_cnt,1\n")

# composite associative array
H = {}

for line in sys.stdin:
    # get all products from the session
    products = line.strip().split(' ')
    size = len(products)
    if size==0:
        continue

    # sort products so the pair is lexicographically sound
    products.sort()

    # get pairs of products
    pairs = [[products[i], products[j]] for i in range(size) for j in range(i+1, size)]

    # emit dummy record
    print '%s\t%s' %('*', 1)

    # build associative arrays
    for w1, w2 in pairs:
        # each pair is lexicographically in order
        if w1 not in H:
            # if w1 is new, add an associative array for it
            H[w1] = {}
            H[w1][w2] = 1
        elif w2 not in H[w1]:
            # w1 is not new, but it doesn't have key for w2
            H[w1][w2] = 1
        else:
            # both are there, increase it
            H[w1][w2] += 1

# emit associative arrays
for h in H:
    print '%s\t%s' %(h, str(H[h]))
```

```
Overwriting mapper.py
```

Reducer

- element-wise sum

In [74]:
```
%%writefile reducer.py
#!/usr/bin/python

# function to combine associative array
def elementSum(H1, H2):
    # make sure H1 is the long one
    if len(H1)<len(H2):
        H0 = H2
        H2 = H1
        H1 = H0
    # merge shorter one H2 into longer one H1
    for h in H2:
        if h not in H1:
            H1[h] = H2[h]
        else:
            H1[h] += H2[h]
```

```
            H1[n] += H2[n]
    # return
    return H1

import sys
import numpy as np

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_5,Reducer_cnt,1\n")

min_support = 100
current_word = None
current_aArray = None
n_total = 0

for line in sys.stdin:
    # parse out keyword and the associative array
    word, aArray = line.strip().split('\t', 1)

    # get total basket
    if word == '*':
        n_total += int(aArray)
        continue

    # get array into variable
    cmdStr = 'aArray = ' + aArray
    exec cmdStr

    # merge the associative array
    if current_word == word:
        current_aArray = elementSum(current_aArray, aArray)
    else:
        # finish one word merge
        if current_word:
            # get the top pairs with heap
            for p in current_aArray:
                if current_aArray[p] > min_support:
                    # get relative freq
                    rf = 100.0*current_aArray[p]/n_total
                    print '%s,%s,%s,%.4f%%' %(current_word, p, current_aArray[p], rf)
        # reset for a new word
        current_word = word
        current_aArray = aArray

#print '\ntotal basket: %d' %n_total
```

```
Overwriting reducer.py
```

Mapper to sort

In [79]:
```
%%writefile mapper_s.py
#!/usr/bin/python
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_5,Mapper_s_cnt,1\n")

for line in sys.stdin:
    # just emit
    print line.strip()
```

```
Overwriting mapper_s.py
```

Reducer to sort

In [80]:
```
%%writefile reducer_s.py
#!/usr/bin/python
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_5,Reducer_s_cnt,1\n")

n_out = 0
n_top = 50

print 'top %d pairs: ' %n_top
```

```
print "top %d pairs:" %n_top

for line in sys.stdin:
    # parse mapper output
    n_out += 1
    if n_out <= n_top:
        print line.strip().replace(',', '\t')
```

```
Overwriting reducer_s.py
```

MapReducing

In [84]:
```
# job 1 - count
!hdfs dfs -rm -r results
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.map.tasks=3 \
-D mapred.reduce.tasks=1 \
-files mapper.py,reducer.py,combiner.py \
-mapper mapper.py \
-reducer reducer.py \
-input /user/leiyang/ProductPurchaseData.txt \
-output results

# job 2 - sort relative frequency
!hdfs dfs -rm -r results2
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
-D map.output.key.field.separator=',' \
-D map.output.key.value.fields.spec=0-2:3- \
-D mapred.text.key.comparator.options='-k3,3nr -k1,1 -k2,2' \
-D mapred.map.tasks=2 \
-D mapred.reduce.tasks=1 \
-files mapper_s.py,reducer_s.py \
-mapper mapper_s.py \
-reducer reducer_s.py \
-input /user/leiyang/results/part-0000* \
-output results2
```

```
Deleted results
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar3782570145270895412/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob870881278814839645.jar tmpDir=null
Deleted results2
packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar5736039512355960039/]
[] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob3937477619845631898.jar tmpDir=null
```

HW3.5 Results

- 3 mappers, 1 reducer
- the combiner was called 1 time by each map process, total 3 times
- with the same configure, the execution time is reduced to 15 sec. from 23 sec. of pair approach, about **33%** improvement

HW3_5 HW3_5

Name	Map	Reduce	Total
Mapper_cnt	3	0	3
Reducer_cnt	0	1	1

Job Name:	streamjob2756454841948235843.
User Name:	leiyang
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sun Jan 31 21:40:35 EST 2016
Started:	Sun Jan 31 21:40:39 EST 2016
Finished:	Sun Jan 31 21:40:54 EST 2016
Elapsed:	15sec
Diagnostics:	
Average Map Time	4sec
Average Shuffle Time	2sec
Average Merge Time	0sec
Average Reduce Time	3sec

In [85]:
```
!hdfs dfs -cat results2/part-0*
```

```
top 50 pairs:
DAI62779        ELE17451        1592    5.1188%
FRO40251        SNA80324        1412    4.5400%
DAI75645        FRO40251        1254    4.0320%
FRO40251        GRO85051        1213    3.9002%
DAI62779        GRO73461        1139    3.6623%
DAI75645        SNA80324        1130    3.6333%
DAI62779        FRO40251        1070    3.4404%
DAI62779        SNA80324        923     2.9678%
DAI62779        DAI85309        918     2.9517%
ELE32164        GRO59710        911     2.9292%
DAI62779        DAI75645        882     2.8359%
FRO40251        GRO73461        882     2.8359%
DAI62779        ELE92920        877     2.8198%
FRO40251        FRO92469        835     2.6848%
DAI62779        ELE32164        832     2.6752%
DAI75645        GRO73461        712     2.2893%
DAI43223        ELE32164        711     2.2861%
DAI62779        GRO30386        709     2.2797%
ELE17451        FRO40251        697     2.2411%
DAI85309        ELE99737        659     2.1189%
DAI62779        ELE26917        650     2.0900%
GRO21487        GRO73461        631     2.0289%
DAI62779        SNA45677        604     1.9421%
ELE17451        SNA80324        597     1.9196%
DAI62779        GRO71621        595     1.9131%
DAI62779        SNA55762        593     1.9067%
DAI62779        DAI83733        586     1.8842%
ELE17451        GRO73461        580     1.8649%
GRO73461        SNA80324        562     1.8070%
DAI62779        GRO59710        561     1.8038%
DAI62779        FRO80039        550     1.7684%
DAI75645        ELE17451        547     1.7588%
DAI62779        SNA93860        537     1.7266%
DAI55148        DAI62779        526     1.6913%
DAI43223        GRO59710        512     1.6462%
ELE17451        ELE32164        511     1.6430%
DAI62779        SNA18336        506     1.6270%
ELE32164        GRO73461        486     1.5627%
DAI62779        FRO78087        482     1.5498%
DAI85309        ELE17451        482     1.5498%
DAI62779        GRO94758        479     1.5401%
DAI62779        GRO21487        471     1.5144%
GRO85051        SNA80324        471     1.5144%
ELE17451        GRO30386        468     1.5048%
FRO85978        SNA95666        463     1.4887%
DAI62779        FRO19221        462     1.4855%
DAI62779        GRO46854        461     1.4823%
DAI43223        DAI62779        459     1.4758%
ELE92920        SNA18336        455     1.4630%
DAI88079        FRO40251        446     1.4340%
```

OPTIONAL: all HW below this are optional

Preliminary information

Much of this homework beyond this point will focus on the Apriori algorithm for frequent itemset mining and the additional step for extracting association rules from these frequent itemsets. Please acquaint yourself with the background information (below) before approaching the remaining assignments.

Apriori background information

Some background material for the Apriori algorithm is located at:

- Slides in Live Session #3
- https://en.wikipedia.org/wiki/Apriori_algorithm (https://en.wikipedia.org/wiki/Apriori_algorithm)
- https://www.dropbox.com/s/k2zm4otych279z2/Apriori-good-slides.pdf?dl=0 (https://www.dropbox.com/s/k2zm4otych279z2/Apriori-good-slides.pdf?dl=0)
- http://snap.stanford.edu/class/cs246-2014/slides/02-assocrules.pdf (http://snap.stanford.edu/class/cs246-2014/slides/02-assocrules.pdf)

Association Rules are frequently used for Market Basket Analysis (MBA) by retailers to understand the purchase behavior of their customers. This information can be then used for many different purposes such as cross-selling and up-selling of products, sales promotions, loyalty programs, store design, discount plans and many others. Evaluation of item sets: Once you have found the frequent itemsets of a dataset, you need to choose a subset of them as your recommendations. Commonly used metrics for measuring significance and interest for selecting rules for

recommendations are: confidence; lift; and conviction.

HW3.6

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

NOTE: For the remaining homework use the online browsing behavior dataset located at (same dataset as used above):

```
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.

Here are the first few lines of the ProductPurchaseData:

- FRO11987 ELE17451 ELE89019 SNA90258 GRO99222
- GRO99222 GRO12298 FRO12685 ELE91550 SNA11465 ELE26917 ELE52966 FRO90334 SNA30755 ELE17451 FRO84225 SNA80192
- ELE17451 GRO73461 DAI22896 SNA99873 FRO86643
- ELE17451 ELE37798 FRO86643 GRO56989 ELE23393 SNA11465
- ELE17451 SNA69641 FRO86643 FRO78087 SNA11465 GRO39357 ELE28573 ELE11375 DAI54444

Answer:

- Aprior algorithm is used to find frequent itemsets, each iteration has two scans of data and a filtering in between:
  1. generate a condidate set C_k for itemsets of size k, based on the output of previous iteration L_{k-1}
  2. remove all members from the set whose support is less than the user specified threshold s_i
  3. generate the final set L_k for frequent itemset of size k, based on output after filtering
- For example, to find itemsets of size k from a basket set, the process is:
  1. count all single words from all baskets, output C_1
  2. remove all words with support below threshold, output L_1
  3. using L_1, generate candidate set for frequent pair set C_2
  4. remove all pairs with support below threshold, get L_2
  5. using L_2, generate candidate set for frequent triple set C_3
  6. remove all triples with support below threshold, get L_3

HW3.7. Shopping Cart Analysis

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 sets need to occur together at least 100 times to be considered frequent) and find itemsets of size 2 and 3.

Then extract association rules from these frequent items. A rule is of the form:

- (item1, item5) ⇒ item2.

List the top 10 discovered rules in descreasing order of confidence in the following format

- (item1, item5) ⇒ item2, supportCount ,support, confidence

Implementation Notes:

- each MapReduce job perform one round of APrior processing:
  - mapper: construct candidate set C_k
  - reducer: filter C_k to get frequent item set L_k
- to find itemsets of size 3, we will need 3 jobs

Mapper 1: get C_1

- emit singleton

In [10]:
```
%%writefile mapper_1.py
#!/usr/bin/python
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_7,Mapper_1_cnt,1\n")

for line in sys.stdin:
    # get words and emit
    for prod in line.strip().split(' '):
        print '%s\t%d' %(prod, 1)
```

```
Overwriting mapper_1.py
```

Reducer 1: get L_1

- only emit words whose frequency is above the support threshold (100)
- can be used as **combiner** too

In [11]:
```
%%writefile reducer_1.py
#!/usr/bin/python
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_7,Reducer_1_cnt,1\n")

current_prod = None
current_count = 0
min_support = 100

for line in sys.stdin:
    # get k-v pair
    prod, count = line.strip().split('\t', 1)

    # skip bad count
    try:
        count = int(count)
    except ValueError:
        continue

    # get count
    if current_prod == prod:
        current_count += count
    else:
        if current_prod and current_count > min_support:
            # emit prod above min support
            print '%s\t%d' %(current_prod, current_count)
        # reset
        current_prod = prod
        current_count = count
```

```
Overwriting reducer_1.py
```

Mapper 2: get C_2

In [12]:
```
%%writefile mapper_2.py
#!/usr/bin/python
import sys, subprocess

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_7,Mapper_2_cnt,1\n")

singleton = []
cat = subprocess.Popen(['hdfs', 'dfs', '-cat', 'results1/part-0*'], stdout=subprocess.PIPE)
for line in cat.stdout:
    singleton.append(line.strip().split('\t')[0])

# read the input data
for line in sys.stdin:

    line = line.strip()

    # get words for each session
```

```
    prod = line.strip().split(' ')

    # keep product from singleton set only
    products = [val for val in prod if val in singleton]
    products.sort()

    # get pairs to emit
    size = len(products)
    pairs = [products[i] + '_' + products[j] for i in range(size) for j in range(i+1, size)]
    for p in pairs:
        print '%s\t%d' %(p, 1)
```

```
Overwriting mapper_2.py
```

Reducer 2: get L_2

- same as Reducer 1, since we have identical k-v format (%s\t%d) from the mapper
- can also be used as combiner

In [13]:
```
### same as reducer_1.py
```

Mapper 3: get C_3

In [12]:
```
%%writefile mapper_3.py
#!/usr/bin/python
import sys, subprocess

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_7,Mapper_3_cnt,1\n")

# load the frequent freqPairs given by Job 2
freqPair = []
cat = subprocess.Popen(['hdfs', 'dfs', '-cat', 'results2/part-0*'], stdout=subprocess.PIPE)
for line in cat.stdout:
    freqPair.append(line.strip().split('\t')[0])

# still read frequent freqPairs first, then session data to generate triples
for line in sys.stdin:

    line = line.strip()

    # get products from each session
    prod = line.split(' ')
    prod.sort()
    n = len(prod)

    # generate freqPairs and triples from the session, in the format of a_b and a_b_c, alphabetica
lly sorted
    triples = [[prod[i],prod[j],prod[k]] for i in range(n) for j in range(i+1,n) for k in range(i+
2,n)]
    pairs = [prod[i]+'_'+prod[j] for i in range(n) for j in range(i+1,n)]

    # processing pairs
    for pair in pairs:
        # if the pair is in freqPair, emit a dummy key a_b_*
        if pair in freqPair:
            print '%s_*\t%d' %(pair, 1)

    # processing triples
    for tri in triples:
        # from each triple a_b_c: check if the 3 child-pairs (a_b, b_c, a_c) are in the freqPair s
et
        if tri[0]+'_'+tri[1] in freqPair and tri[1]+'_'+tri[2] in freqPair and tri[0]+'_'+tri[2] i
n freqPair:
            # if so, emit the triple a_b_c
            print '%s_%s_%s\t%d' %(tri[0], tri[1], tri[2], 1)
```

```
Overwriting mapper_3.py
```

Reducer 3: get L_3

- use order inversion to get confidence

In [13]:
```
%%writefile reducer_3.py
#!/usr/bin/python
import sys

# increase counter for mapper being called
sys.stderr.write("reporter:counter:HW3_7,Reducer_3_cnt,1\n")

current_prod = None
current_dummy = None
current_count = 0
min_support = 100
marginal = 0

for line in sys.stdin:

    # get k-v freqPair
    prod, count = line.strip().split('\t', 1)

    # skip bad count
    try:
        count = int(count)
    except ValueError:
        continue

    # handle marginal with dummy key
    if '*' == prod[-1]:
        if current_dummy == prod:
            # accumulate marginal
            marginal += count
        else:
            # reset marginal for new dummy key
            current_dummy = prod
            marginal = count
        continue

    # processing triple and emit rules
    if current_prod == prod:
        current_count += count
    else:
        if current_prod and current_count > min_support:
            # for debug, check if current dummy matches current triple
            if current_prod[:-8] != current_dummy[:-1]:
                print 'WARNING: mismatch between %s and %s(%d)' %(current_prod, current_dummy, mar
ginal)
            else:
                # emit triples for the rule
                w1,w2,w3 = current_prod.split('_')
                conf = 100.0*current_count/marginal
                print '(%s, %s) => %s, %d, %d, %.2f%%' %(w1, w2, w3, current_count, marginal, con
f)

        # reset for new triple
        current_prod = prod
        current_count = count
```

```
Overwriting reducer_3.py
```

MapReducing

In [16]:
```
# job 1 - get L_1 for frequent singletons
!hdfs dfs -rm -r results1
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.map.tasks=3 \
-D mapred.reduce.tasks=1 \
-files mapper_1.py,reducer_1.py \
-mapper mapper_1.py \
-reducer reducer_1.py \
-combiner reducer_1.py \
-input /user/leiyang/ProductPurchaseData.txt \
-output results1
```

```
Deleted results1
```

In [18]:
```
# job 2 - get L_2 for frequent pairs
!hdfs dfs -rm -r results2
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.map.tasks=3 \
-D mapred.reduce.tasks=1 \
-files mapper_2.py,reducer_1.py \
-mapper mapper_2.py \
-reducer reducer_1.py \
-input /user/leiyang/ProductPurchaseData.txt \
-output results2
```

```
Deleted results2
```

In [14]:
```
# job 3 - get L_3 and calculate association rules
!hdfs dfs -rm -r results3
!hadoop jar /usr/local/Cellar/hadoop/2.*/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar
\
-D mapred.map.tasks=3 \
-D mapred.reduce.tasks=1 \
-files mapper_3.py,reducer_3.py \
-mapper mapper_3.py \
-reducer reducer_3.py \
-input /user/leiyang/ProductPurchaseData.txt \
-output results3
```

```
Deleted results3
```

Association Rules

In [15]:
```
!hdfs dfs -cat results3/part-0*
```

```
(DAI22896, DAI62779) => GRO73461, 101, 297, 34.01%
WARNING: mismatch between DAI23334_DAI62779_ELE92920 and DAI31081_DAI43223_*(123)
(DAI31081, DAI62779) => ELE17451, 103, 364, 28.30%
(DAI31081, DAI75645) => FRO40251, 122, 206, 59.22%
(DAI31081, ELE32164) => GRO59710, 112, 312, 35.90%
(DAI31081, FRO40251) => GRO85051, 102, 280, 36.43%
WARNING: mismatch between DAI31081_FRO40251_SNA80324 and DAI31081_FRO53271_*(161)
(DAI42083, DAI62779) => DAI92600, 105, 117, 89.74%
(DAI42083, DAI92600) => ELE17451, 117, 256, 45.70%
(DAI42493, DAI62779) => ELE17451, 112, 309, 36.25%
(DAI42493, DAI62779) => ELE92920, 112, 309, 36.25%
(DAI42493, DAI62779) => SNA18336, 109, 309, 35.28%
(DAI43223, DAI62779) => ELE17451, 227, 459, 49.46%
(DAI43223, DAI62779) => ELE32164, 287, 459, 62.53%
(DAI43223, DAI62779) => GRO59710, 205, 459, 44.66%
(DAI43223, ELE17451) => ELE32164, 206, 326, 63.19%
(DAI43223, ELE17451) => GRO59710, 156, 326, 47.85%
(DAI43223, ELE32164) => GRO59710, 287, 711, 40.37%
(DAI43223, ELE32164) => GRO73461, 111, 711, 15.61%
(DAI55148, DAI62779) => DAI75645, 163, 526, 30.99%
(DAI55148, DAI62779) => ELE17451, 150, 526, 28.52%
(DAI55148, DAI62779) => FRO40251, 189, 526, 35.93%
WARNING: mismatch between DAI55148_DAI62779_SNA80324 and DAI55148_DAI75645_*(299)
(DAI55148, DAI75645) => ELE17451, 106, 299, 35.45%
(DAI55148, DAI75645) => FRO40251, 120, 299, 40.13%
WARNING: mismatch between DAI55148_DAI75645_SNA80324 and DAI55148_DAI85309_*(131)
(DAI55148, ELE17451) => FRO40251, 120, 305, 39.34%
WARNING: mismatch between DAI55148_ELE17451_SNA80324 and DAI55148_ELE32164_*(134)
(DAI55148, FRO40251) => FRO92469, 105, 343, 30.61%
WARNING: mismatch between DAI55148_FRO40251_SNA80324 and DAI55148_FRO92469_*(108)
(DAI55911, FRO40251) => GRO85051, 133, 232, 57.33%
(DAI62779, DAI75645) => DAI85309, 103, 882, 11.68%
(DAI62779, DAI75645) => ELE17451, 328, 882, 37.19%
(DAI62779, DAI75645) => ELE20847, 115, 882, 13.04%
(DAI62779, DAI75645) => ELE26917, 101, 882, 11.45%
(DAI62779, DAI75645) => ELE92920, 130, 882, 14.74%
(DAI62779, DAI75645) => FRO40251, 412, 882, 46.71%
(DAI62779, DAI75645) => GRO30386, 137, 882, 15.53%
(DAI62779, DAI75645) => GRO73461, 261, 882, 29.59%
(DAI62779, DAI75645) => GRO85051, 154, 882, 17.46%
(DAI62779, DAI75645) => SNA80324, 421, 882, 47.73%
(DAI62779, DAI83733) => DAI85309, 138, 586, 23.55%
(DAI62779, DAI83733) => ELE17451, 147, 586, 25.09%
(DAI62779, DAI83733) => ELE92920, 103, 586, 17.58%
(DAI62779, DAI85309) => ELE17451, 339, 918, 36.93%
```

```
(DAI62779, DAI85309) => ELE32164, 141, 918, 15.36%
(DAI62779, DAI85309) => ELE92920, 191, 918, 20.81%
(DAI62779, DAI85309) => ELE99737, 272, 918, 29.63%
(DAI62779, DAI85309) => FRO40251, 127, 918, 13.83%
(DAI62779, DAI85309) => GRO46854, 110, 918, 11.98%
(DAI62779, DAI85309) => GRO71621, 116, 918, 12.64%
(DAI62779, DAI85309) => GRO73461, 179, 918, 19.50%
(DAI62779, DAI85309) => SNA18336, 151, 918, 16.45%
(DAI62779, DAI85309) => SNA45677, 118, 918, 12.85%
(DAI62779, DAI85309) => SNA55762, 116, 918, 12.64%
(DAI62779, DAI88079) => FRO40251, 117, 117, 100.00%
(DAI62779, DAI88807) => SNA72163, 104, 261, 39.85%
(DAI62779, DAI91290) => ELE17451, 109, 353, 30.88%
(DAI62779, ELE17451) => ELE26917, 160, 1592, 10.05%
(DAI62779, ELE17451) => ELE32164, 277, 1592, 17.40%
(DAI62779, ELE17451) => ELE56788, 107, 1592, 6.72%
(DAI62779, ELE17451) => ELE74009, 112, 1592, 7.04%
(DAI62779, ELE17451) => ELE92920, 345, 1592, 21.67%
(DAI62779, ELE17451) => FRO31317, 106, 1592, 6.66%
(DAI62779, ELE17451) => FRO40251, 406, 1592, 25.50%
(DAI62779, ELE17451) => FRO78087, 121, 1592, 7.60%
(DAI62779, ELE17451) => FRO80039, 130, 1592, 8.17%
(DAI62779, ELE17451) => GRO15017, 111, 1592, 6.97%
(DAI62779, ELE17451) => GRO30386, 218, 1592, 13.69%
(DAI62779, ELE17451) => GRO46854, 109, 1592, 6.85%
(DAI62779, ELE17451) => GRO59710, 213, 1592, 13.38%
(DAI62779, ELE17451) => GRO71621, 159, 1592, 9.99%
(DAI62779, ELE17451) => GRO73461, 245, 1592, 15.39%
(DAI62779, ELE17451) => GRO81087, 160, 1592, 10.05%
(DAI62779, ELE17451) => GRO85051, 178, 1592, 11.18%
(DAI62779, ELE17451) => GRO94758, 180, 1592, 11.31%
(DAI62779, ELE17451) => SNA18336, 244, 1592, 15.33%
(DAI62779, ELE17451) => SNA38068, 118, 1592, 7.41%
(DAI62779, ELE17451) => SNA45677, 158, 1592, 9.92%
(DAI62779, ELE17451) => SNA55762, 157, 1592, 9.86%
(DAI62779, ELE17451) => SNA59903, 202, 1592, 12.69%
(DAI62779, ELE17451) => SNA72163, 107, 1592, 6.72%
(DAI62779, ELE17451) => SNA80324, 417, 1592, 26.19%
(DAI62779, ELE17451) => SNA90094, 103, 1592, 6.47%
WARNING: mismatch between DAI62779_ELE17451_SNA96271 and DAI62779_ELE20398_*(113)
(DAI62779, ELE20847) => FRO40251, 148, 275, 53.82%
(DAI62779, ELE20847) => SNA80324, 153, 275, 55.64%
WARNING: mismatch between DAI62779_ELE21353_FRO19221 and DAI62779_ELE24630_*(132)
(DAI62779, ELE26917) => FRO40251, 109, 650, 16.77%
(DAI62779, ELE32164) => ELE92920, 165, 832, 19.83%
(DAI62779, ELE32164) => GRO30386, 118, 832, 14.18%
(DAI62779, ELE32164) => GRO59710, 301, 832, 36.18%
(DAI62779, ELE32164) => GRO73461, 131, 832, 15.75%
(DAI62779, ELE59028) => FRO85978, 146, 370, 39.46%
WARNING: mismatch between DAI62779_ELE59028_SNA93860 and DAI62779_ELE59935_*(351)
(DAI62779, ELE74009) => ELE92920, 105, 432, 24.31%
(DAI62779, ELE78169) => GRO94758, 109, 213, 51.17%
(DAI62779, ELE92920) => FRO40251, 152, 877, 17.33%
(DAI62779, ELE92920) => GRO15017, 143, 877, 16.31%
(DAI62779, ELE92920) => GRO59710, 116, 877, 13.23%
(DAI62779, ELE92920) => GRO81087, 134, 877, 15.28%
(DAI62779, ELE92920) => SNA18336, 432, 877, 49.26%
(DAI62779, FRO19221) => GRO73461, 142, 462, 30.74%
(DAI62779, FRO19221) => SNA53220, 131, 462, 28.35%
WARNING: mismatch between DAI62779_FRO19221_SNA93860 and DAI62779_FRO24098_*(133)
(DAI62779, FRO40251) => FRO92469, 238, 1070, 22.24%
(DAI62779, FRO40251) => GRO30386, 114, 1070, 10.65%
(DAI62779, FRO40251) => GRO71621, 102, 1070, 9.53%
(DAI62779, FRO40251) => GRO73461, 315, 1070, 29.44%
(DAI62779, FRO40251) => GRO85051, 381, 1070, 35.61%
(DAI62779, FRO40251) => SNA18336, 102, 1070, 9.53%
(DAI62779, FRO40251) => SNA80324, 476, 1070, 44.49%
(DAI62779, FRO85978) => SNA93860, 156, 434, 35.94%
WARNING: mismatch between DAI62779_FRO85978_SNA95666 and DAI62779_FRO92261_*(223)
WARNING: mismatch between DAI62779_FRO92469_SNA80324 and DAI62779_FRO98184_*(118)
(DAI62779, GRO15017) => SNA18336, 105, 391, 26.85%
(DAI62779, GRO21487) => GRO73461, 173, 471, 36.73%
(DAI62779, GRO30386) => GRO59710, 101, 709, 14.25%
(DAI62779, GRO30386) => GRO73461, 186, 709, 26.23%
(DAI62779, GRO30386) => SNA80324, 136, 709, 19.18%
(DAI62779, GRO38814) => GRO73461, 154, 389, 39.59%
(DAI62779, GRO46854) => GRO73461, 135, 461, 29.28%
(DAI62779, GRO71621) => GRO73461, 153, 595, 25.71%
```

```
(DAI62779, GRO73461) => SNA45677, 112, 1139, 9.83%
(DAI62779, GRO73461) => SNA55762, 109, 1139, 9.57%
(DAI62779, GRO73461) => SNA80324, 198, 1139, 17.38%
WARNING: mismatch between DAI62779_GRO73461_SNA96271 and DAI62779_GRO81087_*(396)
WARNING: mismatch between DAI62779_GRO85051_SNA80324 and DAI62779_GRO88324_*(237)
(DAI62779, GRO94758) => SNA45677, 116, 479, 24.22%
WARNING: mismatch between DAI62779_GRO94758_SNA80324 and DAI62779_GRO99222_*(237)
WARNING: mismatch between DAI62779_SNA45677_SNA96271 and DAI62779_SNA53220_*(248)
WARNING: mismatch between DAI62779_SNA53220_SNA93860 and DAI62779_SNA55762_*(593)
WARNING: mismatch between DAI62779_SNA59903_SNA72163 and DAI62779_SNA72163_*(279)
(DAI75645, DAI85309) => FRO40251, 103, 212, 48.58%
(DAI75645, DAI88079) => FRO40251, 148, 149, 99.33%
(DAI75645, ELE17451) => FRO40251, 292, 547, 53.38%
(DAI75645, ELE17451) => GRO73461, 121, 547, 22.12%
(DAI75645, ELE17451) => SNA80324, 300, 547, 54.84%
(DAI75645, ELE20847) => FRO40251, 153, 306, 50.00%
(DAI75645, ELE20847) => SNA80324, 164, 306, 53.59%
(DAI75645, ELE26917) => FRO40251, 128, 278, 46.04%
(DAI75645, ELE26917) => SNA80324, 130, 278, 46.76%
(DAI75645, ELE74009) => FRO40251, 139, 286, 48.60%
(DAI75645, ELE74009) => SNA80324, 106, 286, 37.06%
(DAI75645, FRO40251) => FRO53271, 123, 1254, 9.81%
(DAI75645, FRO40251) => FRO92469, 251, 1254, 20.02%
(DAI75645, FRO40251) => GRO21487, 107, 1254, 8.53%
(DAI75645, FRO40251) => GRO30386, 109, 1254, 8.69%
(DAI75645, FRO40251) => GRO38814, 117, 1254, 9.33%
(DAI75645, FRO40251) => GRO71621, 112, 1254, 8.93%
(DAI75645, FRO40251) => GRO73461, 293, 1254, 23.37%
(DAI75645, FRO40251) => GRO81087, 112, 1254, 8.93%
(DAI75645, FRO40251) => GRO85051, 395, 1254, 31.50%
(DAI75645, FRO40251) => GRO94758, 118, 1254, 9.41%
(DAI75645, FRO40251) => SNA45677, 120, 1254, 9.57%
(DAI75645, FRO40251) => SNA55762, 131, 1254, 10.45%
(DAI75645, FRO40251) => SNA80324, 550, 1254, 43.86%
WARNING: mismatch between DAI75645_FRO47962_GRO73461 and DAI75645_FRO53271_*(210)
WARNING: mismatch between DAI75645_FRO92469_SNA80324 and DAI75645_GRO15017_*(228)
(DAI75645, GRO21487) => GRO73461, 114, 213, 53.52%
(DAI75645, GRO30386) => SNA80324, 131, 239, 54.81%
(DAI75645, GRO38814) => GRO73461, 101, 244, 41.39%
WARNING: mismatch between DAI75645_GRO38814_SNA80324 and DAI75645_GRO44993_*(120)
(DAI75645, GRO46854) => GRO73461, 101, 190, 53.16%
(DAI75645, GRO73461) => SNA80324, 230, 712, 32.30%
WARNING: mismatch between DAI75645_GRO81087_SNA80324 and DAI75645_GRO85051_*(395)
WARNING: mismatch between DAI75645_GRO85051_SNA80324 and DAI75645_GRO94758_*(203)
WARNING: mismatch between DAI75645_GRO94758_SNA80324 and DAI75645_SNA38068_*(124)
(DAI75645, SNA45677) => SNA80324, 111, 245, 45.31%
WARNING: mismatch between DAI75645_SNA55762_SNA80324 and DAI75645_SNA72163_*(131)
(DAI85309, ELE17451) => ELE92920, 137, 482, 28.42%
(DAI85309, ELE17451) => SNA18336, 119, 482, 24.69%
(DAI85309, ELE92920) => SNA18336, 139, 201, 69.15%
(DAI85309, ELE99737) => FRO19221, 111, 659, 16.84%
(DAI85309, ELE99737) => GRO94758, 102, 659, 15.48%
(DAI85309, ELE99737) => SNA45677, 105, 659, 15.93%
(DAI88079, ELE17451) => FRO40251, 123, 124, 99.19%
(DAI88079, FRO40251) => GRO73461, 144, 446, 32.29%
WARNING: mismatch between DAI88079_FRO40251_SNA80324 and DAI88079_GRO73461_*(145)
(DAI88807, ELE17451) => SNA59903, 120, 291, 41.24%
(DAI88807, ELE17451) => SNA72163, 105, 291, 36.08%
(DAI88807, GRO73461) => SNA72163, 110, 313, 35.14%
WARNING: mismatch between DAI88807_SNA59903_SNA72163 and DAI88807_SNA72163_*(394)
(ELE17451, ELE32164) => GRO59710, 202, 511, 39.53%
(ELE17451, ELE92920) => SNA18336, 228, 384, 59.38%
(ELE17451, FRO40251) => FRO92469, 162, 697, 23.24%
(ELE17451, FRO40251) => GRO73461, 159, 697, 22.81%
(ELE17451, FRO40251) => GRO85051, 217, 697, 31.13%
(ELE17451, FRO40251) => SNA80324, 353, 697, 50.65%
(ELE17451, GRO30386) => GRO73461, 103, 468, 22.01%
WARNING: mismatch between ELE17451_GRO85051_SNA80324 and ELE17451_GRO88324_*(117)
WARNING: mismatch between ELE17451_SNA59903_SNA72163 and ELE17451_SNA72163_*(272)
(ELE20847, FRO40251) => FRO92469, 122, 434, 28.11%
(ELE20847, FRO40251) => GRO85051, 139, 434, 32.03%
(ELE20847, FRO40251) => SNA80324, 232, 434, 53.46%
(ELE26917, FRO40251) => GRO85051, 146, 346, 42.20%
(ELE26917, FRO40251) => SNA80324, 133, 346, 38.44%
(ELE32164, FRO53271) => GRO59710, 105, 254, 41.34%
(ELE32164, GRO59710) => GRO73461, 137, 911, 15.04%
WARNING: mismatch between ELE59028_FRO85978_SNA93860 and ELE59935_FRO40251_*(134)
WARNING: mismatch between ELE78169_GRO94758_SNA45677 and ELE91337_FRO35904_*(104)
```

```
WARNING: mismatch between FRO19221_SNA53220_SNA93860 and FRO24098_FRO40251_*(106)
(FRO40251, FRO53271) => GRO85051, 105, 309, 33.98%
WARNING: mismatch between FRO40251_FRO53271_SNA80324 and FRO40251_FRO61354_*(126)
(FRO40251, FRO80039) => SNA80324, 104, 249, 41.77%
(FRO40251, FRO92469) => GRO73461, 211, 835, 25.27%
WARNING: mismatch between FRO40251_FRO92469_SNA80324 and FRO40251_FRO98729_*(107)
(FRO40251, GRO21487) => GRO73461, 182, 375, 48.53%
(FRO40251, GRO21487) => GRO85051, 120, 375, 32.00%
(FRO40251, GRO21487) => SNA80324, 118, 375, 31.47%
(FRO40251, GRO30386) => SNA80324, 103, 224, 45.98%
(FRO40251, GRO38814) => GRO73461, 106, 295, 35.93%
(FRO40251, GRO38814) => GRO85051, 115, 295, 38.98%
WARNING: mismatch between FRO40251_GRO38814_SNA80324 and FRO40251_GRO38983_*(153)
(FRO40251, GRO46854) => GRO73461, 106, 210, 50.48%
(FRO40251, GRO56726) => GRO73461, 103, 247, 41.70%
(FRO40251, GRO69543) => GRO73461, 111, 227, 48.90%
(FRO40251, GRO71621) => SNA80324, 142, 288, 49.31%
(FRO40251, GRO73461) => GRO85051, 147, 882, 16.67%
(FRO40251, GRO73461) => SNA80324, 232, 882, 26.30%
WARNING: mismatch between FRO40251_GRO81087_SNA80324 and FRO40251_GRO85051_*(1213)
(FRO40251, GRO85051) => SNA45677, 107, 1213, 8.82%
WARNING: mismatch between FRO40251_GRO85051_SNA80324 and FRO40251_GRO94758_*(230)
WARNING: mismatch between FRO40251_GRO94758_SNA80324 and FRO40251_GRO99222_*(142)
(FRO40251, SNA45677) => SNA80324, 126, 309, 40.78%
WARNING: mismatch between FRO40251_SNA55762_SNA80324 and FRO40251_SNA72163_*(201)
WARNING: mismatch between FRO40251_SNA80324_SNA96271 and FRO40251_SNA90094_*(201)
WARNING: mismatch between FRO73056_GRO44993_GRO73461 and FRO78087_FRO80039_*(116)
```

HW3.8

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.

HW3.8 (Conceptual Exercise)