Mark von Oven - Homework 2

Data Analytics & Machine Learning at Scale

HW2.0:

How do you merge two sorted lists/arrays of records of the form [key, value]? Where is this used in Hadoop MapReduce? [Hint within the shuffle]

What is a combiner function in the context of Hadoop?

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

What is the Hadoop shuffle?

HW2.0: When you pass records in [key,value] form to the reducer, Hadoop merges them automatically for you. The combiner function is an aggregator that can be invoked before sending records to the reducer. This can be helpful when there is a large amount of information, and combining it can preserve network bandwidth. The Hadoop shuffle is what occurs when Hadoop is preparing the map outputs to be sent to the reducer(s) based on output keys.

HW2.1: Counters as a debugging aid (and for getting work done, but please use sparingly as they are heavy)

Consumer complaints dataset: 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 to complete this question:

```
https://www.dropbox.com/s/vbalm3yva2rr86m/Consumer_Complaints.csv?dl=0
```

Use the following command to grab the file:

```
curl -L https://www.dropbox.com/s/vbalm3yva2rr86m/Consumer_Complaints.csv?dl=0 -o Consumer_Complai
nts.csv
```

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 MapReduce 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.

In [7]:
```
%%writefile MRcomplaint_counter.py
from mrjob.job import MRJob
from mrjob.step import MRStep

class MRComplaintCount(MRJob):
    def mapper(self, _, line):
        complaint_id, complaint_type, _ = line.split(',', 2)
        if complaint_type != "Product":
            if complaint_type in ["Debt collection", "Mortgage"]:
                self.increment_counter(complaint_type, 'Num_mapper_calls', 1)
                yield complaint_type, 1
            else:
                self.increment_counter("Other", 'Num_mapper_calls', 1)
                yield "Other", 1

    def reducer(self, word, counts):
        self.increment_counter(word, 'Num_reducer_calls', 1)
        yield word, sum(counts)

if __name__ == '__main__':
    MRComplaintCount.run()
```

```
Overwriting MRcomplaint_counter.py
```

In [9]:
```
#HW2.1
%reload_ext autoreload
%autoreload 2

from MRcomplaint_counter import MRComplaintCount
mr_job = MRComplaintCount(args=['Consumer_Complaints.csv'])
with mr_job.make_runner() as runner:
    runner.run()
    # stream_output: get access of the output
    print runner.counters()
```

```
[{'Debt collection': {'Num_mapper_calls': 44372, 'Num_reducer_calls': 1}, 'Other': {'Num_map
per_calls': 142788, 'Num_reducer_calls': 1}, 'Mortgage': {'Num_mapper_calls': 125752, 'Num_r
educer_calls': 1}}]
```

HW2.2: 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.

In [25]:
```
!echo foo foo quux labs foo bar quux > WordCount.txt
```

In [10]:
```
%%writefile mr_wc_counter.py
#HW2.2
from mrjob.job import MRJob
from mrjob.step import MRStep
import re

WORD_RE = re.compile(r"[\w']+")

class MRWordFreqCount(MRJob):
    def mapper(self, _, line):
        self.increment_counter('group', 'Num_mapper_calls', 1)
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

    def reducer(self, word, counts):
        self.increment_counter('group', 'Num_reducer_calls', 1)
        yield word, sum(counts)

if __name__ == '__main__':
    MRWordFreqCount.run()
```

```
Overwriting mr_wc_counter.py
```

In [11]:
```
#HW2.2
%reload_ext autoreload
%autoreload 2

from mr_wc_counter import MRWordFreqCount
mr_job = MRWordFreqCount(args=['WordCount.txt'])
with mr_job.make_runner() as runner:
    runner.run()
    # stream_output: get access of the output
    print runner.counters()
```

```
[{'group': {'Num_mapper_calls': 1, 'Num_reducer_calls': 4}}]
```

HW2.2: The mapper is only called once because there is only 1 line being analyzed...the line and the code is sent to a single mapper to be processed. The reducer is called four times because there are four unique keys (i.e. words) in the output of the mapper (i.e. foo, quux, labs, bar). All keys containing a unique word are sent to the same reducer and then reduced to a single version of key, sum(value)

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 times 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.

In [12]:
```
%%writefile mr_CCC_counter.py
#HW2.2
from mrjob.job import MRJob
from mrjob.step import MRStep

class MRWordFreqCount(MRJob):
    def mapper(self, _, line):
        c_id, c_type, sub_prod, issue, _ = line.split(',', 4)
        if issue != "Issue":
            self.increment_counter('group', 'Num_mapper_calls', 1)
            yield issue, 1

    def reducer(self, word, counts):
        self.increment_counter('group', 'Num_reducer_calls', 1)
        yield word, sum(counts)

if __name__ == '__main__':
    MRWordFreqCount.run()
```

```
Overwriting mr_CCC_counter.py
```

In [13]:
```
#HW2.2
%reload_ext autoreload
%autoreload 2

from mr_CCC_counter import MRWordFreqCount
mr_job = MRWordFreqCount(args=['Consumer_Complaints.csv'])
with mr_job.make_runner() as runner:
    runner.run()
    # stream_output: get access of the output
    print runner.counters()
```

```
[{'group': {'Num_mapper_calls': 312912, 'Num_reducer_calls': 80}}]
```

HW2.2 The mapper is called 312912 times and the reducer 80 times. [{'group': {'Num_mapper_calls': 312912, 'Num_reducer_calls': 80}}]

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, combiner counter, and Reducer Counter after completing your word count job.

In [68]:
```
%%writefile mr_CCCCCC_counter.py
#HW2.2
from mrjob.job import MRJob
from mrjob.step import MRStep

class MRWordFreqCount(MRJob):
    def mapper(self, _, line):
        c_id, c_type, sub_prod, issue, _ = line.split(',', 4)
        if issue != "Issue":
            self.increment_counter('group', 'Num_mapper_calls', 1)
            yield issue, 1

    def combiner(self, key, values):
        self.increment_counter('group', 'Num_combiner_calls', 1)
        yield key, sum(values)

    def reducer(self, word, counts):
        self.increment_counter('group', 'Num_reducer_calls', 1)
        yield word, sum(counts)

if __name__ == '__main__':
    MRWordFreqCount.run()
```

```
Writing mr_CCCCCC_counter.py
```

In [69]:
```
#HW2.2
%reload_ext autoreload
%autoreload 2

from mr_CCCCCC_counter import MRWordFreqCount
mr_job = MRWordFreqCount(args=['Consumer_Complaints.csv'])
with mr_job.make_runner() as runner:
    runner.run()
    # stream_output: get access of the output
    print runner.counters()
```

```
[{'group': {'Num_combiner_calls': 147, 'Num_mapper_calls': 312912, 'Num_reducer_calls': 8
0}}]
```

HW2.2.1:

Using a single reducer perform a sort of the words in decreasing order of word counts. Present the top 50 terms and their frequency. If there are ties please sort the tokens in alphanumeric/string order. Present bottom 10 tokens (least frequent items).

HINT: You will need a second MRStep for the sort part. Step 1 will be the usual word count, while step 2 will be a sort step. Please use the Hadoop/MRJob framework to perform the sort. Please do NOT use any of the built-in sorts from python.

In [5]:

```
%%writefile MRWordFreqCountSort.py
#HW2.2.1

from mrjob.job import MRJob
from mrjob.step import MRStep
import sys, re, string, operator

class MRWordFreqCountSort(MRJob):

    def get_words(self, _, line):
        regex = re.compile('[%s]' % re.escape(string.punctuation))
        line = regex.sub(' ', str(line).lower())
        line = re.sub( '\s+', ' ', line )

        words = line.split()
        for w in words:
            yield(w, 1)

    def sum_words(self, word, values):
        yield(word, sum(values))

    def sort_words(self, word, value):
        newkey = '{number:012d}'.format(number=int(value))+word
        yield(newkey, [word, value])

    def red_init(self):
        self.d=[]

    def reduce_words(self, key, values):
        for value in values:
            self.d.append([value[0], value[1]])

    def reduce_words_final(self):
        index = len(self.d)-1
        bottom = 10
#        while index >=0:                    #values in desc order
#            yield(self.d[index][0], self.d[index][1])
#            index -= 1
#        for key, value in self.d:          #if i want the values in asc order
#            yield(key, value)
```

```
        print("***Top 50***")
        for i in range(50):
            print(self.d[index - i][0], self.d[index - i][1])
        print("***Bottom 10***")
        for j in range(10):
            print(self.d[j][0], self.d[j][1] )

    def steps(self):
        return [MRStep(mapper=self.get_words,
                       reducer=self.sum_words),
                MRStep(mapper=self.sort_words,
                       reducer_init=self.red_init,
                       reducer=self.reduce_words,
                       reducer_final=self.reduce_words_final)]

if __name__ == '__main__':
    MRWordFreqCountSort.run()
```

```
Overwriting MRWordFreqCountSort.py
```

In [6]:
```
!python MRWordFreqCountSort.py --jobconf mapred.reduce.tasks=1 Consumer_Complaints.csv
```

```
No configs found; falling back on auto-configuration
Creating temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/MRWordFreqCountSor
t.z013hnv.20160707.212422.141760
Running step 1 of 2...
Running step 2 of 2...
***Top 50***
('yes', 369651)
('closed', 309303)
('with', 288613)
('mortgage', 267787)
('2014', 260871)
('no', 229714)
('explanation', 215782)
('2013', 215714)
('web', 183716)
('loan', 170400)
('credit', 159787)
('2012', 144318)
('account', 143647)
('collection', 118073)
('bank', 108378)
('debt', 98527)
('or', 95366)
('relief', 86076)
('07', 80935)
('other', 80052)
('08', 79720)
('06', 79413)
('10', 78830)
('03', 77247)
('referral', 77087)
('05', 76342)
('04', 75741)
('09', 75317)
('of', 71382)
('modification', 70487)
('foreclosure', 70487)
('02', 68426)
('01', 67421)
('monetary', 62838)
```

```
('12', 56896)
('card', 55750)
('11', 55201)
('conventional', 50516)
('non', 50093)
('not', 48628)
('reporting', 47793)
('ca', 47076)
('servicing', 45986)
('service', 44950)
('america', 42530)
('report', 40675)
('payments', 40000)
('information', 39523)
('phone', 37790)
('fixed', 36857)
***Bottom 10***
('00001', 1)
('00002', 1)
('00005', 1)
('00008', 1)
('00017', 1)
('00018', 1)
('00023', 1)
('00024', 1)
('00026', 1)
('00032', 1)
Streaming final output from /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/MRWordFreqCount
Sort.z013hnv.20160707.212422.141760/output...
Removing temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/MRWordFreqCountSor
t.z013hnv.20160707.212422.141760...
```

HW2.2.2:

Repeat HW2.2.1 using 3 reducers. Use the same code as in HW2.2.1 with just one modification to the command line: just add --jobconf mapred.reduce.tasks=3 as see presented here:

```
python HW2.2WordCount.py --jobconf mapred.reduce.tasks=3 oneLinerTextFile.txt
```

Describe what you see. Is this correct?

In [7]:
```
!python MRWordFreqCountSort.py --jobconf mapred.reduce.tasks=3 Consumer_Complaints.csv
```

```
No configs found; falling back on auto-configuration
Creating temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/MRWordFreqCountSor
t.z013hnv.20160707.212659.962669
Running step 1 of 2...
Running step 2 of 2...
***Top 50***
('375865', 1)
('375861', 1)
('37586', 1)
('375859', 1)
('375856', 1)
('375855', 1)
('375852', 1)
('375851', 1)
('37585', 1)
('375847', 1)
('375845', 1)
('375843', 1)
('375842', 1)
('375841', 1)
('375840', 1)
('375839', 1)
('375838', 1)
('375837', 1)
('375835', 1)
('375834', 1)
('375833', 1)
('375831', 1)
('375829', 1)
('375826', 1)
('375823', 1)
('375820', 1)
('375819', 1)
('375817', 1)
('375816', 1)
('375815', 1)
('375814', 1)
('375813', 1)
('375812', 1)
('375810', 1)
```

```
('375805', 1)
('375804', 1)
('37580', 1)
('375798', 1)
('375796', 1)
('375795', 1)
('375794', 1)
('375791', 1)
('375788', 1)
('375786', 1)
('375784', 1)
('375780', 1)
('375777', 1)
('375776', 1)
('375775', 1)
('375770', 1)
***Bottom 10***
('00001', 1)
('00002', 1)
('00005', 1)
('00008', 1)
('00017', 1)
('00018', 1)
('00023', 1)
('00024', 1)
('00026', 1)
('00032', 1)
***Top 50***
('724050', 1)
('724047', 1)
('724039', 1)
('724034', 1)
('724032', 1)
('724030', 1)
('72403', 1)
('724024', 1)
('724023', 1)
('724022', 1)
('724018', 1)
('724011', 1)
```

```
('724008', 1)
('724007', 1)
('723997', 1)
('723995', 1)
('723990', 1)
('723989', 1)
('723987', 1)
('723978', 1)
('723969', 1)
('723967', 1)
('723966', 1)
('723965', 1)
('723963', 1)
('723958', 1)
('723956', 1)
('723950', 1)
('723947', 1)
('723942', 1)
('723941', 1)
('723930', 1)
('72393', 1)
('723924', 1)
('723920', 1)
('723913', 1)
('723910', 1)
('723905', 1)
('723901', 1)
('723900', 1)
('723895', 1)
('723891', 1)
('723889', 1)
('723886', 1)
('723880', 1)
('723878', 1)
('723874', 1)
('723870', 1)
('72387', 1)
('723869', 1)
***Bottom 10***
('375866', 1)
```

```
('375869', 1)
('375871', 1)
('375874', 1)
('375876', 1)
('375877', 1)
('375881', 1)
('375882', 1)
('375886', 1)
('375887', 1)
***Top 50***
('yes', 369651)
('closed', 309303)
('with', 288613)
('mortgage', 267787)
('2014', 260871)
('no', 229714)
('explanation', 215782)
('2013', 215714)
('web', 183716)
('loan', 170400)
('credit', 159787)
('2012', 144318)
('account', 143647)
('collection', 118073)
('bank', 108378)
('debt', 98527)
('or', 95366)
('relief', 86076)
('07', 80935)
('other', 80052)
('08', 79720)
('06', 79413)
('10', 78830)
('03', 77247)
('referral', 77087)
('05', 76342)
('04', 75741)
('09', 75317)
('of', 71382)
('modification', 70487)
```

```
('foreclosure', 70487)
('02', 68426)
('01', 67421)
('monetary', 62838)
('12', 56896)
('card', 55750)
('11', 55201)
('conventional', 50516)
('non', 50093)
('not', 48628)
('reporting', 47793)
('ca', 47076)
('servicing', 45986)
('service', 44950)
('america', 42530)
('report', 40675)
('payments', 40000)
('information', 39523)
('phone', 37790)
('fixed', 36857)
***Bottom 10***
('724052', 1)
('724053', 1)
('724054', 1)
('724063', 1)
('724065', 1)
('724072', 1)
('724076', 1)
('724077', 1)
('724081', 1)
('724083', 1)
Streaming final output from /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/MRWordFreqCount
Sort.z013hnv.20160707.212659.962669/output...
Removing temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/MRWordFreqCountSor
t.z013hnv.20160707.212659.962669...
```

HW2.2.2 Using 3 reducers for this sort task ends up sorting and printing the list in 3 different chunks. Each list is correct on it's own, but the point of the exercise was to sort and print the entire list - which means this is incorrect

HW2.2.3: [Optional; we will cover this in class]

Solve the "total sort" issue in HW2.2.2.

HW2.2.3 Looking forward to it :)

HW2.3: 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 located at:

```
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 guided by the following questions:.

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, (break ties by sorting the products alphabetical order) etc. using Hadoop Map-Reduce.

In [18]:

```
%%writefile wordcountSCA.py
#HW2.3

from mrjob.job import MRJob
from mrjob.step import MRStep
import sys, re, string, operator

class wordCountSCA(MRJob):

    def get_words(self, _, line):
        regex = re.compile('[%s]' % re.escape(string.punctuation))
        line = regex.sub(' ', str(line).lower())
        line = re.sub( '\s+', ' ', line )

        words = line.split()                                    # split each basket into
items
        basket = ''.join(map(str, words))                       # create a unique key fo
r each basket

        for w in words:
            yield(w, 1)                                         # count each item once f
or item totals
            yield("BASKET "+basket, 1)                          # count each item once f
or basket size

    def sum_words(self, word, values):
        yield(word, sum(values))

    def sort_words(self, word, value):
        newkey = '{number:012d}'.format(number=int(value))+word
        yield(newkey, [word, value])

    def red_init(self):
        self.d=[]
        self.bs=[]

    def reduce_words(self, key, values):
        for value in values:
            if value[0][:6]!="BASKET":
                self.d.append([value[0], value[1]])
```

```
            else:
                self.bs.append([value[0], value[1]])

    def reduce_words_final(self):
        unique_items = len(self.d)
        num_baskets = len(self.bs)
        max_d = unique_items-1
        max_bs = num_baskets-1
        print("Number of unique items is %s" % unique_items)
        print("Largest basket has %s items" % self.bs[max_bs][1])
        print("***Top 50***")
        for i in range(50):
            print(self.d[max_d - i][0], self.d[max_d - i][1])

    def steps(self):
        return [MRStep(mapper=self.get_words,
                       reducer=self.sum_words),
                MRStep(mapper=self.sort_words,
                       reducer_init=self.red_init,
                       reducer=self.reduce_words,
                       reducer_final=self.reduce_words_final)]

if __name__ == '__main__':
    wordCountSCA.run()
```

```
Overwriting wordcountSCA.py
```

In [20]:
```
!python wordCountSCA.py --jobconf mapred.reduce.tasks=1 ProductPurchaseData.txt
```

```
No configs found; falling back on auto-configuration
Creating temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/wordCountSCA.z013hn
v.20160708.011531.218113
Running step 1 of 2...
Running step 2 of 2...
Number of unique items is 12592
Largest basket has 288 items
***Top 50***
('dai62779', 6667)
('fro40251', 3881)
('ele17451', 3875)
('gro73461', 3602)
('sna80324', 3044)
('ele32164', 2851)
('dai75645', 2736)
('sna45677', 2455)
('fro31317', 2330)
('dai85309', 2293)
('ele26917', 2292)
('fro80039', 2233)
('gro21487', 2115)
('sna99873', 2083)
('gro59710', 2004)
('gro71621', 1920)
('fro85978', 1918)
('gro30386', 1840)
('ele74009', 1816)
('gro56726', 1784)
('dai63921', 1773)
('gro46854', 1756)
('ele66600', 1713)
('dai83733', 1712)
('fro32293', 1702)
('ele66810', 1697)
('sna55762', 1646)
('dai22177', 1627)
('fro78087', 1531)
('ele99737', 1516)
('gro94758', 1489)
('ele34057', 1489)
```

```
('fro35904', 1436)
('fro53271', 1420)
('sna93860', 1407)
('sna90094', 1390)
('gro38814', 1352)
('ele56788', 1345)
('gro61133', 1321)
('ele74482', 1316)
('dai88807', 1316)
('ele59935', 1311)
('sna96271', 1295)
('dai43223', 1290)
('ele91337', 1289)
('gro15017', 1275)
('dai31081', 1261)
('gro81087', 1220)
('dai22896', 1219)
('gro85051', 1214)
Streaming final output from /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/wordCountSCA.z0
13hnv.20160708.011531.218113/output...
Removing temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/wordCountSCA.z013hn
v.20160708.011531.218113...
```

2.3.1 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 frequency (break ties by sorting the products alphabetical order) etc. using Hadoop Map-Reduce.

In [21]:
```
!python wordCountSCA.py --jobconf mapred.reduce.tasks=2 ProductPurchaseData.txt
```

```
No configs found; falling back on auto-configuration
Creating temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/wordCountSCA.z013hn
v.20160708.011739.135101
Running step 1 of 2...
Running step 2 of 2...
Number of unique items is 9646
Largest basket has 13 items
***Top 50***
('sna98720', 12)
('sna85641', 12)
('sna84439', 12)
('sna83731', 12)
('sna82197', 12)
('sna81989', 12)
('sna77281', 12)
('sna72305', 12)
('sna71126', 12)
('sna61679', 12)
('sna58201', 12)
('sna54318', 12)
('sna52223', 12)
('sna46099', 12)
('sna37804', 12)
('sna34587', 12)
('sna31046', 12)
('sna29452', 12)
('sna28368', 12)
('sna26649', 12)
('sna23770', 12)
('sna20107', 12)
('gro91087', 12)
('gro88061', 12)
('gro85444', 12)
('gro84449', 12)
('gro84258', 12)
('gro83392', 12)
('gro81597', 12)
('gro80995', 12)
('gro75023', 12)
('gro68313', 12)
```

```
('gro66865', 12)
('gro62704', 12)
('gro59094', 12)
('gro52534', 12)
('gro50736', 12)
('gro49643', 12)
('gro49322', 12)
('gro46512', 12)
('gro43461', 12)
('gro39193', 12)
('gro37355', 12)
('gro34212', 12)
('gro34170', 12)
('gro29410', 12)
('gro29257', 12)
('gro27151', 12)
('gro23893', 12)
('gro19142', 12)
Number of unique items is 2946
Largest basket has 288 items
***Top 50***
('dai62779', 6667)
('fro40251', 3881)
('ele17451', 3875)
('gro73461', 3602)
('sna80324', 3044)
('ele32164', 2851)
('dai75645', 2736)
('sna45677', 2455)
('fro31317', 2330)
('dai85309', 2293)
('ele26917', 2292)
('fro80039', 2233)
('gro21487', 2115)
('sna99873', 2083)
('gro59710', 2004)
('gro71621', 1920)
('fro85978', 1918)
('gro30386', 1840)
('ele74009', 1816)
```

```
('gro56726', 1784)
('dai63921', 1773)
('gro46854', 1756)
('ele66600', 1713)
('dai83733', 1712)
('fro32293', 1702)
('ele66810', 1697)
('sna55762', 1646)
('dai22177', 1627)
('fro78087', 1531)
('ele99737', 1516)
('gro94758', 1489)
('ele34057', 1489)
('fro35904', 1436)
('fro53271', 1420)
('sna93860', 1407)
('sna90094', 1390)
('gro38814', 1352)
('ele56788', 1345)
('gro61133', 1321)
('ele74482', 1316)
('dai88807', 1316)
('ele59935', 1311)
('sna96271', 1295)
('dai43223', 1290)
('ele91337', 1289)
('gro15017', 1275)
('dai31081', 1261)
('gro81087', 1220)
('dai22896', 1219)
('gro85051', 1214)
Streaming final output from /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/wordCountSCA.z0
13hnv.20160708.011739.135101/output...
Removing temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/wordCountSCA.z013hn
v.20160708.011739.135101...
```

HW2.4. (Computationally prohibitive but then again Hadoop can handle this) Pairs

From a data mining perspective (and the aPriori algorihtm in particular), Support and Confidence are defined as follows:

```
   SUPPORT
   In data mining, the support value of X (where X is a collection of cooccurring items sometimes
referred to as
   an item-set. E.g., a basket or subset of a basket) with respect to T  (a transaction database w
here each
   row is a transaction such as a basket of items that have been purchased)  is defined as the pro
portion
   of transactions in the  database which contains  the item-set X. (a relative frequency of sort
s)

   CONFIDENCE
   The confidence value of a rule, X ==>  Y (where X is a collection of cooccurring items and Y is
generally
   a single item. E.g., If Diapers and Beer then Cigars were also purchased), with respect to a se
t of transactions T, is the
   proportion of the transactions that contains X which also contains Y. (Think of it as  tgePr(Y|
X) )

   The pairs/stripes algorithm returns cooccurrence information that can be used directly to  calc
ulate the confidence and support.
   Note that confidence for pair X ==>  Y will  differ from the relative frequency that results fr
om stripes when X occurs by itself in transactions.
```

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 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.

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

```
item1, item2, support count, support  (OPTIONAL Feel free to add in confidence level also)
```

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

In [60]:

```
%%writefile pairsSCA.py
#HW2.4

from mrjob.job import MRJob
from mrjob.step import MRStep
import sys, re, string, operator, itertools

class pairsSCA(MRJob):

    def get_words(self, _, line):
        self.increment_counter('group', 'Num_mapper_step1_calls', 1)
        regex = re.compile('[%s]' % re.escape(string.punctuation))
        line = regex.sub(' ', str(line).lower())
        line = re.sub( '\s+', ' ', line )

        words = line.split()                                        # split each basket into
items
        basket = ''.join(map(str, words))                           # create a unique key fo
r each basket

        for w in words:
            yield(w, 1)                                             # count each item once f
or item totals
            yield("BASKET "+basket, 1)                              # count each item once f
or basket size
        for subset in itertools.combinations(sorted(set(words)), 2):
            yield("PAIRS "+str(subset[0])+" - "+str(subset[1]), 1)    # count each pair once

    def sum_words(self, word, values):
        self.increment_counter('group', 'Num_reducer_step1_calls', 1)
        yield(word, sum(values))

    def sort_words(self, word, value):
        self.increment_counter('group', 'Num_mapper_step2_calls', 1)
        newkey = '{number:012d}'.format(number=int(value))+word
        yield(newkey, [word, value])

    def red_init(self):
        self.d=[]
        self.bs=[]
```

```
        self.pairs=[]
        self.freqpairs=[]

    def reduce_words(self, key, values):
        self.increment_counter('group', 'Num_reducer_step2_calls', 1)
        for value in values:
            if value[0][:6]=="BASKET":
                self.bs.append([value[0], value[1]])
            elif value[0][:5]=="PAIRS":
                self.pairs.append([value[0][6:], value[1]])
            else:
                self.d.append([value[0], value[1]])

    def reduce_words_final(self):
        unique_items = len(self.d)
        num_baskets = len(self.bs)
        num_pairs = len(self.pairs)
        max_d = unique_items-1
        max_bs = num_baskets-1
        max_pairs = num_pairs-1
        print("Number of unique items is %s" % unique_items)
        print("Number of baskets is %s" % num_baskets)
        print("Largest basket has %s items" % self.bs[max_bs][1])
        print("Number of total pairs is %s" % num_pairs)
        for k in range(num_pairs):
            if self.pairs[k][1]>100:
                self.freqpairs.append([self.pairs[0], self.pairs[1]])
        num_freqpairs = len(self.freqpairs)
        print("Number of frequent pairs is %s" % num_freqpairs)
        print("***Top 50 pairs***")
        for i in range(50):
            print(self.pairs[max_pairs - i][0][:8], self.pairs[max_pairs - i][0][11:],
                  self.pairs[max_pairs - i][1], float(self.pairs[max_pairs - i][1])/num_basket
s)


    def steps(self):
        return [MRStep(mapper=self.get_words,
                       reducer=self.sum_words),
                MRStep(mapper=self.sort_words,
```

```
                       reducer_init=self.red_init,
                       reducer=self.reduce_words,
                       reducer_final=self.reduce_words_final)]

if __name__ == '__main__':
    pairsSCA.run()
```

```
Overwriting pairsSCA.py
```

In [62]:
```
%time !python pairsSCA.py --jobconf mapred.reduce.tasks=1 ProductPurchaseData.txt
```

```
No configs found; falling back on auto-configuration
Creating temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/pairsSCA.z013hnv.20
160708.033048.144792
Running step 1 of 2...
Counters: 1
        group
                Num_mapper_step1_calls=31101
Counters: 2
        group
                Num_mapper_step1_calls=31101
                Num_reducer_step1_calls=920384
Running step 2 of 2...
Counters: 1
        group
                Num_mapper_step2_calls=920384
Number of unique items is 12592
Number of baskets is 30697
Largest basket has 288 items
Number of total pairs is 877095
Number of frequent pairs is 1311
***Top 50 pairs***
('dai62779', 'ele17451', 1592, 0.05186174544743786)
('fro40251', 'sna80324', 1412, 0.0459979802586572)
('dai75645', 'fro40251', 1254, 0.040850897481838615)
('fro40251', 'gro85051', 1213, 0.039515262077727466)
('dai62779', 'gro73461', 1139, 0.037104603055673195)
('dai75645', 'sna80324', 1130, 0.03681141479623416)
('dai62779', 'fro40251', 1070, 0.034856826399973936)
('dai62779', 'sna80324', 923, 0.030068084829136397)
('dai62779', 'dai85309', 918, 0.029905202462781378)
('ele32164', 'gro59710', 911, 0.029677167149884352)
('fro40251', 'gro73461', 882, 0.028732449425025248)
('dai62779', 'dai75645', 882, 0.028732449425025248)
('dai62779', 'ele92920', 877, 0.02856956705867023)
('fro40251', 'fro92469', 835, 0.027201355181288075)
('dai62779', 'ele32164', 832, 0.027103625761475063)
('dai75645', 'gro73461', 712, 0.02319444896895462)
('dai43223', 'ele32164', 711, 0.023161872495683616)
('dai62779', 'gro30386', 709, 0.02309671954914161)
('ele17451', 'fro40251', 697, 0.022705801869889564)
```

```
('dai85309', 'ele99737', 659, 0.021467895885591427)
('dai62779', 'ele26917', 650, 0.02117470762615239)
('gro21487', 'gro73461', 631, 0.020555754634003325)
('dai62779', 'sna45677', 604, 0.019676189855686223)
('ele17451', 'sna80324', 597, 0.019448154542789198)
('dai62779', 'gro71621', 595, 0.01938300159624719)
('dai62779', 'sna55762', 593, 0.019317848649705184)
('dai62779', 'dai83733', 586, 0.01908981333680816)
('ele17451', 'gro73461', 580, 0.018894354497182134)
('gro73461', 'sna80324', 562, 0.01830797797830407)
('dai62779', 'gro59710', 561, 0.018275401505033064)
('dai62779', 'fro80039', 550, 0.017917060299052025)
('dai75645', 'ele17451', 547, 0.017819330879239013)
('dai62779', 'sna93860', 537, 0.017493566146528975)
('dai55148', 'dai62779', 526, 0.017135224940547936)
('dai43223', 'gro59710', 512, 0.016679154314753884)
('ele17451', 'ele32164', 511, 0.016646577841482883)
('dai62779', 'sna18336', 506, 0.016483695475127864)
('ele32164', 'gro73461', 486, 0.01583216600970779)
('dai85309', 'ele17451', 482, 0.015701860116623775)
('dai62779', 'fro78087', 482, 0.015701860116623775)
('dai62779', 'gro94758', 479, 0.015604130696810763)
('gro85051', 'sna80324', 471, 0.015343518910642734)
('dai62779', 'gro21487', 471, 0.015343518910642734)
('ele17451', 'gro30386', 468, 0.015245789490829723)
('fro85978', 'sna95666', 463, 0.015082907124474704)
('dai62779', 'fro19221', 462, 0.015050330651203701)
('dai62779', 'gro46854', 461, 0.015017754177932698)
('dai43223', 'dai62779', 459, 0.014952601231390689)
('ele92920', 'sna18336', 455, 0.014822295338306675)
('dai88079', 'fro40251', 446, 0.014529107078867641)
Counters: 2
        group
                Num_mapper_step2_calls=920384
                Num_reducer_step2_calls=920384
Streaming final output from /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/pairsSCA.z013hn
v.20160708.033048.144792/output...
Removing temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/pairsSCA.z013hnv.20
160708.033048.144792...
CPU times: user 692 ms, sys: 293 ms, total: 985 ms
```

```
Wall time: 1min 37s
```

HW2.5: Stripes

Repeat 2.4 using the stripes design pattern for finding cooccuring pairs (and out.

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

In [5]:

```
%%writefile stripes_madSCA.py
#HW2.5

from mrjob.job import MRJob
from mrjob.step import MRStep
import sys, re, string, operator, itertools

class stripes_madSCA(MRJob):

    def get_words(self, _, line):
        self.increment_counter('group', 'Num_mapper_calls', 1)
        regex = re.compile('[%s]' % re.escape(string.punctuation))
        line = regex.sub(' ', str(line).lower())
        line = re.sub( '\s+', ' ', line )

        words = line.split()                                          # split each basket into
items
        basket = ''.join(map(str, words))                             # create a unique key fo
r each basket

        for w in words:
            #yield(w, 1)                                               # count each item once
for item totals
            yield("BASKET "+basket, 1)                                # count each item once f
or basket size
        getdic={ word:{ subset[1]:1 for subset in itertools.combinations(sorted(set(words)),
2) if subset[0]==word} for word in words }
        for k, v in getdic.iteritems():
            if v:
                yield("STRIPE ", [k, v])

    def init_combine_stripes(self):
        self.dic = {}

    def simple_comb(self, key, values):
        self.increment_counter('group', 'Num_combiner_calls', 1)
        if key == "STRIPE ":
            for k, v in values:
                yield("STRIPE ", [k, v])
        else:
```

```
            yield("BASKET ", [key, sum(values)])

    def init_reduce_stripes(self):
        self.reddic = {}
        self.bs = {}
        self.pairs = []

    def simple_red(self, key, values):
        self.increment_counter('group', 'Num_reducer_calls', 1)
        if key == "STRIPE ":
            for k, v in values:
                if self.reddic.get(k):
                    x = self.reddic[k]
                    y = v
                    self.reddic[k] = { n: x.get(n, 0) + y.get(n, 0) for n in set(x) | set(y) }
                else:
                    self.reddic[k]=v
        else:
            for value in values:
                yield(value[0], value[1])

        for k, v in self.reddic.iteritems():
            for i in v:
                yield(k +' - '+ i, v[i])

    def sort_words(self, word, value):
        self.increment_counter('group', 'Num_mapper_step2_calls', 1)
        newkey = '{number:012d}'.format(number=int(value))+word
        yield(newkey, [word, value])

    def red_init(self):
        self.bs=[]
        self.pairs=[]
        self.freqpairs=[]

    def reduce_words(self, key, values):
        self.increment_counter('group', 'Num_reducer_step2_calls', 1)
        for value in values:
            if value[0][:6]=="BASKET":
                self.bs.append([value[0], value[1]])
```

```
            else:
                self.pairs.append([value[0], value[1]])

    def just_print(self):
        print self.pairs

    def reduce_words_final(self):
        #unique_items = len(self.d)
        num_baskets = len(self.bs)
        num_pairs = len(self.pairs)
        #max_d = unique_items-1
        max_bs = num_baskets-1
        max_pairs = num_pairs-1
        #print("Number of unique items is %s" % unique_items)
        print("Number of baskets is %s" % num_baskets)
        print("Largest basket has %s items" % self.bs[max_bs][1])
        print("Number of total pairs is %s" % num_pairs)
        for k in range(num_pairs):
            if self.pairs[k][1]>100:
                self.freqpairs.append([self.pairs[0], self.pairs[1]])
        num_freqpairs = len(self.freqpairs)
        print("Number of frequent pairs is %s" % num_freqpairs)
        print("***Top 50 pairs***")
        for i in range(50):
            print(self.pairs[max_pairs - i][0][:8], self.pairs[max_pairs - i][0][11:],
                  self.pairs[max_pairs - i][1], float(self.pairs[max_pairs - i][1])/num_basket
s)

    def steps(self):
        return [MRStep(mapper=self.get_words,
                       combiner_init=self.init_combine_stripes,
                       combiner=self.simple_comb,
                       reducer_init=self.init_reduce_stripes,
                       reducer=self.simple_red),
                MRStep(mapper=self.sort_words,
                       reducer_init=self.red_init,
                       reducer=self.reduce_words,
                       reducer_final=self.reduce_words_final)]

if __name__ == '__main__':
```

```
    stripes_madSCA.run()
```

```
Overwriting stripes_madSCA.py
```

In [6]:
```
%time !python stripes_madSCA.py --jobconf mapred.reduce.tasks=1 ProductPurchaseData.txt
```

```
No configs found; falling back on auto-configuration
Creating temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/stripes_madSCA.z013
hnv.20160709.024643.456789
Running step 1 of 2...
Counters: 2
        group
                Num_combiner_calls=30739
                Num_mapper_calls=31101
Counters: 3
        group
                Num_combiner_calls=30739
                Num_mapper_calls=31101
                Num_reducer_calls=2
Running step 2 of 2...
Counters: 1
        group
                Num_mapper_step2_calls=907832
Number of baskets is 30737
Largest basket has 288 items
Number of total pairs is 877095
Number of frequent pairs is 1311
***Top 50 pairs***
('dai62779', 'ele17451', 1592, 0.05179425448156944)
('fro40251', 'sna80324', 1412, 0.04593812018088948)
('dai75645', 'fro40251', 1254, 0.0407977356280704)
('fro40251', 'gro85051', 1213, 0.039463838370693304)
('dai62779', 'gro73461', 1139, 0.037056316491524875)
('dai75645', 'sna80324', 1130, 0.03676350977649087)
('dai62779', 'fro40251', 1070, 0.03481146500959755)
('dai62779', 'sna80324', 923, 0.030028955330708918)
('dai62779', 'dai85309', 918, 0.029866284933467806)
('ele32164', 'gro59710', 911, 0.029638546377330252)
('fro40251', 'gro73461', 882, 0.028695058073331815)
('dai62779', 'dai75645', 882, 0.028695058073331815)
('dai62779', 'ele92920', 877, 0.028532387676090704)
('fro40251', 'fro92469', 835, 0.02716595633926538)
('dai62779', 'ele32164', 832, 0.027068354100920716)
('dai75645', 'gro73461', 712, 0.023164264567134073)
('dai43223', 'ele32164', 711, 0.02313173048768585)
('dai62779', 'gro30386', 709, 0.023066662328789407)
```

```
('ele17451', 'fro40251', 697, 0.02267625337541074)
('dai85309', 'ele99737', 659, 0.021439958356378307)
('dai62779', 'ele26917', 650, 0.021147151641344307)
('gro21487', 'gro73461', 631, 0.02052900413182809)
('dai62779', 'sna45677', 604, 0.019650583986726096)
('ele17451', 'sna80324', 597, 0.01942284543058854)
('dai62779', 'gro71621', 595, 0.0193577772716921)
('dai62779', 'sna55762', 593, 0.019292709112795653)
('dai62779', 'dai83733', 586, 0.0190649705566581)
('ele17451', 'gro73461', 580, 0.018869766079968767)
('gro73461', 'sna80324', 562, 0.01828415264990077)
('dai62779', 'gro59710', 561, 0.01825161857045255)
('dai62779', 'fro80039', 550, 0.017893743696522108)
('dai75645', 'ele17451', 547, 0.017796141458177442)
('dai62779', 'sna93860', 537, 0.017470800663695222)
('dai55148', 'dai62779', 526, 0.01711292578976478)
('dai43223', 'gro59710', 512, 0.01665744867748967)
('ele17451', 'ele32164', 511, 0.016624914598041447)
('dai62779', 'sna18336', 506, 0.01646224420080034)
('ele32164', 'gro73461', 486, 0.0158115626118359)
('dai85309', 'ele17451', 482, 0.01568142629404301)
('dai62779', 'fro78087', 482, 0.01568142629404301)
('dai62779', 'gro94758', 479, 0.015583824055698345)
('gro85051', 'sna80324', 471, 0.015323551420112567)
('dai62779', 'gro21487', 471, 0.015323551420112567)
('ele17451', 'gro30386', 468, 0.015225949181767902)
('fro85978', 'sna95666', 463, 0.015063278784526792)
('dai62779', 'fro19221', 462, 0.01503074470507857)
('dai62779', 'gro46854', 461, 0.014998210625630348)
('dai43223', 'dai62779', 459, 0.014933142466733903)
('ele92920', 'sna18336', 455, 0.014803006148941016)
('dai88079', 'fro40251', 446, 0.014510199433907018)
Counters: 2
        group
                Num_mapper_step2_calls=907832
                Num_reducer_step2_calls=907794
Streaming final output from /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/stripes_madSCA.
z013hnv.20160709.024643.456789/output...
Removing temp directory /var/folders/rs/x38wjyw962d7804bsghvbxrn37md3k/T/stripes_madSCA.z013
hnv.20160709.024643.456789...
```

```
CPU times: user 846 ms, sys: 245 ms, total: 1.09 s
Wall time: 2min 21s
```

HW2.6: KMeans Clustering Tweet Dataset [OPTIONAL]

For this problem, please refer to and borrow from the following notebook:

http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/dcw8evd9v0su3xu/K_Means_Unit_Test_Notebook.ipynb (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/dcw8evd9v0su3xu/K_Means_Unit_Test_Notebook.ipynb)

Here you will use a different dataset consisting of word-frequency distributions for 1,000 Twitter users. These Twitter users use language in very different ways, and were classified by hand according to the criteria:

0: Human, where only basic human-human communication is observed.

1: Cyborg, where language is primarily borrowed from other sources (e.g., jobs listings, classifieds postings, advertisements, etc...).

2: Robot, where language is formulaically derived from unrelated sources (e.g., weather/seismology, police/fire event logs, etc...).

3: Spammer, where language is replicated to high multiplicity (e.g., celebrity obsessions, personal promotion, etc...)

Check out the preprints of recent research, which spawned this dataset:

http://arxiv.org/abs/1505.04342 (http://arxiv.org/abs/1505.04342) http://arxiv.org/abs/1508.01843 (http://arxiv.org/abs/1508.01843)

The main data lie in the accompanying file:

topUsers_Apr-Jul_2014_1000-words.txt located at: https://www.dropbox.com/s/6129k2urvbvobkr/topUsers_Apr-Jul_2014_1000-words.txt?dl=0 (https://www.dropbox.com/s/6129k2urvbvobkr/topUsers_Apr-Jul_2014_1000-words.txt?dl=0)

and are of the form:

USERID,CODE,TOTAL,WORD1_COUNT,WORD2_COUNT,... . .

where

USERID = unique user identifier CODE = 0/1/2/3 class code TOTAL = sum of the word counts

Using this data, you will implement a 1000-dimensional K-means algorithm in MrJob on the users by their 1000-dimensional word stripes/vectors using several centroid initializations and values of K.

Note that each "point" is a user as represented by 1000 words, and that word-frequency distributions are generally heavy-tailed power-laws (often called Zipf distributions), and are very rare in the larger class of discrete, random distributions. For each user you will have to normalize by its "TOTAL" column. Try several parameterizations and initializations:

(A) K=4 uniform random centroid-distributions over the 1000 words (generate 1000 random numbers and normalize the vectors) (B) K=2 perturbation-centroids, randomly perturbed from the aggregated (user-wide) distribution (C) K=4 perturbation-centroids, randomly perturbed from the aggregated (user-wide) distribution (D) K=4 "trained" centroids, determined by the sums across the classes. Use use the (row-normalized) class-level aggregates as 'trained' starting centroids (i.e., the training is already done for you!). Note that you do not have to compute the aggregated distribution or the class-aggregated distributions, which are rows in the auxiliary file:

topUsers_Apr-Jul_2014_1000-words_summaries.txt located at: https://www.dropbox.com/s/w4oklbsoqefou3b/topUsers_Apr-Jul_2014_1000-words_summaries.txt?dl=0 (https://www.dropbox.com/s/w4oklbsoqefou3b/topUsers_Apr-Jul_2014_1000-words_summaries.txt?dl=0)

Row 1: Words Row 2: Aggregated distribution across all classes Row 3-6 class-aggregated distributions for clases 0-3 For (A), we select 4 users randomly from a uniform distribution [1,...,1,000] For (B), (C), and (D) you will have to use data from the auxiliary file:

topUsers_Apr-Jul_2014_1000-words_summaries.txt

This file contains 5 special word-frequency distributions:

(1) The 1000-user-wide aggregate, which you will perturb for initializations in parts (B) and (C), and

(2-5) The 4 class-level aggregates for each of the user-type classes (0/1/2/3)

In parts (B) and (C), you will have to perturb the 1000-user aggregate (after initially normalizing by its sum, which is also provided). So if in (B) you want to create 2 perturbations of the aggregate, start with (1), normalize, and generate 1000 random numbers uniformly from the unit interval (0,1) twice (for two centroids), using:

from numpy import random numbers = random.sample(1000)

Take these 1000 numbers and add them (component-wise) to the 1000-user aggregate, and then renormalize to obtain one of your aggregate-perturbed initial centroids.

#

Geneate random initial centroids around the global aggregate

Part (B) and (C) of this question

#

def startCentroidsBC(k): counter = 0 for line in open("topUsers_Apr-Jul_2014_1000-words_summaries.txt").readlines(): if counter == 2: data = re.split(",",line) globalAggregate = [float(data[i+3])/float(data[2]) for i in range(1000)] counter += 1

```
    ## perturb the global aggregate for the four initializations
    centroids = []
    for i in range(k):
        rndpoints = random.sample(1000)
        peturpoints = [rndpoints[n]/10+globalAggregate[n] for n in range(1000)]
        centroids.append(peturpoints)
        total = 0
        for j in range(len(centroids[i])):
            total += centroids[i][j]
        for j in range(len(centroids[i])):
            centroids[i][j] = centroids[i][j]/total
    return centroids
```

—— For experiments A, B, C and D and iterate until a threshold (try 0.001) is reached. After convergence, print out a summary of the classes present in each cluster. In particular, report the composition as measured by the total portion of each class type (0-3) contained in each cluster, and discuss your findings and any differences in outcomes across parts A-D.

HW 2.7: (OPTIONAL) Logfile clean up

This exercise workes Microsoft logfiles data. The logfiles are described are located at:

https://kdd.ics.uci.edu/databases/msweb/msweb.html (https://kdd.ics.uci.edu/databases/msweb/msweb.html)
http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/ (http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/)

This dataset records which areas/pages (Vroots) of www.microsoft.com each user visited in a one-week timeframe in Feburary 1998.

Here, you must preprocess the data on a single node (i.e., not on a cluster of nodes) from the format:

C,"10001",10001 #Visitor id 10001 V,1000,1 #Visit by Visitor 10001 to page id 1000 V,1001,1 #Visit by Visitor 10001 to page id 1001 V,1002,1 #Visit by Visitor 10001 to page id 1002 C,"10002",10002 #Visitor id 10001 V Note: #denotes comments to the format:

V,1000,1,C, 10001 V,1001,1,C, 10001 V,1002,1,C, 10001

Write the python code to accomplish this.

2.7.1 Explain you can not do this clean up on one machine.

HW 2.8: Find the 5 most frequently visited pages

Find the 5 most frequently visited pages using MrJob from the output of HW2.7 (i.e., transfromed log file).

WARNING: per-step jobconf has bugs that affect Total sorts/partitions etc. For MRJob, Sort, partition code via the MRJob config does NOT work in local mode (known bug/feature which I believe has not been fixed as of June 2016). So you will need to run in the cloud (e.g. in AWS). It's issue #616 in github: "Inline and Local modes should support per-step jobconf #616". https://github.com/Yelp/mrjob/issues/616 (https://github.com/Yelp/mrjob/issues/616) To overcome this issue run your MRJob jobs on the cloud using -r hadoop or -r emr:

```
#!python MostFrequentVisits.py -r hadoop anonymous-msweb_converted.data
```

In [ ]: