DATASCI W261: Machine Learning at Scale

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

- **W261**
- **Week-4**
- **Assignment-4**
- **Date of Submission: 29-SEP-2015**

=== Week 4: mrjob & k-means ===

HW 4.0

What is MrJob? How is it different to Hadoop MapReduce?

- mrjob helps in writing MapReduce jobs in Python 2.6+ and run them on several platforms. We can:

  1. Write multi-step MapReduce jobs in pure Python
  2. Test on your local machine
  3. Run on a Hadoop cluster
  4. Run in the cloud using Amazon Elastic MapReduce (EMR)
- mrjob is the easiest route to writing Python programs that run on Hadoop. mrjob helps in testing the code locally without installing Hadoop or run it on a cluster.
- Additionally, mrjob has extensive integration with Amazon Elastic MapReduce. Once it is set up, it’s as easy to run your job in the cloud as it is to run it on the laptop.
- Here are a number of features of mrjob that make writing MapReduce jobs easier:

  1. Keep all MapReduce code for one job in a single class
  2. Easily upload and install code and data dependencies at runtime
  3. Switch input and output formats with a single line of code
  4. Automatically download and parse error logs for Python tracebacks
  5. Put command line filters before or after your Python code
- mrjob is a wrapper over hadoop map/reduce streaming API(s). mrjob provides a consistent interface across every environment it supports. No matter whether you’re running locally, in the cloud, or on your own cluster, your Python code doesn’t change at all. mrjob is a framework that assists in submitting your job to the Hadoop job tracker and in running each individual step under Hadoop Streaming.

What are the mapper_final(), combiner_final(), reducer_final() methods? When are they called?

1. mapper_final(): Used to define an action to run after the mapper reaches the end of input.
2. combiner_final(): Used to define an action to run after the combiner reaches the end of input.
3. reducer_final(): Used to define an action to run after the reducer reaches the end of input.

HW 4.1

What is serialization in the context of MrJob or Hadoop?

- Serialization is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.
- In the context of Hadoop, serialization is process of converting the data into a compact binary format so that vast amounts of data can be quickly transferred over network, and be consumed by programs written in various lamguages. Instead of using Java Serialization Hadoop has its own serialization framework. The main perceived problems with Java Serialization was that it writes the classname of each object being serialized to the stream, with each subsequent instance of that class containing a 5 byte reference to the first, instead of the classname. This reduced the effective bandwidth and caused problems with random access and sorting of records in a serialiazed stream.

When it used in these frameworks?

- Serialization is used while reading and writing data before and after each step (map, combine, shuffle, reduce etc) when data is read from and written to disk and/or wire for transfer over the network.

What is the default serialization mode for input and outputs for MrJob?

- The default input protocol is RawValueProtocol, which reads and writes lines of raw text with no key. So by default, the first step in your job sees (None, <text of the line>) for each line of input.
- The default output and internal protocols are both JSONProtocol, which reads and writes JSON strings separated by a tab character. (Hadoop Streaming uses the tab character to separate keys and values within one line when it sorts your data [1].)

HW 4.2

Recall the Microsoft logfiles data from the async lecture. 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 (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.

In [17]:
```
%%writefile preprocess_hw42.py
#!/usr/bin/env python
import sys
import os

if len(sys.argv) < 2:
    print "No input file is passed, Aborting!!!"
    sys.exit(1)

input_file = sys.argv[1]
output_file = input_file + '.pp'

try:
    os.remove(output_file)
except OSError:
    pass

last_visitor = None
with open(input_file, 'r') as f1:
    with open(output_file, 'a') as f2:
        for line in f1:
            line = line.strip()
            tokens = line.split(",")
            if len(tokens) == 3 and tokens[0] == 'C':
                last_visitor = tokens[2]

            if len(tokens) == 3 and tokens[0] == 'V':
                out_line = '{0},C,{1}\n'.format(line,last_visitor)
                f2.write(out_line)
```

```
Overwriting preprocess_hw42.py
```

In [18]:
```
!chmod a+x preprocess_hw42.py
```

In [24]:
```
!python preprocess_hw42.py anonymous-msweb.data
!echo "### Output (head)\n"
!head -n 10 anonymous-msweb.data.pp
!echo "\n### Output (tail)\n"
!tail -n 10 anonymous-msweb.data.pp
```

```
### Output (head)

V,1000,1,C,10001
V,1001,1,C,10001
V,1002,1,C,10001
V,1001,1,C,10002
V,1003,1,C,10002
V,1001,1,C,10003
V,1003,1,C,10003
V,1004,1,C,10003
V,1005,1,C,10004
V,1006,1,C,10005

### Output (tail)

V,1123,1,C,42708
V,1038,1,C,42708
V,1026,1,C,42708
V,1041,1,C,42708
V,1001,1,C,42709
V,1003,1,C,42709
V,1035,1,C,42710
V,1001,1,C,42710
V,1018,1,C,42710
V,1008,1,C,42711
```

HW 4.3

Find the 5 most frequently visited pages using mrjob from the output of 4.2 (i.e., transfromed log file).

In [72]:
```
%%writefile mrjob_hw43.py
from mrjob.job import MRJob
from mrjob.step import MRStep


class MRVistedPagesCount(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper,
                   combiner=self.combiner,
                   reducer=self.reducer),
            MRStep(reducer=self.reducer_find_top_5)
        ]

    def mapper(self, _, line):
        tokens = line.strip().split(",")
        yield tokens[1], 1

    def combiner(self, page_visted, counts):
        yield page_visted, sum(counts)

    def reducer(self, page_visted, counts):
        yield None, (sum(counts), page_visted)

    # discard the key; it is just None
    def reducer_find_top_5(self, _, page_visted_pairs):
        # Store all the keys into memory (Assumption: Can be loaded into memory)
        pairs = []
        for p in page_visted_pairs:
            pairs.append(p)
        pairs.sort(key=lambda x: x[0], reverse=True)

        for p in pairs[:5]:
            yield p[1],p[0]


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

```
Overwriting mrjob_hw43.py
```

In [73]:
```
!chmod a+x mrjob_hw43.py
```

In [74]:
```
# Running mrjob using command line
!python mrjob_hw43.py -r local anonymous-msweb.data.pp -q
```

```
"1008"  10836
"1034"  9383
"1004"  8463
"1018"  5330
"1017"  5108
```

In [41]:
```
# Running mrjob using a driver
from mrjob_hw43 import MRVistedPagesCount
mr_job = MRVistedPagesCount(args=['-r', 'local', 'anonymous-msweb.data.pp', 'q'])
with mr_job.make_runner() as runner:
    runner.run()
    # stream_output: get access of the output
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defa
ult. It's recommended you run your job with --strict-protocols or set up mrjob.conf as descr
ibed at https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocols
WARNING:mrjob.runner:
ERROR:mrjob.local:STDERR: + __mrjob_PWD=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/jo
b_local_dir/0/mapper/0
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python -c 'import fcntl; fcntl.floc
k(9, fcntl.LOCK_EX)'
ERROR:mrjob.local:STDERR: + export PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.88204
9/job_local_dir/0/mapper/0/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_l
ocal_dir/0/mapper/0/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + cd /tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_local_di
r/0/mapper/0
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python mrjob_hw43.py --step-num=0 --
mapper /tmp/mrjob_hw43.cloudera.20150926.200443.882049/input_part-00000
ERROR:mrjob.local:STDERR: + __mrjob_PWD=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/jo
b_local_dir/0/mapper/0
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python -c 'import fcntl; fcntl.floc
k(9, fcntl.LOCK_EX)'
ERROR:mrjob.local:STDERR: + export PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.88204
9/job_local_dir/0/mapper/0/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_l
ocal_dir/0/mapper/0/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + cd /tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_local_di
r/0/mapper/0
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python mrjob_hw43.py --step-num=0 --
combiner
ERROR:mrjob.local:STDERR: + __mrjob_PWD=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/jo
b_local_dir/0/mapper/1
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python -c 'import fcntl; fcntl.floc
k(9, fcntl.LOCK_EX)'
ERROR:mrjob.local:STDERR: + export PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.88204
9/job_local_dir/0/mapper/1/mrjob.tar.gz:
```

```
ERROR:mrjob.local:STDERR: + PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_l
ocal_dir/0/mapper/1/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + cd /tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_local_di
r/0/mapper/1
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python mrjob_hw43.py --step-num=0 --
mapper /tmp/mrjob_hw43.cloudera.20150926.200443.882049/input_part-00001
ERROR:mrjob.local:STDERR: + __mrjob_PWD=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/jo
b_local_dir/0/mapper/1
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python -c 'import fcntl; fcntl.floc
k(9, fcntl.LOCK_EX)'
ERROR:mrjob.local:STDERR: + export PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.88204
9/job_local_dir/0/mapper/1/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_l
ocal_dir/0/mapper/1/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + cd /tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_local_di
r/0/mapper/1
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python mrjob_hw43.py --step-num=0 --
combiner
ERROR:mrjob.local:STDERR: + __mrjob_PWD=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/jo
b_local_dir/0/reducer/0
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python -c 'import fcntl; fcntl.floc
k(9, fcntl.LOCK_EX)'
ERROR:mrjob.local:STDERR: + export PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.88204
9/job_local_dir/0/reducer/0/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_l
ocal_dir/0/reducer/0/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + cd /tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_local_di
r/0/reducer/0
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python mrjob_hw43.py --step-num=0 --
reducer /tmp/mrjob_hw43.cloudera.20150926.200443.882049/input_part-00000
ERROR:mrjob.local:STDERR: + __mrjob_PWD=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/jo
b_local_dir/0/reducer/1
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python -c 'import fcntl; fcntl.floc
k(9, fcntl.LOCK_EX)'
```

```
ERROR:mrjob.local:STDERR: + export PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.88204
9/job_local_dir/0/reducer/1/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_l
ocal_dir/0/reducer/1/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + cd /tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_local_di
r/0/reducer/1
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python mrjob_hw43.py --step-num=0 --
reducer /tmp/mrjob_hw43.cloudera.20150926.200443.882049/input_part-00001
ERROR:mrjob.local:STDERR: + __mrjob_PWD=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/jo
b_local_dir/1/reducer/0
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python -c 'import fcntl; fcntl.floc
k(9, fcntl.LOCK_EX)'
ERROR:mrjob.local:STDERR: + export PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.88204
9/job_local_dir/1/reducer/0/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + PYTHONPATH=/tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_l
ocal_dir/1/reducer/0/mrjob.tar.gz:
ERROR:mrjob.local:STDERR: + exec
ERROR:mrjob.local:STDERR: + cd /tmp/mrjob_hw43.cloudera.20150926.200443.882049/job_local_di
r/1/reducer/0
ERROR:mrjob.local:STDERR: + /home/cloudera/anaconda/bin/python mrjob_hw43.py --step-num=1 --
reducer /tmp/mrjob_hw43.cloudera.20150926.200443.882049/input_part-00000
```

```
('1008', 10836)
('1034', 9383)
('1004', 8463)
('1018', 5330)
('1017', 5108)
```

HW 4.4

Find the most frequent visitor of each page using mrjob and the output of 4.2 (i.e., transfromed log file). In this output please include the webpage URL, webpageID and Visitor ID.

Approach

1. Create a separate file with URLS, i.e with records that start with A. This will be passed to the mrjob as an additional file for joining datasets
2. Following happens in the first pass of mrjob:
   - Mapper: Emits (pageId,visitorId), 1
   - Combiner: Combines the counts
   - Reducer: Combines the counts and emits records like (p1,v1) 100 | (p1, v2), 101 | (p1,v3), 202
   - Partitioner is used to that all keys with the same pageId goes to the same reducer
3. Following happens in the second pass of mrjob:
   - Mapper: Emits (pageId,visit_count), visitorId
   - Reducer_Init: Loads the url file into a dict
   - Reducer: Emits (pageId, pageURL),(vists, vistorId)
   - Partitioner is used to that all keys with the same pageId goes to the same reducer
   - Secondary sorting (Descending Order) is done on the visit_count. This ensures that pages with highest page visits come first, before other records which has a lower page visit for the same page.

Final Output Format(output_hw44.txt): (pageId, pageURL),(vists, vistorId)

In [1]:
```
# Create a file with only URL(s), i.e. records starting with 'A'
!rm -v url
!grep ^A anonymous-msweb.data > url
```

```
removed `url'
```

In [21]:
```
%%writefile mrjob_hw44.py
from mrjob.job import MRJob
from mrjob.step import MRStep


class MRFrequentVisitor(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper,
                   combiner=self.combiner,
                   reducer=self.reducer),
             MRStep(mapper=self.mapper_frequent_visitor,
                   reducer_init=self.reducer_frequent_visitor_init,
                   reducer=self.reducer_frequent_visitor)
        ]

    def mapper(self, _, line):
        tokens = line.strip().split(",")
        key = "{0},{1}".format(tokens[1],tokens[4])
        yield key, 1

    def combiner(self, key, counts):
        yield key, sum(counts)

    def reducer(self, key, counts):
        yield key, sum(counts)

    # 2nd Pass

    def mapper_frequent_visitor(self, key, value):
        tokens = key.strip().split(",")
        modified_key = "{0},{1}".format(tokens[0],value)
        yield modified_key, tokens[1]


    def reducer_frequent_visitor_init(self):
        # Reads the 'url' file into a Dict for displaying additional information
        self.last_page = None
        self.pageDict = {}
        with open('url','r') as f:
```

```
            for line in f:
                tokens = line.strip().split(",")
                self.pageDict[tokens[1]] = tokens[4]

    def reducer_frequent_visitor(self, key, values):
        tokens = key.strip().split(",")
        page = tokens[0]
        visits = int(tokens[1])

        if self.last_page != page:
            self.last_page = page
            # values might be a list, if there is a tie for same key => (p1, 1000), [v1,v2,v
3..]
            for value in values:
                k = '{0},{1}'.format(page,
                                    self.pageDict.get(page, 'NA').replace("\"",""))
                v = '{0},{1}'.format(visits,
                                    value)
                yield k,v

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

```
Overwriting mrjob_hw44.py
```

In [22]:
```
!chmod a+x mrjob_hw44.py
```

In [23]:

```
# Running mrjob using a Hadoop Runner in local cluster
from mrjob_hw44 import MRFrequentVisitor
import os

# Passing Hadoop Streaming parameters to:
# partition by leftmost part of composite key
# secodary sort by rightmost part of the same composite key

mr_job = MRFrequentVisitor(args=['-r', 'hadoop',
                                 '--hadoop-home', '/usr/lib/hadoop-0.20-mapreduce',
                                 '--hadoop-bin', '/usr/bin/hadoop',
                                 '--file', 'url',
                                 '--jobconf', 'stream.num.map.output.key.fields=2',
                                 '--jobconf', 'map.output.key.field.separator=,',
                                 '--jobconf', 'mapred.text.key.partitioner.options=-k1,1',
                                 '--jobconf', 'mapred.text.key.comparator.options=-k1,1 -k2,2n
r',
                                 '--jobconf', 'mapred.output.key.comparator.class=org.apache.h
adoop.mapred.lib.KeyFieldBasedComparator',
                                 '--partitioner', 'org.apache.hadoop.mapred.lib.KeyFieldBasedP
artitioner',
                                 'anonymous-msweb.data.pp', '-v'])


output_file = "output_hw44.txt"
try:
    os.remove(output_file)
except OSError:
    pass

with mr_job.make_runner() as runner, open(output_file, 'a') as f:
    runner.run()
    # stream_output: get access of the output
    for line in runner.stream_output():
        #print mr_job.parse_output_line(line)
        f.write(line)
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defa
ult. It's recommended you run your job with --strict-protocols or set up mrjob.conf as descr
ibed at https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocols
WARNING:mrjob.runner:
WARNING:mrjob.compat:Detected hadoop configuration property names that do not match hadoop v
ersion 2.0.0:
The have been translated as follows
 mapred.text.key.partitioner.options: mapreduce.partition.keypartitioner.options
map.output.key.field.separator: mapreduce.map.output.key.field.separator
mapred.output.key.comparator.class: mapreduce.job.output.key.comparator.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.options
WARNING:mrjob.compat:Detected hadoop configuration property names that do not match hadoop v
ersion 2.0.0:
The have been translated as follows
 mapred.text.key.partitioner.options: mapreduce.partition.keypartitioner.options
map.output.key.field.separator: mapreduce.map.output.key.field.separator
mapred.output.key.comparator.class: mapreduce.job.output.key.comparator.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.options
```

In [27]:

```
# Output (Max page visit for all pages is 1; All ties have been reported)
!wc -l output_hw44.txt
!echo "\n### Output (head)\n"
!head -n 10 output_hw44.txt
!echo "\n### Output (tail)\n"
!tail -n 10 output_hw44.txt
```

```
   98654 output_hw44.txt

### Output (head)

"1000,/regwiz"  "1,42411"
"1000,/regwiz"  "1,42381"
"1000,/regwiz"  "1,42320"
"1000,/regwiz"  "1,42291"
"1000,/regwiz"  "1,42285"
"1000,/regwiz"  "1,42260"
"1000,/regwiz"  "1,42213"
"1000,/regwiz"  "1,42198"
"1000,/regwiz"  "1,42176"
"1000,/regwiz"  "1,42160"

### Output (tail)

"1295,/train_cert"      "1,10345"
"1295,/train_cert"      "1,10340"
"1295,/train_cert"      "1,10325"
"1295,/train_cert"      "1,10316"
"1295,/train_cert"      "1,10271"
"1295,/train_cert"      "1,10208"
"1295,/train_cert"      "1,10205"
"1295,/train_cert"      "1,10204"
"1295,/train_cert"      "1,10090"
"1295,/train_cert"      "1,10028"
```

HW 4.5

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 our 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

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 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 (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.

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.

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

Generate Initial Centroids for different use cases

- (A) K=4 uniform random centroid-distributions over the 1000 words
- (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.

The following mrjob implements kmeans for 1000 dimensions and is common for each of the above use cases

In [2]:
```
%%writefile mrjob_hw45.py
from numpy import argmin, array, random
from mrjob.job import MRJob
from mrjob.step import MRStep
from mrjob.compat import get_jobconf_value
from itertools import chain
import sys

#Calculate find the nearest centroid for data point
def MinDist(datapoint, centroid_points):
    datapoint = array(datapoint)
    centroid_points = array(centroid_points)
    diff = datapoint - centroid_points
    diffsq = diff*diff
    # Get the nearest centroid for each instance
    minidx = argmin(list(diffsq.sum(axis = 1)))
    return minidx

#Check whether centroids converge
def stop_criterion(centroid_points_old, centroid_points_new,T):
    oldvalue = list(chain(*centroid_points_old))
    newvalue = list(chain(*centroid_points_new))
    Diff = [abs(x-y) for x, y in zip(oldvalue, newvalue)]
    Flag = True
    for i in Diff:
        if(i>T):
            Flag = False
            break
    return Flag

class MRKmeans(MRJob):
    centroid_points=[]
    #k=0
    def steps(self):
        return [
            MRStep(mapper_init = self.mapper_init, mapper=self.mapper,combiner = self.combine
r,reducer=self.reducer)
               ]
    #load centroids info from file
    def mapper_init(self):
```

```
        self.centroid_points = [map(float,s.split('\n')[0].split(',')) for s in open("Centroid
s.txt").readlines()]
        open('Centroids.txt', 'w').close()
    #load data and output the nearest centroid index and data point
    def mapper(self, _, line):
        D = (map(float,line.split(',')))
        yield int(MinDist(D[3:],self.centroid_points)), (D[3:],1)
    #Combine sum of data points locally
    def combiner(self, idx, inputdata):
        num = 0
        sum_n = [0 for i in xrange(1000)]
        for d, n in inputdata:
            num = num + n
            sum_n = [x + y for x,y in zip(d,sum_n)]
        yield idx,(sum_n,num)
    #Aggregate sum for each cluster and then calculate the new centroids
    def reducer(self, idx, inputdata):
        centroids = []
        k = int(get_jobconf_value('k'))
        num = [0] * k
        for i in range(k):
            centroids.append([0 for i in xrange(1000)])
        for d, n in inputdata:
            num[idx] = num[idx] + n
            for i in xrange(1000):
                centroids[idx][i] = centroids[idx][i] + d[i]
        for i in xrange(1000):
            centroids[idx][i] = centroids[idx][i]/num[idx]

        with open('Centroids.txt', 'a') as f:
            f.writelines(",".join(str(i) for i in centroids[idx]) + '\n')
        yield idx,(centroids[idx], num)

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

```
Overwriting mrjob_hw45.py
```

In [3]:
```
!chmod a+x mrjob_hw45.py
```

Pre-Process File By Normalizing the values in 'topUsers_Apr-Jul_2014_1000-words.txt'

Input: topUsers_Apr-Jul_2014_1000-words.txt

Output: topUsers_Apr-Jul_2014_1000-words.txt.pp

In [5]:
```
# Pre-Process the input by normalizing the values
from sets import Set
import os

input_file = 'topUsers_Apr-Jul_2014_1000-words.txt'
output_file = input_file + '.pp'

try:
    os.remove(output_file)
except OSError:
    pass

values = Set()

# Word Count has been normalized by the Total
with open(input_file, 'r') as f1, open(output_file, 'w') as f2:
    for line in f1:
        count = 0
        elements = line.strip().split(",")
        total = int(elements[2])
        for i in xrange(3, len(elements)):
            value = round(float(elements[i]) * 1.0 / total, 6)
            elements[i] = str(value)
            values.add(value)
        f2.write(",".join(elements) + "\n")

print 'Min: {0}'.format(min(values))
print 'Max: {0}'.format(max(values))
```

```
Min: 0.0
Max: 0.40991
```

Generic Function with runner for running mrjob

In [6]:
```
%load_ext autoreload
%autoreload 2
import numpy as np
from numpy import random
from mrjob_hw45 import MRKmeans, stop_criterion

def kmeans(K, centroid_points):
    mr_job = MRKmeans(args=['--file', output_file,
                            '--jobconf', 'k={0}'.format(K),
                            '--no-strict-protocol',
                            'topUsers_Apr-Jul_2014_1000-words.txt.pp', '-v'])

    # Update centroids iteratively
    i = 0
    while(1):
        # save previous centoids to check convergency
        centroid_points_old = centroid_points[:]
        print "iteration " + str(i) + ":"
        with mr_job.make_runner() as runner:
            runner.run()
            # stream_output: get access of the output
            for line in runner.stream_output():
                key,value =  mr_job.parse_output_line(line)
                centroid_points[key] = value[0]
                print '[K={0}] Cluster IDX: {1}, \
                        Number of Elements: {2}, \
                        % of Elements: {3}'.format(K, key,
                                                     value[1][key],
                                                    value[1][key] * 100.0 / 1000)
                #print key, len(value[0]), value[1]

        print "\n"
        i = i + 1
        if(stop_criterion(centroid_points_old,centroid_points,0.001)):
            break
    print "--- Done ---"
    #print centroid_points
```

Case: (A) K=4 uniform random centroid-distributions over the 1000 words

In [7]:
```
import os

K = 4
output_file = 'Centroids.txt'

try:
    os.remove(output_file)
except OSError:
    pass

centroid_points = []
for k in xrange(K):
    centroid_points.append(np.random.uniform(0.0, 0.40991, 1000)/1000)

with open(output_file, 'w+') as f:
        f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_points)

kmeans(K, centroid_points)
```

```
WARNING:mrjob.compat:get_jobconf_value() has been renamed to jobconf_from_env(). get_jobcon
f_value() will be removed in v0.5.0
```

```
iteration 0:
[K=4] Cluster IDX: 0,                              Number of Elements: 649,
% of Elements: 64.9
[K=4] Cluster IDX: 1,                              Number of Elements: 153,
% of Elements: 15.3
[K=4] Cluster IDX: 2,                              Number of Elements: 34,
% of Elements: 3.4
[K=4] Cluster IDX: 3,                              Number of Elements: 164,
% of Elements: 16.4


iteration 1:
[K=4] Cluster IDX: 0,                              Number of Elements: 577,
% of Elements: 57.7
[K=4] Cluster IDX: 1,                              Number of Elements: 252,
% of Elements: 25.2
[K=4] Cluster IDX: 2,                              Number of Elements: 26,
% of Elements: 2.6
[K=4] Cluster IDX: 3,                              Number of Elements: 145,
% of Elements: 14.5


iteration 2:
[K=4] Cluster IDX: 0,                              Number of Elements: 617,
% of Elements: 61.7
[K=4] Cluster IDX: 1,                              Number of Elements: 233,
% of Elements: 23.3
[K=4] Cluster IDX: 2,                              Number of Elements: 20,
% of Elements: 2.0
[K=4] Cluster IDX: 3,                              Number of Elements: 130,
% of Elements: 13.0


iteration 3:
[K=4] Cluster IDX: 0,                              Number of Elements: 634,
% of Elements: 63.4
[K=4] Cluster IDX: 1,                              Number of Elements: 219,
% of Elements: 21.9
[K=4] Cluster IDX: 2,                              Number of Elements: 16,
% of Elements: 1.6
```

```
[K=4] Cluster IDX: 3,                    Number of Elements: 131,
% of Elements: 13.1


iteration 4:
[K=4] Cluster IDX: 0,                    Number of Elements: 677,
% of Elements: 67.7
[K=4] Cluster IDX: 1,                    Number of Elements: 178,
% of Elements: 17.8
[K=4] Cluster IDX: 2,                    Number of Elements: 14,
% of Elements: 1.4
[K=4] Cluster IDX: 3,                    Number of Elements: 131,
% of Elements: 13.1


iteration 5:
[K=4] Cluster IDX: 0,                    Number of Elements: 736,
% of Elements: 73.6
[K=4] Cluster IDX: 1,                    Number of Elements: 121,
% of Elements: 12.1
[K=4] Cluster IDX: 2,                    Number of Elements: 12,
% of Elements: 1.2
[K=4] Cluster IDX: 3,                    Number of Elements: 131,
% of Elements: 13.1


iteration 6:
[K=4] Cluster IDX: 0,                    Number of Elements: 767,
% of Elements: 76.7
[K=4] Cluster IDX: 1,                    Number of Elements: 90,
% of Elements: 9.0
[K=4] Cluster IDX: 2,                    Number of Elements: 12,
% of Elements: 1.2
[K=4] Cluster IDX: 3,                    Number of Elements: 131,
% of Elements: 13.1


iteration 7:
[K=4] Cluster IDX: 0,                    Number of Elements: 781,
% of Elements: 78.1
```

```
[K=4] Cluster IDX: 1,                          Number of Elements: 76,
% of Elements: 7.6
[K=4] Cluster IDX: 2,                          Number of Elements: 12,
% of Elements: 1.2
[K=4] Cluster IDX: 3,                          Number of Elements: 131,
% of Elements: 13.1


iteration 8:
[K=4] Cluster IDX: 0,                          Number of Elements: 788,
% of Elements: 78.8
[K=4] Cluster IDX: 1,                          Number of Elements: 69,
% of Elements: 6.9
[K=4] Cluster IDX: 2,                          Number of Elements: 12,
% of Elements: 1.2
[K=4] Cluster IDX: 3,                          Number of Elements: 131,
% of Elements: 13.1


iteration 9:
[K=4] Cluster IDX: 0,                          Number of Elements: 793,
% of Elements: 79.3
[K=4] Cluster IDX: 1,                          Number of Elements: 64,
% of Elements: 6.4
[K=4] Cluster IDX: 2,                          Number of Elements: 12,
% of Elements: 1.2
[K=4] Cluster IDX: 3,                          Number of Elements: 131,
% of Elements: 13.1


iteration 10:
[K=4] Cluster IDX: 0,                          Number of Elements: 794,
% of Elements: 79.4
[K=4] Cluster IDX: 1,                          Number of Elements: 63,
% of Elements: 6.3
[K=4] Cluster IDX: 2,                          Number of Elements: 12,
% of Elements: 1.2
[K=4] Cluster IDX: 3,                          Number of Elements: 131,
% of Elements: 13.1
```

```
--- Done ---
```

(B) K=2 perturbation-centroids, randomly perturbed from the aggregated (user-wide) distribution

In [13]:
```
K = 2
output_file = 'Centroids.txt'

try:
    os.remove(output_file)
except OSError:
    pass

centroid_points = []
with open('topUsers_Apr-Jul_2014_1000-words_summaries.txt', 'r') as f:
    for line in f:
        if line.startswith('ALL_CODES'):
            elements = line.strip().split(",")
            total = int(elements[2])
            array = [int(e) * 1.0 / total for e in elements[3:]]
            for k in xrange(K):
                centroid_points.append(array)
            break;

print [sum(i) for i in centroid_points]
std = np.std(centroid_points[0])
print "Standard Deviation", std
mean = np.mean(centroid_points[0])
print "Mean", mean
# Add Random Noise
centroid_points = centroid_points + np.random.sample(K * 1000).reshape(K, 1000)
for k in xrange(K):
    # Normalize Again
    centroid_points[k] = centroid_points[k] * 1.0 / np.sum(centroid_points[k])
print [sum(i) for i in centroid_points]

with open(output_file, 'w+') as f:
        f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_points)

kmeans(K, centroid_points)
```

```
[0.999999999999999, 0.999999999999999]
Standard Deviation 0.0024990758202
Mean 0.001
[1.0000000000000002, 1.0000000000000007]
iteration 0:
[K=2] Cluster IDX: 0,                         Number of Elements: 447,
% of Elements: 44.7
[K=2] Cluster IDX: 1,                         Number of Elements: 553,
% of Elements: 55.3


--- Done ---
```

(C) K=4 perturbation-centroids, randomly perturbed from the aggregated (user-wide) distribution

In [12]:
```
K = 4
output_file = 'Centroids.txt'

try:
    os.remove(output_file)
except OSError:
    pass

centroid_points = []
with open('topUsers_Apr-Jul_2014_1000-words_summaries.txt', 'r') as f:
    for line in f:
        if line.startswith('ALL_CODES'):
            elements = line.strip().split(",")
            total = int(elements[2])
            array = [int(e) * 1.0 / total for e in elements[3:]]
            for k in xrange(K):
                centroid_points.append(array)
            break;

print [sum(i) for i in centroid_points]
std = np.std(centroid_points[0])
print "Standard Deviation", std
mean = np.mean(centroid_points[0])
print "Mean", mean
# Add Random Noise
centroid_points = centroid_points + np.random.sample(K * 1000).reshape(K, 1000)
for k in xrange(K):
    # Normalize Again
    centroid_points[k] = centroid_points[k] * 1.0 / np.sum(centroid_points[k])
print [sum(i) for i in centroid_points]

with open(output_file, 'w+') as f:
        f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_points)

kmeans(K, centroid_points)
```

```
[0.999999999999999, 0.999999999999999, 0.999999999999999, 0.999999999999999]
Standard Deviation 0.0024990758202
Mean 0.001
[0.99999999999999989, 1.0000000000000018, 1.0000000000000004, 0.99999999999999889]
iteration 0:
[K=4] Cluster IDX: 0,                         Number of Elements: 544,
% of Elements: 54.4
[K=4] Cluster IDX: 1,                         Number of Elements: 354,
% of Elements: 35.4
[K=4] Cluster IDX: 2,                         Number of Elements: 99,
% of Elements: 9.9
[K=4] Cluster IDX: 3,                         Number of Elements: 3,
% of Elements: 0.3


--- Done ---
```

(D) K=4 "trained" centroids, determined by the sums across the classes.

In [245]:
```
K = 4
output_file = 'Centroids.txt'

try:
    os.remove(output_file)
except OSError:
    pass

centroid_points = []
with open('topUsers_Apr-Jul_2014_1000-words_summaries.txt', 'r') as f:
    for line in f:
        if line.startswith('CODE'):
            elements = line.strip().split(",")
            total = int(elements[2])
            array = [int(e) * 1.0 / total for e in elements[3:]]
            centroid_points.append(array)

print [sum(i) for i in centroid_points]

with open(output_file, 'w+') as f:
        f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_points)

kmeans(K, centroid_points)
```

```
[0.9999999999999996, 1.0000000000000002, 1.0, 1.0]
iteration 0:
[K=4] Cluster IDX: 0,                    Number of Elements: 787,
% of Elements: 78.7
[K=4] Cluster IDX: 1,                    Number of Elements: 61,
% of Elements: 6.1
[K=4] Cluster IDX: 2,                    Number of Elements: 82,
% of Elements: 8.2
[K=4] Cluster IDX: 3,                    Number of Elements: 70,
% of Elements: 7.0


iteration 1:
[K=4] Cluster IDX: 0,                    Number of Elements: 796,
% of Elements: 79.6
[K=4] Cluster IDX: 1,                    Number of Elements: 54,
% of Elements: 5.4
[K=4] Cluster IDX: 2,                    Number of Elements: 85,
% of Elements: 8.5
[K=4] Cluster IDX: 3,                    Number of Elements: 65,
% of Elements: 6.5


iteration 2:
[K=4] Cluster IDX: 0,                    Number of Elements: 802,
% of Elements: 80.2
[K=4] Cluster IDX: 1,                    Number of Elements: 51,
% of Elements: 5.1
[K=4] Cluster IDX: 2,                    Number of Elements: 84,
% of Elements: 8.4
[K=4] Cluster IDX: 3,                    Number of Elements: 63,
% of Elements: 6.3


iteration 3:
[K=4] Cluster IDX: 0,                    Number of Elements: 804,
% of Elements: 80.4
[K=4] Cluster IDX: 1,                    Number of Elements: 51,
% of Elements: 5.1
[K=4] Cluster IDX: 2,                    Number of Elements: 82,
```

```
% of Elements: 8.2
[K=4] Cluster IDX: 3,                              Number of Elements: 63,
% of Elements: 6.3


iteration 4:
[K=4] Cluster IDX: 0,                              Number of Elements: 804,
% of Elements: 80.4
[K=4] Cluster IDX: 1,                              Number of Elements: 51,
% of Elements: 5.1
[K=4] Cluster IDX: 2,                              Number of Elements: 82,
% of Elements: 8.2
[K=4] Cluster IDX: 3,                              Number of Elements: 63,
% of Elements: 6.3


--- Done ---
```

The distributions are very similar in Case A & D with bulk of the users falling into 1 cluster (80%)

In Case B, the users are much more evenly distributed

In Case C, the users mostly fall into 2 clusters

In [ ]: