W261 Machine Learning at Scale

Names Safyre Anderson, Howard Wen , Vamsi Sakhamuri

Emails safyre@berkeley.edu, howard.wen1@gmail.com, vamsi@ischool.berkeley.edu

Time of Initial Submission: February 18th, 2016 8am PST

Section W261-3, Spring 2016

Week 5 Homework

HW 5.0

What is a data warehouse? What is a Star schema? When is it used?

1. *Data Warehouse:*

https://en.wikipedia.org/wiki/Data_warehouse (https://en.wikipedia.org/wiki/Data_warehouse). A data warehouse is an enterprise-level repository for historical and current datasets. Typically, a business has many sources of data that need to be processed for a variety of applications: e.g. reporting, business analyics, and/ or predictive analytics. For example, in mobile advertising data pipelines, an ad request from an app on a customer's mobile device will require a decision that involves streaming usage data in addition to historical data. The historical data would have been used to determine which types of ads are relevant to the end user, while the streaming data used to generate a prediction. Data collected from these different sources and various purposes need to be passed through some form of transformation and storage in order to be processed in subsequent parts of the pipeline. The gathering of data, transforming them and storing them is known as ETL or extract, transform, and load. The loading piece of ETL refers to storing processed data into the data warehouse--typically, a relational database, but can also include semi-stuctured and unstructured data storage.

Two traditional examples of data warehousing strategies are OLAP (online analytical processing) and OLTP (online transactional processing).

1. *Star Schema:*

The star schema is a framework for organizing data in a relational database where a central fact table acts as a means to connect data between different dimensional tables. The fact table will contain foreign keys for each of the branching tables. Additionally, each of the branching tables focusses on data related to one dimension that an analysis could splice on. For example an ecommerce schema might have a transactions-based fact table that contains foreign keys for a customers table, and a payment table. The transactions table will be able to join with both the customers and payment tables directly, but the payment table and customer table would need to join with each other indirectly through the transactions table since they would not (in this example) contain each others foreign keys.

1. *When is it used?:*

The star schema is useful for highly structured data that may have numerous dimensions that could be used to break down the data during analyses. The main reason this is useful in most reporting applications is because it makes querying data much more efficient. Instead of having one large table with all the data and many rows, joining a smaller table with a larger one can shrink the size of the data significantly before it is extracted.

HW 5.1

1. *In the database world What is 3NF? Does machine learning use data in 3NF? If so why? In what form does ML consume data?*

3NF stands for "Third normal form". This refers to how data are organized with their respective tables and keys: Attributes in a table should provide a fact about the table's key and nothing else ("Nothing but the key" https://en.wikipedia.org/wiki/Third_normal_form (https://en.wikipedia.org/wiki/Third_normal_form)). This structure allows a database to avoid duplicating data as much as possible and therefore save storage space. Machine learning would not be able to utilize data in this form very well--features would be scattered across different sources so the classifier would not be able to train on features in different tables. ML classifiers need to have all the data they depend on in one location (distributed or not) and therefore need the data to be denormalized.

1. *Why would one use log files that are denormalized?*

Log files are normally stored as semi-structured or unstructured data. They may have some regular format, but for the most part, log files contain plain English that would need to be parsed during analysis. If log files were placed into a structure and normalized, closely related pieces of information that could be useful would be separated from each other--for instance, an error message with a time stamp. Furthermore, a denormalized log file is more usable because log files are all text. Normalized data typically contains data that can be strongly typed, but if a log file is more intuitively parsed as a document then normalization may just overcomplicate things.

HW 5.2

Using MRJob, implement a hashside join (memory-backed map-side) for left, right and inner joins. Run your code on the data used in HW 4.4: (Recall 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.) :

Justify which table you chose as the Left table in this hashside join.

Please report the number of rows resulting from:

(1) Left joining Table Left with Table Right (2) Right joining Table Left with Table Right (3) Inner joining Table Left with Table Right

In [2]:
```
## first get 4.2 data and preprocess it:
!wget -O anonymous-msweb.data http://archive.ics.uci.edu/ml/machine-le
arning-databases/anonymous/anonymous-msweb.data

## python code to reformat the data:
# check format of whole file
#!cat anonymous-msweb.data
write_file_loc = "msweb_processed.txt"
writefile = open(write_file_loc, 'wb')

with open("anonymous-msweb.data", 'rU') as f:
    for line in f.readlines():
        line = line.strip().split(',')

        # skip all the header rows
        if line[0] not in ['C', 'V']:
            continue

        if line[0] == 'C':
            user_id = line[2]

        elif line[0] == 'V':
            writefile.write('V,{},1,C,{}\n'.format(line[1], user_id))

        else:
            continue
```

```
--2016-02-15 14:26:38--  http://archive.ics.uci.edu/ml/machine-learn
ing-databases/anonymous/anonymous-msweb.data
Resolving archive.ics.uci.edu... 128.195.10.249
Connecting to archive.ics.uci.edu|128.195.10.249|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1423098 (1.4M) [text/plain]
Saving to: 'anonymous-msweb.data'

anonymous-msweb.dat 100%[=====================>]   1.36M  1.91MB/s
in 0.7s

2016-02-15 14:26:39 (1.91 MB/s) - 'anonymous-msweb.data' saved [1423
098/1423098]
```

In [4]:
```
# preprocessed file is:
!head -n10 msweb_processed.txt
```

```
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
```

In [6]:
```
!head -n15 anonymous-msweb.data
```

```
I,4,"www.microsoft.com","created by getlog.pl"
T,1,"VRoot",0,0,"VRoot"
N,0,"0"
N,1,"1"
T,2,"Hide1",0,0,"Hide"
N,0,"0"
N,1,"1"
A,1287,1,"International AutoRoute","/autoroute"
A,1288,1,"library","/library"
A,1289,1,"Master Chef Product Information","/masterchef"
A,1297,1,"Central America","/centroam"
A,1215,1,"For Developers Only Info","/developer"
A,1279,1,"Multimedia Golf","/msgolf"
A,1239,1,"Microsoft Consulting","/msconsult"
A,1282,1,"home","/home"
```

In [12]:
```
## now process a new set of data with just URL and webpageID for mergi
ng
## first get 4.2 data and preprocess it:
!wget -O anonymous-msweb.data http://archive.ics.uci.edu/ml/machine-le
arning-databases/anonymous/anonymous-msweb.data

## python code to reformat the data:
# check format of whole file
#!cat anonymous-msweb.data
write_file_loc = "msweb_dim_url.txt"
writefile = open(write_file_loc, 'wb')

with open("anonymous-msweb.data", 'rU') as f:
    for line in f.readlines():
        line = line.strip().split(',')

        # skip all the header rows
        if line[0] == 'A':
            # attach the relative url to the home url. Remove ""'s
            full_url = "http://www.microsoft.com"+line[4].strip('"')
            writefile.write('{url_id}\t{url}\n'.format(url_id=line[1],
url=full_url))

        #elif line[0] == 'V':
        #    writefile.write('V,{},1,C,{}\n'.format(line[1], user_id))

        #else:
        #    continue
```

```
--2016-02-15 15:20:24--  http://archive.ics.uci.edu/ml/machine-learn
ing-databases/anonymous/anonymous-msweb.data
Resolving archive.ics.uci.edu... 128.195.10.249
Connecting to archive.ics.uci.edu|128.195.10.249|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1423098 (1.4M) [text/plain]
Saving to: 'anonymous-msweb.data'

anonymous-msweb.dat 100%[=====================>]   1.36M  1.88MB/s
in 0.7s

2016-02-15 15:20:25 (1.88 MB/s) - 'anonymous-msweb.data' saved [1423
098/1423098]
```

In [60]:
```
!head -n10 msweb_dim_url.txt
```

```
1287    http://www.microsoft.com/autoroute
1288    http://www.microsoft.com/library
1289    http://www.microsoft.com/masterchef
1297    http://www.microsoft.com/centroam
1215    http://www.microsoft.com/developer
1279    http://www.microsoft.com/msgolf
1239    http://www.microsoft.com/msconsult
1282    http://www.microsoft.com/home
1251    http://www.microsoft.com/referencesupport
1121    http://www.microsoft.com/magazine
```

In [83]:
```
!wc -l msweb_dim_url.txt
!wc -l msweb_processed.txt
```

```
     304 msweb_dim_url.txt
   98654 msweb_processed.txt
```

In [107]:
```
%%writefile MR_HW52_joins.py
#!/usr/bin/env python
# find the most frequent visitors per url
from mrjob.job import MRJob
from mrjob.step import MRStep
import csv
from heapq import heappush, heappop,nlargest

class MapSideJoins(MRJob):
    def steps(self):
        return [MRStep(
            mapper_init = self.mapper_init,
            mapper = self.mapper,
            mapper_final = self.mapper_final
            )]

    def mapper_init(self):
        # set variables
        # inspired by example on lecture slide 121
        self.URLS = {}

        # for right outer join
        self.URLS_ON_RIGHT = {}

        # initialize row counts
        self.left = 0
        self.right = 0
        self.inner = 0

        # read in the page_url table
        # formatted like : 1287 http://www.microsoft.com/autoroute
        f = open("msweb_dim_url.txt", 'rU')
        for line in f:
            line = line.strip()
            data = line.split()
            #store page-id as key in URLS dictionary
            self.URLS[data[0]] = data[1]

    def mapper(self, _, line):
        """stream in output from 4.2 (msweb_processed.txt):

        V,1000,1,C,10001

        and match with keys in self.URLS"""
        self.increment_counter('group', 'Number_mapper_calls',1)
        line = line.strip().split(',')
        key = line[1]

        if key in self.URLS.keys():
            ## outputs are : JOIN_TYPE, URL_ID, URL, CUSTOMER_ID
            # map inner join
            self.inner += 1
            yield None, ("inner", key, self.URLS[key], line[4])

            # map visits left join urls on url_id
            # this code assumes each customer has a visit
            # if there are visits to no URLS, we will catch it in the
second mapper
            # although this probably isn't the case...
            self.left += 1
            yield None, ("left", key, self.URLS[key], line[4])

            # map visits right join urls on url_id
            # keep track of url_ids that have not been visited
            self.right += 1
            yield None, ("right", key, self.URLS[key], line[4])
            self.URLS_ON_RIGHT[key] = self.URLS[key]

        #right join will also include sites that have not been visited
        # no customer_id
        else:
            self.right += 1
            yield None, ("right", key, self.URLS[key], "NA")

    def mapper_final(self):
        for key in self.URLS.keys():
            # in a left join, if a key exists on the left table, but n
ot the right table
            # then we need a row with data from the left table and nul
l values from the missing
```

```
            # data on the right
            if key not in self.URLS_ON_RIGHT.keys():
                self.left += 1
                yield None, ("left", key, "NA",  self.URLS[key])

        ## final outputs:
        yield None, ()
        yield None, ("left-joined rows: " + str(self.left))
        yield None, ("right-joined rows: " + str(self.right))
        yield None, ("inner-joined rows: " + str(self.inner))

if __name__ == "__main__":
    MapSideJoins.run()
```

```
Overwriting MR_HW52_joins.py
```

In [108]:

```
!rm hw52_out.txt
!./MR_HW52_joins.py msweb_processed.txt \
    --file msweb_dim_url.txt > hw52_out.txt
```

```
no configs found; falling back on auto-configuration
no configs found; falling back on auto-configuration
creating tmp directory /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000g
n/T/MR_HW52_joins.Safyre.20160218.162259.447239

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by d
efault. It's recommended you run your job with --strict-protocols or
set up mrjob.conf as described at https://pythonhosted.org/mrjob/wha
ts-new.html#ready-for-strict-protocols

writing to /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000gn/T/MR_HW52_
joins.Safyre.20160218.162259.447239/step-0-mapper_part-00000
Counters from step 1:
  group:
    Number_mapper_calls: 98654
Moving /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000gn/T/MR_HW52_join
s.Safyre.20160218.162259.447239/step-0-mapper_part-00000 -> /var/fol
ders/jz/dhc2gzfj2091nhfmrbvv3cjh0000gn/T/MR_HW52_joins.Safyre.201602
18.162259.447239/output/part-00000
Streaming final output from /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh
0000gn/T/MR_HW52_joins.Safyre.20160218.162259.447239/output
removing tmp directory /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000g
n/T/MR_HW52_joins.Safyre.20160218.162259.447239
```

In [93]:
```
!tail -n10 hw52_out.txt
```

```
null    ["left", "1291", "NA", "http://www.microsoft.com/news"]
null    ["left", "1297", "NA", "http://www.microsoft.com/centroam"]
null    ["left", "1294", "NA", "http://www.microsoft.com/bookshelf"]
null    ["left", "1287", "NA", "http://www.microsoft.com/autoroute"]
null    ["left", "1289", "NA", "http://www.microsoft.com/masterchef"
]
null    ["left", "1288", "NA", "http://www.microsoft.com/library"]
null    []
null    "left-joined rows: 98663"
null    "right-joined rows: 98654"
null    "inner-joined rows: 98654"
```

In [170]:
```
!chmod +x MR_HW52_joins.py
!python MR_HW52_joins.py -r emr s3://w261-safyre-hw52-input/msweb_proc
essed.txt \
    --conf-path /Users/Safyre/cc-mrjob/mrjob.conf\
    --file s3://w261-safyre-hw52-input/msweb_dim_url.txt \
    --output-dir= s3://w261-safyre-hw52-output/output/out/ \
    --no-output \
    --cleanup=NONE\
    --no-strict-protocol
```

```
Got unexpected keyword arguments: ssh_tunnel
using existing scratch bucket mrjob-62fd57571f35a64e
using s3://mrjob-62fd57571f35a64e/tmp/ as our scratch dir on S3
creating tmp directory /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000g
n/T/MR_HW52_joins.Safyre.20160219.153400.354127
writing master bootstrap script to /var/folders/jz/dhc2gzfj2091nhfmr
bvv3cjh0000gn/T/MR_HW52_joins.Safyre.20160219.153400.354127/b.py
Copying non-input files into s3://mrjob-62fd57571f35a64e/tmp/MR_HW52
_joins.Safyre.20160219.153400.354127/files/
Waiting 5.0s for S3 eventual consistency
Creating Elastic MapReduce job flow
Job flow created with ID: j-PBLI033NYNV6
Created new job flow j-PBLI033NYNV6
Job launched 30.5s ago, status STARTING: Provisioning Amazon EC2 cap
acity
Job launched 61.4s ago, status STARTING: Provisioning Amazon EC2 cap
acity
Job launched 91.9s ago, status STARTING: Provisioning Amazon EC2 cap
acity
Job launched 122.5s ago, status STARTING: Provisioning Amazon EC2 ca
pacity
Job launched 153.0s ago, status STARTING: Provisioning Amazon EC2 ca
pacity
Job launched 184.0s ago, status STARTING: Provisioning Amazon EC2 ca
pacity
```

```
Job launched 214.5s ago, status STARTING: Provisioning Amazon EC2 ca
pacity
Job launched 245.1s ago, status STARTING: Provisioning Amazon EC2 ca
pacity
Job launched 275.6s ago, status STARTING: Provisioning Amazon EC2 ca
pacity
Job launched 306.5s ago, status STARTING: Provisioning Amazon EC2 ca
pacity
Job launched 337.0s ago, status STARTING: Provisioning Amazon EC2 ca
pacity
Job launched 368.4s ago, status STARTING: Provisioning Amazon EC2 ca
pacity
Job launched 399.7s ago, status STARTING: Configuring cluster softwa
re
Job launched 430.6s ago, status STARTING: Configuring cluster softwa
re
Job launched 461.1s ago, status STARTING: Configuring cluster softwa
re
Job launched 492.0s ago, status STARTING: Configuring cluster softwa
re
Job launched 522.6s ago, status BOOTSTRAPPING: Running bootstrap act
ions
Job launched 553.2s ago, status BOOTSTRAPPING: Running bootstrap act
ions
Job launched 583.7s ago, status RUNNING: Running step
Job launched 614.3s ago, status RUNNING: Running step
Job launched 644.8s ago, status RUNNING: Running step (MR_HW52_joins
.Safyre.20160219.153400.354127: Step 1 of 1)
Opening ssh tunnel to Hadoop job tracker
Connect to job tracker at: http://localhost:40328/jobtracker.jsp
Job launched 677.1s ago, status RUNNING: Running step (MR_HW52_joins
.Safyre.20160219.153400.354127: Step 1 of 1)
Unable to load progress from job tracker
Job launched 707.7s ago, status RUNNING: Running step (MR_HW52_joins
.Safyre.20160219.153400.354127: Step 1 of 1)
Job completed.
Running time was 100.0s (not counting time spent waiting for the EC2
instances)
Fetching counters from S3...
Waiting 5.0s for S3 eventual consistency
Counters from step 1:
  (no counters found)
Killing our SSH tunnel (pid 4419)
Terminating job flow: j-PBLI033NYNV6
```

In [137]:
```
#persistent job flow
!python -m mrjob.tools.emr.create_job_flow --conf-path /Users/Safyre/c
c-mrjob/mrjob.conf
```

```
using existing scratch bucket mrjob-62fd57571f35a64e
using s3://mrjob-62fd57571f35a64e/tmp/ as our scratch dir on S3
Creating persistent job flow to run several jobs in...
creating tmp directory /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000g
n/T/no_script.Safyre.20160219.061229.360977
writing master bootstrap script to /var/folders/jz/dhc2gzfj2091nhfmr
bvv3cjh0000gn/T/no_script.Safyre.20160219.061229.360977/b.py
Copying non-input files into s3://mrjob-62fd57571f35a64e/tmp/no_scri
pt.Safyre.20160219.061229.360977/files/
Waiting 5.0s for S3 eventual consistency
Creating Elastic MapReduce job flow
Job flow created with ID: j-4B9KNCB48TTT
j-4B9KNCB48TTT
```

HW 5.3 EDA of Google n-grams dataset

A large subset of the Google n-grams dataset

https://aws.amazon.com/datasets/google-books-ngrams/ (https://aws.amazon.com/datasets/google-books-ngrams/)

which we have placed in a bucket/folder on Dropbox on s3:

https://www.dropbox.com/sh/tmqpc4o0xswhkvz/AACUifrl6wrMrlK6a3X3lZ9Ea?dl=0 (https://www.dropbox.com/sh/tmqpc4o0xswhkvz/AACUifrl6wrMrlK6a3X3lZ9Ea?dl=0)

`s3://filtered-5grams/`

In particular, this bucket contains (~200) files (10Meg each) in the format:

```
(ngram) \t (count) \t (pages_count) \t (books_count)
```

For HW 5.3-5.5, for the Google n-grams dataset unit test and regression test your code using the first 10 lines of the following file:

googlebooks-eng-all-5gram-20090715-0-filtered.txt

Once you are happy with your test results proceed to generating your results on the Google n-grams dataset.

Do some EDA on this dataset using mrjob, e.g.,

- Longest 5-gram (number of characters)
- Top 10 most frequent words (please use the count information), i.e., unigrams
- 20 Most/Least densely appearing words (count/pages_count) sorted in decreasing order of relative frequency
- Distribution of 5-gram sizes (character length). E.g., count (using the count field) up how many times a 5-gram of 50 characters shows up. Plot the data graphically using a histogram.

In [47]:
```
## download dev data
!wget -O google-5grams-test.txt "https://www.dropbox.com/sh/tmqpc4o0xs
whkvz/AACr50woxiBWoaiiLmnwduX8a/googlebooks-eng-all-5gram-20090715-0-f
iltered.txt?dl=0"
```

```
--2016-02-15 23:42:40--  https://www.dropbox.com/sh/tmqpc4o0xswhkvz/
AACr50woxiBWoaiiLmnwduX8a/googlebooks-eng-all-5gram-20090715-0-filte
red.txt?dl=0
Resolving www.dropbox.com... 108.160.172.206, 108.160.172.238
Connecting to www.dropbox.com|108.160.172.206|:443... connected.
HTTP request sent, awaiting response... 302 FOUND
Location: https://dl.dropboxusercontent.com/content_link/aIZVHfoBfzA
CzWxyagqbBmxeGTl601USgFACattc1MpkSXDxZ9gk5hMyZTQFqWBn/file [followin
g]
--2016-02-15 23:42:41--  https://dl.dropboxusercontent.com/content_l
ink/aIZVHfoBfzACzWxyagqbBmxeGTl601USgFACattc1MpkSXDxZ9gk5hMyZTQFqWBn
/file
Resolving dl.dropboxusercontent.com... 199.47.217.69
Connecting to dl.dropboxusercontent.com|199.47.217.69|:443... connec
ted.
HTTP request sent, awaiting response... 200 OK
Length: 11444614 (11M) [text/plain]
Saving to: 'google-5grams-test.txt'

google-5grams-test. 100%[=====================>]  10.91M  1.03MB/s
in 8.3s

2016-02-15 23:42:50 (1.31 MB/s) - 'google-5grams-test.txt' saved [11
444614/11444614]
```

In [48]:
```
#ngram \t count \t page_count \t book_count
!head -n10 google-5grams-test.txt
```

```
A BILL FOR ESTABLISHING RELIGIOUS       59      59      54
A Biography of General George   92      90      74
A Case Study in Government      102     102     78
A Case Study of Female  447     447     327
A Case Study of Limited 55      55      43
A Child's Christmas in Wales    1099    1061    866
A Circumstantial Narrative of the       62      62      50
A City by the Sea       62      60      49
A Collection of Fairy Tales     123     117     80
A Collection of Forms of        116     103     82
```

In [77]:
```
## upload test set to s3 bucket
import boto
from boto.s3.connection import S3Connection
from boto.s3.key import Key

# AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY are environment variable
s
# S3Connection argument can be empty
conn = S3Connection()
bucket = conn.create_bucket('w261-safyre-hw53-input')  # sub-datasets
bucket already exists
myBucket = conn.get_bucket('w261-safyre-hw53-input')

k = Key(myBucket)
k.key = 'google-5grams-test.txt'
k.set_contents_from_filename('google-5grams-test.txt')
#googlebooks-eng-all-5gram-20090715-0-filtered.txt
```

Out[77]:
```
11444614
```

In [201]:
```
%%writefile MR_HW53.py
#!/usr/bin/env python
# mapper for 5.3 word count
# find the most frequent visitors per url
from mrjob.job import MRJob
from mrjob.step import MRStep
import csv
from heapq import heappush, heappop,nlargest, nsmallest
import re

class WordCount(MRJob):
    def steps(self):
        return [MRStep(
            mapper_init = self.mapper_init,
            mapper = self.mapper,
            combiner = self.combiner,
            reducer = self.reducer,
            reducer_final = self.reducer_final
            )]

    def mapper_init(self):
        self.counts = {}
        self.pages = {}
        #self.lengths = [] # heap queue
        self.unigram_counts = []
        self.density_counts = []
        #self.counts_final = {}
        #self.density_final = {}
    def mapper(self, _, line):
        line.strip()
        [ngram, gram_count, pages,books] = re.split("\t",line)
        gram_count = int(gram_count)
        pages = int(pages)
        words = re.split(" ",ngram)


        # emit words, counts and pages (for density calcs)
        for word in words:
            self.counts.setdefault(word,0)
            self.pages.setdefault(word,0)
            self.counts[word] += gram_count
            self.pages[word] += pages

        for word in self.counts.keys():
            yield word, (self.counts[word], self.pages[word], ngram)


    def combiner(self, word, counts_pages):
        sum_counts = sum_pages = num = 0
        for count, pages, ngram in counts_pages:
            sum_counts += count
            sum_pages += pages
        yield word, (sum_counts, sum_pages, ngram)

    def reducer(self, word, sums):
        sum_counts = sum_pages = 0
        self.counts_final = {}
        self.density_final = {}
        self.lengths = []
        for count, pages, ngram in sums:
            self.counts_final.setdefault(word,0)
            self.density_final.setdefault(word,0)
            sum_counts += count
            sum_pages += pages
            self.counts_final[word] = sum_counts
            self.density_final[word] = float(sum_counts)/sum_pages
            heappush(self.lengths, (len(ngram), ngram))
        #yield word, (sum_counts, sum_pages)
    '''
    def reducer_final(self, word, sums):
        density = 0
        for count_sums, page_sums in sums:
            density = float(count_sums, page_sums)
        yield word, (count_sums, density)
     '''
    def reducer_final(self):
        unigram_counts = []
        density_counts = []
        #for word, counts, density in word_count_density:

        for word in self.counts_final.keys():
            heappush(unigram_counts, (self.counts_final[word], word))
            heappush(density_counts, (self.density_final[word], word))
        top10_counts = nlargest(10, unigram_counts)
        top10_density = nlargest(10, density_counts)
        bottom10_density = nsmallest(10, density_counts)

        yield None, top10_counts # top 10 most freq words
        yield None, top10_density # top 10 most dense words
        yield None, bottom10_density # bottom 10 dense words
        yield None, nlargest(1, self.lengths) # max ngram

        for i in range(len(self.lengths)):
            # pop out each item for histogram,
            # is sorted already in asc order
            yield heappop(self.lengths)

if __name__ == "__main__":
    WordCount.run()
```

```
Overwriting MR_HW53.py
```

In [202]:
```
#test with a small file
#!head -n100 google-5grams-test.txt > test_input52.txt
```

In [203]:
```
!chmod +x MR_HW53.py
!python MR_HW53.py test_input52.txt > hw53b_out.txt
```

```
no configs found; falling back on auto-configuration
no configs found; falling back on auto-configuration
creating tmp directory /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000g
n/T/MR_HW53.Safyre.20160219.161734.789699

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by d
efault. It's recommended you run your job with --strict-protocols or
set up mrjob.conf as described at https://pythonhosted.org/mrjob/wha
ts-new.html#ready-for-strict-protocols

writing to /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000gn/T/MR_HW53.
Safyre.20160219.161734.789699/step-0-mapper_part-00000
Counters from step 1:
  (no counters found)
writing to /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000gn/T/MR_HW53.
Safyre.20160219.161734.789699/step-0-mapper-sorted
> sort /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000gn/T/MR_HW53.Safy
re.20160219.161734.789699/step-0-mapper_part-00000
writing to /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000gn/T/MR_HW53.
Safyre.20160219.161734.789699/step-0-reducer_part-00000
Counters from step 1:
  (no counters found)
Moving /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000gn/T/MR_HW53.Safy
re.20160219.161734.789699/step-0-reducer_part-00000 -> /var/folders/
jz/dhc2gzfj2091nhfmrbvv3cjh0000gn/T/MR_HW53.Safyre.20160219.161734.7
89699/output/part-00000
Streaming final output from /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh
0000gn/T/MR_HW53.Safyre.20160219.161734.789699/output
removing tmp directory /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000g
n/T/MR_HW53.Safyre.20160219.161734.789699
```

In [204]:
```
!cat hw53b_out.txt
```

```
null    [[676, "where"]]
null    [[1.0, "where"]]
null    [[1.0, "where"]]
null    [[27, "A branch was established at"]]
27      "A branch was established at"
```

In [ ]:

In [141]:
```
!chmod +x MR_HW53bc.py
!python MR_HW53bc.py -r emr \
    --emr-job-flow-id=j-4B9KNCB48TTT \
    --conf-path /Users/Safyre/cc-mrjob/mrjob.conf\
    --file s3://w261-safyre-hw53-input/google-5grams-test.txt \
    --output-dir= s3://w261-safyre-hw53-output/output/out/ \
    --no-output \
    --cleanup=NONE\
    --no-strict-protocol
```

```
Got unexpected keyword arguments: ssh_tunnel
using existing scratch bucket mrjob-62fd57571f35a64e
using s3://mrjob-62fd57571f35a64e/tmp/ as our scratch dir on S3
creating tmp directory /var/folders/jz/dhc2gzfj2091nhfmrbvv3cjh0000g
n/T/MR_HW53bc.Safyre.20160219.064554.578414
Copying non-input files into s3://mrjob-62fd57571f35a64e/tmp/MR_HW53
bc.Safyre.20160219.064554.578414/files/
Adding our job to existing job flow j-4B9KNCB48TTT
^C
```

In [118]:
```
!tail -n10 hw53b_out.txt
```

HW 5.4 Synonym detection over 2Gig of Data

For the remainder of this assignment you will work with two datasets:

1: unit/systems test data set: SYSTEMS TEST DATASET

*Three terms, A,B,C and their corresponding strip-docs of co-occurring terms

DocA {X:20, Y:30, Z:5} DocB {X:100, Y:20} DocC {M:5, N:20, Z:5}*

2: A large subset of the Google n-grams dataset as was described above

For each HW 5.4 -5.5.1 Please unit test and system test your code with respect to SYSTEMS TEST DATASET and show the results. Please compute the expected answer by hand and show your hand calculations for the SYSTEMS TEST DATASET. Then show the results you get with you system.

In this part of the assignment we will focus on developing methods for detecting synonyms, using the Google 5-grams dataset. To accomplish this you must script two main tasks using MRJob:

(*1) Build stripes for the most frequent 10,000 words using cooccurence informationa based on the words ranked from 9001,-10,000 as a basis/vocabulary (drop stopword-like terms), and output to a file in your bucket on s3 (bigram analysis, though the words are non-contiguous).*

(2) Using two (symmetric) comparison methods of your choice (e.g., correlations, distances, similarities), pairwise compare all stripes (vectors), and output to a file in your bucket on s3.

==Design notes for (1)== For this task you will be able to modify the pattern we used in HW 3.2 (feel free to use the solution as reference). To total the word counts across the 5-grams, output the support from the mappers using the total order inversion pattern:

<word,count>*

to ensure that the support arrives before the cooccurrences.

In addition to ensuring the determination of the total word counts, the mapper must also output co-occurrence counts for the pairs of words inside of each 5-gram. Treat these words as a basket, as we have in HW 3, but count all stripes or pairs in both orders, i.e., count both orderings: (word1,word2), and (word2,word1), to preserve symmetry in our output for (2).

==Design notes for (2)== For this task you will have to determine a method of comparison. Here are a few that you might consider:

- Jaccard
- Cosine similarity
- Spearman correlation
- Euclidean distance
- Taxicab (Manhattan) distance
- Shortest path graph distance (a graph, because our data is symmetric!)
- Pearson correlation
- Kendall correlation ...

However, be cautioned that some comparison methods are more difficult to parallelize than others, and do not perform more associations than is necessary, since your choice of association will be symmetric.

Please use the inverted index (discussed in live session #5) based pattern to compute the pairwise (term-by-term) similarity matrix.

Please report the size of the cluster used and the amount of time it takes to run for the index construction task and for the synonym calculation task. How many pairs need to be processed (HINT: use the posting list length to calculate directly)? Report your Cluster configuration!

In [ ]:

In [205]:

```
%%writefile HW_54.py
#!/usr/bin/python
from mrjob.job import MRJob
from mrjob.step import MRStep
from mrjob.protocol import RawValueProtocol
import re
import operator
from collections import Counter, OrderedDict

class BigramCooccurrence(MRJob):

    OUTPUT_PROTOCOL = RawValueProtocol

    URLs = {}

    def steps(self):
        return [MRStep(
                mapper = self.mapper,
                combiner = self.combiner,
                reducer_init = self.reducer_init,
                reducer = self.reducer
                )]

    def mapper_1(self, _, line):

        # emit stripes with coocurrence line count
        for line in sys.stdin:
            line=line.strip()
            co_words=line.split()

            # prevent double counts
            co_words.sort()
            for i, first_word in enumerate(co_words):
                stripe={}
                for following_word in co_words[i+1:]:
                    stripe.setdefault(following_word, 1)
                    stripe[following_word] += 1

                yield first_word, stripe

    def combiner_1(self,first_word,stripe):
        combined_coorrences=Counter({}) #Counters make tracking indivi
dual product counts easier
        for line in sys.stdin:

        cooccurrences=Counter(eval(stripe))

        '''
        combine counts in stripe with same first word:
        inverted index: word --> docA, docB, docC, etc
        sort in ascending order and insert into tuple of word-stripes
with length 1000 (initialized prior)
        create binarized version for jaccardian and euclidean

        Need another MR step to calculate similarities, euclidean is e
asiest
```

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

```
Writing HW_54.py
```

In [ ]: