Homework 5

Alex Smith
June 11, 2016
MIDS261 - Machine Learning at Scale
Professor Shanahan
Due: June 19, 2016

Useful resources

The following resources were particularly useful.

- Wikipedia article on data warehouses (https://en.wikipedia.org/wiki/Data_warehouse)
- Async 5.4

Libraries

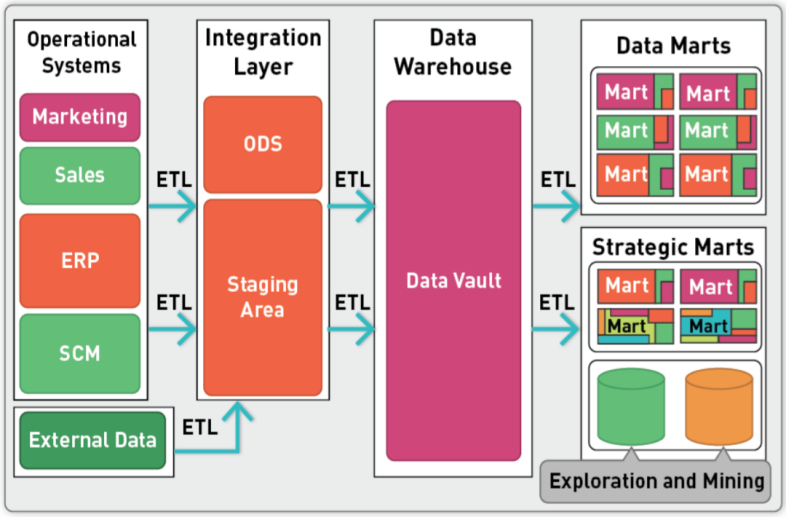
The following libraries must be installed before running the below code. They can all be installed through Pip (https://github.com/pypa/pip).

- Scikit Learn (http://scikit-learn.org/stable/)
- Numpy (http://www.numpy.org/)
- Regular Expression (https://docs.python.org/2/library/re.html)
- Pretty Table (https://pypi.python.org/pypi/PrettyTable)
- Random (https://docs.python.org/2/library/random.html)
- Datetime (https://docs.python.org/2/library/datetime.html)
- NLTK (http://www.nltk.org/)

HW 5.0

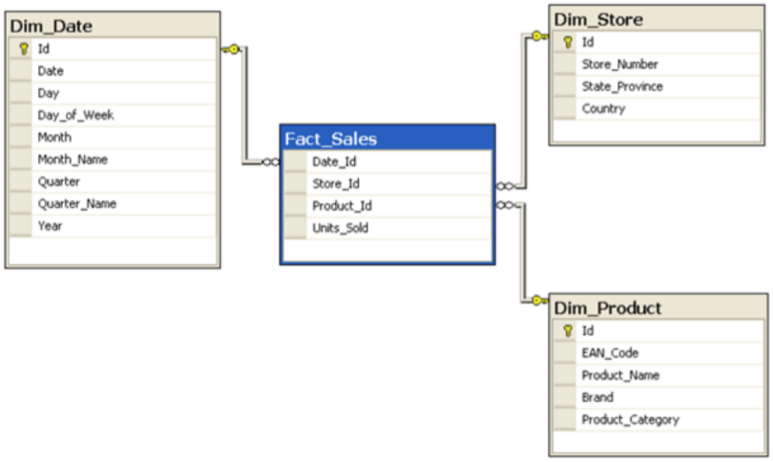
What is a data warehouse? What is a Star schema? When is it used?

A **data warehouse** is a central repository of data from diverse sources. Traditionally, data warehouse have stored relational data. However, they are increasingly also storing semi-structured (e.g. logs) and unstructured (e.g. tweets) data. Data warehouses have historically formed the foundation of business intelligence. Now, they are increasingly used for data science. More simply, we can think of a data warehouse as a system for reporting and data analysis.



Source: Async slides 5.4

A **star schema** is a manner of organizing data by using one or more fact tables to reference any number of dimension tables (Async 5.4). For example, in the image below we see a a single fact table that stores the sales. Each sales record has a date, store, product, and units sold field. The date, store, and product fields all reference dimension tables. For example, the store_id for a given transaction will be linked to information on the store number and the location. Star schemas are used when needing to organize large amounts of data for quick querying and editing. By linking to multiple dimension tables, the star schema is easily updatable. Let's continue to consisder the example below. If we realize that our information for a given product is inaccurate and needs to be updated, it is easy to update the data with a star schema. We just need to update the information in the product dimension table for this specific product. If instead, we had stored the information about the product in the fact table, then we would have to search for every instance of that product information in the fact table and update it there.



Source: Async slides 5.4

HW 5.1

In the database world What is 3NF? Does machine learning use data in 3NF? If so why?
In what form does machine learning consume data?
Why would one use log files that are denormalized?

In the database world, **3NF** stands for third normal form. The 3rd normal form is the third step in normalizing a database that builds on the 1st and 2nd normal forms. The 1st normal form requires that a record in a database with attributes that contain only single indivisible elements. For example, if we are attempting to store the telephone information for an individual, we might need to store multiple telephone numbers for a single individual. The simplest, and possibly most intuitive, approach would be to just list two telephone numbers in the telephone field.

Customer

Customer ID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659 555-776-4100
789	Maria	Fernandez	555-808-9633

Source: Wikipedia, First Normal Form (https://en.wikipedia.org/wiki/First_normal_form)

However, this would not be in 1st normal form because the entry for the telephone is not a single indivisible element. To make this data compliant with 1st normal form, we could split the record for that individual so that we had two records each with a telephone number.
Data is in 2nd normal form if it meets all the requirements of 1st normal form and all the non-key attributes depend on the whole key and not just part of the key. This can best be illustrated by an example. Consider the table below. Each record has a primary key composed of the customer_id and store_id.

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

Source: Keydata.com (https://www.1keydata.com/database-normalization/second-normal-form-2nf.php)

However, the purchase location is dependent only part of the key, just the store_id. This table fails to meet the standards of the 2nd normal form for this reason. To make the data fit into 2nd normal form requires us to split the data into two tables, like below, where each attribute depends on the whole primary key.

TABLE_PURCHASE

Customer ID	Store ID
1	1
1	3
2	1
3	2
4	3

TABLE_STORE

Store ID	Purchase Location
1	Los Angeles
2	New York
3	San Francisco

Source: Keydata.com (https://www.1keydata.com/database-normalization/second-normal-form-2nf.php)

Only after meeting the requirements of the 1st and 2nd normal forms, can we attempt to normalize our data to

the 3rd normal form. 3rd normal form adds the additional requirement that all attributes are determined solely by the key value. This is best illustrated through an example. Consider the following table below that shows the winner for a few tournaments. The tournament and the year combined key that is required to identify the row.

Tournament Winners

Tournament	Year	Winner	Winner Date of Birth
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

Source: Wikipedia, Third Normal Form (https://en.wikipedia.org/wiki/Third_normal_form)

However, when looking at the data, it is clear that the year of birth is dependent on the winner, a non-key attribute. This is risky because it introduces the possibility that the years of birth could be different for the same winner. To bring this data into compliance with 3rd normal form, we would need to split the data into 2 tables like below.

Tournament Winners

Tournament	Year	Winner
Indiana Invitational	1998	Al Fredrickson
Cleveland Open	1999	Bob Albertson
Des Moines Masters	1999	Al Fredrickson
Indiana Invitational	1999	Chip Masterson

Winner Dates of Birth

Winner	Date of Birth
Chip Masterson	14 March 1977
Al Fredrickson	21 July 1975
Bob Albertson	28 September 1968

Source: Wikipedia, Third Normal Form (https://en.wikipedia.org/wiki/Third_normal_form)

Machine learning consumes data in denormalized form. This is because we want our model to consume the data with all the featues available. Normalizing the data begins hiding the features in other related tables. If we do get normalized data and need to do some machine learning, we should first de-normalize the data. We can do this by joining the multiple tables together. A denormalized **log file** would be very useful precisely for this reason. A denormalized log file would have all the features for each entry and would provide all the features to the machine learning model.

HW 5.2

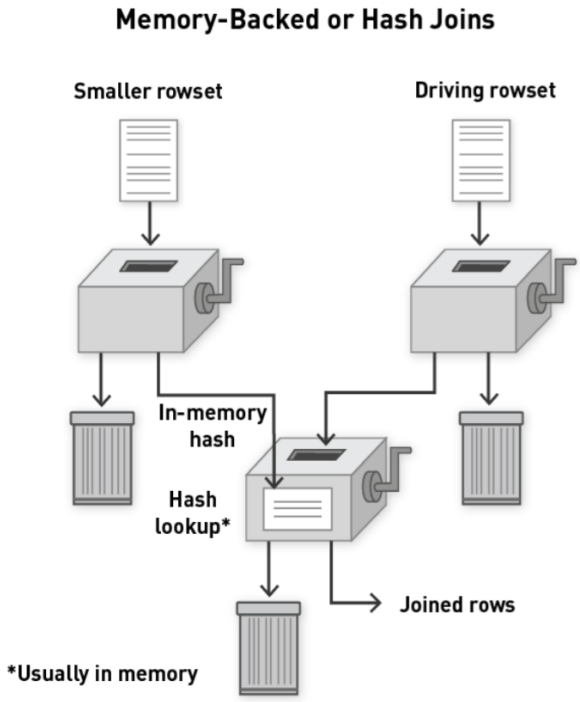
Using MRJob, implement a hashside join (memory-backed map-side) for left, right and inner joins. Run your code on the data used in HW 4.4: (Recall HW 4.4: Find the most frequent visitor of each page using mrjob and the output of 4.2 (i.e., transfromed log file). In this output please include the webpage 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*

Defining our files

We have two files that we will be joining: "MS_weblog.txt" and "MS_webpages.txt". The weblog file has a list of visitor ids and webpage ids. The webpages file has a list of webpage ids and the URLs. We choose the URLs data as the left table that we hold in memory because it is smaller. We want to put the smaller table in memory. See the diagram from Async 5.10 slides:



Source: Async 5.10 Slides

Create the MRJob class that performs an right join

```
%%writefile right_mapper.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep

class right_mapper(MRJob):

    # create a value to dictionary
    # to hold the data of the left
    # table
    left_table = {}


    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper_init=self.mapper_init,\
                       mapper=self.mapper,\
                       reducer=None)]


    # we need to initalize our mapper by
    # storing the left table in memory
    def mapper_init(self):

        # open the left table, the list
        # urls with their ids
        with open('MS_webpages.txt','r') as\
        myfile:

            # read each line in the table
            for line in myfile.readlines():

                # split the line by the commas
                # and set the id and url values
                line = line.split(',')
                _id = int(line[0].strip())
                _url = line[1].strip()

                # add the url and id to the
                # dictionary
                self.left_table[_id] = _url


    # we read each line being fed into the
    # the mapper and output the record and
    # also include the webpage url if it
    # exists
    def mapper(self, _, line):

        # split the line by commas
        # and set the web_id and customer_id
        # values
        line = line.split(',')
        web_id = int(line[1].strip())
        cust_id = int(line[4].strip())
```

```
        # set a blank URL value because
        # with a right join we want to
        # output all log values regarless
        # of whether we can actually find
        # a URL
        url = ""

        # if the web_id exists in the url
        # list, set to the url variable to
        # that value
        if web_id in self.left_table.keys():
            url = self.left_table[web_id]

        # set the keys and value we want to lead
        # where the key is tuple of web_id and
        # url and the value is the customer_id
        _key = web_id,url
        _value = cust_id

        # yield key and value
        yield _key,_value


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

```
Overwriting right_mapper.py
```

Create the runner to run the class within this notebook

In [6]:
```
%reload_ext autoreload
%autoreload 2

# import the MRJob that we created
from right_mapper import right_mapper

# import the pandas library to help us
# export our data
import pandas as pd

# set the data that we're going to pull
mr_job = right_mapper(args=['MS_weblog.txt',\
                        '--file=MS_webpages.txt'])

# create the runner and run it
with mr_job.make_runner() as runner:
    runner.run()

    # set a counter for counting the lines of
    # output
    count = 0

    # create an array to hold the output
    output = []

    # stream_output: get access to the output
    for line in runner.stream_output():

        # increment the counter
        count = count + 1

        # grab the key,value
        key,value =  mr_job.parse_output_line(line)

        # set the key,value to terms we
        # understand
        web_id,web_url = key
        cust_id = value

        # set the output line we want and append
        # to our output array
        output_line = [web_id,web_url,cust_id]
        output.append(output_line)

    # convert the output to a pandas and export
    # it as a CSV
    output_pd = pd.DataFrame(output)
    output_pd.to_csv('HW5.2_Output_Right',\
                    header=False,index=False)

    # print out the number of lines we processed
    print "Total records after a right join:",count
```

```
Total records after a right join: 98654
```

Create an MRJob class that performs a left join

```
%%writefile left_mapper.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
import mrjob

class left_mapper(MRJob):

    # create a value to dictionary
    # to hold the data of the left
    # table
    left_table = {}

    # create an output array to store
    # our data
    outputs = []

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):

        return [MRStep(mapper_init=self.mapper_init,\
                       mapper=self.mapper,\
                      mapper_final=self.mapper_final),\
                MRStep(reducer=self.reducer)]

    # set 2 mappers for this function
    def jobconf(self):

        orig_jobconf = super(left_mapper, self).jobconf()
        custom_jobconf = {
            'mapreduce.job.maps': '2',
        }

        return mrjob.conf.combine_dicts(orig_jobconf, custom_jobconf)


    # we need to initalize our mapper by
    # storing the left table in memory
    def mapper_init(self):

        # open the left table, the list
        # urls with their ids
        with open('MS_webpages.txt','r') as\
        myfile:

            # read each line in the table
            for line in myfile.readlines():

                # split the line by the commas
                # and set the id and url values
                line = line.split(',')
                _id = int(line[0].strip())
                _url = line[1].strip()

                # add the url and id to the
                # dictionary. we also add a flag
```

```
            # indicator that tells us whether
            # this particular site has been
            # outputted or not
            self.left_table[_id] = [_url,False]


# we read each line being fed into the
# the mapper and update the outputs array
# and the left-table dictionary
def mapper(self, _, line):

    # split the line by commas
    # and set the web_id and customer_id
    # values
    line = line.split(',')
    web_id = int(line[1].strip())
    cust_id = int(line[4].strip())

    # if the web_id exists in the url
    # list, set to the url variable to
    # that value, remember we only
    # want to consider values that have a
    # match in the left table
    if web_id in self.left_table.keys():
        url = self.left_table[web_id][0]

        # set the indicator in the table
        # to true now that we've used this
        # web page
        self.left_table[web_id][1] = True

        # set the keys and value we want to
        # use to update the output array
        _key = web_id,url
        _value = cust_id

        # update the outputs table
        yield _key,_value


# we want to now add any webpages that
# we were missing, we'll yield them
# with a very special key
def mapper_final(self):

    # set the special key
    special_key = 99

    # loop through the left table dictionary
    for web_id in self.left_table.keys():

        # if the page has not been viewed,
        # output it
        if self.left_table[web_id][1]==False:
            url = self.left_table[web_id][0]
            _value = web_id,url
```

```
            yield special_key,_value


# our final reducer really just passes
# the outputs from the mappers and does
# very little work on its own
def reducer(self, key, values):

    # set the special key
    special_key = 99

    # if it's our special key, we'll add
    # the websites to our dictionary
    if key == special_key:

        # create a dictionary for the
        # unseen websites
        unseen = {}

        # loop through all the websites
        # and add them to the dictionary
        for web_id,url in values:

            # check to see if we've already
            # added it
            if web_id not in unseen.keys():
                unseen[web_id] = [url,0]

            # increment the count for the
            # unseen
            unseen[web_id][1] = \
            unseen[web_id][1] + 1

        # loop through the dictionary
        # and yield each unseen webpage
        for web_id in unseen.keys():

            # check to see if the unseen has a
            # value of 2 since we want it to
            # be unseen by both mappers
            if unseen[web_id][1] == 2:
                _key = web_id,unseen[web_id][0]
                _value = ""
                yield _key,_value

    # otherwise we're just going to yield
    # each website
    else:

        # set the web id and the web url
        web_id,web_url = key
        _key = web_id,web_url

        # loop through each customer visit
        for cust in values:
            _value = cust
```

```
                yield _key,_value


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

```
Overwriting left_mapper.py
```

Create the runner and run the job in the notebook

In [287]:
```
%reload_ext autoreload
%autoreload 2

# import the MRJob that we created
from left_mapper import left_mapper

# import the pandas library to help us
# export our data
import pandas as pd

# set the data that we're going to pull
mr_job = left_mapper(args=['MS_weblog.txt',\
                           '--file=MS_webpages.txt'])

# create the runner and run it
with mr_job.make_runner() as runner:
    runner.run()

    # initalize a counter to keep track of where we
    # are
    count = 0

    # create an output array to hold our findings
    output = []

    # stream_output: get access to the output
    for line in runner.stream_output():

        # increment the counter
        count = count + 1

        # grab the key,value
        key,value =  mr_job.parse_output_line(line)

        # set the key,value to terms we
        # understand
        web_id,web_url = key
        cust_id = value

        # set the output line we want and append
        # to our output array
        output_line = [web_id,web_url,cust_id]
        output.append(output_line)

    # convert the output to a pandas and export
    # it as a CSV
    output_pd = pd.DataFrame(output)
    output_pd.to_csv('HW5.2_Output_Left',\
                     header=False,index=False)

    # print out the number of lines we processed
    print "Total records after a left join:",len(output)
```

```
Total records after a left join: 98663
```

Create an MRJob class that complete an inner join

```
%%writefile inner_mapper.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep

class inner_mapper(MRJob):

    # create a value to dictionary
    # to hold the data of the left
    # table
    left_table = {}


    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper_init=self.mapper_init,\
                       mapper=self.mapper,\
                       reducer=None)]


    # we need to initalize our mapper by
    # storing the left table in memory
    def mapper_init(self):

        # open the left table, the list
        # urls with their ids
        with open('MS_webpages.txt','r') as\
        myfile:

            # read each line in the table
            for line in myfile.readlines():

                # split the line by the commas
                # and set the id and url values
                line = line.split(',')
                _id = int(line[0].strip())
                _url = line[1].strip()

                # add the url and id to the
                # dictionary
                self.left_table[_id] = _url


    # we read each line being fed into the
    # the mapper and output the record only
    # if the webpage url of it also exists
    def mapper(self, _, line):

        # split the line by commas
        # and set the web_id and customer_id
        # values
        line = line.split(',')
        web_id = int(line[1].strip())
        cust_id = int(line[4].strip())

        # if the web_id exists in the url
```

```
        # list, set to the url variable to
        # that value
        if web_id in self.left_table.keys():
            url = self.left_table[web_id]

            # set the keys and value we want to lead
            # where the key is tuple of web_id and
            # url and the value is the customer_id
            _key = web_id,url
            _value = cust_id

            # yield key and value
            yield _key,_value


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

```
Overwriting inner_mapper.py
```

Create a runner and run the job in the notebook

In [84]:
```
%reload_ext autoreload
%autoreload 2

# import the MRJob that we created
from inner_mapper import inner_mapper

# import the pandas library to help us
# export our data
import pandas as pd

# set the data that we're going to pull
mr_job = inner_mapper(args=['MS_weblog.txt',\
                            '--file=MS_webpages.txt'])

# create the runner and run it
with mr_job.make_runner() as runner:
    runner.run()

    # set a counter for counting the lines of
    # output
    count = 0

    # create an array to hold the output
    output = []

    # stream_output: get access to the output
    for line in runner.stream_output():

        # increment the counter
        count = count + 1

        # grab the key,value
        key,value =  mr_job.parse_output_line(line)

        # set the key,value to terms we
        # understand
        web_id,web_url = key
        cust_id = value

        # set the output line we want and append
        # to our output array
        output_line = [web_id,web_url,cust_id]
        output.append(output_line)

    # convert the output to a pandas and export
    # it as a CSV
    output_pd = pd.DataFrame(output)
    output_pd.to_csv('HW5.2_Output_Inner',\
                     header=False,index=False)

    # print out the number of lines we processed
    print "Total records after an inner join:",count
```

```
Total records after an inner join: 98654
```

Based on the joins we completed above, we can see that every customer log record has an associated website (hence right join equals inner join), but not all websites have a customer visit (hence left join greater than right join).

Disclaimer: All MRJobs below use multiple reducers, with the exception of any final sorts. Any final sorts use a single reducer and an identity mapper only for the very last step of sorting.

HW 5.3 Exploratory data analysis of Google n-grams dataset

A large subset (https://aws.amazon.com/datasets/google-books-ngrams/) of the Google n-grams dataset, which we have placed in a bucket/folder on Dropbox (https://www.dropbox.com/sh/tmqpc4o0xswhkvz/AACUifrl6wrMrlK6a3X3lZ9Ea?dl=0) on s3 (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.*

Longest 5-gram (number of characters)

Create a testing data set that we can test locally and is relatively small

In [237]:
```
!head -100 data/googlebooks-eng-all-5gram-20090715-0-filtered.txt > tes
t.txt
print "Finished creating test data"
```

```
Finished creating test data
```

In [327]:
```
# move the the testing data to s3
!aws s3 cp test.txt s3://aks-w261-hw5/test.txt
```

```
upload: ./test.txt to s3://aks-w261-hw5/test.txt
```

MRJob Class that finds the longest n-gram

We have multiple choices for how to implement this MRJob problem. We choose one that outputs multiple maximums, one per reducer. We can then find the maximum of these. This will be trivial because now finding the maximum will be over a very small dataset.

```
%%writefile longestNgram.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep

class longestNgram(MRJob):

    # longest ngram length and the
    # actual ngram
    max_length = 0
    max_ngram = None

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper=self.mapper,\
                       reducer=self.reducer,\
                       reducer_final=self.reducer_final)]


    # our mapper reads in each line of data
    # calculates the length and returns the
    # length of the line and the ngram
    def mapper(self, _, line):

        # split the line by the tabs
        line = line.split("\t")
        ngram = line[0]

        # calculate the length of the ngram
        length = len(ngram)

        # yield the length and the ngram
        # as a key,value pair
        yield length,ngram


    # our reducer reads in the ngrams for
    # each length
    def reducer(self, length, ngrams):

        # compare the length to the maximum
        # length and only execute if its bigger
        if length > self.max_length:

            # set the new max length and ngram
            self.max_length = length

            # set the max ngram to an array
            # and fill with all the ngrams that
            # meet this max
            self.max_ngram = []
            for ngram in ngrams:
                self.max_ngram.append(ngram)


    # our reducer final then outputs the maximum
```

```
    # ngram and the ngrams that make up this length
    def reducer_final(self):

        # yield the maximum ngram and its length
        yield self.max_length,self.max_ngram


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

```
Overwriting longestNgram.py
```

Create a runner to test on a small data set

In [95]:
```
%reload_ext autoreload
%autoreload 2

# import the MRJob that we created
from longestNgram import longestNgram

# import numpy to help us with the last step of
# finding the maximum
import numpy as np

# set the data that we're going to pull
mr_job = longestNgram(args=['test.txt'])

# create the runner and run it
with mr_job.make_runner() as runner:
    runner.run()

    # create arrays to hold the outputs from
    # our reducers, we allow multiple reducers
    # so that our program is easily scalable
    lengths = []
    ngrams = []

    # stream_output: get access to the output
    for line in runner.stream_output():

        # grab the key,value
        _length,_ngrams =  mr_job.parse_output_line(line)

        # add the length to the lengths and add
        # the ngrams to our ngram list
        lengths.append(_length)
        ngrams.append(list(_ngrams))

    # get the index of the maximum ngram by
    # converting our arrays to numpy arrays
    lengths = np.array(lengths)
    ngrams = np.array(ngrams)
    max_index = np.argmax(lengths)

    # use the index to grab the maximum values
    max_length = lengths[max_index]
    max_ngrams = ngrams[max_index]

    # print out the maximum ngram and its length
    print "The longest ngrams had a length of",\
    max_length
    print "These ngrams are:",max_ngrams
```

```
The longest ngrams had a length of 34
These ngrams are: ['A HANDBOOK ON THEODOLITE SURVEYING']
```

Test the runner on the test data but on the cluster

In [331]:
```
# create the cluster
!mrjob create-cluster \
--max-hours-idle 1 \
--aws-region=us-west-1 -c ~/.mrjob.conf
```

```
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating persistent cluster to run several jobs in...
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/no_script.Alex.20160618.174830.395929
Copying local files to s3://mrjob-f8c316b67324528f/tmp/no_script.Alex.2
0160618.174830.395929/files/...
j-1V97UD5N6422Z
```

In [335]:
```
# run the program with the cluster we
# just spun up
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_3
!python longestNgram.py -r emr s3://aks-w261-hw5/test.txt \
    --cluster-id=j-1V97UD5N6422Z \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_3 \
    --no-output
```

```
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/longestNgram.Alex.20160618.175957.065679
Copying local files to s3://mrjob-f8c316b67324528f/tmp/longestNgram.Ale
x.20160618.175957.065679/files/...
Adding our job to existing cluster j-1V97UD5N6422Z
Waiting for step 1 of 1 (s-C77A3L5EDWFP) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40458/cluster
  RUNNING for 19.2s
     0.0% complete
  RUNNING for 54.2s
    10.6% complete
  RUNNING for 84.9s
   100.0% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-C77A3L5EDWFP on ec2
-54-183-30-12.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-30-12.us-west-1.compute.amazonaws.
com/mnt/var/log/hadoop/steps/s-C77A3L5EDWFP/syslog
Counters: 54
        File Input Format Counters
                Bytes Read=45912
        File Output Format Counters
                Bytes Written=1835
        File System Counters
                FILE: Number of bytes read=2750
                FILE: Number of bytes written=3639121
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1872
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=24
                HDFS: Number of write operations=0
                S3: Number of bytes read=45912
                S3: Number of bytes written=1835
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=24
                Launched map tasks=24
                Launched reduce tasks=11
                Total megabyte-seconds taken by all map tasks=799482240
                Total megabyte-seconds taken by all reduce tasks=388719
360
                Total time spent by all map tasks (ms)=555196
                Total time spent by all maps in occupied slots (ms)=249
83820
                Total time spent by all reduce tasks (ms)=134972
                Total time spent by all reduces in occupied slots (ms)=
12147480
```

```
                Total vcore-seconds taken by all map tasks=555196
                Total vcore-seconds taken by all reduce tasks=134972
        Map-Reduce Framework
                CPU time spent (ms)=34780
                Combine input records=0
                Combine output records=0
                Failed Shuffles=0
                GC time elapsed (ms)=8998
                Input split bytes=1872
                Map input records=100
                Map output bytes=3119
                Map output materialized bytes=7417
                Map output records=100
                Merged Map outputs=264
                Physical memory (bytes) snapshot=14714015744
                Reduce input groups=18
                Reduce input records=100
                Reduce output records=11
                Reduce shuffle bytes=7417
                Shuffled Maps =264
                Spilled Records=200
                Total committed heap usage (bytes)=16750477312
                Virtual memory (bytes) snapshot=82899214336
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/longestNgra
m.Alex.20160618.175957.065679/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/longestNgram.Alex.20160618.175957.065679...
Killing our SSH tunnel (pid 15589)
```

In [336]:
```
# list our clusters
!aws emr list-clusters
```

```
CLUSTERS        j-1V97UD5N6422Z no_script.Alex.20160618.174830.395929
        26
STATUS  WAITING
STATECHANGEREASON       Cluster ready after last step completed.
TIMELINE        1466272124.47   1466272475.9
CLUSTERS        j-2ZBRZB00G8QK0 no_script.Alex.20160618.174309.971368
        0
STATUS  TERMINATED_WITH_ERRORS
STATECHANGEREASON       VALIDATION_ERROR        Subnet is required : Th
e specified instance type c4.large can only be used in a VPC.
TIMELINE        1466271809.74   1466271814.99
CLUSTERS        j-1AKN0BR62Y9V3 W261    8
STATUS  TERMINATED
STATECHANGEREASON       USER_REQUEST    Terminated by user request
TIMELINE        1466093499.95   1466096435.34   1466094081.38
CLUSTERS        j-2N44R4Q31HKC3 W261    0
STATUS  TERMINATED_WITH_ERRORS
STATECHANGEREASON       VALIDATION_ERROR        Service role  not autho
rized to call EC2
TIMELINE        1466088056.78   1466088170.12
```

Analyze the entire corpus of data

We've completed unit testing both locally and in the could. Let's try running our program over the entire thing!

In [337]:
```
# run the program with the cluster we
# just spun up
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_3
!python longestNgram.py -r emr s3://filtered-5grams/ \
    --cluster-id=j-1V97UD5N6422Z \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_3 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_3/_SUCCESS
delete: s3://aks-w261-hw5/out_5_3/part-00009
delete: s3://aks-w261-hw5/out_5_3/part-00010
delete: s3://aks-w261-hw5/out_5_3/part-00000
delete: s3://aks-w261-hw5/out_5_3/part-00001
delete: s3://aks-w261-hw5/out_5_3/part-00006
delete: s3://aks-w261-hw5/out_5_3/part-00007
delete: s3://aks-w261-hw5/out_5_3/part-00003
delete: s3://aks-w261-hw5/out_5_3/part-00008
delete: s3://aks-w261-hw5/out_5_3/part-00005
delete: s3://aks-w261-hw5/out_5_3/part-00004
delete: s3://aks-w261-hw5/out_5_3/part-00002
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/longestNgram.Alex.20160618.184415.363713
Copying local files to s3://mrjob-f8c316b67324528f/tmp/longestNgram.Ale
x.20160618.184415.363713/files/...
Adding our job to existing cluster j-1V97UD5N6422Z
Waiting for step 1 of 1 (s-1KUZSP23LJJ5O) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40458/cluster
  RUNNING for 14.4s
   100.0% complete
  RUNNING for 47.4s
     5.0% complete
  RUNNING for 78.6s
     7.6% complete
  RUNNING for 109.8s
    11.3% complete
  RUNNING for 141.4s
    16.0% complete
  RUNNING for 172.8s
    19.8% complete
  RUNNING for 204.0s
    23.7% complete
  RUNNING for 235.6s
    28.1% complete
  RUNNING for 267.0s
    33.1% complete
  RUNNING for 298.8s
    37.1% complete
  RUNNING for 329.8s
    40.8% complete
  RUNNING for 360.9s
    44.4% complete
  RUNNING for 391.8s
    48.4% complete
  RUNNING for 422.5s
    52.3% complete
  RUNNING for 453.4s
    56.0% complete
  RUNNING for 484.6s
    80.2% complete
  RUNNING for 515.2s
    82.0% complete
```

```
  RUNNING for 546.9s
    93.4% complete
  RUNNING for 578.2s
   100.0% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-1KUZSP23LJJ5O on ec
2-54-183-30-12.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-30-12.us-west-1.compute.amazonaws.
com/mnt/var/log/hadoop/steps/s-1KUZSP23LJJ5O/syslog
Counters: 56
        File Input Format Counters
                Bytes Read=2156069116
        File Output Format Counters
                Bytes Written=1759
        File System Counters
                FILE: Number of bytes read=1019827603
                FILE: Number of bytes written=2062294771
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=23450
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=190
                HDFS: Number of write operations=0
                S3: Number of bytes read=2156069116
                S3: Number of bytes written=1759
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=188
                Killed reduce tasks=2
                Launched map tasks=190
                Launched reduce tasks=13
                Other local map tasks=2
                Total megabyte-seconds taken by all map tasks=844308864
0
                Total megabyte-seconds taken by all reduce tasks=683714
5920
                Total time spent by all map tasks (ms)=5863256
                Total time spent by all maps in occupied slots (ms)=263
846520
                Total time spent by all reduce tasks (ms)=2374009
                Total time spent by all reduces in occupied slots (ms)=
213660810
                Total vcore-seconds taken by all map tasks=5863256
                Total vcore-seconds taken by all reduce tasks=2374009
        Map-Reduce Framework
                CPU time spent (ms)=2623420
                Combine input records=0
                Combine output records=0
                Failed Shuffles=0
                GC time elapsed (ms)=63082
                Input split bytes=23450
                Map input records=58682266
```

```
                Map output bytes=1862272363
                Map output materialized bytes=1021611542
                Map output records=58682266
                Merged Map outputs=2090
                Physical memory (bytes) snapshot=103364632576
                Reduce input groups=80
                Reduce input records=58682266
                Reduce output records=11
                Reduce shuffle bytes=1021611542
                Shuffled Maps =2090
                Spilled Records=117364532
                Total committed heap usage (bytes)=124261498880
                Virtual memory (bytes) snapshot=409737420800
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/longestNgra
m.Alex.20160618.184415.363713/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/longestNgram.Alex.20160618.184415.363713...
Killing our SSH tunnel (pid 15695)
```

Output the longest ngram

In [343]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_3_longest

# sync the files to a local directory
!aws s3 sync s3://aks-w261-hw5/out_5_3 /Users/Alex/Documents/Berkeley/16
02Summer/W261/HW5/5_3_longest
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_3_longest
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_3_longest/_SUCCESS
```

```
mkdir: /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_3_longest:
 File exists
_SUCCESS   part-00001 part-00003 part-00005 part-00007 part-00009
part-00000 part-00002 part-00004 part-00006 part-00008 part-00010
```

Each reducer has already spit out the ngram it finds as the longest. We now simply loop through the output files and grab the longest one. This task is quicker done locally than in the cloud because the number of outputs is small enough that we can easily hold the potential longest ngrams in memory.

```
# import the os and ast libraries
# to help us get the files and read
# in the files
import os
import ast

# create a dictionary to hold the longest
# ngrams outputted by each reducer
longest_ngrams = {}

# loop through our files and add each the longest
# ngrams to the dictionary, where the key is the
# length and the value is the ngrams
indir = '/Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_3_longes
t/'
for root, dirs, filenames in os.walk(indir):

    # loop through each file
    for filename in filenames:

        # set the filename
        filename = indir+filename

        # open the file
        with open(filename,'r') as myfile:

            # read the line in the file
            line = myfile.readline()

            # split the line by tabs
            line = line.split("\t")

            # set the count and the ngrams
            count = int(line[0])
            ngrams = ast.literal_eval(line[1])

            # check to see if we have this key
            # in the dictionary and if not, let's
            # add it
            if count not in longest_ngrams:
                longest_ngrams[count] = []

            # add the ngrams to this list
            longest_ngrams[count].append(ngrams)

# initalize a maximum value that we'll overwrite
max_count = 0
max_ngrams = None

# loop through each element in the dictionary
# compare it to the max value and if it's a new
# maximum update our values
for count in longest_ngrams:
    if count > max_count:
        max_count = count
        max_ngrams = longest_ngrams[count]
```

```
# print out the maximum ngram
print "The maximum ngram has a length of",max_count
print "The ngrams of this length are",max_ngrams
```

```
The maximum ngram has a length of 159
The ngrams of this length are [['ROPLEZIMPREDASTRODONBRASLPKLSON YHROAC
LMPARCHEYXMMIOUDAVESAURUS PIOFPILOCOWERSURUASOGETSESNEGCP TYRAVOPSIFENG
OQUAPIALLOBOSKENUO OWINFUYAIOKENECKSASXHYILPOYNUAT', 'AIOPJUMRXUYVASLYH
YPSIBEMAPODIKR UFRYDIUUOLBIGASUAURUSREXLISNAYE RNOONDQSRUNSUBUNOUGRABBE
RYAIRTC UTAHRAPTOREDILEIPMILBDUMMYUVERI SYEVRAHVELOCYALLOSAURUSLINROTS
R']]
```

Top 10 most frequently occuring words

Create the MRJob class that will perform a word count

```
%%writefile wordcount.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
from mrjob.protocol import RawProtocol

class wordcount(MRJob):

    MRJob.SORT_VALUES = True
    INTERNAL_PROTOCOL = RawProtocol
    OUTPUT_PROTOCOL = RawProtocol

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):

        # set how we want to sort, we use a single reducer
        # only so that we can do a total sort, we use multiple
        # reducers for all other steps
        JOBCONF = {
            'mapred.output.key.comparator.class': 'org.apache.hadoop.map
red.lib.KeyFieldBasedComparator',
            'mapred.text.key.comparator.options': '-k1,1nr',
            'mapreduce.job.reduces': 1,
        }

        return [MRStep(mapper=self.mapper,\
                       combiner=self.combiner,\
                       reducer=self.reducer),\
                MRStep(jobconf = JOBCONF,\
                       mapper=None,\
                       reducer=self.reducer_final)]


    # our mapper reads in each line of data
    # splits each line into multiple words
    # and then returns the word with a count
    def mapper(self, _, line):

        # split the line by the tabs, set the
        # ngram and lower-case it
        line = line.split("\t")
        ngram = line[0].lower()
        count = int(line[1])

        # split the ngram by the spaces
        ngram = ngram.split()

        # loop through each word in the
        # ngram and yield the word and
        # its count
        for word in ngram:
            yield word, str(count)


    # our combiner sums the values for each
    # word
```

```
    def combiner(self, word, counts):

        # loop through the counts, summing
        # as we go along
        sum_count = 0
        for count in counts:
            sum_count = sum_count+int(count)

        # yield the counts for each word
        yield word, str(sum_count)


    # our reducer performs the final count
    # for all words
    def reducer(self, word, counts):

        # loop through the counts, summing
        # as we go along
        sum_count = 0
        for count in counts:
            sum_count = sum_count+int(count)

        # convert the count to a string
        sum_count = str(sum_count)

        # we want to make sure that the count
        # string is at least 10 characters, if
        # its not, we add leading zeros. this
        # helps us sort
        extend = "0"

        # while the count lenght is less than 8
        while len(sum_count) < 10:
            sum_count = extend + sum_count

        # yield the counts as the key so that
        # we can the maximum
        yield str(sum_count), word


    # our reducer final then outputs a sorted
    # list of the words
    def reducer_final(self, count, words):

        # loop through the list with the words
        for word in words:
            yield str(int(count)), word

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

```
Overwriting wordcount.py
```

In [467]:
```
# Test the program on the small dataset
!python wordcount.py test.txt > 5_3_frequentWords
!head 5_3_frequentWords
```

```
Using configs in /Users/Alex/.mrjob.conf
ignoring partitioner keyword arg (requires real Hadoop): 'org.apache.ha
doop.mapred.lib.KeyFieldBasedPartitioner'
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/wordcount.Alex.20160619.010610.690478
Running step 1 of 2...
Running step 2 of 2...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/wordcount.Alex.20160619.010610.690478/output...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/wordcount.Alex.20160619.010610.690478...
42      critique
44      based
44      method
45      assessment
45      methodology
48      develop
51      further
51      her
51      lakota
51      look
```

Test the runner on the test data but on the cluster

In [457]:
```
# create the cluster
!mrjob create-cluster \
--max-hours-idle 1 \
--aws-region=us-west-1 -c ~/.mrjob.conf
```

```
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating persistent cluster to run several jobs in...
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/no_script.Alex.20160619.003604.699420
Copying local files to s3://mrjob-f8c316b67324528f/tmp/no_script.Alex.2
0160619.003604.699420/files/...
j-2C5FHH6GSAKSB
```

In [468]:
```
# run the program with the cluster we
# just spun up, still on the test data
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_3
!python wordcount.py -r emr s3://aks-w261-hw5/test.txt \
    --cluster-id=j-2C5FHH6GSAKSB \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_3 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_3/_SUCCESS
delete: s3://aks-w261-hw5/out_5_3/part-00009
delete: s3://aks-w261-hw5/out_5_3/part-00010
delete: s3://aks-w261-hw5/out_5_3/part-00000
delete: s3://aks-w261-hw5/out_5_3/part-00006
delete: s3://aks-w261-hw5/out_5_3/part-00002
delete: s3://aks-w261-hw5/out_5_3/part-00005
delete: s3://aks-w261-hw5/out_5_3/part-00003
delete: s3://aks-w261-hw5/out_5_3/part-00008
delete: s3://aks-w261-hw5/out_5_3/part-00001
delete: s3://aks-w261-hw5/out_5_3/part-00004
delete: s3://aks-w261-hw5/out_5_3/part-00007
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/wordcount.Alex.20160619.010639.049441
Copying local files to s3://mrjob-f8c316b67324528f/tmp/wordcount.Alex.2
0160619.010639.049441/files/...
Adding our job to existing cluster j-2C5FHH6GSAKSB
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Waiting for step 1 of 2 (s-2HKMIDN787KO6) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40531/cluster
  RUNNING for 27.9s
     5.0% complete
  RUNNING for 61.5s
    46.3% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-2HKMIDN787KO6 on ec
2-54-183-220-200.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-220-200.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-2HKMIDN787KO6/syslog
Counters: 54
        File Input Format Counters
                Bytes Read=45912
        File Output Format Counters
                Bytes Written=3957
        File System Counters
```

```
                FILE: Number of bytes read=3878
                FILE: Number of bytes written=3671830
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1872
                HDFS: Number of bytes written=3957
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=81
                HDFS: Number of write operations=22
                S3: Number of bytes read=45912
                S3: Number of bytes written=0
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=24
                Launched map tasks=24
                Launched reduce tasks=11
                Total megabyte-seconds taken by all map tasks=883091520
                Total megabyte-seconds taken by all reduce tasks=291297
600
                Total time spent by all map tasks (ms)=613258
                Total time spent by all maps in occupied slots (ms)=275
96610
                Total time spent by all reduce tasks (ms)=101145
                Total time spent by all reduces in occupied slots (ms)=
9103050
                Total vcore-seconds taken by all map tasks=613258
                Total vcore-seconds taken by all reduce tasks=101145
        Map-Reduce Framework
                CPU time spent (ms)=153870
                Combine input records=500
                Combine output records=347
                Failed Shuffles=0
                GC time elapsed (ms)=8007
                Input split bytes=1872
                Map input records=100
                Map output bytes=4879
                Map output materialized bytes=8308
                Map output records=500
                Merged Map outputs=264
                Physical memory (bytes) snapshot=14131720192
                Reduce input groups=214
                Reduce input records=347
                Reduce output records=214
                Reduce shuffle bytes=8308
                Shuffled Maps =264
                Spilled Records=694
                Total committed heap usage (bytes)=16125001728
                Virtual memory (bytes) snapshot=82820009984
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
```

```
                WRONG_REDUCE=0
Waiting for step 2 of 2 (s-3I1UOHMXF5VS0) to complete...
  RUNNING for 57.0s
    39.1% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-3I1UOHMXF5VS0 on ec
2-54-183-220-200.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-220-200.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-3I1UOHMXF5VS0/syslog
Counters: 55
        File Input Format Counters
                Bytes Read=6492
        File Output Format Counters
                Bytes Written=2594
        File System Counters
                FILE: Number of bytes read=2610
                FILE: Number of bytes written=3137349
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=10871
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=58
                HDFS: Number of write operations=0
                S3: Number of bytes read=0
                S3: Number of bytes written=2594
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=25
                Launched map tasks=29
                Launched reduce tasks=1
                Rack-local map tasks=4
                Total megabyte-seconds taken by all map tasks=927367200
                Total megabyte-seconds taken by all reduce tasks=542073
60
                Total time spent by all map tasks (ms)=644005
                Total time spent by all maps in occupied slots (ms)=289
80225
                Total time spent by all reduce tasks (ms)=18822
                Total time spent by all reduces in occupied slots (ms)=
1693980
                Total vcore-seconds taken by all map tasks=644005
                Total vcore-seconds taken by all reduce tasks=18822
        Map-Reduce Framework
                CPU time spent (ms)=31540
                Combine input records=0
                Combine output records=0
                Failed Shuffles=0
                GC time elapsed (ms)=12652
                Input split bytes=4379
                Map input records=214
                Map output bytes=4171
                Map output materialized bytes=4020
```

```
                Map output records=214
                Merged Map outputs=29
                Physical memory (bytes) snapshot=15663816704
                Reduce input groups=214
                Reduce input records=214
                Reduce output records=214
                Reduce shuffle bytes=4020
                Shuffled Maps =29
                Spilled Records=428
                Total committed heap usage (bytes)=16690708480
                Virtual memory (bytes) snapshot=60293427200
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/wordcount.Al
ex.20160619.010639.049441/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/wordcount.Alex.20160619.010639.049441...
Killing our SSH tunnel (pid 16409)
```

Run the word count on the entire corpus using AWS

Now that we've finished testing on a subset of the data, let's try it on the whole data set.

In [469]:
```
# run the program with the cluster we
# just spun up, still on the test data
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_3
!python wordcount.py -r emr s3://filtered-5grams/ \
    --cluster-id=j-2C5FHH6GSAKSB \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_3 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_3/_SUCCESS
delete: s3://aks-w261-hw5/out_5_3/part-00000
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/wordcount.Alex.20160619.011047.204054
Copying local files to s3://mrjob-f8c316b67324528f/tmp/wordcount.Alex.2
0160619.011047.204054/files/...
Adding our job to existing cluster j-2C5FHH6GSAKSB
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Waiting for step 1 of 2 (s-2CR3SF4BXE3GG) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40531/cluster
  RUNNING for 1.1s
   100.0% complete
  RUNNING for 33.4s
     5.0% complete
  RUNNING for 64.3s
     5.3% complete
  RUNNING for 96.1s
     8.1% complete
  RUNNING for 127.7s
     9.0% complete
  RUNNING for 158.8s
    11.1% complete
  RUNNING for 190.0s
    13.6% complete
  RUNNING for 221.4s
    14.6% complete
  RUNNING for 252.4s
    17.0% complete
  RUNNING for 284.5s
    19.2% complete
  RUNNING for 315.2s
    20.6% complete
  RUNNING for 346.4s
    23.0% complete
  RUNNING for 377.3s
    25.4% complete
```

```
  RUNNING for 408.1s
    27.3% complete
  RUNNING for 439.0s
    29.1% complete
  RUNNING for 470.4s
    30.5% complete
  RUNNING for 501.6s
    32.8% complete
  RUNNING for 532.8s
    34.2% complete
  RUNNING for 563.6s
    35.7% complete
  RUNNING for 595.4s
    37.5% complete
  RUNNING for 626.6s
    38.7% complete
  RUNNING for 659.3s
    40.5% complete
  RUNNING for 690.2s
    42.2% complete
  RUNNING for 721.9s
    43.7% complete
  RUNNING for 754.6s
    45.4% complete
  RUNNING for 785.4s
    47.2% complete
  RUNNING for 817.6s
    48.6% complete
  RUNNING for 849.6s
    50.3% complete
  RUNNING for 881.7s
    52.0% complete
  RUNNING for 912.9s
    53.3% complete
  RUNNING for 943.9s
    55.2% complete
  RUNNING for 975.7s
    63.3% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-2CR3SF4BXE3GG on ec
2-54-183-220-200.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-220-200.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-2CR3SF4BXE3GG/syslog
Counters: 56
        File Input Format Counters
                Bytes Read=2156069116
        File Output Format Counters
                Bytes Written=5467696
        File System Counters
                FILE: Number of bytes read=38202070
                FILE: Number of bytes written=142843203
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=23450
                HDFS: Number of bytes written=5467696
```

```
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=413
                HDFS: Number of write operations=22
                S3: Number of bytes read=2156069116
                S3: Number of bytes written=0
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=188
                Killed reduce tasks=1
                Launched map tasks=190
                Launched reduce tasks=12
                Other local map tasks=2
                Total megabyte-seconds taken by all map tasks=191887444
80
                Total megabyte-seconds taken by all reduce tasks=104455
78560
                Total time spent by all map tasks (ms)=13325517
                Total time spent by all maps in occupied slots (ms)=599
648265
                Total time spent by all reduce tasks (ms)=3626937
                Total time spent by all reduces in occupied slots (ms)=
326424330
                Total vcore-seconds taken by all map tasks=13325517
                Total vcore-seconds taken by all reduce tasks=3626937
        Map-Reduce Framework
                CPU time spent (ms)=5990990
                Combine input records=293411330
                Combine output records=6822745
                Failed Shuffles=0
                GC time elapsed (ms)=89585
                Input split bytes=23450
                Map input records=58682266
                Map output bytes=2843318430
                Map output materialized bytes=83609265
                Map output records=293411330
                Merged Map outputs=2090
                Physical memory (bytes) snapshot=128929230848
                Reduce input groups=269339
                Reduce input records=6822745
                Reduce output records=269339
                Reduce shuffle bytes=83609265
                Shuffled Maps =2090
                Spilled Records=13645490
                Total committed heap usage (bytes)=144772694016
                Virtual memory (bytes) snapshot=409824456704
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Waiting for step 2 of 2 (s-3GFHSPFPB67HD) to complete...
  RUNNING for 45.8s
     5.0% complete
```

```
  RUNNING for 76.4s
    54.1% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-3GFHSPFPB67HD on ec
2-54-183-220-200.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-220-200.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-3GFHSPFPB67HD/syslog
Counters: 55
        File Input Format Counters
                Bytes Read=6217412
        File Output Format Counters
                Bytes Written=3889400
        File System Counters
                FILE: Number of bytes read=3167675
                FILE: Number of bytes written=10151479
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=6222395
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=66
                HDFS: Number of write operations=0
                S3: Number of bytes read=0
                S3: Number of bytes written=3889400
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=31
                Launched map tasks=33
                Launched reduce tasks=1
                Rack-local map tasks=2
                Total megabyte-seconds taken by all map tasks=114830208
0
                Total megabyte-seconds taken by all reduce tasks=979660
80
                Total time spent by all map tasks (ms)=797432
                Total time spent by all maps in occupied slots (ms)=358
84440
                Total time spent by all reduce tasks (ms)=34016
                Total time spent by all reduces in occupied slots (ms)=
3061440
                Total vcore-seconds taken by all map tasks=797432
                Total vcore-seconds taken by all reduce tasks=34016
        Map-Reduce Framework
                CPU time spent (ms)=75760
                Combine input records=0
                Combine output records=0
                Failed Shuffles=0
                GC time elapsed (ms)=14627
                Input split bytes=4983
                Map input records=269339
                Map output bytes=5737035
                Map output materialized bytes=3435644
                Map output records=269339
```

```
                Merged Map outputs=33
                Physical memory (bytes) snapshot=17235947520
                Reduce input groups=269339
                Reduce input records=269339
                Reduce output records=269339
                Reduce shuffle bytes=3435644
                Shuffled Maps =33
                Spilled Records=538678
                Total committed heap usage (bytes)=18669371392
                Virtual memory (bytes) snapshot=68186292224
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/wordcount.Al
ex.20160619.011047.204054/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/wordcount.Alex.20160619.011047.204054...
Killing our SSH tunnel (pid 16434)
```

In [470]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_3_wordcount

# sync the files to a local directory
!aws s3 sync s3://aks-w261-hw5/out_5_3 /Users/Alex/Documents/Berkeley/16
02Summer/W261/HW5/5_3_wordcount
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_3_wordcount
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_3_wordcount/_SUCCESS
```

```
download: s3://aks-w261-hw5/out_5_3/_SUCCESS to 5_3_wordcount/_SUCCESS
download: s3://aks-w261-hw5/out_5_3/part-00000 to 5_3_wordcount/part-00
000
_SUCCESS   part-00000
```

Print out the top 10 words by word count

In [471]:
```
!head 5_3_wordcount/part-00000
```

```
5490815394      the
3698583299      of
2227866570      to
1421312776      in
1361123022      a
1149577477      and
802921147       that
758328796       is
688707130       be
492170314       as
```

20 Most/Least densely appearing words (count/pages_count) sorted in decreasing order of relative frequency

Create an MRJob class to calculate the density of words

```
%%writefile wordDensity.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
import numpy as np
import ast
from mrjob.protocol import RawProtocol

class wordDensity(MRJob):

    MRJob.SORT_VALUES = True
    INTERNAL_PROTOCOL = RawProtocol
    OUTPUT_PROTOCOL = RawProtocol

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):

        # set how we want to sort, we use a single reducer
        # only so that we can do a total sort, we use multiple
        # reducers for all other steps
        JOBCONF = {
            'mapred.output.key.comparator.class': 'org.apache.hadoop.map
red.lib.KeyFieldBasedComparator',
            'mapred.text.key.comparator.options': '-k1,1nr',
            'mapreduce.job.reduces': 1,
        }

        return [MRStep(mapper=self.mapper,\
                       combiner=self.combiner,\
                       reducer=self.reducer),\
               MRStep(jobconf = JOBCONF,\
                       mapper=None,\
                       reducer=self.reducer_final)]


    # our mapper reads in each line of data
    # splits each line into multiple words
    # and then returns the word with a count
    def mapper(self, _, line):

        # split the line by the tabs, set the
        # ngram and lower-case it
        line = line.split("\t")
        ngram = line[0].lower()

        # set the counts of interest
        count = int(line[1])
        page_count = int(line[2])
        counts = (count,page_count)

        # split the ngram by the spaces
        ngram = ngram.split()

        # loop through each word in the
        # ngram and yield the word and a
        # count of 1
```

```
        for word in ngram:
            yield word, str(counts)


    # our combiner sums the values for each
    # word
    def combiner(self, word, counts):

        # create an array to hold the counts
        # for each element and convert it to
        # a numpy array to allow summing
        totals = [0,0]
        totals = np.array(totals)

        # loop through each element in the
        # counts generator
        for _count in counts:

            # read in the count
            _count = ast.literal_eval(_count)
            count,page_count = _count

            # create an array to hold the
            # count and page count
            sub_total = [count,page_count]
            sub_total = np.array(sub_total)

            # add the values to our total
            # array
            totals = totals + sub_total

        # conver the counts list to a tuple
        counts = (totals[0],totals[1])

        # yield the counts for each word
        yield word, str(counts)


    # our reducer performs the final count
    # for all words
    def reducer(self, word, counts):

        # create an array to hold the counts
        # for each element and convert it to
        # a numpy array to allow summing
        totals = [0,0]
        totals = np.array(totals)

        # loop through each element in the
        # counts generator
        for _count in counts:

            # read in the count
            _count = ast.literal_eval(_count)
            count,page_count = _count

            # create an array to hold the
```

```
            # count and page count
            sub_total = [count,page_count]
            sub_total = np.array(sub_total)

            # add the values to our total
            # array
            totals = totals + sub_total

        # for each word divide the count
        # by the page_count to calculate
        # the density
        density = float(totals[0])/\
        float(totals[1])

        # we want to make sure that the count
        # string is at least 15 characters, if
        # its not, we add leading zeros. this
        # helps us sort
        extend = "0"

        # convert density to a string
        density = str(density)

        # while the count lenght is less than 15
        while len(density) < 15:
            density = density + extend

        # yield each word with its density
        yield str(density),word


    # our reducer final simply yields the inputs
    # from the indentity mapper and outputs a
    # completely sorted list
    def reducer_final(self, density, words):

        # loop through the list with the words
        for word in words:
            yield str(density), word


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

```
Overwriting wordDensity.py
```

Test on a small dataset

In [498]:
```
# Test the program on the small dataset
!python wordDensity.py test.txt > 5_3_density_prelim
!head 5_3_density_prelim
```

```
Using configs in /Users/Alex/.mrjob.conf
ignoring partitioner keyword arg (requires real Hadoop): 'org.apache.ha
doop.mapred.lib.KeyFieldBasedPartitioner'
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/wordDensity.Alex.20160619.024917.633110
Running step 1 of 2...
Running step 2 of 2...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/wordDensity.Alex.20160619.024917.633110/output...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/wordDensity.Alex.20160619.024917.633110...
1.0000000000000 aerial
1.0000000000000 america's
1.0000000000000 analysis
1.0000000000000 apology
1.0000000000000 approximation
1.0000000000000 are
1.0000000000000 arithmetic
1.0000000000000 aspiration
1.0000000000000 assessment
1.0000000000000 b
```

Test on the cloud with a small dataset

In [499]:
```
# run the program with the cluster we
# just spun up, still on the test data
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_3
!python wordDensity.py -r emr s3://aks-w261-hw5/test.txt \
    --cluster-id=j-2C5FHH6GSAKSB \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_3 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_3/_SUCCESS
delete: s3://aks-w261-hw5/out_5_3/part-00000
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/wordDensity.Alex.20160619.025039.146114
Copying local files to s3://mrjob-f8c316b67324528f/tmp/wordDensity.Ale
x.20160619.025039.146114/files/...
Adding our job to existing cluster j-2C5FHH6GSAKSB
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Waiting for step 1 of 2 (s-HAGLW978UQGO) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40531/cluster
  RUNNING for 19.5s
Unable to connect to resource manager
  RUNNING for 52.0s
  RUNNING for 83.3s
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-HAGLW978UQGO on ec2
-54-183-220-200.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-220-200.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-HAGLW978UQGO/syslog
Counters: 54
        File Input Format Counters
                Bytes Read=45912
        File Output Format Counters
                Bytes Written=5027
        File System Counters
                FILE: Number of bytes read=5547
                FILE: Number of bytes written=3676424
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1872
                HDFS: Number of bytes written=5027
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=81
                HDFS: Number of write operations=22
```

```
                S3: Number of bytes read=45912
                S3: Number of bytes written=0
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=24
                Launched map tasks=24
                Launched reduce tasks=11
                Total megabyte-seconds taken by all map tasks=927961920
                Total megabyte-seconds taken by all reduce tasks=304044
480
                Total time spent by all map tasks (ms)=644418
                Total time spent by all maps in occupied slots (ms)=289
98810
                Total time spent by all reduce tasks (ms)=105571
                Total time spent by all reduces in occupied slots (ms)=
9501390
                Total vcore-seconds taken by all map tasks=644418
                Total vcore-seconds taken by all reduce tasks=105571
        Map-Reduce Framework
                CPU time spent (ms)=157320
                Combine input records=500
                Combine output records=347
                Failed Shuffles=0
                GC time elapsed (ms)=7596
                Input split bytes=1872
                Map input records=100
                Map output bytes=8139
                Map output materialized bytes=10540
                Map output records=500
                Merged Map outputs=264
                Physical memory (bytes) snapshot=14180884480
                Reduce input groups=214
                Reduce input records=347
                Reduce output records=214
                Reduce shuffle bytes=10540
                Shuffled Maps =264
                Spilled Records=694
                Total committed heap usage (bytes)=16625172480
                Virtual memory (bytes) snapshot=82872950784
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Waiting for step 2 of 2 (s-GIEET9FFSCJ6) to complete...
  RUNNING for 74.4s
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-GIEET9FFSCJ6 on ec2
-54-183-220-200.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-220-200.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-GIEET9FFSCJ6/syslog
Counters: 55
```

```
        File Input Format Counters
                Bytes Read=8232
        File Output Format Counters
                Bytes Written=5241
        File System Counters
                FILE: Number of bytes read=2887
                FILE: Number of bytes written=3138665
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=12669
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=58
                HDFS: Number of write operations=0
                S3: Number of bytes read=0
                S3: Number of bytes written=5241
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=25
                Launched map tasks=29
                Launched reduce tasks=1
                Rack-local map tasks=4
                Total megabyte-seconds taken by all map tasks=895520160
                Total megabyte-seconds taken by all reduce tasks=529171
20
                Total time spent by all map tasks (ms)=621889
                Total time spent by all maps in occupied slots (ms)=279
85005
                Total time spent by all reduce tasks (ms)=18374
                Total time spent by all reduces in occupied slots (ms)=
1653660
                Total vcore-seconds taken by all map tasks=621889
                Total vcore-seconds taken by all reduce tasks=18374
        Map-Reduce Framework
                CPU time spent (ms)=29150
                Combine input records=0
                Combine output records=0
                Failed Shuffles=0
                GC time elapsed (ms)=10036
                Input split bytes=4437
                Map input records=214
                Map output bytes=5241
                Map output materialized bytes=4557
                Map output records=214
                Merged Map outputs=29
                Physical memory (bytes) snapshot=15172284416
                Reduce input groups=214
                Reduce input records=214
                Reduce output records=214
                Reduce shuffle bytes=4557
                Shuffled Maps =29
                Spilled Records=428
                Total committed heap usage (bytes)=16104030208
                Virtual memory (bytes) snapshot=60257116160
```

```
                Shuffle Errors
                        BAD_ID=0
                        CONNECTION=0
                        IO_ERROR=0
                        WRONG_LENGTH=0
                        WRONG_MAP=0
                        WRONG_REDUCE=0
        Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/wordDensity.
        Alex.20160619.025039.146114/...
        Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
        T/wordDensity.Alex.20160619.025039.146114...
        Killing our SSH tunnel (pid 16729)
```

Run the word count on the entire corpus using AWS

Now that we've finished testing on a subset of the data, let's try it on the whole data set.

In [500]:
```
# run the program with the cluster we
# just spun up, still on the test data
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_3
!python wordDensity.py -r emr s3://filtered-5grams/ \
    --cluster-id=j-2C5FHH6GSAKSB \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_3 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_3/_SUCCESS
delete: s3://aks-w261-hw5/out_5_3/part-00000
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/wordDensity.Alex.20160619.025656.535057
Copying local files to s3://mrjob-f8c316b67324528f/tmp/wordDensity.Ale
x.20160619.025656.535057/files/...
Adding our job to existing cluster j-2C5FHH6GSAKSB
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Waiting for step 1 of 2 (s-23RT1DKUT3Z8P) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40531/cluster
  RUNNING for 9.3s
   100.0% complete
  RUNNING for 41.7s
     5.0% complete
  RUNNING for 73.0s
     6.1% complete
  RUNNING for 104.4s
     8.3% complete
  RUNNING for 136.0s
     8.5% complete
  RUNNING for 167.4s
     8.5% complete
  RUNNING for 198.5s
     8.5% complete
  RUNNING for 230.3s
     8.9% complete
  RUNNING for 261.2s
     9.5% complete
  RUNNING for 293.3s
    11.1% complete
  RUNNING for 324.4s
    11.9% complete
  RUNNING for 356.2s
    13.5% complete
  RUNNING for 387.4s
    13.5% complete
```

```
RUNNING for 418.2s
  14.0% complete
RUNNING for 450.3s
  14.6% complete
RUNNING for 481.6s
  15.4% complete
RUNNING for 512.8s
  15.8% complete
RUNNING for 544.4s
  17.1% complete
RUNNING for 575.9s
  17.9% complete
RUNNING for 608.3s
  19.0% complete
RUNNING for 640.0s
  19.6% complete
RUNNING for 671.1s
  20.4% complete
RUNNING for 702.9s
  21.3% complete
RUNNING for 734.5s
  22.1% complete
RUNNING for 766.4s
  22.6% complete
RUNNING for 797.6s
  23.8% complete
RUNNING for 828.8s
  25.0% complete
RUNNING for 860.9s
  25.6% complete
RUNNING for 892.2s
  25.6% complete
RUNNING for 924.1s
  26.9% complete
RUNNING for 955.7s
  28.2% complete
RUNNING for 987.2s
  29.1% complete
RUNNING for 1018.5s
  29.8% complete
RUNNING for 1050.4s
  30.7% complete
RUNNING for 1082.1s
  32.3% complete
RUNNING for 1113.6s
  32.5% complete
RUNNING for 1144.9s
  32.8% complete
RUNNING for 1176.9s
  34.0% complete
RUNNING for 1208.0s
  35.4% complete
RUNNING for 1239.8s
  35.9% complete
RUNNING for 1271.2s
  36.4% complete
RUNNING for 1302.9s
```

```
  36.4% complete
RUNNING for 1334.4s
  37.1% complete
RUNNING for 1366.3s
  38.7% complete
RUNNING for 1397.5s
  39.6% complete
RUNNING for 1428.9s
  39.6% complete
RUNNING for 1460.0s
  40.7% complete
RUNNING for 1491.2s
  41.3% complete
RUNNING for 1522.4s
  41.3% complete
RUNNING for 1553.9s
  42.3% complete
RUNNING for 1585.2s
  43.9% complete
RUNNING for 1616.7s
  44.5% complete
RUNNING for 1648.4s
  44.8% complete
RUNNING for 1681.9s
  45.1% complete
RUNNING for 1713.4s
  45.7% complete
RUNNING for 1744.6s
  47.2% complete
RUNNING for 1775.8s
  48.3% complete
RUNNING for 1807.3s
  48.4% complete
RUNNING for 1838.6s
  49.0% complete
RUNNING for 1870.5s
  49.9% complete
RUNNING for 1901.8s
  50.3% complete
RUNNING for 1933.3s
  51.1% complete
RUNNING for 1965.1s
  52.6% complete
RUNNING for 1996.6s
  53.1% complete
RUNNING for 2027.6s
  53.4% complete
RUNNING for 2059.0s
  53.8% complete
RUNNING for 2090.4s
  54.4% complete
RUNNING for 2121.2s
  58.8% complete
RUNNING for 2152.5s
  60.4% complete
RUNNING for 2183.7s
  60.5% complete
```

```
  RUNNING for 2214.8s
    82.1% complete
  RUNNING for 2246.3s
   100.0% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-23RT1DKUT3Z8P on ec
2-54-183-220-200.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-220-200.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-23RT1DKUT3Z8P/syslog.2016-06-19-02
  Parsing step log: ssh://ec2-54-183-220-200.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-23RT1DKUT3Z8P/syslog
Counters: 56
        File Input Format Counters
                Bytes Read=2156069116
        File Output Format Counters
                Bytes Written=6814391
        File System Counters
                FILE: Number of bytes read=61032463
                FILE: Number of bytes written=192933161
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=23450
                HDFS: Number of bytes written=6814391
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=413
                HDFS: Number of write operations=22
                S3: Number of bytes read=2156069116
                S3: Number of bytes written=0
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=188
                Killed reduce tasks=1
                Launched map tasks=190
                Launched reduce tasks=12
                Other local map tasks=2
                Total megabyte-seconds taken by all map tasks=453630441
60
                Total megabyte-seconds taken by all reduce tasks=236587
59360
                Total time spent by all map tasks (ms)=31502114
                Total time spent by all maps in occupied slots (ms)=141
7595130
                Total time spent by all reduce tasks (ms)=8214847
                Total time spent by all reduces in occupied slots (ms)=
739336230
                Total vcore-seconds taken by all map tasks=31502114
                Total vcore-seconds taken by all reduce tasks=8214847
        Map-Reduce Framework
                CPU time spent (ms)=18484720
                Combine input records=293411330
                Combine output records=6822745
                Failed Shuffles=0
                GC time elapsed (ms)=94201
```

```
                Input split bytes=23450
                Map input records=58682266
                Map output bytes=4703685170
                Map output materialized bytes=110864887
                Map output records=293411330
                Merged Map outputs=2090
                Physical memory (bytes) snapshot=138155126784
                Reduce input groups=269339
                Reduce input records=6822745
                Reduce output records=269339
                Reduce shuffle bytes=110864887
                Shuffled Maps =2090
                Spilled Records=13645490
                Total committed heap usage (bytes)=143625027584
                Virtual memory (bytes) snapshot=409769627648
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Waiting for step 2 of 2 (s-1M1VY9KIS6BNS) to complete...
  RUNNING for 74.3s
    80.0% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-1M1VY9KIS6BNS on ec
2-54-183-220-200.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-220-200.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-1M1VY9KIS6BNS/syslog
Counters: 56
        File Input Format Counters
                Bytes Read=7773723
        File Output Format Counters
                Bytes Written=7083730
        File System Counters
                FILE: Number of bytes read=3398114
                FILE: Number of bytes written=10677793
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=7778772
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=66
                HDFS: Number of write operations=0
                S3: Number of bytes read=0
                S3: Number of bytes written=7083730
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=32
                Killed map tasks=1
                Launched map tasks=34
                Launched reduce tasks=1
```

```
                Rack-local map tasks=2
                Total megabyte-seconds taken by all map tasks=112540752
0
                Total megabyte-seconds taken by all reduce tasks=676944
00
                Total time spent by all map tasks (ms)=781533
                Total time spent by all maps in occupied slots (ms)=351
68985
                Total time spent by all reduce tasks (ms)=23505
                Total time spent by all reduces in occupied slots (ms)=
2115450
                Total vcore-seconds taken by all map tasks=781533
                Total vcore-seconds taken by all reduce tasks=23505
        Map-Reduce Framework
                CPU time spent (ms)=73680
                Combine input records=0
                Combine output records=0
                Failed Shuffles=0
                GC time elapsed (ms)=13042
                Input split bytes=5049
                Map input records=269339
                Map output bytes=7083730
                Map output materialized bytes=3730948
                Map output records=269339
                Merged Map outputs=33
                Physical memory (bytes) snapshot=17034039296
                Reduce input groups=269339
                Reduce input records=269339
                Reduce output records=269339
                Reduce shuffle bytes=3730948
                Shuffled Maps =33
                Spilled Records=538678
                Total committed heap usage (bytes)=18386780160
                Virtual memory (bytes) snapshot=68171264000
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/wordDensity.
Alex.20160619.025656.535057/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/wordDensity.Alex.20160619.025656.535057...
Killing our SSH tunnel (pid 16757)
```

In [501]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_3_worddensit
y

# sync the files to a local directory
!aws s3 sync s3://aks-w261-hw5/out_5_3 /Users/Alex/Documents/Berkeley/16
02Summer/W261/HW5/5_3_worddensity
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_3_worddensity
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_3_worddensity/_SUCCESS
```

```
mkdir: /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_3_worddensi
ty: File exists
download: s3://aks-w261-hw5/out_5_3/_SUCCESS to 5_3_worddensity/_SUCCES
S
download: s3://aks-w261-hw5/out_5_3/part-00000 to 5_3_worddensity/part-
00000
_SUCCESS   part-00000
```

Top 20 and bottom 20 words by density

Print out the top and bottom 20 words by density

In [502]:
```
print "Top 20 words by density\n"
!head -20 5_3_worddensity/part-00000
print "\n"
print "Botton 20 words by density\n"
!tail -20 5_3_worddensity/part-00000
```

```
Top 20 words by density

11.557291666700 xxxx
8.0741599073000 blah
7.5333333333300 nnn
6.2017491314200 na
4.9218750000000 oooooooooooooo
4.8543057272400 nd
4.5116279069800 llll
4.1696500133600 oooooo
3.8586371934700 ooooo
3.7624521072800 lillelu
3.5769230769200 pfeffermann
3.5769230769200 madarassy
3.5600000000000 meteoritical
3.5000000000000 xxxxxxxx
3.2290388548100 beep
3.1886792452800 latha
2.9191176470600 iyengar
2.8250000000000 counterfeiteth
2.8198198198200 nonsquamous
2.8198198198200 nonmorular


Botton 20 words by density

1.0000000000000 abdulmejid
1.0000000000000 abdominales
1.0000000000000 abdolola
1.0000000000000 abderitish
1.0000000000000 abcs
1.0000000000000 abbott's
1.0000000000000 abbiamo
1.0000000000000 abbes
1.0000000000000 abbaside
1.0000000000000 abbanysh
1.0000000000000 abbagliato
1.0000000000000 abasheth
1.0000000000000 abases
1.0000000000000 abare
1.0000000000000 interkinetic
1.0000000000000 aaya
1.0000000000000 aaws
1.0000000000000 gladded
1.0000000000000 glacken's
1.0000000000000 concertino
```

Distribution of 5-gram sizes (character length)

In [503]:
```
%%writefile distSize.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep

class distSize(MRJob):

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper=self.mapper,\
                       combiner=self.combiner,\
                       reducer=self.reducer)]


    # our mapper reads in each line of data
    # splits each line, calculates the length
    # of the line and returns the length with
    # a count of 1
    def mapper(self, _, line):

        # split the line by the tabs, set the
        # ngram and lower-case it
        line = line.split("\t")
        ngram = line[0].lower()
        count = int(line[1])

        # grab the ngram length
        length = len(ngram)

        # yield the length and the count
        yield length,count


    # our combiner sums the counts for each
    # length
    def combiner(self, length, counts):

        # yield the counts for each ngram
        yield length, sum(counts)


    # our reducer performs the final count
    # for all words
    def reducer(self, length, counts):

        # yield the final lengths for
        # each ngram
        yield length, sum(counts)

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

```
Overwriting distSize.py
```

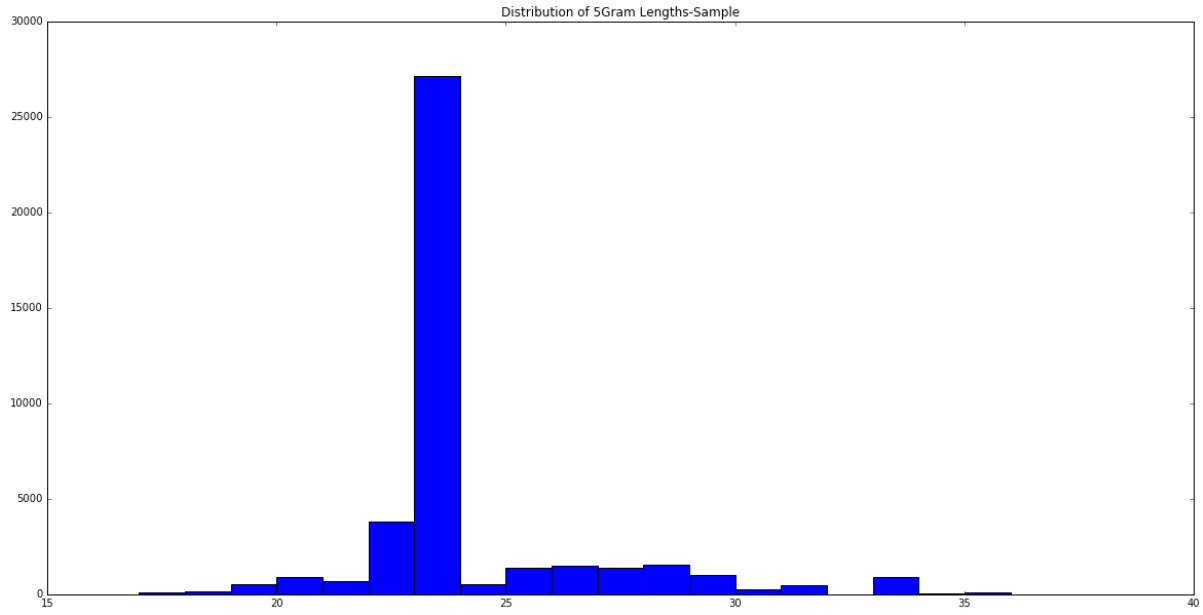
Create a runner to test our MRJob on a sample data

In [504]:
```
%reload_ext autoreload
%autoreload 2

# import the MRJob that we created
from distSize import distSize

# set the data that we're going to pull
mr_job = distSize(args=['test.txt'])

# create the runner and run it
with mr_job.make_runner() as runner:
    runner.run()

    # create a file to store the output of this
    # step
    with open('length_dist','w') as myfile:

        # stream_output: get access to the output
        for line in runner.stream_output():

            # grab the key,value
            length,count =  \
            mr_job.parse_output_line(line)

            # set the information we want to
            # write
            info = str(length)+"\t"+str(count)+"\n"

            # write the word,density to an
            # preliminary output file
            myfile.write(info)

# preview the top of the preliminary file
# that we created
print "Preview of length distribution file:"
!head length_dist
```

```
Preview of length distribution file:
17      114
18      146
19      528
20      888
21      678
22      3804
23      27144
24      518
25      1404
26      1489
```

Plot a histogram of the length distribution for just a sample of the data

In [39]:
```
# tell matplotlib not to open a new window
%matplotlib inline

# import pandas and matplotlib to load
# the data and plot the histogram
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# read in the data as a data frame
data = pd.read_table('length_dist',header=None)
data_count = data.iloc[:,1]
data_lengt = data.iloc[:,0]

# gives histogram bars a width
width = 1.0

# plot the histogram
plt.figure(figsize=(20,10))
plt.bar(data_lengt,data_count, width, color='b')
plt.title("Distribution of 5Gram Lengths-Sample")
plt.show()
```



Run the program on the complete data

Now that we've unit tested, let's move on to actually running the analysis on the complete data.

In [506]:
```
# run the program with the cluster we
# just spun up, still on the test data
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_3
!python distSize.py -r emr s3://filtered-5grams/ \
    --cluster-id=j-2C5FHH6GSAKSB \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_3 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_3/_SUCCESS
delete: s3://aks-w261-hw5/out_5_3/part-00000
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/distSize.Alex.20160619.034106.675317
Copying local files to s3://mrjob-f8c316b67324528f/tmp/distSize.Alex.20
160619.034106.675317/files/...
Adding our job to existing cluster j-2C5FHH6GSAKSB
Waiting for step 1 of 1 (s-YGA8ZOFVHT7K) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40531/cluster
  RUNNING for 16.1s
   100.0% complete
  RUNNING for 48.8s
     5.0% complete
  RUNNING for 79.6s
     7.8% complete
  RUNNING for 111.5s
    10.4% complete
  RUNNING for 143.2s
    13.7% complete
  RUNNING for 174.9s
    16.2% complete
  RUNNING for 206.2s
    19.4% complete
  RUNNING for 237.9s
    22.7% complete
  RUNNING for 270.0s
    25.8% complete
  RUNNING for 301.9s
    29.2% complete
  RUNNING for 333.1s
    32.6% complete
  RUNNING for 364.5s
    34.8% complete
  RUNNING for 395.9s
    38.1% complete
  RUNNING for 429.4s
    40.7% complete
  RUNNING for 460.9s
    43.2% complete
  RUNNING for 492.5s
    46.2% complete
  RUNNING for 523.4s
    48.8% complete
  RUNNING for 555.2s
    51.9% complete
  RUNNING for 586.8s
    54.3% complete
  RUNNING for 619.5s
    77.2% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-YGA8ZOFVHT7K on ec2
-54-183-220-200.us-west-1.compute.amazonaws.com...
```

```
  Parsing step log: ssh://ec2-54-183-220-200.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-YGA8ZOFVHT7K/syslog
Counters: 55
        File Input Format Counters
                Bytes Read=2156069116
        File Output Format Counters
                Bytes Written=780
        File System Counters
                FILE: Number of bytes read=56725
                FILE: Number of bytes written=21113283
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=23450
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=190
                HDFS: Number of write operations=0
                S3: Number of bytes read=2156069116
                S3: Number of bytes written=780
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=188
                Launched map tasks=190
                Launched reduce tasks=11
                Other local map tasks=2
                Total megabyte-seconds taken by all map tasks=118539043
20
                Total megabyte-seconds taken by all reduce tasks=622731
4560
                Total time spent by all map tasks (ms)=8231878
                Total time spent by all maps in occupied slots (ms)=370
434510
                Total time spent by all reduce tasks (ms)=2162262
                Total time spent by all reduces in occupied slots (ms)=
194603580
                Total vcore-seconds taken by all map tasks=8231878
                Total vcore-seconds taken by all reduce tasks=2162262
        Map-Reduce Framework
                CPU time spent (ms)=2545500
                Combine input records=58682266
                Combine output records=9172
                Failed Shuffles=0
                GC time elapsed (ms)=55284
                Input split bytes=23450
                Map input records=58682266
                Map output bytes=372255868
                Map output materialized bytes=136284
                Map output records=58682266
                Merged Map outputs=2090
                Physical memory (bytes) snapshot=101322788864
                Reduce input groups=80
                Reduce input records=9172
                Reduce output records=80
                Reduce shuffle bytes=136284
```

```
                Shuffled Maps =2090
                Spilled Records=18344
                Total committed heap usage (bytes)=123371782144
                Virtual memory (bytes) snapshot=409652019200
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/distSize.Ale
x.20160619.034106.675317/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/distSize.Alex.20160619.034106.675317...
Killing our SSH tunnel (pid 16867)
```

In [507]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_3_distributi
on

# sync the files to a local directory
!aws s3 sync s3://aks-w261-hw5/out_5_3 /Users/Alex/Documents/Berkeley/16
02Summer/W261/HW5/5_3_distribution
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_3_distribution
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_3_distribution/_SUCCESS
```

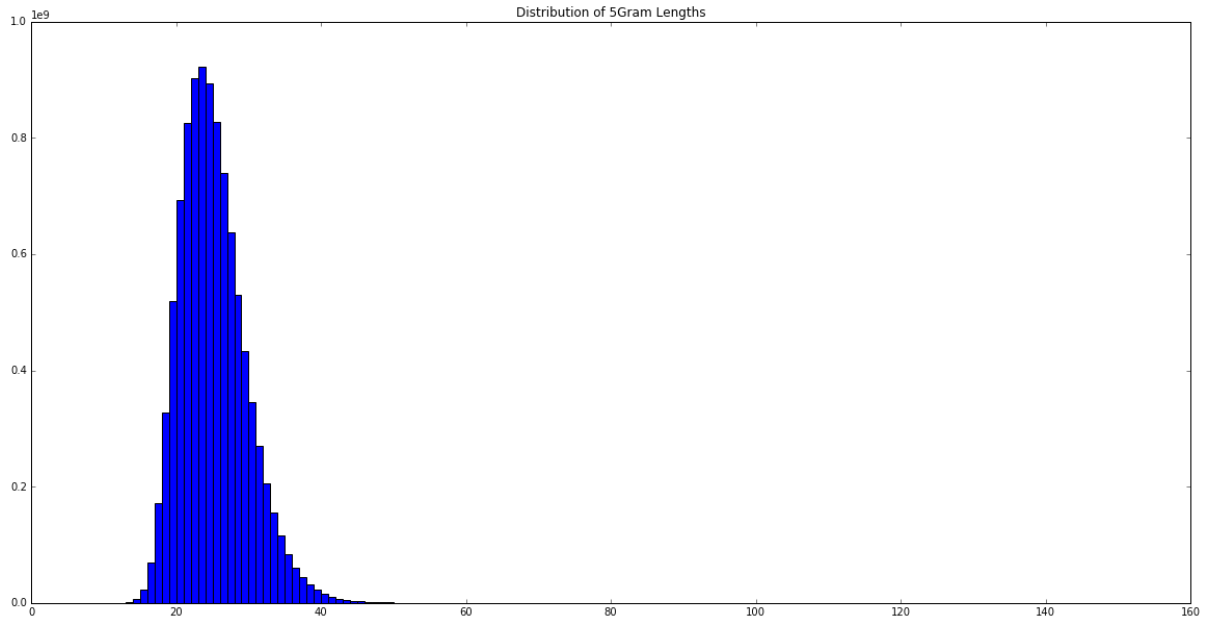
```
download: s3://aks-w261-hw5/out_5_3/_SUCCESS to 5_3_distribution/_SUCCE
SS
download: s3://aks-w261-hw5/out_5_3/part-00000 to 5_3_distribution/part
-00000
download: s3://aks-w261-hw5/out_5_3/part-00004 to 5_3_distribution/part
-00004
download: s3://aks-w261-hw5/out_5_3/part-00002 to 5_3_distribution/part
-00002
download: s3://aks-w261-hw5/out_5_3/part-00005 to 5_3_distribution/part
-00005
download: s3://aks-w261-hw5/out_5_3/part-00006 to 5_3_distribution/part
-00006
download: s3://aks-w261-hw5/out_5_3/part-00001 to 5_3_distribution/part
-00001
download: s3://aks-w261-hw5/out_5_3/part-00008 to 5_3_distribution/part
-00008
download: s3://aks-w261-hw5/out_5_3/part-00003 to 5_3_distribution/part
-00003
download: s3://aks-w261-hw5/out_5_3/part-00007 to 5_3_distribution/part
-00007
download: s3://aks-w261-hw5/out_5_3/part-00009 to 5_3_distribution/part
-00009
download: s3://aks-w261-hw5/out_5_3/part-00010 to 5_3_distribution/part
-00010
_SUCCESS   part-00001 part-00003 part-00005 part-00007 part-00009
part-00000 part-00002 part-00004 part-00006 part-00008 part-00010
```

In [508]:
```
# import the os and pandas libraries
# to help us get the files and read them in
import os
import pandas as pd

# loop through our files and add each the longest
# ngrams to the dictionary, where the key is the
# length and the value is the ngrams
indir = '/Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_3_distribu
tion/'
for root, dirs, filenames in os.walk(indir):

    # create a file that stores all the distriubtions
    with open('5_3_length_distr','w') as myfile:

        # loop through each file
        for filename in filenames:

            # set the filename
            filename = indir+filename

            # write every line in the file to
            # our new file
            with open(filename, 'r') as myoldfile:
                for line in myoldfile.readlines():

                    # split the line and set the values
                    line = line.split()
                    length = int(line[0])
                    count = int(line[1])

                    # set the string we're going to write
                    info = str(length) + "\t" + str(count) + "\n"

                    myfile.write(info)

# preview the file we've created
!head 5_3_length_distr
```

```
103     91
12      114018
24      894422079
36      60137979
48      802590
61      4984
73      182
9       14140
128     92
13      888866
```

Plot the histogram for the complete data

In [49]:
```
# tell matplotlib not to open a new window
%matplotlib inline

# import pandas and matplotlib to load
# the data and plot the histogram
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# read in the data
data = pd.read_table('5_3_length_distr',header=None)

# read in the data as a data frame
data_count = data.iloc[:,1]
data_lengt = data.iloc[:,0]

# gives histogram bars a width
width = 1.0

# plot the histogram
plt.figure(figsize=(20,10))
plt.bar(data_lengt,data_count,width,color='b')
plt.title("Distribution of 5Gram Lengths")
plt.show()
```



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

Generate the systems testing data set

This is borrowed and slightly modified from Annie's notebook posted to the GoogleGroup.

```
DOC         WORDS-STRIPE (count of occurrences of words M,N,X,Y,Z in documen
ts A,B,C)

A           {X:20, Y: 30, Z:5}
B           {X:100, Y:20 }
C           {M:5, N:20, Z:5, Y:1}
```

In [197]:
```
# generate our system_test file
!echo \
"A\t{'X':20, 'Y':30, 'Z':5}\n"\
"B\t{'X':100, 'Y':20}\n"\
"C\t{'M':5, 'N':20, 'Z':5, 'Y':1}" > systems_test.txt
print "systems_test.txt file generated"
!cat systems_test.txt
```

```
systems_test.txt file generated
A       {'X':20, 'Y':30, 'Z':5}
B       {'X':100, 'Y':20}
C       {'M':5, 'N':20, 'Z':5, 'Y':1}
```

Binarize the stripes so that we can run cosine similarity and jaccard similarity metrics on the data.

In [198]:
```
# import abstract synthax trees to help
# read in the data
import ast

# Create a method that binarizes the stripes.
def binarizeStripes():

    # Create an output file to write binary stripes to.
    with open ('binary_stripes.txt', 'w') as output_file:

        # Read each line of stripes file.
        for line in open('systems_test.txt', 'r'):
            line = line.strip()
            contents = line.split("\t")
            word = contents[0]

            # Read in stripe as a dictionary
            stripe = dict(ast.literal_eval(contents[1]))

            # Iterate through each value in
            # stripes dictionary.
            for key in stripe:

                # Set all values in stripe to
                # binary value (aka 1).
                stripe[key] = 1

            # write to the output file each words
            # binarized document appearance
            output_file.write("\t".join([word, str(stripe)]) + "\n")
```

In [199]:
```
binarizeStripes()
!cat binary_stripes.txt
```

```
A       {'Y': 1, 'X': 1, 'Z': 1}
B       {'Y': 1, 'X': 1}
C       {'Y': 1, 'Z': 1, 'M': 1, 'N': 1}
```

Normalize the stripes

The normalization is the inverse of the square root of the sum of squares of the items in the vector. In our case, each item is equal to 1. So, we end up with the square root of the length.

In [201]:
```
import ast
import math

# Create a method that normalizes the binary stripes.
def NormalizeStripes():

    # Create an output file to write normalized
    # stripes to.
    with open('normalized_stripes.txt', 'w') \
    as output_file:

        # Read each line of binarized stripes file.
        for line in open('binary_stripes.txt', 'r'):
            line = line.strip()
            contents = line.split("\t")
            word = contents[0]

            # Read in stripe as a dictionary
            stripe = dict(ast.literal_eval(contents[1]))

            # Get length of stripe to use for normalization.
            stripeLen = len(stripe)

            # Iterate through each value in stripes dictionary.
            for key in stripe:

                # Normalize each value by dividing by
                # sqrt(stripeLength).
                stripe[key] = float(1)/math.sqrt(stripeLen)

            # write to the output file
            output_file.write("\t".join([word, str(stripe)]) + "\n")
```

In [202]:
```
NormalizeStripes()
!cat normalized_stripes.txt
```

```
A       {'Y': 0.5773502691896258, 'X': 0.5773502691896258, 'Z': 0.57735
02691896258}
B       {'Y': 0.7071067811865475, 'X': 0.7071067811865475}
C       {'Y': 0.5, 'Z': 0.5, 'M': 0.5, 'N': 0.5}
```

Create the inverted index

The inverted index is organized by word and shows what document each word appears on.

```
WORD              DOCUMENT LIST (POSTINGS LIST)

M                 {C: 1/2}
N                 {C: 1/2}
X                 {A:1/sqrt(3), B: 1/sqrt(2)}
Y                 {A:1/sqrt(3), B:1/sqrt(2), C: 1/2}
Z                 {A:1/sqrt(3), C:1/2}
```

In [203]:
```
import ast
import math

# Create a method that computes the inverted index.
def CreateInvertedIndex():

    # Create an output file to write inverted index to.
    with open('inverted_index.txt', 'w') as outputFile:

        # In-memory inverted index dictionary.
        invertedIndex = {}

        # Read each line of normalized stripes file.
        for line in open('normalized_stripes.txt', 'r'):
            line = line.strip()
            contents = line.split("\t")
            word = contents[0]

            # Read in stripe as a dictionary
            stripe = dict(ast.literal_eval(contents[1]))

            # Iterate through each value in stripes dictionary and inver
t.
            for key,value in stripe.items():

                # Create inverted index for each value in stripe.
                if key in invertedIndex:
                    invertedIndex[key][word] = value
                else:
                    invertedIndex[key] = {word: value}

        # write to the output file
        for key,value in invertedIndex.iteritems():
            outputFile.write("\t".join([key, str(value)]) + "\n")
```

In [204]:
```
CreateInvertedIndex()
!cat inverted_index.txt
```

```
Y       {'A': 0.5773502691896258, 'C': 0.5, 'B': 0.7071067811865475}
X       {'A': 0.5773502691896258, 'B': 0.7071067811865475}
Z       {'A': 0.5773502691896258, 'C': 0.5}
M       {'C': 0.5}
N       {'C': 0.5}
```

Calculate the cosine similarity

To calculate the cosine similarity of two stripe documents, say A and C, compute partial similarities using dot products and aggregating partial similarities to get overall cosine similarity.

cosine(stripe A, stripe C) = A[Y]*C[Y]* + *A[Z]* C[Z]

Note: A[M] = A[N] = 0; C[X] = 0, hence M,N,X do not contribute to the cosine similarity.

```
Cosine(A,B) = (1/sqrt(3))*(1/sqrt(2)) + (1/sqrt(3))*(1/sqrt(2)) + (1/sqrt
(3))*0 = 2/sqrt(6)
Cosine(A,C) = 0*(1/sqrt(2)) + 0*(1/sqrt(2)) + (1/sqrt(3))*0 + (1/sqrt(3))*1/
2 + (1/sqrt(3))*1/2 = 1/sqrt(3)
Cosine(B,C) = 0*(1/sqrt(2)) + 0*(1/sqrt(2)) + (1/sqrt(2))*0 + (1/sqrt(2))*1/
2 + 0*1/2 = 1/2sqrt(2)
```

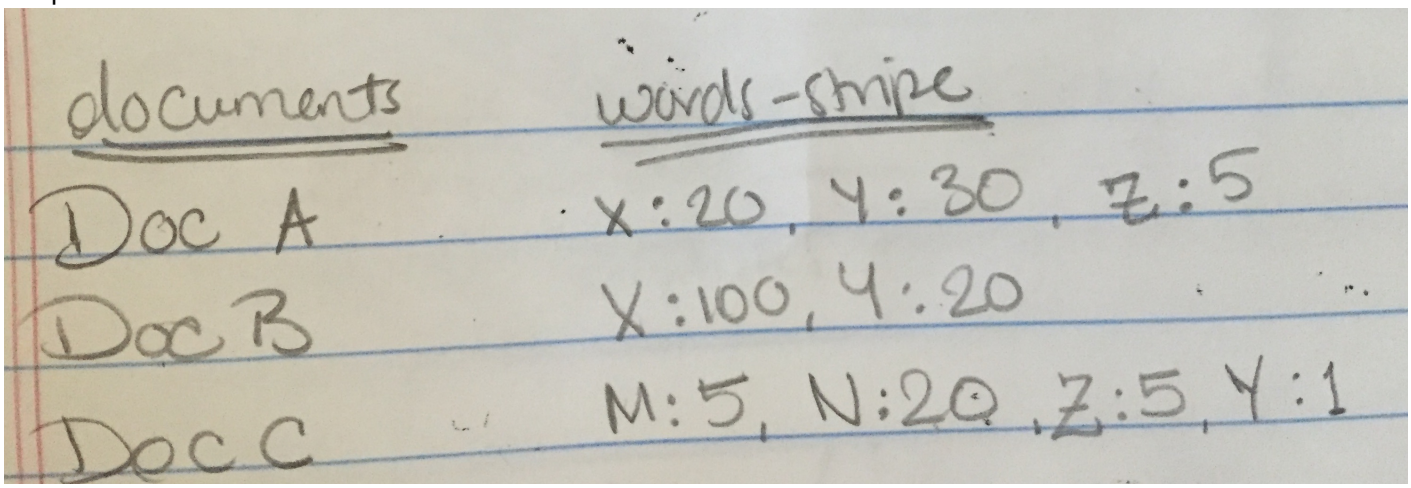
In [205]:
```
import ast
import math

# Create a method that computes Cosine similarity from inverted index.
def ComputeCosineSim():

    # Create an output file to write cosine similarities to.
    with open('cosine_sim.txt', 'w') as outputFile:

        # In-memory inverted index dictionary.
        cosineSim = {}

        # Read each line of inverted index file.
        for line in open('inverted_index.txt', 'r'):
            line = line.strip()
            contents = line.split("\t")
            word = contents[0]

            # Read in stripe as a dictionary
            stripe = dict(ast.literal_eval(contents[1]))

            # Iterate through each pair in stripes to compute dot produc
t.
            for key1,value1 in stripe.items():
                for key2,value2 in stripe.items():
                    if key1 != key2:
                        if (key1, key2) not in cosineSim:
                            cosineSim[(key1, key2)] = value1*value2
                        else:
                            cosineSim[(key1, key2)] += value1*value2

        # write the cosine similarities to the output file
        for key,value in cosineSim.iteritems():
            outputFile.write("\t".join([str(key), str(value)]) + "\n")
```

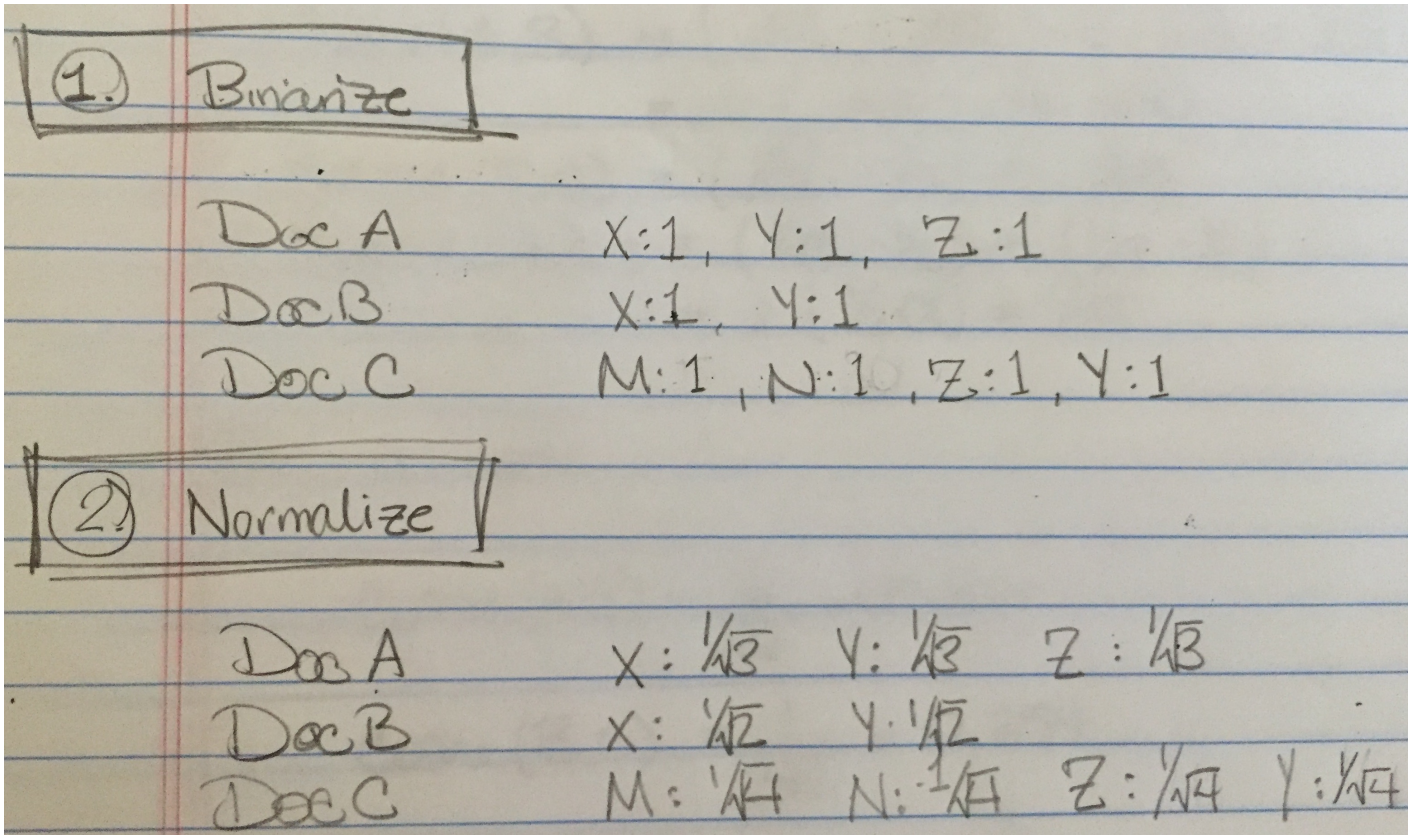
In [206]:
```
ComputeCosineSim()
!cat cosine_sim.txt
```

```
('B', 'C')      0.353553390593
('B', 'A')      0.816496580928
('C', 'A')      0.57735026919
('C', 'B')      0.353553390593
('A', 'B')      0.816496580928
('A', 'C')      0.57735026919
```

Hand calculations

To make sur that we are on the same page as our program, we also calculate this by one, on paper. We upload the pictures here:





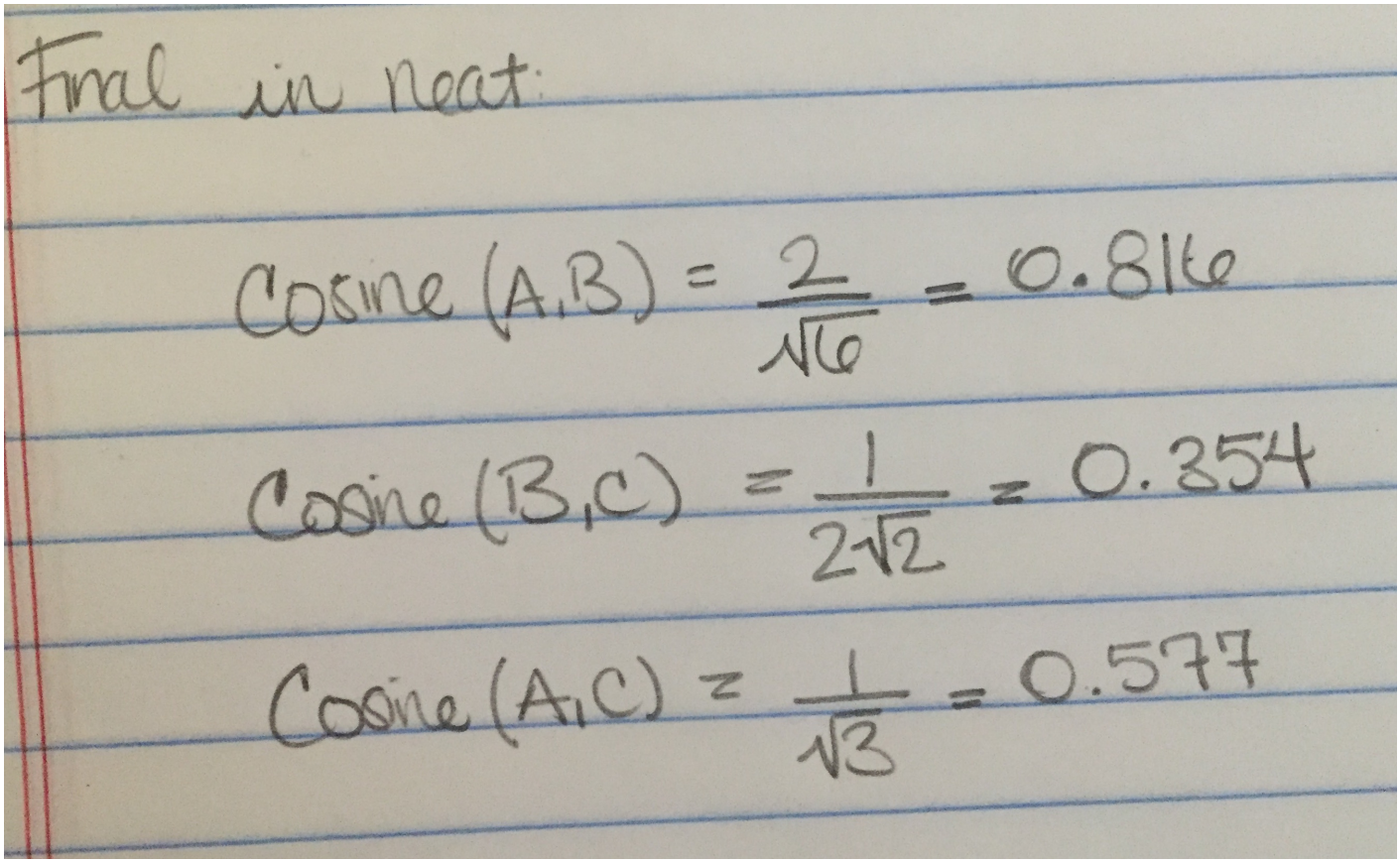
(3.) Invert the Index

X	A: $1/\sqrt{3}$	B: $1/\sqrt{2}$	
Y	A: $1/\sqrt{3}$	B: $1/\sqrt{2}$	C: $1/2$
Z	A: $1/\sqrt{3}$	C: $1/2$	
M	C: $1/2$		
N	C: $1/2$		

(4.) Calculate Cosine Similarity

↳ this simplifies to the dot product of the vector pairs

$$\text{Cosine}(A,B) = (1/\sqrt{3} \cdot 1/\sqrt{2}) + (1/\sqrt{3} + 1/\sqrt{2})$$
$$= 2(1/\sqrt{6}) = 2/\sqrt{6}$$
$$\text{Cosine}(B,C) = (1/\sqrt{2} \cdot 1/2) = 1/2\sqrt{2}$$
$$\text{Cosine}(A,C) = (1/\sqrt{3} \cdot 1/2) + (1/\sqrt{3} \cdot 1/2)$$
$$= 2(1/2\sqrt{3}) = 1/\sqrt{3}$$



Create our top 10,000 and synonym dictionary lists

In [512]:
```
!head -10000 5_3_wordcount/part-00000 > top10000
!tail -1000 top10000 > topVocab
print "Files created."
```

```
Files created.
```

Size of cluster

We use a four m3.xlarge instances on Amazon Web Services.

We now build our MRJob functions based around the simple Python functions we did above.

MRJob class to convert each document into a stripe

We remove the stop words and output the inverted index for each word.

```
%%writefile stripeMaker.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
import ast

class stripeMaker(MRJob):

    # we initalize arrays to hold the words we care
    # about. stop_words holds stop words that we wish
    # to exclude. vocab holds the vocabularly that we
    # care about each word forming stripes with.
    # words_interest holds the words we want to form
    # stripes for
    stop_words = []
    vocab = []
    words_interest = []

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper_init = self.mapper_init,\
                       mapper=self.mapper,\
                       combiner=self.reducer,\
                       reducer=self.reducer)]


    # set the stop words that we want to compare
    # all the words to
    def mapper_init(self):

        # read in the file of stop words
        with open('stopwords.txt','r') as myfile:

            # read in all the lines
            for line in myfile.readlines():

                # if the line is not blank, append
                # it to our list of stop words
                if line.strip() != "":
                    self.stop_words.append(line.strip())

        # read in the file of vocab
        with open('topVocab','r') as myfile:

            # read all the lines
            for line in myfile.readlines():

                # split the line and read
                # in the vocab
                line = line.split("\t")
                word = line[1].strip()

                # append the word to the list
                self.vocab.append(word)

        # read in the file of words of interest
```

```
    with open('top10000','r') as myfile:

        # read all the lines
        for line in myfile.readlines():

            # split the line and read
            # in the word
            line = line.split("\t")
            word = line[1].strip()

            # append the word to the list
            self.words_interest.append(word)


# our mapper reads in each line of data
# and splits out the count for each word
# and the associated document
def mapper(self, _, line):

    # split the line by the tabs, set the
    # ngram and lower-case it
    line = line.split("\t")
    ngram = line[0].lower()
    count = int(line[1])

    # set the words
    words = ngram.split()

    # loop through each word in the ngram
    for word in words:

        # check and make sure it's in our
        # words of interests
        if word in self.words_interest:

            # make sure it's not in our stop
            # words
            if word not in self.stop_words:

                # create a dictionary to hold the
                # word
                word_coocur = {}

                # loop through all the other words
                for other_word in words:

                    # make sure the other_word is in
                    # our vocab
                    if other_word in self.vocab:

                        # make sure word not in our stop
                        # words
                        if other_word not in self.stop_words:

                            # if it's not the same word
                            if word != other_word:
```

```
                                    # if we haven't aready found
                                    # this other word for this word
                                    if other_word not in word_coocur:
                                        word_coocur[other_word] = count
                                    else:
                                        word_coocur[other_word] = \
                                        word_coocur[other_word] + count

                    # yield word and its associated
                    # co-occurences if we've actually
                    # add a co-occurence that's not a stop word
                    if len(word_coocur) > 0:
                        yield word, str(word_coocur)


    # our reducer combines the counts for all
    # words
    def reducer(self, word, cooccurs):

        # create a dictionary to hold the
        # co-occurence counts for each word
        cooccur = {}

        # loop through the coocurrences
        # dictionaries, combining as we
        # go along
        for _cooccur in cooccurs:

            # read in the dictionary
            _cooccur = ast.literal_eval(_cooccur)

            # loop through each other word
            for other_word in _cooccur:

                # check to see if the cooccurring words
                # are already in the coocurring dictionary,
                # if not, then add them
                if other_word not in cooccur:
                    cooccur[other_word] = 0

                # increment the count for the word
                cooccur[other_word] = \
                cooccur[other_word] + \
                _cooccur[other_word]

        # yield the word and its cooccruence
        yield word, str(cooccur)

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

```
Overwriting stripeMaker.py
```

In [3]:
```
# make a larger file to test
!head -1000 data/googlebooks-eng-all-5gram-20090715-0-filtered.txt > tes
t.txt
print "done making test.txt"
```

```
done making test.txt
```

In [6]:
```
%reload_ext autoreload
%autoreload 2

# import the MRJob that we created
from stripeMaker import stripeMaker

# set the data that we're going to pull
mr_job = stripeMaker(args=['test.txt','--file=stopwords.txt',\
                           '--file=top10000','--file=topVocab'])

# create the runner and run it
with mr_job.make_runner() as runner:
    runner.run()

    # create a file to store the stripes for each
    # word
    with open('stripe_coocur','w') as myfile:

        # stream_output: get access to the output
        for line in runner.stream_output():

            # grab the key,value
            word,coocur =  \
            mr_job.parse_output_line(line)

            # set the information we want to
            # write
            info = word+"\t"+str(coocur)+"\n"

            # write the word,density to an
            # preliminary output file
            myfile.write(info)

# preview the top of the preliminary file
# that we created
print "Preview the sripes file:"
!head stripe_coocur
```

```
Preview the sripes file:
additions       {'sanctioned': 51}
african {'campaigns': 50}
age     {'franklin': 77}
albert  {'belgium': 46}
author  {'sanctioned': 51}
cells   {'predominance': 44}
clinical        {'pathological': 257}
coast   {'peru': 206}
complete        {'catalog': 49}
evidence        {'admitting': 42}
```

In [14]:
```
!python stripeMaker.py test.txt \
--file=stopwords.txt \
--file=top10000 \
--file=topVocab
```

```
Using configs in /Users/Alex/.mrjob.conf
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker.Alex.20160619.153449.596917
Running step 1 of 1...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/stripeMaker.Alex.20160619.153449.596917/output...
"additions"     "{'sanctioned': 51}"
"african"       "{'campaigns': 50}"
"age"   "{'franklin': 77}"
"albert"        "{'belgium': 46}"
"author"        "{'sanctioned': 51}"
"cells" "{'predominance': 44}"
"clinical"      "{'pathological': 257}"
"coast" "{'peru': 206}"
"complete"      "{'catalog': 49}"
"evidence"      "{'admitting': 42}"
"glance"        "{'hasty': 79}"
"hundred"       "{'crowns': 54}"
"manufacturing" "{'establishments': 44}"
"narrative"     "{'voyages': 532}"
"opera" "{'extant': 47}"
"response"      "{'secreted': 51}"
"roosevelt"     "{'franklin': 77}"
"section"       "{'appointments': 79}"
"shaft" "{'penetrating': 45}"
"tendon"        "{'attaches': 44}"
"thousand"      "{'crowns': 54}"
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker.Alex.20160619.153449.596917...
```

Stripe the entire file on the cloud

In [7]:
```
# create the cluster
!mrjob create-cluster \
--max-hours-idle 1 \
--aws-region=us-west-1 -c ~/.mrjob.conf
```

```
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating persistent cluster to run several jobs in...
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/no_script.Alex.20160619.151659.982911
Copying local files to s3://mrjob-f8c316b67324528f/tmp/no_script.Alex.2
0160619.151659.982911/files/...
j-25Y2I2CYBIZC3
```

In [11]:
```
# add my files to s3
!aws s3 cp stopwords.txt s3://aks-w261-hw5/stopwords.txt
!aws s3 cp top10000 s3://aks-w261-hw5/top10000
!aws s3 cp topVocab s3://aks-w261-hw5/topVocab
```

```
upload: ./stopwords.txt to s3://aks-w261-hw5/stopwords.txt
upload: ./top10000 to s3://aks-w261-hw5/top10000
upload: ./topVocab to s3://aks-w261-hw5/topVocab
```

In [15]:
```
# run the program with the cluster we
# just spun up, still on the test data
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_4
!python stripeMaker.py -r emr s3://aks-w261-hw5/test.txt \
    --file=s3://aks-w261-hw5/stopwords.txt \
    --file=s3://aks-w261-hw5/top10000 \
    --file=s3://aks-w261-hw5/topVocab \
    --cluster-id=j-25Y2I2CYBIZC3 \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_4 \
    --no-output
```

```
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker.Alex.20160619.153535.785909
Copying local files to s3://mrjob-f8c316b67324528f/tmp/stripeMaker.Ale
x.20160619.153535.785909/files/...
Adding our job to existing cluster j-25Y2I2CYBIZC3
Waiting for step 1 of 1 (s-UAZ6NDT24I3W) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40140/cluster
  RUNNING for 11.0s
  RUNNING for 45.1s
     5.0% complete
  RUNNING for 75.9s
    58.2% complete
  RUNNING for 107.6s
   100.0% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-UAZ6NDT24I3W on ec2
-54-183-228-122.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-UAZ6NDT24I3W/syslog
Counters: 54
        File Input Format Counters
                Bytes Read=45912
        File Output Format Counters
                Bytes Written=31
        File System Counters
                FILE: Number of bytes read=253
                FILE: Number of bytes written=3677482
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1872
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=24
                HDFS: Number of write operations=0
                S3: Number of bytes read=45912
                S3: Number of bytes written=31
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=24
                Launched map tasks=24
                Launched reduce tasks=11
                Total megabyte-seconds taken by all map tasks=802637280
                Total megabyte-seconds taken by all reduce tasks=384693
120
                Total time spent by all map tasks (ms)=557387
                Total time spent by all maps in occupied slots (ms)=250
82415
                Total time spent by all reduce tasks (ms)=133574
                Total time spent by all reduces in occupied slots (ms)=
```

```
        12021660
                Total vcore-seconds taken by all map tasks=557387
                Total vcore-seconds taken by all reduce tasks=133574
        Map-Reduce Framework
                CPU time spent (ms)=34540
                Combine input records=1
                Combine output records=1
                Failed Shuffles=0
                GC time elapsed (ms)=9096
                Input split bytes=1872
                Map input records=100
                Map output bytes=31
                Map output materialized bytes=4257
                Map output records=1
                Merged Map outputs=264
                Physical memory (bytes) snapshot=14723215360
                Reduce input groups=1
                Reduce input records=1
                Reduce output records=1
                Reduce shuffle bytes=4257
                Shuffled Maps =264
                Spilled Records=2
                Total committed heap usage (bytes)=16626221056
                Virtual memory (bytes) snapshot=82893606912
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/stripeMaker.
Alex.20160619.153535.785909/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker.Alex.20160619.153535.785909...
Killing our SSH tunnel (pid 17253)
```

Run the word count on the entire corpus using AWS

Now that we've finished testing on a subset of the data, let's try it on the whole data set.

In [16]:
```
# run the program with the cluster we
# just spun up, still on the test data
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_4
!python stripeMaker.py -r emr s3://filtered-5grams/ \
    --file=s3://aks-w261-hw5/stopwords.txt \
    --file=s3://aks-w261-hw5/top10000 \
    --file=s3://aks-w261-hw5/topVocab \
    --cluster-id=j-25Y2I2CYBIZC3 \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_4 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_4/_SUCCESS
delete: s3://aks-w261-hw5/out_5_4/part-00009
delete: s3://aks-w261-hw5/out_5_4/part-00010
delete: s3://aks-w261-hw5/out_5_4/part-00000
delete: s3://aks-w261-hw5/out_5_4/part-00001
delete: s3://aks-w261-hw5/out_5_4/part-00005
delete: s3://aks-w261-hw5/out_5_4/part-00008
delete: s3://aks-w261-hw5/out_5_4/part-00002
delete: s3://aks-w261-hw5/out_5_4/part-00004
delete: s3://aks-w261-hw5/out_5_4/part-00007
delete: s3://aks-w261-hw5/out_5_4/part-00003
delete: s3://aks-w261-hw5/out_5_4/part-00006
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker.Alex.20160619.154118.501113
Copying local files to s3://mrjob-f8c316b67324528f/tmp/stripeMaker.Ale
x.20160619.154118.501113/files/...
Adding our job to existing cluster j-25Y2I2CYBIZC3
Waiting for step 1 of 1 (s-7HCF6V7VKGG2) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40140/cluster
  RUNNING for 24.5s
   100.0% complete
  RUNNING for 56.4s
     5.0% complete
  RUNNING for 88.2s
     5.4% complete
  RUNNING for 119.2s
     5.9% complete
  RUNNING for 150.6s
     6.4% complete
  RUNNING for 182.2s
     6.9% complete
  RUNNING for 213.2s
     7.4% complete
  RUNNING for 244.8s
     8.3% complete
  RUNNING for 275.8s
     8.7% complete
  RUNNING for 307.6s
     9.5% complete
  RUNNING for 338.5s
    10.6% complete
  RUNNING for 369.5s
    11.1% complete
  RUNNING for 400.7s
    11.6% complete
  RUNNING for 431.8s
    12.1% complete
  RUNNING for 462.5s
    13.0% complete
  RUNNING for 494.4s
    13.6% complete
  RUNNING for 526.0s
    14.1% complete
```

```
RUNNING for 557.3s
  14.7% complete
RUNNING for 588.8s
  16.0% complete
RUNNING for 620.4s
  17.1% complete
RUNNING for 651.6s
  17.6% complete
RUNNING for 683.9s
  18.1% complete
RUNNING for 714.9s
  18.6% complete
RUNNING for 745.9s
  19.5% complete
RUNNING for 777.4s
  20.4% complete
RUNNING for 809.9s
  21.3% complete
RUNNING for 840.6s
  22.1% complete
RUNNING for 872.7s
  22.9% complete
RUNNING for 903.7s
  23.7% complete
RUNNING for 934.9s
  24.4% complete
RUNNING for 965.8s
  24.9% complete
RUNNING for 996.8s
  25.5% complete
RUNNING for 1028.5s
  26.8% complete
RUNNING for 1060.0s
  27.4% complete
RUNNING for 1091.0s
  27.8% complete
RUNNING for 1122.8s
  28.9% complete
RUNNING for 1153.7s
  29.7% complete
RUNNING for 1184.5s
  30.8% complete
RUNNING for 1216.0s
  31.2% complete
RUNNING for 1247.7s
  31.5% complete
RUNNING for 1279.0s
  32.4% complete
RUNNING for 1310.1s
  32.7% complete
RUNNING for 1340.9s
  33.1% complete
RUNNING for 1372.1s
  33.5% complete
RUNNING for 1403.1s
  34.9% complete
RUNNING for 1435.0s
```

```
  35.2% complete
RUNNING for 1467.2s
  35.6% complete
RUNNING for 1499.5s
  36.2% complete
RUNNING for 1531.0s
  36.7% complete
RUNNING for 1562.5s
  37.1% complete
RUNNING for 1593.4s
  37.5% complete
RUNNING for 1625.8s
  38.2% complete
RUNNING for 1657.3s
  39.0% complete
RUNNING for 1688.3s
  39.6% complete
RUNNING for 1720.7s
  40.1% complete
RUNNING for 1751.7s
  40.7% complete
RUNNING for 1782.8s
  41.1% complete
RUNNING for 1813.8s
  41.6% complete
RUNNING for 1844.8s
  42.3% complete
RUNNING for 1876.4s
  42.7% complete
RUNNING for 1908.1s
  43.0% complete
RUNNING for 1939.7s
  44.1% complete
RUNNING for 1970.6s
  44.8% complete
RUNNING for 2002.5s
  45.1% complete
RUNNING for 2033.4s
  45.5% complete
RUNNING for 2065.2s
  46.2% complete
RUNNING for 2096.0s
  46.7% complete
RUNNING for 2127.6s
  47.1% complete
RUNNING for 2158.3s
  47.4% complete
RUNNING for 2189.8s
  48.2% complete
RUNNING for 2220.5s
  49.2% complete
RUNNING for 2252.2s
  49.5% complete
RUNNING for 2283.1s
  49.9% complete
RUNNING for 2314.7s
  50.6% complete
```

```
  RUNNING for 2346.2s
    51.0% complete
  RUNNING for 2377.7s
    51.5% complete
  RUNNING for 2408.9s
    52.1% complete
  RUNNING for 2439.6s
    52.5% complete
  RUNNING for 2470.8s
    53.1% complete
  RUNNING for 2501.7s
    53.9% complete
  RUNNING for 2533.7s
    56.4% complete
  RUNNING for 2565.1s
    56.6% complete
  RUNNING for 2596.4s
    56.8% complete
  RUNNING for 2628.1s
    58.4% complete
  RUNNING for 2659.2s
    59.9% complete
  RUNNING for 2690.3s
    60.0% complete
  RUNNING for 2721.0s
    82.7% complete
  RUNNING for 2752.2s
   100.0% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-7HCF6V7VKGG2 on ec2
-54-183-228-122.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-7HCF6V7VKGG2/syslog.2016-06-19-15
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-7HCF6V7VKGG2/syslog
Counters: 56
        File Input Format Counters
                Bytes Read=2156069116
        File Output Format Counters
                Bytes Written=8054521
        File System Counters
                FILE: Number of bytes read=12190479
                FILE: Number of bytes written=53710803
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=23450
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=190
                HDFS: Number of write operations=0
                S3: Number of bytes read=2156069116
                S3: Number of bytes written=8054521
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
```

```
        Job Counters
                Data-local map tasks=188
                Killed reduce tasks=1
                Launched map tasks=190
                Launched reduce tasks=12
                Other local map tasks=2
                Total megabyte-seconds taken by all map tasks=544550630
40
                Total megabyte-seconds taken by all reduce tasks=291155
41440
                Total time spent by all map tasks (ms)=37816016
                Total time spent by all maps in occupied slots (ms)=170
1720720
                Total time spent by all reduce tasks (ms)=10109563
                Total time spent by all reduces in occupied slots (ms)=
909860670
                Total vcore-seconds taken by all map tasks=37816016
                Total vcore-seconds taken by all reduce tasks=10109563
        Map-Reduce Framework
                CPU time spent (ms)=21108040
                Combine input records=1284587
                Combine output records=627653
                Failed Shuffles=0
                GC time elapsed (ms)=66408
                Input split bytes=23450
                Map input records=58682266
                Map output bytes=37265318
                Map output materialized bytes=20411684
                Map output records=1284587
                Merged Map outputs=2090
                Physical memory (bytes) snapshot=114165063680
                Reduce input groups=9582
                Reduce input records=627653
                Reduce output records=9582
                Reduce shuffle bytes=20411684
                Shuffled Maps =2090
                Spilled Records=1255306
                Total committed heap usage (bytes)=123632353280
                Virtual memory (bytes) snapshot=409766621184
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/stripeMaker.
Alex.20160619.154118.501113/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker.Alex.20160619.154118.501113...
Killing our SSH tunnel (pid 17274)
```

In [21]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_stripes1

# sync the files to a local directory and save them on s3
!aws s3 cp s3://aks-w261-hw5/out_5_4/ s3://aks-w261-hw5/5_4_stripes1/ --
recursive
!aws s3 sync s3://aks-w261-hw5/5_4_stripes1 /Users/Alex/Documents/Berkel
ey/1602Summer/W261/HW5/5_4_stripes1
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_stripes1
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_stripes1/_SUCCESS
```

```
mkdir: /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_stripes1:
 File exists
copy: s3://aks-w261-hw5/out_5_4/part-00000 to s3://aks-w261-hw5/5_4_str
ipes1/part-00000
copy: s3://aks-w261-hw5/out_5_4/_SUCCESS to s3://aks-w261-hw5/5_4_strip
es1/_SUCCESS
copy: s3://aks-w261-hw5/out_5_4/part-00006 to s3://aks-w261-hw5/5_4_str
ipes1/part-00006
copy: s3://aks-w261-hw5/out_5_4/part-00003 to s3://aks-w261-hw5/5_4_str
ipes1/part-00003
copy: s3://aks-w261-hw5/out_5_4/part-00002 to s3://aks-w261-hw5/5_4_str
ipes1/part-00002
copy: s3://aks-w261-hw5/out_5_4/part-00007 to s3://aks-w261-hw5/5_4_str
ipes1/part-00007
copy: s3://aks-w261-hw5/out_5_4/part-00005 to s3://aks-w261-hw5/5_4_str
ipes1/part-00005
copy: s3://aks-w261-hw5/out_5_4/part-00008 to s3://aks-w261-hw5/5_4_str
ipes1/part-00008
copy: s3://aks-w261-hw5/out_5_4/part-00004 to s3://aks-w261-hw5/5_4_str
ipes1/part-00004
copy: s3://aks-w261-hw5/out_5_4/part-00001 to s3://aks-w261-hw5/5_4_str
ipes1/part-00001
copy: s3://aks-w261-hw5/out_5_4/part-00010 to s3://aks-w261-hw5/5_4_str
ipes1/part-00010
copy: s3://aks-w261-hw5/out_5_4/part-00009 to s3://aks-w261-hw5/5_4_str
ipes1/part-00009
download: s3://aks-w261-hw5/5_4_stripes1/_SUCCESS to 5_4_stripes1/_SUCC
ESS
download: s3://aks-w261-hw5/5_4_stripes1/part-00000 to 5_4_stripes1/par
t-00000
download: s3://aks-w261-hw5/5_4_stripes1/part-00008 to 5_4_stripes1/par
t-00008
download: s3://aks-w261-hw5/5_4_stripes1/part-00002 to 5_4_stripes1/par
t-00002
download: s3://aks-w261-hw5/5_4_stripes1/part-00007 to 5_4_stripes1/par
t-00007
download: s3://aks-w261-hw5/5_4_stripes1/part-00009 to 5_4_stripes1/par
t-00009
download: s3://aks-w261-hw5/5_4_stripes1/part-00003 to 5_4_stripes1/par
t-00003
download: s3://aks-w261-hw5/5_4_stripes1/part-00006 to 5_4_stripes1/par
t-00006
download: s3://aks-w261-hw5/5_4_stripes1/part-00001 to 5_4_stripes1/par
t-00001
download: s3://aks-w261-hw5/5_4_stripes1/part-00004 to 5_4_stripes1/par
t-00004
download: s3://aks-w261-hw5/5_4_stripes1/part-00010 to 5_4_stripes1/par
t-00010
download: s3://aks-w261-hw5/5_4_stripes1/part-00005 to 5_4_stripes1/par
t-00005
_SUCCESS   part-00001 part-00003 part-00005 part-00007 part-00009
part-00000 part-00002 part-00004 part-00006 part-00008 part-00010
```

My stripe maker took 45 minutes running on a 4 node cluster, 1 master at m1.medium and 3 cores at m3.xlarge. This is by far my most intensive MRJob.

Binarize the stripes

In [46]:
```
%%writefile binarizeStripes.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
import ast

class binarizeStripes(MRJob):

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper=self.mapper)]


    # our mapper reads in each line of data
    # and binarizes each stripe
    def mapper(self, _, line):

        # we use a try in case in case any
        # data go written in a weird format
        try:
            # split the line by the tabs, set the
            # word and its cooccurence words
            line = line.split("\t")
            word = line[0].strip().strip("\"").strip("'")
            other_words = line[1].strip().strip("\"").strip("'")
            other_words = ast.literal_eval(other_words)

            # loop through each other word in
            # the cooccurence matrix
            for other_word in other_words:

                # set the value equal to 1
                other_words[other_word]=1

            # yield the word with its cooccurence
            # matrix
            yield word, str(other_words)
        except:
            pass


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

```
Overwriting binarizeStripes.py
```

In [23]:
```
%reload_ext autoreload
%autoreload 2

# import the MRJob that we created
from binarizeStripes import binarizeStripes

# set the data that we're going to pull
mr_job = binarizeStripes(args=['stripe_coocur'])

# create the runner and run it
with mr_job.make_runner() as runner:
    runner.run()

    # create a file to store the stripes for each
    # word
    with open('stripe_binarized','w') as myfile:

        # stream_output: get access to the output
        for line in runner.stream_output():

            # grab the key,value
            word,coocur =  \
            mr_job.parse_output_line(line)

            # set the information we want to
            # write
            info = word+"\t"+str(coocur)+"\n"

            # write the word,density to an
            # preliminary output file
            myfile.write(info)

# preview the top of the preliminary file
# that we created
print "Preview the binarized stripes file:"
!head stripe_binarized
```

```
Preview the binarized stripes file:
additions       {'sanctioned': 1}
african {'campaigns': 1}
age     {'franklin': 1}
albert  {'belgium': 1}
author  {'sanctioned': 1}
cells   {'predominance': 1}
clinical        {'pathological': 1}
coast   {'peru': 1}
complete        {'catalog': 1}
evidence        {'admitting': 1}
```

In [25]:
```
# test the binarize on a subset of the data
!head -500 5_4_stripes1/part-00004 > testing_stripes
```

In [2]:
```
# test the binarize function on this
# larger output locally
!python binarizeStripes.py testing_stripes > temp.txt
!head -1 temp.txt
```

```
Using configs in /Users/Alex/.mrjob.conf
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/binarizeStripes.Alex.20160620.232750.602429
Running step 1 of 1...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/binarizeStripes.Alex.20160620.232750.602429/output...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/binarizeStripes.Alex.20160620.232750.602429...
"abandoned"     "{'restless': 1, 'humiliation': 1, 'tents': 1, 'defende
rs': 1, 'pursuits': 1, 'athenians': 1, 'traitor': 1, 'nova': 1, 'habita
tion': 1, 'anarchy': 1, 'forts': 1, 'metaphysical': 1, 'damp': 1, 'enum
eration': 1, 'speedily': 1, 'mate': 1, 'judicious': 1, 'cunning': 1, 'o
fficially': 1, 'exhaustion': 1, 'jungle': 1, 'hue': 1, 'metropolis': 1,
 'relic': 1, 'housed': 1, 'cart': 1, 'questioning': 1, 'domains': 1, 't
ravellers': 1, 'ordinances': 1, 'sinners': 1, 'licence': 1, 'lutheran':
 1, 'forfeited': 1, 'arduous': 1, 'detroit': 1, 'fide': 1, 'natures':
 1, 'barren': 1, 'conqueror': 1, 'intolerable': 1, 'sinner': 1}"
```

Run the binarize on the entire dataset in the cloud

In [48]:
```
# run the program with the cluster we
# just spun up, on the stripes that we
# just made
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_4
!python binarizeStripes.py -r emr s3://aks-w261-hw5/5_4_stripes1/ \
    --cluster-id=j-25Y2I2CYBIZC3 \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_4 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_4/part-00000
delete: s3://aks-w261-hw5/out_5_4/part-00010
delete: s3://aks-w261-hw5/out_5_4/part-00011
delete: s3://aks-w261-hw5/out_5_4/part-00004
delete: s3://aks-w261-hw5/out_5_4/part-00003
delete: s3://aks-w261-hw5/out_5_4/part-00012
delete: s3://aks-w261-hw5/out_5_4/part-00002
delete: s3://aks-w261-hw5/out_5_4/part-00006
delete: s3://aks-w261-hw5/out_5_4/part-00009
delete: s3://aks-w261-hw5/out_5_4/part-00008
delete: s3://aks-w261-hw5/out_5_4/part-00007
delete: s3://aks-w261-hw5/out_5_4/part-00005
delete: s3://aks-w261-hw5/out_5_4/part-00001
delete: s3://aks-w261-hw5/out_5_4/part-00013
delete: s3://aks-w261-hw5/out_5_4/part-00014
delete: s3://aks-w261-hw5/out_5_4/part-00015
delete: s3://aks-w261-hw5/out_5_4/part-00016
delete: s3://aks-w261-hw5/out_5_4/part-00017
delete: s3://aks-w261-hw5/out_5_4/part-00018
delete: s3://aks-w261-hw5/out_5_4/part-00019
delete: s3://aks-w261-hw5/out_5_4/part-00020
delete: s3://aks-w261-hw5/out_5_4/part-00021
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/binarizeStripes.Alex.20160619.170721.256749
Copying local files to s3://mrjob-f8c316b67324528f/tmp/binarizeStripes.
Alex.20160619.170721.256749/files/...
Adding our job to existing cluster j-25Y2I2CYBIZC3
Waiting for step 1 of 1 (s-E2I55CB9L0YY) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40140/cluster
  RUNNING for 22.2s
Unable to connect to resource manager
  RUNNING for 55.1s
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-E2I55CB9L0YY on ec2
-54-183-228-122.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-E2I55CB9L0YY/syslog
Counters: 35
        File Input Format Counters
                Bytes Read=8268288
        File Output Format Counters
                Bytes Written=3823264
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=3197418
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2883
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=31
```

```
                HDFS: Number of write operations=0
                S3: Number of bytes read=8268288
                S3: Number of bytes written=3823264
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=31
                Launched map tasks=31
                Total megabyte-seconds taken by all map tasks=935320320
                Total time spent by all map tasks (ms)=649528
                Total time spent by all maps in occupied slots (ms)=292
28760
                Total time spent by all reduces in occupied slots (ms)=
0
                Total vcore-seconds taken by all map tasks=649528
        Map-Reduce Framework
                CPU time spent (ms)=37140
                Failed Shuffles=0
                GC time elapsed (ms)=8388
                Input split bytes=2883
                Map input records=9582
                Map output records=7251
                Merged Map outputs=0
                Physical memory (bytes) snapshot=8516853760
                Spilled Records=0
                Total committed heap usage (bytes)=10136059904
                Virtual memory (bytes) snapshot=60884066304
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/binarizeStri
pes.Alex.20160619.170721.256749/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/binarizeStripes.Alex.20160619.170721.256749...
Killing our SSH tunnel (pid 17576)
```

Store the binarize stripes locally and make a folder for them on S3

In [49]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_bin_stripe
s1

# sync the files to a local directory and save them on s3
!aws s3 cp s3://aks-w261-hw5/out_5_4/ s3://aks-w261-hw5/5_4_bin_stripes
1/ --recursive
!aws s3 sync s3://aks-w261-hw5/5_4_bin_stripes1 /Users/Alex/Documents/Be
rkeley/1602Summer/W261/HW5/5_4_bin_stripes1
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_bin_stripes1
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_bin_stripes1/_SUCCESS
```

```
copy: s3://aks-w261-hw5/out_5_4/_SUCCESS to s3://aks-w261-hw5/5_4_bin_s
tripes1/_SUCCESS
copy: s3://aks-w261-hw5/out_5_4/part-00003 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00003
copy: s3://aks-w261-hw5/out_5_4/part-00006 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00006
copy: s3://aks-w261-hw5/out_5_4/part-00001 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00001
copy: s3://aks-w261-hw5/out_5_4/part-00007 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00007
copy: s3://aks-w261-hw5/out_5_4/part-00005 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00005
copy: s3://aks-w261-hw5/out_5_4/part-00002 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00002
copy: s3://aks-w261-hw5/out_5_4/part-00004 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00004
copy: s3://aks-w261-hw5/out_5_4/part-00009 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00009
copy: s3://aks-w261-hw5/out_5_4/part-00008 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00008
copy: s3://aks-w261-hw5/out_5_4/part-00010 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00010
copy: s3://aks-w261-hw5/out_5_4/part-00000 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00000
copy: s3://aks-w261-hw5/out_5_4/part-00013 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00013
copy: s3://aks-w261-hw5/out_5_4/part-00011 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00011
copy: s3://aks-w261-hw5/out_5_4/part-00012 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00012
copy: s3://aks-w261-hw5/out_5_4/part-00016 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00016
copy: s3://aks-w261-hw5/out_5_4/part-00014 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00014
copy: s3://aks-w261-hw5/out_5_4/part-00017 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00017
copy: s3://aks-w261-hw5/out_5_4/part-00020 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00020
copy: s3://aks-w261-hw5/out_5_4/part-00018 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00018
copy: s3://aks-w261-hw5/out_5_4/part-00024 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00024
copy: s3://aks-w261-hw5/out_5_4/part-00022 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00022
copy: s3://aks-w261-hw5/out_5_4/part-00019 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00019
copy: s3://aks-w261-hw5/out_5_4/part-00025 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00025
copy: s3://aks-w261-hw5/out_5_4/part-00015 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00015
copy: s3://aks-w261-hw5/out_5_4/part-00023 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00023
copy: s3://aks-w261-hw5/out_5_4/part-00021 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00021
copy: s3://aks-w261-hw5/out_5_4/part-00027 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00027
copy: s3://aks-w261-hw5/out_5_4/part-00026 to s3://aks-w261-hw5/5_4_bin
```

```
_stripes1/part-00026
copy: s3://aks-w261-hw5/out_5_4/part-00030 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00030
copy: s3://aks-w261-hw5/out_5_4/part-00028 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00028
copy: s3://aks-w261-hw5/out_5_4/part-00029 to s3://aks-w261-hw5/5_4_bin
_stripes1/part-00029
download: s3://aks-w261-hw5/5_4_bin_stripes1/_SUCCESS to 5_4_bin_stripe
s1/_SUCCESS
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00000 to 5_4_bin_stri
pes1/part-00000
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00002 to 5_4_bin_stri
pes1/part-00002
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00008 to 5_4_bin_stri
pes1/part-00008
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00006 to 5_4_bin_stri
pes1/part-00006
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00003 to 5_4_bin_stri
pes1/part-00003
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00004 to 5_4_bin_stri
pes1/part-00004
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00014 to 5_4_bin_stri
pes1/part-00014
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00010 to 5_4_bin_stri
pes1/part-00010
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00015 to 5_4_bin_stri
pes1/part-00015
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00009 to 5_4_bin_stri
pes1/part-00009
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00018 to 5_4_bin_stri
pes1/part-00018
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00005 to 5_4_bin_stri
pes1/part-00005
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00001 to 5_4_bin_stri
pes1/part-00001
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00013 to 5_4_bin_stri
pes1/part-00013
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00020 to 5_4_bin_stri
pes1/part-00020
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00019 to 5_4_bin_stri
pes1/part-00019
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00023 to 5_4_bin_stri
pes1/part-00023
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00024 to 5_4_bin_stri
pes1/part-00024
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00025 to 5_4_bin_stri
pes1/part-00025
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00011 to 5_4_bin_stri
pes1/part-00011
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00017 to 5_4_bin_stri
pes1/part-00017
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00027 to 5_4_bin_stri
pes1/part-00027
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00029 to 5_4_bin_stri
pes1/part-00029
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00030 to 5_4_bin_stri
pes1/part-00030
```

```
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00026 to 5_4_bin_stri
pes1/part-00026
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00007 to 5_4_bin_stri
pes1/part-00007
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00022 to 5_4_bin_stri
pes1/part-00022
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00028 to 5_4_bin_stri
pes1/part-00028
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00021 to 5_4_bin_stri
pes1/part-00021
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00016 to 5_4_bin_stri
pes1/part-00016
download: s3://aks-w261-hw5/5_4_bin_stripes1/part-00012 to 5_4_bin_stri
pes1/part-00012
_SUCCESS   part-00004 part-00009 part-00014 part-00019 part-00024 part-
00029
part-00000 part-00005 part-00010 part-00015 part-00020 part-00025 part-
00030
part-00001 part-00006 part-00011 part-00016 part-00021 part-00026
part-00002 part-00007 part-00012 part-00017 part-00022 part-00027
part-00003 part-00008 part-00013 part-00018 part-00023 part-00028
```

Write the MRJob class to normalize the words in the stripes

The normalization is the inverse of the square root of the sum of squares of the items in the vector. **Because we have already binarized, we can just take the inverse of the square root of the length of each cooccurence matrix.**

```
%%writefile normalizeStripes.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
import ast
import math

class normalizeStripes(MRJob):
    """This class normalizes successfully if
    we've already binarized"""


    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper=self.mapper)]


    # our mapper reads in each line of data
    # and normalizes each stripe
    def mapper(self, _, line):

        # we use a try in case in case any
        # data go written in a weird format
        try:
            # split the line by the tabs, set the
            # word and its cooccurence words
            line = line.split("\t")
            word = line[0].strip().strip("\"").strip("'")
            other_words = line[1].strip().strip("\"").strip("'")
            other_words = ast.literal_eval(other_words)

            # initalize a sum value
            total = len(other_words)

            # loop through each other word again,
            # this time using the total to normalize
            # remember that we normalize by the inverse
            # of the square root of the sum of the squares
            # of the values, but since we've binarized
            # we can just take the inverse of the square
            # root of the length of each coocurence matrix
            for other_word in other_words:

                # set the value equal to the value
                # divided by the squareroot of the length
                other_words[other_word] = 1.0/math.sqrt(total)

            # yield the word with its cooccurence
            # matrix
            yield word, str(other_words)

        except:
            pass


if __name__ == '__main__':
```

```
    normalizeStripes.run()
```

```
Overwriting normalizeStripes.py
```

In [51]:
```
%reload_ext autoreload
%autoreload 2

# import the MRJob that we created
from normalizeStripes import normalizeStripes

# set the data that we're going to pull
mr_job = normalizeStripes(args=['stripe_binarized'])

# create the runner and run it
with mr_job.make_runner() as runner:
    runner.run()

    # create a file to store the stripes for each
    # word
    with open('stripe_normalized','w') as myfile:

        # stream_output: get access to the output
        for line in runner.stream_output():

            # grab the key,value
            word,coocur =  \
            mr_job.parse_output_line(line)

            # set the information we want to
            # write
            info = word+"\t"+str(coocur)+"\n"

            # write the word,density to an
            # preliminary output file
            myfile.write(info)

# preview the top of the preliminary file
# that we created
print "Preview the normalized stripes file:"
!head stripe_normalized
```

```
Preview the normalized stripes file:
additions       {'sanctioned': 1.0}
african {'campaigns': 1.0}
age     {'franklin': 1.0}
albert  {'belgium': 1.0}
author  {'sanctioned': 1.0}
cells   {'predominance': 1.0}
clinical        {'pathological': 1.0}
coast   {'peru': 1.0}
complete        {'catalog': 1.0}
evidence        {'admitting': 1.0}
```

In [58]:
```
# test on the command line
!python normalizeStripes.py stripe_binarized
```

```
Using configs in /Users/Alex/.mrjob.conf
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/normalizeStripes.Alex.20160619.172649.464379
Running step 1 of 1...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/normalizeStripes.Alex.20160619.172649.464379/output...
"additions"     "{'sanctioned': 1.0}"
"african"       "{'campaigns': 1.0}"
"age"   "{'franklin': 1.0}"
"albert"        "{'belgium': 1.0}"
"author"        "{'sanctioned': 1.0}"
"cells" "{'predominance': 1.0}"
"clinical"      "{'pathological': 1.0}"
"coast" "{'peru': 1.0}"
"complete"      "{'catalog': 1.0}"
"evidence"      "{'admitting': 1.0}"
"glance"        "{'hasty': 1.0}"
"hundred"       "{'crowns': 1.0}"
"manufacturing" "{'establishments': 1.0}"
"narrative"     "{'voyages': 1.0}"
"opera" "{'extant': 1.0}"
"response"      "{'secreted': 1.0}"
"roosevelt"     "{'franklin': 1.0}"
"section"       "{'appointments': 1.0}"
"shaft" "{'penetrating': 1.0}"
"tendon"        "{'attaches': 1.0}"
"thousand"      "{'crowns': 1.0}"
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/normalizeStripes.Alex.20160619.172649.464379...
```

Run normalize on the stripes in the cloud

In [59]:
```
# run the program with the cluster we
# just spun up, on the stripes that we
# just binarized
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_4
!python normalizeStripes.py -r emr s3://aks-w261-hw5/5_4_bin_stripes1/ \
    --cluster-id=j-25Y2I2CYBIZC3 \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_4 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_4/part-00001
delete: s3://aks-w261-hw5/out_5_4/part-00010
delete: s3://aks-w261-hw5/out_5_4/part-00011
delete: s3://aks-w261-hw5/out_5_4/part-00004
delete: s3://aks-w261-hw5/out_5_4/part-00003
delete: s3://aks-w261-hw5/out_5_4/part-00012
delete: s3://aks-w261-hw5/out_5_4/part-00002
delete: s3://aks-w261-hw5/out_5_4/part-00007
delete: s3://aks-w261-hw5/out_5_4/part-00000
delete: s3://aks-w261-hw5/out_5_4/part-00009
delete: s3://aks-w261-hw5/out_5_4/part-00005
delete: s3://aks-w261-hw5/out_5_4/part-00008
delete: s3://aks-w261-hw5/out_5_4/part-00006
delete: s3://aks-w261-hw5/out_5_4/part-00013
delete: s3://aks-w261-hw5/out_5_4/part-00014
delete: s3://aks-w261-hw5/out_5_4/part-00015
delete: s3://aks-w261-hw5/out_5_4/part-00016
delete: s3://aks-w261-hw5/out_5_4/part-00017
delete: s3://aks-w261-hw5/out_5_4/part-00018
delete: s3://aks-w261-hw5/out_5_4/part-00020
delete: s3://aks-w261-hw5/out_5_4/part-00019
delete: s3://aks-w261-hw5/out_5_4/part-00022
delete: s3://aks-w261-hw5/out_5_4/part-00021
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/normalizeStripes.Alex.20160619.172656.662960
Copying local files to s3://mrjob-f8c316b67324528f/tmp/normalizeStripe
s.Alex.20160619.172656.662960/files/...
Adding our job to existing cluster j-25Y2I2CYBIZC3
Waiting for step 1 of 1 (s-1YVWQU33RSGZ0) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40140/cluster
  RUNNING for 4.0s
Unable to connect to resource manager
  RUNNING for 36.4s
  RUNNING for 67.3s
  RUNNING for 98.5s
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-1YVWQU33RSGZ0 on ec
2-54-183-228-122.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-1YVWQU33RSGZ0/syslog
Counters: 35
        File Input Format Counters
                Bytes Read=3876600
        File Output Format Counters
                Bytes Written=8076942
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=3610423
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=3395
```

```
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=35
                HDFS: Number of write operations=0
                S3: Number of bytes read=3876600
                S3: Number of bytes written=8076942
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=35
                Launched map tasks=35
                Total megabyte-seconds taken by all map tasks=105845184
0
                Total time spent by all map tasks (ms)=735036
                Total time spent by all maps in occupied slots (ms)=330
76620
                Total time spent by all reduces in occupied slots (ms)=
0
                Total vcore-seconds taken by all map tasks=735036
        Map-Reduce Framework
                CPU time spent (ms)=41100
                Failed Shuffles=0
                GC time elapsed (ms)=9227
                Input split bytes=3395
                Map input records=7251
                Map output records=7251
                Merged Map outputs=0
                Physical memory (bytes) snapshot=9631232000
                Spilled Records=0
                Total committed heap usage (bytes)=11705253888
                Virtual memory (bytes) snapshot=68643151872
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/normalizeStr
ipes.Alex.20160619.172656.662960/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/normalizeStripes.Alex.20160619.172656.662960...
Killing our SSH tunnel (pid 17693)
```

Store the normalized stripes locally and make a folder for them on S3

In [60]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_norm_stripes1

# sync the files to a local directory and save them on s3
!aws s3 cp s3://aks-w261-hw5/out_5_4/ s3://aks-w261-hw5/5_4_norm_stripes1/ --recursive
!aws s3 sync s3://aks-w261-hw5/5_4_norm_stripes1 /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_norm_stripes1
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_norm_stripes1
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_norm_stripes1/_SUCCESS
```

```
copy: s3://aks-w261-hw5/out_5_4/_SUCCESS to s3://aks-w261-hw5/5_4_norm_
stripes1/_SUCCESS
copy: s3://aks-w261-hw5/out_5_4/part-00000 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00000
copy: s3://aks-w261-hw5/out_5_4/part-00007 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00007
copy: s3://aks-w261-hw5/out_5_4/part-00002 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00002
copy: s3://aks-w261-hw5/out_5_4/part-00003 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00003
copy: s3://aks-w261-hw5/out_5_4/part-00004 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00004
copy: s3://aks-w261-hw5/out_5_4/part-00001 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00001
copy: s3://aks-w261-hw5/out_5_4/part-00008 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00008
copy: s3://aks-w261-hw5/out_5_4/part-00006 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00006
copy: s3://aks-w261-hw5/out_5_4/part-00005 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00005
copy: s3://aks-w261-hw5/out_5_4/part-00009 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00009
copy: s3://aks-w261-hw5/out_5_4/part-00011 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00011
copy: s3://aks-w261-hw5/out_5_4/part-00012 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00012
copy: s3://aks-w261-hw5/out_5_4/part-00016 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00016
copy: s3://aks-w261-hw5/out_5_4/part-00010 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00010
copy: s3://aks-w261-hw5/out_5_4/part-00018 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00018
copy: s3://aks-w261-hw5/out_5_4/part-00013 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00013
copy: s3://aks-w261-hw5/out_5_4/part-00015 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00015
copy: s3://aks-w261-hw5/out_5_4/part-00017 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00017
copy: s3://aks-w261-hw5/out_5_4/part-00019 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00019
copy: s3://aks-w261-hw5/out_5_4/part-00014 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00014
copy: s3://aks-w261-hw5/out_5_4/part-00020 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00020
copy: s3://aks-w261-hw5/out_5_4/part-00026 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00026
copy: s3://aks-w261-hw5/out_5_4/part-00027 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00027
copy: s3://aks-w261-hw5/out_5_4/part-00021 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00021
copy: s3://aks-w261-hw5/out_5_4/part-00023 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00023
copy: s3://aks-w261-hw5/out_5_4/part-00028 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00028
copy: s3://aks-w261-hw5/out_5_4/part-00024 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00024
copy: s3://aks-w261-hw5/out_5_4/part-00025 to s3://aks-w261-hw5/5_4_nor
```

```
m_stripes1/part-00025
copy: s3://aks-w261-hw5/out_5_4/part-00030 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00030
copy: s3://aks-w261-hw5/out_5_4/part-00029 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00029
copy: s3://aks-w261-hw5/out_5_4/part-00031 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00031
copy: s3://aks-w261-hw5/out_5_4/part-00033 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00033
copy: s3://aks-w261-hw5/out_5_4/part-00034 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00034
copy: s3://aks-w261-hw5/out_5_4/part-00032 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00032
copy: s3://aks-w261-hw5/out_5_4/part-00022 to s3://aks-w261-hw5/5_4_nor
m_stripes1/part-00022
download: s3://aks-w261-hw5/5_4_norm_stripes1/_SUCCESS to 5_4_norm_stri
pes1/_SUCCESS
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00001 to 5_4_norm_st
ripes1/part-00001
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00002 to 5_4_norm_st
ripes1/part-00002
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00000 to 5_4_norm_st
ripes1/part-00000
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00007 to 5_4_norm_st
ripes1/part-00007
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00009 to 5_4_norm_st
ripes1/part-00009
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00005 to 5_4_norm_st
ripes1/part-00005
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00008 to 5_4_norm_st
ripes1/part-00008
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00010 to 5_4_norm_st
ripes1/part-00010
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00003 to 5_4_norm_st
ripes1/part-00003
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00006 to 5_4_norm_st
ripes1/part-00006
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00012 to 5_4_norm_st
ripes1/part-00012
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00015 to 5_4_norm_st
ripes1/part-00015
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00011 to 5_4_norm_st
ripes1/part-00011
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00004 to 5_4_norm_st
ripes1/part-00004
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00017 to 5_4_norm_st
ripes1/part-00017
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00020 to 5_4_norm_st
ripes1/part-00020
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00013 to 5_4_norm_st
ripes1/part-00013
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00021 to 5_4_norm_st
ripes1/part-00021
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00014 to 5_4_norm_st
ripes1/part-00014
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00016 to 5_4_norm_st
ripes1/part-00016
```

```
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00029 to 5_4_norm_st
ripes1/part-00029
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00026 to 5_4_norm_st
ripes1/part-00026
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00025 to 5_4_norm_st
ripes1/part-00025
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00030 to 5_4_norm_st
ripes1/part-00030
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00031 to 5_4_norm_st
ripes1/part-00031
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00027 to 5_4_norm_st
ripes1/part-00027
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00024 to 5_4_norm_st
ripes1/part-00024
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00028 to 5_4_norm_st
ripes1/part-00028
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00022 to 5_4_norm_st
ripes1/part-00022
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00033 to 5_4_norm_st
ripes1/part-00033
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00034 to 5_4_norm_st
ripes1/part-00034
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00032 to 5_4_norm_st
ripes1/part-00032
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00023 to 5_4_norm_st
ripes1/part-00023
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00019 to 5_4_norm_st
ripes1/part-00019
download: s3://aks-w261-hw5/5_4_norm_stripes1/part-00018 to 5_4_norm_st
ripes1/part-00018
_SUCCESS   part-00005 part-00011 part-00017 part-00023 part-00029
part-00000 part-00006 part-00012 part-00018 part-00024 part-00030
part-00001 part-00007 part-00013 part-00019 part-00025 part-00031
part-00002 part-00008 part-00014 part-00020 part-00026 part-00032
part-00003 part-00009 part-00015 part-00021 part-00027 part-00033
part-00004 part-00010 part-00016 part-00022 part-00028 part-00034
```

MRJob Class to create the inverted matrix

```
%%writefile invertIndex.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
import ast
import math

class invertIndex(MRJob):

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper=self.mapper,\
                       combiner=self.reducer,\
                       reducer=self.reducer)]


    # our mapper reads in each line of data
    # and inverts each stripe
    def mapper(self, _, line):

        # we use a try in case in case any
        # data go written in a weird format
        try:
            # split the line by the tabs, set the
            # word and its cooccurence words
            line = line.split("\t")
            word = line[0].strip().strip("\"").strip("'")
            other_words = line[1].strip().strip("\"").strip("'")
            other_words = ast.literal_eval(other_words)

            # loop through each of the other words
            for other_word in other_words:

                # create a dictionary for the
                # invert, holds the word and
                # inverted value
                inverted_value = \
                other_words[other_word]
                invert = {word:inverted_value}

                # yield the inverted word with
                # its inverted value
                yield other_word, str(invert)

        except:
            pass

    # our reducer combines the inverted values
    # for multiple words
    def reducer(self, word, cooccurs):

        # create a dictionary to hold the
        # co-occurence counts for each word
        cooccur = {}

        # loop through the coocurrences
```

```
        # dictionaries, combining as we
        # go along
        for _cooccur in cooccurs:

            # read in the dictionary
            _cooccur = ast.literal_eval(_cooccur)

            # loop through each other word
            for other_word in _cooccur:

                cooccur[other_word] = \
                _cooccur[other_word]

        # yield the word and its cooccruence
        yield word, str(cooccur)

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

```
Overwriting invertIndex.py
```

In [255]:
```
%reload_ext autoreload
%autoreload 2

# import the MRJob that we created
from invertIndex import invertIndex

# set the data that we're going to pull
mr_job = invertIndex(args=['stripe_normalized'])

# create the runner and run it
with mr_job.make_runner() as runner:
    runner.run()

    # create a file to store the stripes for each
    # word
    with open('stripe_invert','w') as myfile:

        # stream_output: get access to the output
        for line in runner.stream_output():

            # grab the key,value
            word,coocur =  \
            mr_job.parse_output_line(line)

            # set the information we want to
            # write
            info = word+"\t"+str(coocur)+"\n"

            # write the word,density to an
            # preliminary output file
            myfile.write(info)

# preview the top of the preliminary file
# that we created
print "Preview the inverted stripes file:"
!head stripe_invert
```

```
Preview the inverted stripes file:
addressed       {'sermon': 1.0}
aerial  {'navigation': 0.7071067811865475, 'history': 0.301511344577763
63}
america's       {'guide': 0.4472135954999579, 'censorship': 0.707106781
1865475}
american        {'postwar': 0.7071067811865475, 'narrative': 0.57735026
91896258, 'history': 0.30151134457776363}
analysis        {'physiological': 1.0}
apology {'commentary': 1.0}
arithmetic      {'key': 0.5773502691896258}
aspiration      {'marrow': 0.7071067811865475, 'bone': 0.70710678118654
75}
assessment      {'methodology': 1.0}
based   {'method': 1.0}
```

In [62]:
```
# test on the commandn line
!python invertIndex.py stripe_normalized
```

```
Using configs in /Users/Alex/.mrjob.conf
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/invertIndex.Alex.20160619.173352.648877
Running step 1 of 1...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/invertIndex.Alex.20160619.173352.648877/output...
"admitting"     "{'evidence': 1.0}"
"appointments"  "{'section': 1.0}"
"attaches"      "{'tendon': 1.0}"
"belgium"       "{'albert': 1.0}"
"campaigns"     "{'african': 1.0}"
"catalog"       "{'complete': 1.0}"
"crowns"        "{'thousand': 1.0, 'hundred': 1.0}"
"establishments"        "{'manufacturing': 1.0}"
"extant"        "{'opera': 1.0}"
"franklin"      "{'age': 1.0, 'roosevelt': 1.0}"
"hasty" "{'glance': 1.0}"
"pathological"  "{'clinical': 1.0}"
"penetrating"   "{'shaft': 1.0}"
"peru"  "{'coast': 1.0}"
"predominance"  "{'cells': 1.0}"
"sanctioned"    "{'additions': 1.0, 'author': 1.0}"
"secreted"      "{'response': 1.0}"
"voyages"       "{'narrative': 1.0}"
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/invertIndex.Alex.20160619.173352.648877...
```

Run inversion in the cloud

In [63]:
```
# run the program with the cluster we
# just spun up, on the stripes that we
# just binarized
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_4
!python invertIndex.py -r emr s3://aks-w261-hw5/5_4_norm_stripes1/ \
    --cluster-id=j-25Y2I2CYBIZC3 \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_4 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_4/_SUCCESS
delete: s3://aks-w261-hw5/out_5_4/part-00009
delete: s3://aks-w261-hw5/out_5_4/part-00010
delete: s3://aks-w261-hw5/out_5_4/part-00002
delete: s3://aks-w261-hw5/out_5_4/part-00005
delete: s3://aks-w261-hw5/out_5_4/part-00004
delete: s3://aks-w261-hw5/out_5_4/part-00001
delete: s3://aks-w261-hw5/out_5_4/part-00007
delete: s3://aks-w261-hw5/out_5_4/part-00000
delete: s3://aks-w261-hw5/out_5_4/part-00008
delete: s3://aks-w261-hw5/out_5_4/part-00011
delete: s3://aks-w261-hw5/out_5_4/part-00003
delete: s3://aks-w261-hw5/out_5_4/part-00006
delete: s3://aks-w261-hw5/out_5_4/part-00012
delete: s3://aks-w261-hw5/out_5_4/part-00015
delete: s3://aks-w261-hw5/out_5_4/part-00017
delete: s3://aks-w261-hw5/out_5_4/part-00014
delete: s3://aks-w261-hw5/out_5_4/part-00013
delete: s3://aks-w261-hw5/out_5_4/part-00016
delete: s3://aks-w261-hw5/out_5_4/part-00018
delete: s3://aks-w261-hw5/out_5_4/part-00020
delete: s3://aks-w261-hw5/out_5_4/part-00019
delete: s3://aks-w261-hw5/out_5_4/part-00021
delete: s3://aks-w261-hw5/out_5_4/part-00022
delete: s3://aks-w261-hw5/out_5_4/part-00023
delete: s3://aks-w261-hw5/out_5_4/part-00024
delete: s3://aks-w261-hw5/out_5_4/part-00026
delete: s3://aks-w261-hw5/out_5_4/part-00025
delete: s3://aks-w261-hw5/out_5_4/part-00027
delete: s3://aks-w261-hw5/out_5_4/part-00028
delete: s3://aks-w261-hw5/out_5_4/part-00030
delete: s3://aks-w261-hw5/out_5_4/part-00029
delete: s3://aks-w261-hw5/out_5_4/part-00031
delete: s3://aks-w261-hw5/out_5_4/part-00032
delete: s3://aks-w261-hw5/out_5_4/part-00033
delete: s3://aks-w261-hw5/out_5_4/part-00034
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/invertIndex.Alex.20160619.173519.902821
Copying local files to s3://mrjob-f8c316b67324528f/tmp/invertIndex.Ale
x.20160619.173519.902821/files/...
Adding our job to existing cluster j-25Y2I2CYBIZC3
Waiting for step 1 of 1 (s-1WG2HXZ285DRR) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40140/cluster
  RUNNING for 24.9s
     0.0% complete
  RUNNING for 58.0s
    15.3% complete
  RUNNING for 89.4s
    41.6% complete
  RUNNING for 121.0s
   100.0% complete
  COMPLETED
Attempting to fetch counters from logs...
```

```
Looking for step log in /mnt/var/log/hadoop/steps/s-1WG2HXZ285DRR on ec
2-54-183-228-122.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-1WG2HXZ285DRR/syslog
Counters: 55
        File Input Format Counters
                Bytes Read=8076942
        File Output Format Counters
                Bytes Written=7811660
        File System Counters
                FILE: Number of bytes read=3628172
                FILE: Number of bytes written=10867590
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=3430
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=35
                HDFS: Number of write operations=0
                S3: Number of bytes read=8076942
                S3: Number of bytes written=7811660
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=36
                Killed map tasks=1
                Launched map tasks=36
                Launched reduce tasks=11
                Total megabyte-seconds taken by all map tasks=141806016
0
                Total megabyte-seconds taken by all reduce tasks=596168
640
                Total time spent by all map tasks (ms)=984764
                Total time spent by all maps in occupied slots (ms)=443
14380
                Total time spent by all reduce tasks (ms)=207003
                Total time spent by all reduces in occupied slots (ms)=
18630270
                Total vcore-seconds taken by all map tasks=984764
                Total vcore-seconds taken by all reduce tasks=207003
        Map-Reduce Framework
                CPU time spent (ms)=105330
                Combine input records=247430
                Combine output records=31787
                Failed Shuffles=0
                GC time elapsed (ms)=11361
                Input split bytes=3430
                Map input records=7251
                Map output bytes=11274980
                Map output materialized bytes=2452420
                Map output records=247430
                Merged Map outputs=385
                Physical memory (bytes) snapshot=20170137600
                Reduce input groups=991
                Reduce input records=31787
```

```
                Reduce output records=991
                Reduce shuffle bytes=2452420
                Shuffled Maps =385
                Spilled Records=63574
                Total committed heap usage (bytes)=22599958528
                Virtual memory (bytes) snapshot=104544759808
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/invertIndex.
Alex.20160619.173519.902821/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/invertIndex.Alex.20160619.173519.902821...
Killing our SSH tunnel (pid 17729)
```

Store the inverted index locally and make a folder on S3 for it

In [64]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_invert1

# sync the files to a local directory and save them on s3
!aws s3 cp s3://aks-w261-hw5/out_5_4/ s3://aks-w261-hw5/5_4_invert1/ --r
ecursive
!aws s3 sync s3://aks-w261-hw5/5_4_invert1 /Users/Alex/Documents/Berkele
y/1602Summer/W261/HW5/5_4_invert1
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_invert1
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_invert1/_SUCCESS
```

```
copy: s3://aks-w261-hw5/out_5_4/_SUCCESS to s3://aks-w261-hw5/5_4_inver
t1/_SUCCESS
copy: s3://aks-w261-hw5/out_5_4/part-00003 to s3://aks-w261-hw5/5_4_inv
ert1/part-00003
copy: s3://aks-w261-hw5/out_5_4/part-00008 to s3://aks-w261-hw5/5_4_inv
ert1/part-00008
copy: s3://aks-w261-hw5/out_5_4/part-00001 to s3://aks-w261-hw5/5_4_inv
ert1/part-00001
copy: s3://aks-w261-hw5/out_5_4/part-00004 to s3://aks-w261-hw5/5_4_inv
ert1/part-00004
copy: s3://aks-w261-hw5/out_5_4/part-00007 to s3://aks-w261-hw5/5_4_inv
ert1/part-00007
copy: s3://aks-w261-hw5/out_5_4/part-00002 to s3://aks-w261-hw5/5_4_inv
ert1/part-00002
copy: s3://aks-w261-hw5/out_5_4/part-00006 to s3://aks-w261-hw5/5_4_inv
ert1/part-00006
copy: s3://aks-w261-hw5/out_5_4/part-00000 to s3://aks-w261-hw5/5_4_inv
ert1/part-00000
copy: s3://aks-w261-hw5/out_5_4/part-00009 to s3://aks-w261-hw5/5_4_inv
ert1/part-00009
copy: s3://aks-w261-hw5/out_5_4/part-00010 to s3://aks-w261-hw5/5_4_inv
ert1/part-00010
copy: s3://aks-w261-hw5/out_5_4/part-00005 to s3://aks-w261-hw5/5_4_inv
ert1/part-00005
download: s3://aks-w261-hw5/5_4_invert1/_SUCCESS to 5_4_invert1/_SUCCES
S
download: s3://aks-w261-hw5/5_4_invert1/part-00001 to 5_4_invert1/part-
00001
download: s3://aks-w261-hw5/5_4_invert1/part-00004 to 5_4_invert1/part-
00004
download: s3://aks-w261-hw5/5_4_invert1/part-00007 to 5_4_invert1/part-
00007
download: s3://aks-w261-hw5/5_4_invert1/part-00010 to 5_4_invert1/part-
00010
download: s3://aks-w261-hw5/5_4_invert1/part-00005 to 5_4_invert1/part-
00005
download: s3://aks-w261-hw5/5_4_invert1/part-00003 to 5_4_invert1/part-
00003
download: s3://aks-w261-hw5/5_4_invert1/part-00000 to 5_4_invert1/part-
00000
download: s3://aks-w261-hw5/5_4_invert1/part-00002 to 5_4_invert1/part-
00002
download: s3://aks-w261-hw5/5_4_invert1/part-00006 to 5_4_invert1/part-
00006
download: s3://aks-w261-hw5/5_4_invert1/part-00009 to 5_4_invert1/part-
00009
download: s3://aks-w261-hw5/5_4_invert1/part-00008 to 5_4_invert1/part-
00008
_SUCCESS   part-00001 part-00003 part-00005 part-00007 part-00009
part-00000 part-00002 part-00004 part-00006 part-00008 part-00010
```

MRJob to calculate the cosine similarity between terms

To calculate the cosine similarity of two stripe documents, say A and C, compute partial similarities using dot products and aggregating partial similarities to get overall cosine similarity.

```
%%writefile cosineSim.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
import ast
import math
from mrjob.protocol import RawProtocol

class cosineSim(MRJob):

    MRJob.SORT_VALUES = True
    INTERNAL_PROTOCOL = RawProtocol
    OUTPUT_PROTOCOL = RawProtocol

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):

        # set how we want to sort, we use a single reducer
        # only so that we can do a total sort, we use multiple
        # reducers for all other steps
        JOBCONF = {
            'mapred.output.key.comparator.class': 'org.apache.hadoop.map
red.lib.KeyFieldBasedComparator',
            'mapred.text.key.comparator.options': '-k1,1nr',
            'mapreduce.job.reduces': 1,
        }

        return [MRStep(mapper=self.mapper,\
                       reducer=self.reducer),\
               MRStep(jobconf = JOBCONF,\
                       mapper=None,\
                       reducer=self.reducer_final)]


    # our mapper reads in each line of data
    # and outputs the partial aggregation
    # for each pair combination
    def mapper(self, _, line):


        # we use a try in case in case any
        # data go written in a weird format
        try:
            # split the line by the tabs, set the
            # word and its cooccurence words
            line = line.split("\t")
            key = line[0].strip().strip("\"").strip("'")
            words = line[1].strip().strip("\"").strip("'")
            words = ast.literal_eval(words)

            # loop through each of the words
            for word in words:

                # loop through the rest of the
                # rest of the words in words
                for other_word in words:
```

```
                # as long as it isn't this
                # word
                if word != other_word:

                    # get the two word values
                    word_val = words[word]
                    other_word_val = words[other_word]

                    # multiple the two together
                    # dot product, for partial
                    # similarity
                    partial = \
                    word_val * other_word_val

                    # set the pair
                    pair = word,other_word

                    # yeild the pair and its
                    # partial similarity
                    yield str(pair),str(partial)
        except:
            pass


    # our reducer combines the partial
    # similarities to create an aggregated
    # similarity
    def reducer(self, pair, similarities):

        # create a sum variable
        sum_sim = 0.0

        # loop through the similarities
        for similarity in similarities:
            sum_sim = sum_sim + float(similarity)

        # we want to make sure that the count
        # string is at least 15 characters, if
        # its not, we add trailing zeros. this
        # helps us sort
        extend = "0"

        # convert density to a string
        sum_sim = str(sum_sim)

        # while the count lenght is less than 15
        while len(sum_sim) < 15:
            sum_sim = sum_sim + extend

        # yield the pair and its partial
        # similarity
        yield sum_sim, pair

    # our reducer final simply outputs the sorted
    # list that hadoop has conviently shuffled and
    # sorted for us
```

```
    def reducer_final(self, similarity, pairs):
        for pair in pairs:
            yield pair,similarity

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

```
Overwriting cosineSim.py
```

In [272]:
```
%reload_ext autoreload
%autoreload 2

# import the MRJob that we created
from cosineSim import cosineSim

# set the data that we're going to pull
mr_job = cosineSim(args=['stripe_invert'])

# create the runner and run it
with mr_job.make_runner() as runner:
    runner.run()

    # create a file to store the stripes for each
    # word
    with open('cosineSimilarities','w') as myfile:

        # stream_output: get access to the output
        for line in runner.stream_output():

            # grab the key,value
            pair,similarity =  \
            mr_job.parse_output_line(line)

            # set the information we want to
            # write
            info = str(pair)+"\t"+str(similarity)+"\n"

            # write the word,density to an
            # preliminary output file
            myfile.write(info)

# preview the top of the preliminary file
# that we created
print "Preview the cosine similarities file:"
!head cosineSimilarities
```

```
Preview the cosine similarities file:
['aerial', 'american']  0.408248290464
['aerial', 'eurobond']  0.707106781187
['aerial', 'history']   0.213200716356
['aerial', 'modern']    0.408248290464
['aerial', 'navigation']        0.5
['aerial', 'postwar']   0.5
['aerial', 'railroads'] 0.408248290464
['aerial', 'southeast'] 0.5
['aerial', 'travel']    0.707106781187
['aerial', 'united']    0.707106781187
```

In [3]:
```
# run on the command line
!python cosineSim.py stripe_invert > temp.txt
!head temp.txt
```

```
Using configs in /Users/Alex/.mrjob.conf
ignoring partitioner keyword arg (requires real Hadoop): 'org.apache.ha
doop.mapred.lib.KeyFieldBasedPartitioner'
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/cosineSim.Alex.20160620.232846.271681
Running step 1 of 2...
Running step 2 of 2...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/cosineSim.Alex.20160620.232846.271681/output...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/cosineSim.Alex.20160620.232846.271681...
('life', 'study')       0.0912870929175
('study', 'life')       0.0912870929175
('key', 'study')        0.1290994448740
('study', 'key')        0.1290994448740
('history', 'practical')        0.1507556722890
('practical', 'history')        0.1507556722890
('bibliographical', 'study')    0.1581138830080
('communities', 'study')        0.1581138830080
('comparative', 'study')        0.1581138830080
('conceptual', 'study') 0.1581138830080
```

Run the cosine similarity across the entire dataset on the cloud

In [74]:
```
# run the program with the cluster we
# just spun up, on the stripes that we
# just binarized
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_4
!python cosineSim.py -r emr s3://aks-w261-hw5/5_4_invert1/ \
    --cluster-id=j-25Y2I2CYBIZC3 \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_4 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_4/_SUCCESS
delete: s3://aks-w261-hw5/out_5_4/part-00009
delete: s3://aks-w261-hw5/out_5_4/part-00010
delete: s3://aks-w261-hw5/out_5_4/part-00001
delete: s3://aks-w261-hw5/out_5_4/part-00005
delete: s3://aks-w261-hw5/out_5_4/part-00003
delete: s3://aks-w261-hw5/out_5_4/part-00006
delete: s3://aks-w261-hw5/out_5_4/part-00002
delete: s3://aks-w261-hw5/out_5_4/part-00008
delete: s3://aks-w261-hw5/out_5_4/part-00004
delete: s3://aks-w261-hw5/out_5_4/part-00007
delete: s3://aks-w261-hw5/out_5_4/part-00000
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/cosineSim.Alex.20160619.180114.892883
Copying local files to s3://mrjob-f8c316b67324528f/tmp/cosineSim.Alex.2
0160619.180114.892883/files/...
Adding our job to existing cluster j-25Y2I2CYBIZC3
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Waiting for step 1 of 2 (s-1D01BTQMXLWND) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40140/cluster
  RUNNING for 5.9s
Unable to connect to resource manager
  RUNNING for 38.0s
  RUNNING for 68.9s
  RUNNING for 100.7s
  RUNNING for 131.4s
  RUNNING for 162.7s
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-1D01BTQMXLWND on ec
2-54-183-228-122.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-1D01BTQMXLWND/syslog
Counters: 54
        File Input Format Counters
                Bytes Read=7959349
```

```
        File Output Format Counters
                Bytes Written=463819278
        File System Counters
                FILE: Number of bytes read=239151381
                FILE: Number of bytes written=628078955
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2576
                HDFS: Number of bytes written=463819278
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=89
                HDFS: Number of write operations=22
                S3: Number of bytes read=7959349
                S3: Number of bytes written=0
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=28
                Launched map tasks=28
                Launched reduce tasks=11
                Total megabyte-seconds taken by all map tasks=183001968
0
                Total megabyte-seconds taken by all reduce tasks=147391
2000
                Total time spent by all map tasks (ms)=1270847
                Total time spent by all maps in occupied slots (ms)=571
88115
                Total time spent by all reduce tasks (ms)=511775
                Total time spent by all reduces in occupied slots (ms)=
46059750
                Total vcore-seconds taken by all map tasks=1270847
                Total vcore-seconds taken by all reduce tasks=511775
        Map-Reduce Framework
                CPU time spent (ms)=860340
                Combine input records=0
                Combine output records=0
                Failed Shuffles=0
                GC time elapsed (ms)=13612
                Input split bytes=2576
                Map input records=991
                Map output bytes=774230900
                Map output materialized bytes=384863023
                Map output records=19049886
                Merged Map outputs=308
                Physical memory (bytes) snapshot=20778643456
                Reduce input groups=11638896
                Reduce input records=19049886
                Reduce output records=11638896
                Reduce shuffle bytes=384863023
                Shuffled Maps =308
                Spilled Records=38099772
                Total committed heap usage (bytes)=23703584768
                Virtual memory (bytes) snapshot=90721210368
        Shuffle Errors
                BAD_ID=0
```

```
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Waiting for step 2 of 2 (s-2GLCX0UOUTSOB) to complete...
  RUNNING for 75.9s
  RUNNING for 106.9s
  RUNNING for 138.1s
  RUNNING for 168.6s
  RUNNING for 199.9s
  RUNNING for 230.5s
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-2GLCX0UOUTSOB on ec
2-54-183-228-122.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-2GLCX0UOUTSOB/syslog
Counters: 54
        File Input Format Counters
                Bytes Read=463976259
        File Output Format Counters
                Bytes Written=475458174
        File System Counters
                FILE: Number of bytes read=149205149
                FILE: Number of bytes written=340654144
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=463981242
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=66
                HDFS: Number of write operations=0
                S3: Number of bytes read=0
                S3: Number of bytes written=475458174
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=33
                Launched map tasks=33
                Launched reduce tasks=1
                Total megabyte-seconds taken by all map tasks=155212704
0
                Total megabyte-seconds taken by all reduce tasks=452689
920
                Total time spent by all map tasks (ms)=1077866
                Total time spent by all maps in occupied slots (ms)=485
03970
                Total time spent by all reduce tasks (ms)=157184
                Total time spent by all reduces in occupied slots (ms)=
14146560
                Total vcore-seconds taken by all map tasks=1077866
                Total vcore-seconds taken by all reduce tasks=157184
        Map-Reduce Framework
                CPU time spent (ms)=458010
```

```
                Combine input records=0
                Combine output records=0
                Failed Shuffles=0
                GC time elapsed (ms)=19089
                Input split bytes=4983
                Map input records=11638896
                Map output bytes=475458174
                Map output materialized bytes=187900821
                Map output records=11638896
                Merged Map outputs=33
                Physical memory (bytes) snapshot=22045343744
                Reduce input groups=11638896
                Reduce input records=11638896
                Reduce output records=11638896
                Reduce shuffle bytes=187900821
                Shuffled Maps =33
                Spilled Records=23277792
                Total committed heap usage (bytes)=25322586112
                Virtual memory (bytes) snapshot=68117573632
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/cosineSim.Al
ex.20160619.180114.892883/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/cosineSim.Alex.20160619.180114.892883...
Killing our SSH tunnel (pid 17847)
```

Save the similarity files locally and create a bucket for them in the cloud

This bucket can be accessed here: s3://aks-w261-hw5/5_4_cosineSim/

In [75]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_cosineSim

# sync the files to a local directory and save them on s3
!aws s3 cp s3://aks-w261-hw5/out_5_4/ s3://aks-w261-hw5/5_4_cosineSim/ --recursive
!aws s3 sync s3://aks-w261-hw5/5_4_cosineSim /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_cosineSim
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_cosineSim
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_cosineSim/_SUCCESS
```

```
copy: s3://aks-w261-hw5/out_5_4/_SUCCESS to s3://aks-w261-hw5/5_4_cosineSim/_SUCCESS
copy: s3://aks-w261-hw5/out_5_4/part-00000 to s3://aks-w261-hw5/5_4_cosineSim/part-00000
download: s3://aks-w261-hw5/5_4_cosineSim/_SUCCESS to 5_4_cosineSim/_SUCCESS
download: s3://aks-w261-hw5/5_4_cosineSim/part-00000 to 5_4_cosineSim/part-00000
_SUCCESS   part-00000
```

Preview the top similarities

In [76]:
```
!head ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_cosineSim/part-00000
```

```
('predicated', 'substantive')           0.7071067811870
('substantive', 'predicated')           0.7071067811870
('ammonia', 'hydrochloric')             0.5962847940000
('hydrochloric', 'ammonia')             0.5962847940000
('oxidation', 'solubility')             0.5685352436150
('solubility', 'oxidation')             0.5685352436150
('flexor', 'tendon')            0.5590169943750
('tendon', 'flexor')            0.5590169943750
('hydrochloric', 'hydroxide')           0.5477225575050
('hydroxide', 'hydrochloric')           0.5477225575050
```

Now let's move on to Jaccard Similarities

We've completed cosine similarities. Let's know calculate the Jaccard similarities.

Make the MRJob class that indexes the data for Jaccard Similarities

```
%%writefile stripeMaker_jac.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
import ast

class stripeMaker_jac(MRJob):

    # we initalize arrays to hold the words we care
    # about. stop_words holds stop words that we wish
    # to exclude. vocab holds the vocabularly that we
    # care about each word forming stripes with.
    # words_interest holds the words we want to form
    # stripes for
    stop_words = []
    vocab = []
    words_interest = []

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper_init = self.mapper_init,\
                       mapper=self.mapper,\
                       combiner=self.reducer,\
                       reducer=self.reducer)]


    # set the stop words that we want to compare
    # all the words to
    def mapper_init(self):

        # read in the file of stop words
        with open('stopwords.txt','r') as myfile:

            # read in all the lines
            for line in myfile.readlines():

                # if the line is not blank, append
                # it to our list of stop words
                if line.strip() != "":
                    self.stop_words.append(line.strip())

        # read in the file of vocab
        with open('topVocab','r') as myfile:

            # read all the lines
            for line in myfile.readlines():

                # split the line and read
                # in the vocab
                line = line.split("\t")
                word = line[1].strip()

                # append the word to the list
                self.vocab.append(word)

        # read in the file of words of interest
```

```
    with open('top10000','r') as myfile:

        # read all the lines
        for line in myfile.readlines():

            # split the line and read
            # in the word
            line = line.split("\t")
            word = line[1].strip()

            # append the word to the list
            self.words_interest.append(word)


# our mapper reads in each line of data
# and splits out the count for each word
# and the associated document
def mapper(self, _, line):

    # split the line by the tabs, set the
    # ngram and lower-case it
    line = line.split("\t")
    ngram = line[0].lower()
    count = int(line[1])

    # set the words
    words = ngram.split()

    # loop through each word in the ngram
    for word in words:

        # check and make sure it's in our
        # words of interests
        if word in self.words_interest:

            # make sure it's not in our stop
            # words
            if word not in self.stop_words:

                # create a dictionary to hold the
                # word
                word_coocur = {}

                # loop through all the other words
                for other_word in words:

                    # make sure the other_word is in
                    # our vocab
                    if other_word in self.vocab:

                        # make sure word not in our stop
                        # words
                        if other_word not in self.stop_words:

                            # if it's not the same word
                            if word != other_word:
```

```
                                        # if we haven't aready found
                                        # this other word for this word
                                        if other_word not in word_coocur:
                                            word_coocur[other_word] = count
                                        else:
                                            word_coocur[other_word] = \
                                            word_coocur[other_word] + count

                                # create an entry for the word itself, it's
                                # cardinality
                                word_star = "*"+word
                                word_coocur[word_star] = count

                            # yield word and its associated
                            # co-occurences if we've actually
                            # add a co-occurence that's not a stop word
                            if len(word_coocur) > 0:
                                yield word, str(word_coocur)


    # our reducer combines the counts for all
    # words
    def reducer(self, word, cooccurs):

        # create a dictionary to hold the
        # co-occurence counts for each word
        cooccur = {}

        # loop through the coocurrences
        # dictionaries, combining as we
        # go along
        for _cooccur in cooccurs:

            # read in the dictionary
            _cooccur = ast.literal_eval(_cooccur)

            # loop through each other word
            for other_word in _cooccur:

                # check to see if the cooccurring words
                # are already in the coocurring dictionary,
                # if not, then add them
                if other_word not in cooccur:
                    cooccur[other_word] = 0

                # increment the count for the word
                cooccur[other_word] = \
                cooccur[other_word] + \
                _cooccur[other_word]

        # yield the word and its cooccruence
        yield word, str(cooccur)

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

```
Writing stripeMaker_jac.py
```

Test on a sample file, first locally and then on the cloud

In [9]:
```
# test locally using the command line
!python stripeMaker_jac.py test.txt \
--file=stopwords.txt \
--file=top10000 \
--file=topVocab \
> temp.txt
!head temp.txt
```

```
Using configs in /Users/Alex/.mrjob.conf
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker_jac.Alex.20160620.233130.110608
Running step 1 of 1...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/stripeMaker_jac.Alex.20160620.233130.110608/output...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker_jac.Alex.20160620.233130.110608...
"ab"    "{'*ab': 125}"
"abandonment"   "{'*abandonment': 71}"
"ability"       "{'*ability': 41}"
"able"  "{'*able': 105}"
"abnormal"      "{'*abnormal': 53}"
"abolition"     "{'*abolition': 62}"
"abraham"       "{'*abraham': 109}"
"absorption"    "{'*absorption': 148}"
"abstract"      "{'*abstract': 77}"
"academic"      "{'*academic': 54}"
```

In [81]:
```
# test on the cloud
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_4
!python stripeMaker_jac.py -r emr s3://aks-w261-hw5/test.txt \
    --file=s3://aks-w261-hw5/stopwords.txt \
    --file=s3://aks-w261-hw5/top10000 \
    --file=s3://aks-w261-hw5/topVocab \
    --cluster-id=j-25Y2I2CYBIZC3 \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_4 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_4/_SUCCESS
delete: s3://aks-w261-hw5/out_5_4/part-00000
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker_jac.Alex.20160619.182212.879141
Copying local files to s3://mrjob-f8c316b67324528f/tmp/stripeMaker_jac.
Alex.20160619.182212.879141/files/...
Adding our job to existing cluster j-25Y2I2CYBIZC3
Waiting for step 1 of 1 (s-1PLHFL5BCDD10) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40140/cluster
  RUNNING for 21.9s
Unable to connect to resource manager
  RUNNING for 54.3s
  RUNNING for 84.9s
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-1PLHFL5BCDD10 on ec
2-54-183-228-122.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-1PLHFL5BCDD10/syslog
Counters: 55
        File Input Format Counters
                Bytes Read=45912
        File Output Format Counters
                Bytes Written=4412
        File System Counters
                FILE: Number of bytes read=3905
                FILE: Number of bytes written=3687156
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1872
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=24
                HDFS: Number of write operations=0
                S3: Number of bytes read=45912
                S3: Number of bytes written=4412
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=25
                Killed map tasks=1
                Launched map tasks=25
                Launched reduce tasks=11
                Total megabyte-seconds taken by all map tasks=838183680
                Total megabyte-seconds taken by all reduce tasks=370272
960
                Total time spent by all map tasks (ms)=582072
                Total time spent by all maps in occupied slots (ms)=261
93240
                Total time spent by all reduce tasks (ms)=128567
                Total time spent by all reduces in occupied slots (ms)=
```

```
11571030
                Total vcore-seconds taken by all map tasks=582072
                Total vcore-seconds taken by all reduce tasks=128567
        Map-Reduce Framework
                CPU time spent (ms)=41260
                Combine input records=193
                Combine output records=173
                Failed Shuffles=0
                GC time elapsed (ms)=8382
                Input split bytes=1872
                Map input records=100
                Map output bytes=5693
                Map output materialized bytes=8991
                Map output records=193
                Merged Map outputs=264
                Physical memory (bytes) snapshot=14656004096
                Reduce input groups=147
                Reduce input records=173
                Reduce output records=147
                Reduce shuffle bytes=8991
                Shuffled Maps =264
                Spilled Records=346
                Total committed heap usage (bytes)=16625696768
                Virtual memory (bytes) snapshot=82953224192
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/stripeMaker_
jac.Alex.20160619.182212.879141/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker_jac.Alex.20160619.182212.879141...
Killing our SSH tunnel (pid 17909)
```

Run the index maker on the entire corpus on the cloud

In [82]:
```
# run on the cloud
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_4
!python stripeMaker_jac.py -r emr s3://filtered-5grams/ \
    --file=s3://aks-w261-hw5/stopwords.txt \
    --file=s3://aks-w261-hw5/top10000 \
    --file=s3://aks-w261-hw5/topVocab \
    --cluster-id=j-25Y2I2CYBIZC3 \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_4 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_4/part-00000
delete: s3://aks-w261-hw5/out_5_4/part-00009
delete: s3://aks-w261-hw5/out_5_4/part-00010
delete: s3://aks-w261-hw5/out_5_4/part-00007
delete: s3://aks-w261-hw5/out_5_4/part-00003
delete: s3://aks-w261-hw5/out_5_4/part-00001
delete: s3://aks-w261-hw5/out_5_4/part-00002
delete: s3://aks-w261-hw5/out_5_4/_SUCCESS
delete: s3://aks-w261-hw5/out_5_4/part-00004
delete: s3://aks-w261-hw5/out_5_4/part-00008
delete: s3://aks-w261-hw5/out_5_4/part-00005
delete: s3://aks-w261-hw5/out_5_4/part-00006
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker_jac.Alex.20160619.182649.857818
Copying local files to s3://mrjob-f8c316b67324528f/tmp/stripeMaker_jac.
Alex.20160619.182649.857818/files/...
Adding our job to existing cluster j-25Y2I2CYBIZC3
Waiting for step 1 of 1 (s-3SQKHHVKFXV9M) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40140/cluster
  RUNNING for 4.7s
   100.0% complete
  RUNNING for 37.1s
     0.0% complete
  RUNNING for 68.1s
     5.0% complete
  RUNNING for 99.5s
     5.5% complete
  RUNNING for 130.3s
     5.9% complete
  RUNNING for 162.3s
     6.4% complete
  RUNNING for 193.4s
     6.9% complete
  RUNNING for 224.6s
     7.3% complete
  RUNNING for 256.0s
     7.8% complete
  RUNNING for 287.8s
     8.1% complete
  RUNNING for 319.3s
     8.9% complete
  RUNNING for 350.2s
     9.1% complete
  RUNNING for 381.1s
     9.3% complete
  RUNNING for 411.9s
    10.6% complete
  RUNNING for 443.0s
    11.2% complete
  RUNNING for 474.7s
    11.7% complete
  RUNNING for 505.9s
    12.1% complete
```

```
RUNNING for 537.9s
  12.5% complete
RUNNING for 568.8s
  13.0% complete
RUNNING for 600.1s
  13.7% complete
RUNNING for 631.4s
  14.0% complete
RUNNING for 662.2s
  14.8% complete
RUNNING for 693.7s
  15.2% complete
RUNNING for 725.2s
  15.5% complete
RUNNING for 756.8s
  16.4% complete
RUNNING for 788.2s
  17.2% complete
RUNNING for 819.5s
  17.6% complete
RUNNING for 850.9s
  18.4% complete
RUNNING for 881.7s
  18.7% complete
RUNNING for 913.5s
  19.0% complete
RUNNING for 945.0s
  20.3% complete
RUNNING for 976.7s
  20.7% complete
RUNNING for 1008.3s
  21.6% complete
RUNNING for 1039.8s
  22.0% complete
RUNNING for 1071.1s
  22.3% complete
RUNNING for 1102.4s
  23.3% complete
RUNNING for 1133.3s
  23.8% complete
RUNNING for 1164.7s
  24.3% complete
RUNNING for 1195.7s
  24.7% complete
RUNNING for 1227.4s
  25.1% complete
RUNNING for 1259.1s
  25.7% complete
RUNNING for 1290.7s
  27.2% complete
RUNNING for 1322.3s
  27.6% complete
RUNNING for 1353.2s
  28.0% complete
RUNNING for 1384.7s
  28.8% complete
RUNNING for 1416.3s
```

```
  29.2% complete
RUNNING for 1447.2s
  30.0% complete
RUNNING for 1479.2s
  31.2% complete
RUNNING for 1510.1s
  31.6% complete
RUNNING for 1541.6s
  32.0% complete
RUNNING for 1572.6s
  32.4% complete
RUNNING for 1604.2s
  32.8% complete
RUNNING for 1635.0s
  33.8% complete
RUNNING for 1666.1s
  34.3% complete
RUNNING for 1697.6s
  34.7% complete
RUNNING for 1729.0s
  35.0% complete
RUNNING for 1760.7s
  35.6% complete
RUNNING for 1792.2s
  35.9% complete
RUNNING for 1824.0s
  36.6% complete
RUNNING for 1855.3s
  37.0% complete
RUNNING for 1886.4s
  37.7% complete
RUNNING for 1917.9s
  38.3% complete
RUNNING for 1948.8s
  38.7% complete
RUNNING for 1980.7s
  39.1% complete
RUNNING for 2011.9s
  39.7% complete
RUNNING for 2044.0s
  40.3% complete
RUNNING for 2074.9s
  40.7% complete
RUNNING for 2106.5s
  41.0% complete
RUNNING for 2138.0s
  41.4% complete
RUNNING for 2169.0s
  42.4% complete
RUNNING for 2200.2s
  43.0% complete
RUNNING for 2232.0s
  43.3% complete
RUNNING for 2262.8s
  43.7% complete
RUNNING for 2294.7s
  44.1% complete
```

```
  RUNNING for 2326.3s
    44.6% complete
  RUNNING for 2357.1s
    45.3% complete
  RUNNING for 2388.6s
    45.6% complete
  RUNNING for 2420.4s
    46.2% complete
  RUNNING for 2451.3s
    46.8% complete
  RUNNING for 2483.4s
    47.4% complete
  RUNNING for 2514.6s
    48.0% complete
  RUNNING for 2545.4s
    48.4% complete
  RUNNING for 2576.7s
    48.8% complete
  RUNNING for 2608.6s
    49.3% complete
  RUNNING for 2639.9s
    49.7% complete
  RUNNING for 2670.9s
    50.1% complete
  RUNNING for 2702.4s
    51.1% complete
  RUNNING for 2734.6s
    51.5% complete
  RUNNING for 2766.0s
    52.0% complete
  RUNNING for 2797.8s
    52.4% complete
  RUNNING for 2829.1s
    52.8% complete
  RUNNING for 2860.4s
    53.1% complete
  RUNNING for 2892.0s
    53.9% complete
  RUNNING for 2924.0s
    54.3% complete
  RUNNING for 2954.8s
    58.3% complete
  RUNNING for 2986.4s
    59.9% complete
  RUNNING for 3018.0s
    60.1% complete
  RUNNING for 3049.8s
    78.5% complete
  RUNNING for 3081.5s
   100.0% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-3SQKHHVKFXV9M on ec
2-54-183-228-122.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-3SQKHHVKFXV9M/syslog.2016-06-19-18
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
```

```
s.com/mnt/var/log/hadoop/steps/s-3SQKHHVKFXV9M/syslog
Counters: 56
        File Input Format Counters
                Bytes Read=2156069116
        File Output Format Counters
                Bytes Written=8253473
        File System Counters
                FILE: Number of bytes read=26146092
                FILE: Number of bytes written=98175924
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=23450
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=190
                HDFS: Number of write operations=0
                S3: Number of bytes read=2156069116
                S3: Number of bytes written=8253473
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=188
                Killed reduce tasks=1
                Launched map tasks=190
                Launched reduce tasks=12
                Other local map tasks=2
                Total megabyte-seconds taken by all map tasks=635866272
00
                Total megabyte-seconds taken by all reduce tasks=314973
01440
                Total time spent by all map tasks (ms)=44157380
                Total time spent by all maps in occupied slots (ms)=198
7082100
                Total time spent by all reduce tasks (ms)=10936563
                Total time spent by all reduces in occupied slots (ms)=
984290670
                Total vcore-seconds taken by all map tasks=44157380
                Total vcore-seconds taken by all reduce tasks=10936563
        Map-Reduce Framework
                CPU time spent (ms)=26316040
                Combine input records=89901041
                Combine output records=1800897
                Failed Shuffles=0
                GC time elapsed (ms)=65848
                Input split bytes=23450
                Map input records=58682266
                Map output bytes=2604755159
                Map output materialized bytes=50913871
                Map output records=89901041
                Merged Map outputs=2090
                Physical memory (bytes) snapshot=117924012032
                Reduce input groups=9588
                Reduce input records=1800897
                Reduce output records=9588
                Reduce shuffle bytes=50913871
```

```
                Shuffled Maps =2090
                Spilled Records=3601794
                Total committed heap usage (bytes)=125491478528
                Virtual memory (bytes) snapshot=409768972288
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/stripeMaker_
jac.Alex.20160619.182649.857818/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/stripeMaker_jac.Alex.20160619.182649.857818...
Killing our SSH tunnel (pid 17929)
```

My stripe maker took 50 minutes running on a 4 node cluster, 1 master at m1.medium and 3 cores at m3.xlarge. This is by far my most intensive MRJob. This took 5 minutes longer than my stripe maker for the cosine similarities likely because this stripemaker outputted more information.

Copy the index locally and save it on S3

In [83]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_stripes2

# sync the files to a local directory and save them on s3
!aws s3 cp s3://aks-w261-hw5/out_5_4/ s3://aks-w261-hw5/5_4_stripes2/ --
recursive
!aws s3 sync s3://aks-w261-hw5/5_4_stripes2 /Users/Alex/Documents/Berkel
ey/1602Summer/W261/HW5/5_4_stripes2
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_stripes2
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_stripes2/_SUCCESS
```

```
copy: s3://aks-w261-hw5/out_5_4/_SUCCESS to s3://aks-w261-hw5/5_4_strip
es2/_SUCCESS
copy: s3://aks-w261-hw5/out_5_4/part-00003 to s3://aks-w261-hw5/5_4_str
ipes2/part-00003
copy: s3://aks-w261-hw5/out_5_4/part-00006 to s3://aks-w261-hw5/5_4_str
ipes2/part-00006
copy: s3://aks-w261-hw5/out_5_4/part-00005 to s3://aks-w261-hw5/5_4_str
ipes2/part-00005
copy: s3://aks-w261-hw5/out_5_4/part-00000 to s3://aks-w261-hw5/5_4_str
ipes2/part-00000
copy: s3://aks-w261-hw5/out_5_4/part-00008 to s3://aks-w261-hw5/5_4_str
ipes2/part-00008
copy: s3://aks-w261-hw5/out_5_4/part-00004 to s3://aks-w261-hw5/5_4_str
ipes2/part-00004
copy: s3://aks-w261-hw5/out_5_4/part-00007 to s3://aks-w261-hw5/5_4_str
ipes2/part-00007
copy: s3://aks-w261-hw5/out_5_4/part-00002 to s3://aks-w261-hw5/5_4_str
ipes2/part-00002
copy: s3://aks-w261-hw5/out_5_4/part-00001 to s3://aks-w261-hw5/5_4_str
ipes2/part-00001
copy: s3://aks-w261-hw5/out_5_4/part-00009 to s3://aks-w261-hw5/5_4_str
ipes2/part-00009
copy: s3://aks-w261-hw5/out_5_4/part-00010 to s3://aks-w261-hw5/5_4_str
ipes2/part-00010
download: s3://aks-w261-hw5/5_4_stripes2/_SUCCESS to 5_4_stripes2/_SUCC
ESS
download: s3://aks-w261-hw5/5_4_stripes2/part-00005 to 5_4_stripes2/par
t-00005
download: s3://aks-w261-hw5/5_4_stripes2/part-00006 to 5_4_stripes2/par
t-00006
download: s3://aks-w261-hw5/5_4_stripes2/part-00002 to 5_4_stripes2/par
t-00002
download: s3://aks-w261-hw5/5_4_stripes2/part-00004 to 5_4_stripes2/par
t-00004
download: s3://aks-w261-hw5/5_4_stripes2/part-00000 to 5_4_stripes2/par
t-00000
download: s3://aks-w261-hw5/5_4_stripes2/part-00009 to 5_4_stripes2/par
t-00009
download: s3://aks-w261-hw5/5_4_stripes2/part-00003 to 5_4_stripes2/par
t-00003
download: s3://aks-w261-hw5/5_4_stripes2/part-00010 to 5_4_stripes2/par
t-00010
download: s3://aks-w261-hw5/5_4_stripes2/part-00001 to 5_4_stripes2/par
t-00001
download: s3://aks-w261-hw5/5_4_stripes2/part-00008 to 5_4_stripes2/par
t-00008
download: s3://aks-w261-hw5/5_4_stripes2/part-00007 to 5_4_stripes2/par
t-00007
_SUCCESS   part-00001 part-00003 part-00005 part-00007 part-00009
part-00000 part-00002 part-00004 part-00006 part-00008 part-00010
```

In [84]:
```
# save a portion of one of the files for testing the next MRJob
!head -500 ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_stripes2/part-00000 > test_stripes2
print "Testing stripes for Jaccard created."
```

```
Testing stripes for Jaccard created.
```

Invert the index through MRJob

```
%%writefile invert_jac.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
import ast
import math

class invert_jac(MRJob):

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper=self.mapper,\
                       reducer=self.reducer)]


    # our mapper reads in each line of data
    # and inverts each stripe
    def mapper(self, _, line):

        # we use a try in case in case any
        # data go written in a weird format
        try:
            # split the line by the tabs, set the
            # word and its cooccurence words
            line = line.split("\t")
            word = line[0].strip().strip("\"").strip("'")
            other_words = line[1].strip().strip("\"").strip("'")
            other_words = ast.literal_eval(other_words)


            # loop through each of the other words
            for other_word in other_words:

                # make sure we don't grab a star
                # word
                if other_word[0] != "*":

                    # create a dictionary for the
                    # invert, holds the word and
                    # inverted value, and that words
                    # star value
                    inverted_value = \
                    other_words[other_word]
                    word_star = "*"+word
                    word_star_val = other_words[word_star]

                    # set the dictionary
                    invert = {word:inverted_value,\
                              word_star:word_star_val}

                    # yield the inverted word with
                    # its inverted value
                    yield other_word, str(invert)

        except:
            pass
```

```
    # our reducer combines the inverted values
    # for multiple words
    def reducer(self, word, cooccurs):

        # create a dictionary to hold the
        # co-occurence counts for each word
        cooccur = {}

        # loop through the coocurrences
        # dictionaries, combining as we
        # go along
        for _cooccur in cooccurs:

            # read in the dictionary
            _cooccur = ast.literal_eval(_cooccur)

            # loop through each other word
            for other_word in _cooccur:

                # add the element to the dictionary
                if other_word not in cooccur:
                    cooccur[other_word] = 0

                # add the values to the dictionary
                cooccur[other_word] = cooccur[other_word] + \
                _cooccur[other_word]

        # yield the word and its cooccruence
        yield word, str(cooccur)

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

```
Overwriting invert_jac.py
```

Unit test

In [11]:
```
!python invert_jac.py test_stripes2 > temp.txt
!head -1 temp.txt
```

```
Using configs in /Users/Alex/.mrjob.conf
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/invert_jac.Alex.20160620.233201.061278
Running step 1 of 1...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/invert_jac.Alex.20160620.233201.061278/output...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/invert_jac.Alex.20160620.233201.061278...
"abnormalities" "{'indicated': 141, '*expected': 10810318, '*genesis':
 439326, '*glucose': 435882, 'epithelial': 205, 'female': 320, '*connec
tive': 334489, '*indicated': 2714938, 'carbonate': 50, 'connective': 4
8, '*gastric': 303646, '*intrinsic': 470104, 'expected': 70, 'genesis':
 76, 'glucose': 94, 'dominant': 49, '*dominant': 1338319, 'facial': 35
2, 'intrinsic': 176, 'gastric': 98, '*facial': 242173, '*female': 20747
76, '*epithelial': 209755, '*carbonate': 246688}"
```

Invert the index on the cloud for the entire corpus

In [93]:
```
# run on the cloud
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_4
!python invert_jac.py -r emr s3://aks-w261-hw5/5_4_stripes2/ \
    --cluster-id=j-25Y2I2CYBIZC3 \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_4 \
    --no-output
```

```
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/invert_jac.Alex.20160619.193554.228592
Copying local files to s3://mrjob-f8c316b67324528f/tmp/invert_jac.Alex.
20160619.193554.228592/files/...
Adding our job to existing cluster j-25Y2I2CYBIZC3
Waiting for step 1 of 1 (s-353PE6R70CQLB) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40140/cluster
  RUNNING for 1.3s
   100.0% complete
  RUNNING for 33.7s
     5.0% complete
  RUNNING for 65.1s
    39.8% complete
  RUNNING for 96.1s
   100.0% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-353PE6R70CQLB on ec
2-54-183-228-122.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-54-183-228-122.us-west-1.compute.amazonaw
s.com/mnt/var/log/hadoop/steps/s-353PE6R70CQLB/syslog
Counters: 54
        File Input Format Counters
                Bytes Read=8327276
        File Output Format Counters
                Bytes Written=9028528
        File System Counters
                FILE: Number of bytes read=6228713
                FILE: Number of bytes written=15991897
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2883
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=31
                HDFS: Number of write operations=0
                S3: Number of bytes read=8327276
                S3: Number of bytes written=9028528
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=31
                Launched map tasks=31
                Launched reduce tasks=11
                Total megabyte-seconds taken by all map tasks=994927680
                Total megabyte-seconds taken by all reduce tasks=566432
640
                Total time spent by all map tasks (ms)=690922
                Total time spent by all maps in occupied slots (ms)=310
91490
                Total time spent by all reduce tasks (ms)=196678
```

```
                Total time spent by all reduces in occupied slots (ms)=
17701020
                Total vcore-seconds taken by all map tasks=690922
                Total vcore-seconds taken by all reduce tasks=196678
        Map-Reduce Framework
                CPU time spent (ms)=80250
                Combine input records=0
                Combine output records=0
                Failed Shuffles=0
                GC time elapsed (ms)=10353
                Input split bytes=2883
                Map input records=9588
                Map output bytes=12472720
                Map output materialized bytes=5408277
                Map output records=246064
                Merged Map outputs=341
                Physical memory (bytes) snapshot=18333659136
                Reduce input groups=991
                Reduce input records=246064
                Reduce output records=991
                Reduce shuffle bytes=5408277
                Shuffled Maps =341
                Spilled Records=492128
                Total committed heap usage (bytes)=20689453056
                Virtual memory (bytes) snapshot=96741961728
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/invert_jac.A
lex.20160619.193554.228592/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/invert_jac.Alex.20160619.193554.228592...
Killing our SSH tunnel (pid 18079)
```

Store the files locally and transfer them to a safe place on S3

In [94]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_invert2

# sync the files to a local directory and save them on s3
!aws s3 cp s3://aks-w261-hw5/out_5_4/ s3://aks-w261-hw5/5_4_invert2/ --r
ecursive
!aws s3 sync s3://aks-w261-hw5/5_4_invert2 /Users/Alex/Documents/Berkele
y/1602Summer/W261/HW5/5_4_invert2
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_invert2
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_invert2/_SUCCESS
```

```
copy: s3://aks-w261-hw5/out_5_4/_SUCCESS to s3://aks-w261-hw5/5_4_inver
t2/_SUCCESS
copy: s3://aks-w261-hw5/out_5_4/part-00006 to s3://aks-w261-hw5/5_4_inv
ert2/part-00006
copy: s3://aks-w261-hw5/out_5_4/part-00007 to s3://aks-w261-hw5/5_4_inv
ert2/part-00007
copy: s3://aks-w261-hw5/out_5_4/part-00002 to s3://aks-w261-hw5/5_4_inv
ert2/part-00002
copy: s3://aks-w261-hw5/out_5_4/part-00004 to s3://aks-w261-hw5/5_4_inv
ert2/part-00004
copy: s3://aks-w261-hw5/out_5_4/part-00000 to s3://aks-w261-hw5/5_4_inv
ert2/part-00000
copy: s3://aks-w261-hw5/out_5_4/part-00003 to s3://aks-w261-hw5/5_4_inv
ert2/part-00003
copy: s3://aks-w261-hw5/out_5_4/part-00005 to s3://aks-w261-hw5/5_4_inv
ert2/part-00005
copy: s3://aks-w261-hw5/out_5_4/part-00008 to s3://aks-w261-hw5/5_4_inv
ert2/part-00008
copy: s3://aks-w261-hw5/out_5_4/part-00001 to s3://aks-w261-hw5/5_4_inv
ert2/part-00001
copy: s3://aks-w261-hw5/out_5_4/part-00009 to s3://aks-w261-hw5/5_4_inv
ert2/part-00009
copy: s3://aks-w261-hw5/out_5_4/part-00010 to s3://aks-w261-hw5/5_4_inv
ert2/part-00010
download: s3://aks-w261-hw5/5_4_invert2/_SUCCESS to 5_4_invert2/_SUCCES
S
download: s3://aks-w261-hw5/5_4_invert2/part-00004 to 5_4_invert2/part-
00004
download: s3://aks-w261-hw5/5_4_invert2/part-00006 to 5_4_invert2/part-
00006
download: s3://aks-w261-hw5/5_4_invert2/part-00009 to 5_4_invert2/part-
00009
download: s3://aks-w261-hw5/5_4_invert2/part-00001 to 5_4_invert2/part-
00001
download: s3://aks-w261-hw5/5_4_invert2/part-00005 to 5_4_invert2/part-
00005
download: s3://aks-w261-hw5/5_4_invert2/part-00000 to 5_4_invert2/part-
00000
download: s3://aks-w261-hw5/5_4_invert2/part-00007 to 5_4_invert2/part-
00007
download: s3://aks-w261-hw5/5_4_invert2/part-00003 to 5_4_invert2/part-
00003
download: s3://aks-w261-hw5/5_4_invert2/part-00002 to 5_4_invert2/part-
00002
download: s3://aks-w261-hw5/5_4_invert2/part-00008 to 5_4_invert2/part-
00008
download: s3://aks-w261-hw5/5_4_invert2/part-00010 to 5_4_invert2/part-
00010
_SUCCESS   part-00001 part-00003 part-00005 part-00007 part-00009
part-00000 part-00002 part-00004 part-00006 part-00008 part-00010
```

In [95]:
```
# save a chunk of the output for testing
# the next part
!head -250 ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_invert2/part-000
00 > jac_invert.txt
print "Created test file for jaccard inverted indexes"
```

```
Created test file for jacard inverted indexes
```

MRJob class for calculated the jaccard similarity for pairs

```
%%writefile jaccardSim.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
import ast
import math
from mrjob.protocol import RawProtocol

class jaccardSim(MRJob):

    MRJob.SORT_VALUES = True
    INTERNAL_PROTOCOL = RawProtocol
    OUTPUT_PROTOCOL = RawProtocol

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):

        # set how we want to sort, we use a single reducer
        # only so that we can do a total sort, we use multiple
        # reducers for all other steps
        JOBCONF = {
            'mapred.output.key.comparator.class': 'org.apache.hadoop.map
red.lib.KeyFieldBasedComparator',
            'mapred.text.key.comparator.options': '-k1,1nr',
            'mapreduce.job.reduces': 1,
        }

        return [MRStep(mapper=self.mapper,\
                       reducer=self.reducer),\
                MRStep(jobconf = JOBCONF,\
                       mapper=None,\
                       reducer=self.reducer_final)]


    # our mapper reads in each line of data
    # and outputs the partial aggregation
    # for each pair combination
    def mapper(self, _, line):

        # we use a try in case in case any
        # data go written in a weird format
        try:
            # split the line by the tabs, set the
            # word and its cooccurence words
            line = line.split("\t")
            key = line[0].strip().strip("\"").strip("'")
            words = line[1].strip().strip("\"").strip("'")
            words = ast.literal_eval(words)

            # gather the length of this word,
            # its cardinality
            key_length = 0
            for word in words:
                # exclude star cardinalities
                if word[0] != "*":
                    key_length = key_length + words[word]
```

```
            # loop through each of the words
            for word in words:

                # make sure it's not the star
                # word
                if word[0] != "*":

                    # set the star word
                    star_word = "*"+word

                    # as long as it isn't this
                    # word
                    if word != key:

                        # get the intersection value
                        intersection = words[word]

                        # get the star word's values
                        star_word_val = words[star_word]

                        # take the intersection divided
                        # by the total count of both
                        # words minus the intersection
                        similarity = float(intersection) / \
                        (float(star_word_val) + \
                         float(key_length) - \
                         float(intersection))

                        # set the pair
                        pair = key,word

                        # yeild the pair and its
                        # partial similarity
                        yield str(pair),str(similarity)

        except:
            pass


    # our reducer makes sure that all the
    # similarities are of a consistent length
    # for sorting
    def reducer(self, pair, similarity):

        # each word pair should only really
        # produce one similarity
        similarity=list(similarity)[0]

        # we want to make sure that the count
        # string is at least 15 characters, if
        # its not, we add leading zeros. this
        # helps us sort
        extend = "0"

        # convert density to a string
        similarity = str(similarity)
```

```
        # while the count lenght is less than 15
        while len(similarity) < 15:
            similarity = similarity + extend

        # yield the pair and its partial
        # similarity
        yield str(similarity),pair


    # our reducer final sorts these
    # similarities
    def reducer_final(self,similarity,pairs):

        # simply pass out each pair
        for pair in pairs:
            yield pair,similarity

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

```
Overwriting jaccardSim.py
```

Unit test locally

In [17]:
```
# test using the command line
!python jaccardSim.py jac_invert.txt > jac_prelim
!head jac_prelim
```

```
Using configs in /Users/Alex/.mrjob.conf
ignoring partitioner keyword arg (requires real Hadoop): 'org.apache.ha
doop.mapred.lib.KeyFieldBasedPartitioner'
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/jaccardSim.Alex.20160619.211947.817016
Running step 1 of 2...
Running step 2 of 2...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/jaccardSim.Alex.20160619.211947.817016/output...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/jaccardSim.Alex.20160619.211947.817016...
('schooling', 'americans')      0.00010000800064
('politically', 'ignored')      0.000100040593395
('dew', 'adam') 0.00010005769994
('illustrating', 'colored')     0.000100076516837
('softly', 'lap')       0.000100100851608
('stead', 'universal')  0.000100110488434
('softly', 'song')      0.0001001124993
('conditional', 'granting')     0.000100114386011
('schooling', 'esteem') 0.000100155240623
('imputed', 'suggestions')      0.000100168113472
```

Create a cluster and let's get this started in the cloud

In [14]:
```
# create the cluster
!mrjob create-cluster \
--max-hours-idle 1 \
--aws-region=us-west-1 -c ~/.mrjob.conf
```

```
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating persistent cluster to run several jobs in...
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/no_script.Alex.20160619.210835.478883
Copying local files to s3://mrjob-f8c316b67324528f/tmp/no_script.Alex.2
0160619.210835.478883/files/...
j-1WX5D4SG2HNQB
```

In [18]:
```
# run on the cloud
!aws s3 rm --recursive s3://aks-w261-hw5/out_5_4
!python jaccardSim.py -r emr s3://aks-w261-hw5/5_4_invert2/ \
    --cluster-id=j-1WX5D4SG2HNQB \
    --aws-region=us-west-1 \
    --output-dir=s3://aks-w261-hw5/out_5_4 \
    --no-output
```

```
delete: s3://aks-w261-hw5/out_5_4/_SUCCESS
delete: s3://aks-w261-hw5/out_5_4/part-00009
delete: s3://aks-w261-hw5/out_5_4/part-00010
delete: s3://aks-w261-hw5/out_5_4/part-00002
delete: s3://aks-w261-hw5/out_5_4/part-00000
delete: s3://aks-w261-hw5/out_5_4/part-00001
delete: s3://aks-w261-hw5/out_5_4/part-00005
delete: s3://aks-w261-hw5/out_5_4/part-00003
delete: s3://aks-w261-hw5/out_5_4/part-00006
delete: s3://aks-w261-hw5/out_5_4/part-00007
delete: s3://aks-w261-hw5/out_5_4/part-00004
delete: s3://aks-w261-hw5/out_5_4/part-00008
Using configs in /Users/Alex/.mrjob.conf
Unexpected option hadoop from /Users/Alex/.mrjob.conf
Using s3://mrjob-f8c316b67324528f/tmp/ as our temp dir on S3
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/jaccardSim.Alex.20160619.212012.913464
Copying local files to s3://mrjob-f8c316b67324528f/tmp/jaccardSim.Alex.
20160619.212012.913464/files/...
Adding our job to existing cluster j-1WX5D4SG2HNQB
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Detected hadoop configuration property names that do not match hadoop v
ersion 2.4.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparato
r.class
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.o
ptions
mapred.text.key.partitioner.options: mapreduce.partition.keypartitione
r.options
Waiting for step 1 of 2 (s-2N4Z5ONFLOQ7G) to complete...
  Opening ssh tunnel to resource manager...
  Connect to resource manager at: http://localhost:40671/cluster
  RUNNING for 26.6s
     5.0% complete
  RUNNING for 59.1s
    32.3% complete
  RUNNING for 90.4s
   100.0% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-2N4Z5ONFLOQ7G on ec
2-52-53-211-37.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-52-53-211-37.us-west-1.compute.amazonaws.
com/mnt/var/log/hadoop/steps/s-2N4Z5ONFLOQ7G/syslog
Counters: 54
        File Input Format Counters
                Bytes Read=9209186
        File Output Format Counters
```

```
                Bytes Written=10631276
        File System Counters
                FILE: Number of bytes read=6139305
                FILE: Number of bytes written=16434961
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2576
                HDFS: Number of bytes written=10631276
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=89
                HDFS: Number of write operations=22
                S3: Number of bytes read=9209186
                S3: Number of bytes written=0
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=28
                Launched map tasks=28
                Launched reduce tasks=11
                Total megabyte-seconds taken by all map tasks=945288000
                Total megabyte-seconds taken by all reduce tasks=463320
000
                Total time spent by all map tasks (ms)=656450
                Total time spent by all maps in occupied slots (ms)=295
40250
                Total time spent by all reduce tasks (ms)=160875
                Total time spent by all reduces in occupied slots (ms)=
14478750
                Total vcore-seconds taken by all map tasks=656450
                Total vcore-seconds taken by all reduce tasks=160875
        Map-Reduce Framework
                CPU time spent (ms)=213900
                Combine input records=0
                Combine output records=0
                Failed Shuffles=0
                GC time elapsed (ms)=10667
                Input split bytes=2576
                Map input records=991
                Map output bytes=10631188
                Map output materialized bytes=6231307
                Map output records=246064
                Merged Map outputs=308
                Physical memory (bytes) snapshot=16435113984
                Reduce input groups=246064
                Reduce input records=246064
                Reduce output records=246064
                Reduce shuffle bytes=6231307
                Shuffled Maps =308
                Spilled Records=492128
                Total committed heap usage (bytes)=18817220608
                Virtual memory (bytes) snapshot=90734755840
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
```

```
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Waiting for step 2 of 2 (s-1VFFAT6SVE4W7) to complete...
  RUNNING for 73.2s
    80.0% complete
  COMPLETED
Attempting to fetch counters from logs...
Looking for step log in /mnt/var/log/hadoop/steps/s-1VFFAT6SVE4W7 on ec
2-52-53-211-37.us-west-1.compute.amazonaws.com...
  Parsing step log: ssh://ec2-52-53-211-37.us-west-1.compute.amazonaws.
com/mnt/var/log/hadoop/steps/s-1VFFAT6SVE4W7/syslog
Counters: 54
        File Input Format Counters
                Bytes Read=10977523
        File Output Format Counters
                Bytes Written=10631276
        File System Counters
                FILE: Number of bytes read=6936953
                FILE: Number of bytes written=17481573
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=10982506
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=66
                HDFS: Number of write operations=0
                S3: Number of bytes read=0
                S3: Number of bytes written=10631276
                S3: Number of large read operations=0
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=33
                Launched map tasks=33
                Launched reduce tasks=1
                Total megabyte-seconds taken by all map tasks=114131232
0
                Total megabyte-seconds taken by all reduce tasks=628531
20
                Total time spent by all map tasks (ms)=792578
                Total time spent by all maps in occupied slots (ms)=356
66010
                Total time spent by all reduce tasks (ms)=21824
                Total time spent by all reduces in occupied slots (ms)=
1964160
                Total vcore-seconds taken by all map tasks=792578
                Total vcore-seconds taken by all reduce tasks=21824
        Map-Reduce Framework
                CPU time spent (ms)=72910
                Combine input records=0
                Combine output records=0
                Failed Shuffles=0
                GC time elapsed (ms)=10083
                Input split bytes=4983
                Map input records=246064
```

```
                Map output bytes=10631276
                Map output materialized bytes=6996645
                Map output records=246064
                Merged Map outputs=33
                Physical memory (bytes) snapshot=16387506176
                Reduce input groups=245673
                Reduce input records=246064
                Reduce output records=246064
                Reduce shuffle bytes=6996645
                Shuffled Maps =33
                Spilled Records=492128
                Total committed heap usage (bytes)=17554735104
                Virtual memory (bytes) snapshot=68166922240
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Removing s3 temp directory s3://mrjob-f8c316b67324528f/tmp/jaccardSim.A
lex.20160619.212012.913464/...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/jaccardSim.Alex.20160619.212012.913464...
Killing our SSH tunnel (pid 717)
```

Copy the jaccard similarities to a local file and save to a bucket on S3

In [22]:
```
# make a directory to hold the longest ngram
# files
!mkdir /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_jaccardSim

# sync the files to a local directory and save them on s3
!aws s3 cp s3://aks-w261-hw5/out_5_4/ s3://aks-w261-hw5/5_4_jaccardSim/
 --recursive
!aws s3 sync s3://aks-w261-hw5/5_4_jaccardSim /Users/Alex/Documents/Berk
eley/1602Summer/W261/HW5/5_4_jaccardSim
!ls ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_jaccardSim
!rm ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_jaccardSim/_SUCCESS
```

```
mkdir: /Users/Alex/Documents/Berkeley/1602Summer/W261/HW5/5_4_jaccardSi
m: File exists
copy: s3://aks-w261-hw5/out_5_4/_SUCCESS to s3://aks-w261-hw5/5_4_jacca
rdSim/_SUCCESS
copy: s3://aks-w261-hw5/out_5_4/part-00000 to s3://aks-w261-hw5/5_4_jac
cardSim/part-00000
download: s3://aks-w261-hw5/5_4_jaccardSim/_SUCCESS to 5_4_jaccardSim/_
SUCCESS
download: s3://aks-w261-hw5/5_4_jaccardSim/part-00000 to 5_4_jaccardSi
m/part-00000
_SUCCESS   part-00000
```

In [23]:
```
# preview the jaccard similarities
!head ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_jaccardSim/part-00000
```

```
('practitioners', 'esteem')     9.9999733334e-05
('practise', 'duties')  9.99953259975e-05
('lighting', 'unfortunate')     9.9993809907e-05
('defines', 'bureau')   9.9993173543e-05
('expects', 'submission')       9.9992857653e-05
('crucified', 'hill')   9.99925621296e-05
('economists', 'useful')        9.99914419089e-06
('kinship', 'seventh')  9.99908857865e-05
('israelites', 'possessions')   9.99902009603e-05
('sac', 'extra')        9.99900009999e-05
```

Finding the files

The Jaccard similarity scores can be found at s3://aks-w261-hw5/5_4_jaccardSim

---

HW 5.5 Evaluation of synonyms that your discovered

In this part of the assignment you will evaluate the success of you synonym detector (developed in response to HW5.4). Take the top 1,000 closest/most similar/correlative pairs of words as determined by your measure in HW5.4, and use the synonyms function in the accompanying python code:
nltk_synonyms.py

Note: This will require installing the python nltk package:
http://www.nltk.org/install.html (http://www.nltk.org/install.html)
and downloading its data with nltk.download().

For each (word1,word2) pair, check to see if word1 is in the list, synonyms(word2), and vice-versa. If one of the two is a synonym of the other, then consider this pair a 'hit', and then report the precision, recall, and F1 measure of your detector across your 1,000 best guesses. Report the macro averages of these measures.

NLTK's synonyms

We use the python code provided to check to see the synomyms of various words that we calculated. We copy the code of the python program into our notebook here because it is easier to read and understand if it's in the notebook. Lukily, it's short and won't take up too much space.

In [276]:
```
%%writefile nltk_synonyms.py
#!/usr/bin/python2.7
''' pass a string to this funciton ( eg 'car') and it will give you a li
st of
words which is related to cat, called lemma of CAT. '''
import nltk
from nltk.corpus import wordnet as wn
import sys
#print all the synset element of an element
def synonyms(string):
    syndict = {}
    for i,j in enumerate(wn.synsets(string)):
        syns = j.lemma_names()
        for syn in syns:
            syndict.setdefault(syn,1)
    return syndict.keys()
```

In [279]:
```
# let's test it the synonyms with a
# sample word, dog
print "Synonyms for the word dog"
print synonyms('dog')
```

```
Synonyms for the word dog
[u'go_after', u'chase_after', u'pawl', u'dog', u'wiener', u'tag', u'fra
nkfurter', u'hound', u'click', u'chase', u'andiron', u'hot_dog', u'tai
l', u'Canis_familiaris', u'give_chase', u'wienerwurst', u'bounder', u'd
omestic_dog', u'track', u'frank', u'trail', u'blackguard', u'weenie',
 u'frump', u'firedog', u'detent', u'dog-iron', u'cad', u'heel', u'hotdo
g']
```

Sort the similarity scores

In [27]:
```
# top 1,0000 words are considered synonyms
!head -1000 ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_jaccardSim/part
-00000 > SYNON_jaccard
!head -1000 ~/Documents/Berkeley/1602Summer/W261/HW5/5_4_cosineSim/part-
00000 > SYNON_cosine
print "Synonym files created"
```

```
Synonym files created
```

COSINE ACCURACY:

MRJob class to calculate the percision (FOR COSINE SIMILARITY)

We classify the top 1,000 words as synonyms. Before explaining how we calculate this, we define some terms:

- TP: true positive, our model placed this pair of words in the top 1,000 words and the NLTK corpus *agreed* that they were synonyms
- FP: false positive, our model placed this pair of words in the top 1,000 words and the NLTK corpus *disagreed* that they were synonyms

Percision can be calculated as TP / (TP + FP). In this case, we calculate the average percision value for each term that had a synonym.

```
%%writefile percision.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
from nltk_synonyms import synonyms
import mrjob
import numpy as np

import ast
import math

class percision(MRJob):

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper=self.mapper,\
                       reducer=self.reducer)]


    # our mapper reads in each line of data
    # and outputs whether the record is a
    # true positive or a false positive
    def mapper(self, _, line):

        # split the line by the tabs, set the
        # word and its cooccurence words
        line = line.split("\t")
        word1,word2 = ast.literal_eval(line[0])
        simil = float(line[2])

        # find the synonyms for the words
        synon1 = synonyms(word1)
        synon2 = synonyms(word2)

        # only do all this work if NLTK
        # actually has a synonym for this
        # word
        if len(synon1) > 0:

            # set a boolean to tell if its a
            # true synonym
            correct = False

            # check to see if any of the synonyms
            # match the either word
            for word in synon1:
                if word == word2:
                    correct = True
            for word in synon2:
                if word == word1:
                    correct = True


            # if we're correct yield a true
            # positive, otherwise yield a false
            # positive, for both words, either
```

```
            # the pair is a hit or its not
            if correct:
                yield word1,[1,0]
                yield word2,[1,0]
            else:
                yield word1,[0,1]
                yield word2,[0,1]


    # our reducer combines the values
    # of the true and false positives
    # and sets the class indicator
    def reducer(self, word, counts):

        # set an array to store the counts
        # for each word, accuracy counts
        accuracy = [0]*2
        accuracy = np.array(accuracy)

        # loop through each count
        for count in counts:

            # convert each count to a numpy
            # array and sum it with our existing
            # totals
            count = np.array(count)
            accuracy = accuracy + count

        # send the true positives and false
        # positives
        TP = float(accuracy[0])
        FP = float(accuracy[1])

        # calculate the average percision
        # for the word
        precis = TP / (TP + FP)

        # pull the the number of synonyms
        # for this word
        total_synon = len(synonyms(word))

        # only emit for words that actually
        # have a chance of having a synonym
        if total_synon > 0:

            # recall is the number of correct over
            # total number of synonyms
            recall = float(TP)/float(total_synon)

            # we define the F1 score as
            # 2 * percisions * recall, all divided
            # by percision plus the recall
            if precis + recall !=0:
                f1_score = (2 * precis * recall) / \
                (precis + recall)
            else:
                f1_score = "NA"
```

```
            # turn the precision, recall, and f1
            # score into a tuple
            scoring = precis,recall,f1_score

            # yield the average percision
            yield word, scoring


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

```
Overwriting percision.py
```

Calculate the accuracy of cosine similarities

Important to note that for a job this small, we don't need to run it on the cluster and spend resources. Our local machine can easily handle this task.

In [60]:
```
# create a smaller data set
!head -250 SYNON_jaccard > SYNON_jaccard_test
!head -250 SYNON_cosine > SYNON_cosine_test
print "Testing files created"
```

```
Testing files created
```

In [68]:
```
# unit test the cosine file
!python percision.py SYNON_cosine_test > synon_output_test
!cat synon_output_test | sort -k3,3nr | head
```

```
Using configs in /Users/Alex/.mrjob.conf
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/percision.Alex.20160619.222031.989378
Running step 1 of 1...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/percision.Alex.20160619.222031.989378/output...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/percision.Alex.20160619.222031.989378...
"tumor" [1.0, 0.6666666666666666, 0.8]
"tumors"        [1.0, 0.6666666666666666, 0.8]
"acetic"        [0.0, 0.0, "NA"]
"alkaline"      [0.0, 0.0, "NA"]
"amenable"      [0.0, 0.0, "NA"]
"ammonia"       [0.0, 0.0, "NA"]
"ammonium"      [0.0, 0.0, "NA"]
"amplifier"     [0.0, 0.0, "NA"]
"anterior"      [0.0, 0.0, "NA"]
"armature"      [0.0, 0.0, "NA"]
```

Run on the whole data set

In [77]:
```
# run on the full data set
!python percision.py SYNON_cosine > synon_output_cosine
!cat synon_output_test | sort -k3,3nr | head
```

```
Using configs in /Users/Alex/.mrjob.conf
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/percision.Alex.20160619.222806.738585
Running step 1 of 1...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/percision.Alex.20160619.222806.738585/output...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/percision.Alex.20160619.222806.738585...
"abdominal"     [0.0, 0.0, "NA"]
"accent"        [0.0, 0.0, "NA"]
"accompaniment" [0.0, 0.0, "NA"]
"accomplished"  [0.0, 0.0, "NA"]
"accustomed"    [0.0, 0.0, "NA"]
"acquainted"    [0.0, 0.0, "NA"]
"administering" [0.0, 0.0, "NA"]
"adversely"     [0.0, 0.0, "NA"]
"advertisement" [0.0, 0.0, "NA"]
"adviser"       [0.0, 0.0, "NA"]
```

JACCARD ACCURACY:

MRJob class for jaccard similarity scoring

We have already done cosine scoring. Now let's move on to Jaccard similarity.

```
%%writefile percisionJac.py
# import the MRJob class
from mrjob.job import MRJob
from mrjob.step import MRStep
from nltk_synonyms import synonyms
import mrjob
import numpy as np

import ast
import math

class percisionJac(MRJob):

    # define the steps of the job and the order in
    # which they will be executed
    def steps(self):
        return [MRStep(mapper=self.mapper,\
                       reducer=self.reducer)]


    # our mapper reads in each line of data
    # and outputs whether the record is a
    # true positive or a false positive
    def mapper(self, _, line):

        # split the line by the tabs, set the
        # word and its cooccurence words
        line = line.split("\t")
        word1,word2 = ast.literal_eval(line[1])
        simil = float(line[2])

        # find the synonyms for the words
        synon1 = synonyms(word1)
        synon2 = synonyms(word2)

        # only do all this work if NLTK
        # actually has a synonym for this
        # word
        if len(synon1) > 0:

            # set a boolean to tell if its a
            # true synonym
            correct = False

            # check to see if any of the synonyms
            # match the either word
            for word in synon1:
                if word == word2:
                    correct = True
            for word in synon2:
                if word == word1:
                    correct = True


            # if we're correct yield a true
            # positive, otherwise yield a false
            # positive, for both words, either
```

```
            # the pair is a hit or its not
            if correct:
                yield word1,[1,0]
                yield word2,[1,0]
            else:
                yield word1,[0,1]
                yield word2,[0,1]


    # our reducer combines the values
    # of the true and false positives
    # and sets the class indicator
    def reducer(self, word, counts):

        # set an array to store the counts
        # for each word, accuracy counts
        accuracy = [0]*2
        accuracy = np.array(accuracy)

        # loop through each count
        for count in counts:

            # convert each count to a numpy
            # array and sum it with our existing
            # totals
            count = np.array(count)
            accuracy = accuracy + count

        # send the true positives and false
        # positives
        TP = float(accuracy[0])
        FP = float(accuracy[1])

        # calculate the average percision
        # for the word
        precis = TP / (TP + FP)

        # pull the the number of synonyms
        # for this word
        total_synon = len(synonyms(word))

        # only emit for words that actually
        # have a chance of having a synonym
        if total_synon > 0:

            # recall is the number of correct over
            # total number of synonyms
            recall = float(TP)/float(total_synon)

            # we define the F1 score as
            # 2 * percisions * recall, all divided
            # by percision plus the recall
            if precis + recall !=0:
                f1_score = (2 * precis * recall) / \
                (precis + recall)
            else:
                f1_score = "NA"
```

```
            # turn the precision, recall, and f1
            # score into a tuple
            scoring = precis,recall,f1_score

            # yield the average percision
            yield word, scoring


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

```
Overwriting percisionJac.py
```

Unit test

In [75]:
```
# unit test the cosine file
!python percisionJac.py SYNON_jaccard_test > synon_output_test
!cat synon_output_test | sort -k3,3nr | head
```

```
Using configs in /Users/Alex/.mrjob.conf
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/percisionJac.Alex.20160619.222504.010377
Running step 1 of 1...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/percisionJac.Alex.20160619.222504.010377/output...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/percisionJac.Alex.20160619.222504.010377...
"abdominal"     [0.0, 0.0, "NA"]
"accent"        [0.0, 0.0, "NA"]
"accompaniment" [0.0, 0.0, "NA"]
"accomplished"  [0.0, 0.0, "NA"]
"accustomed"    [0.0, 0.0, "NA"]
"acquainted"    [0.0, 0.0, "NA"]
"administering" [0.0, 0.0, "NA"]
"adversely"     [0.0, 0.0, "NA"]
"advertisement" [0.0, 0.0, "NA"]
"adviser"       [0.0, 0.0, "NA"]
```

Run on the whole data set

In [76]:
```
# run the whole file
!python percisionJac.py SYNON_jaccard > synon_output_jaccard
!cat synon_output_test | sort -k3,3nr | head
```

```
Using configs in /Users/Alex/.mrjob.conf
Creating temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/percisionJac.Alex.20160619.222755.927357
Running step 1 of 1...
Streaming final output from /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm000
0gn/T/percisionJac.Alex.20160619.222755.927357/output...
Removing temp directory /var/folders/k8/fy2j66nj4xsczx6cbcxhjlvm0000gn/
T/percisionJac.Alex.20160619.222755.927357...
"abdominal"     [0.0, 0.0, "NA"]
"accent"        [0.0, 0.0, "NA"]
"accompaniment" [0.0, 0.0, "NA"]
"accomplished"  [0.0, 0.0, "NA"]
"accustomed"    [0.0, 0.0, "NA"]
"acquainted"    [0.0, 0.0, "NA"]
"administering" [0.0, 0.0, "NA"]
"adversely"     [0.0, 0.0, "NA"]
"advertisement" [0.0, 0.0, "NA"]
"adviser"       [0.0, 0.0, "NA"]
```

In [ ]: