DATASCI W261: Machine Learning at Scale

Nick Hamlin (nickhamlin@gmail.com)
Tigi Thomas (tgthomas@berkeley.edu)
Rock Baek (rockb1017@gmail.com)
Hussein Danish (husseindanish@gmail.com)

Time of Submission: 8:23 AM EST, Friday, Feb 19, 2016
W261-3, Spring 2016
Week 5 Homework

Submission Notes:

- For each problem, we've included a summary of the question as posed in the instructions. In many cases, we have not included the full text to keep the final submission as uncluttered as possible. For reference, we've included a link to the original instructions in the "Useful Reference" below.
- Problem statements are listed in *italics*, while our responses are shown in plain text.
- We've included the full output of the mapreduce jobs in our responses so that counter results are shown. However, these don't always render nicely into PDF form. In these situations, please reference the complete rendered notebook on Github (https://github.com/nickhamlin/mids_261_homework/blob/master/HW5/MIDS-W261-2015-HWK-Week05-Hamlin-Thomas-Baek-Danish.ipynb)

Useful References:

- **Original Assignment Instructions (https://www.dropbox.com/sh/0cv65h44zylqwe3/AADyEEBMPGezLplMmNwAFIkba/h Questions.txt?dl=0)**

In [1]:
```
#Use this to make sure we reload the MrJob code when we make changes
%load_ext autoreload
%autoreload 2
```

HW5.0.

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

A data warehouse is a central repository of data coming from one or multiple sources. Current and historical data is kept in the data warehouse. This data is used for creating analytical reports that can help spread knowledge through an enterprise. Relational data is stored but increasing semi-structured data, suchs as logs and unstructured data, such as tweets, are also being kept. This forms a foundation for business intelligence and now data science. There exists different types of systems within a data warehouse depending on the data pipeline:

- Datamarts: simple form of data warehouse covering only one subject
- Online Analytical Processing (OLAP): offline data, logging data, used for reporting and modeling
- Online Transactional Processing (OLTP): online data, used in real time, used for offer serving
- Predictive analytics: offline batch modeling, real-time ad serving

A star schema is a model in which data are represented by facts and dimensions. The schema consists of one or more facts tables each referencing any number of dimension tables. The star schema gets its name from the resemblence of its physical model to a star shape. A dimension table is at the middle and is surrounded by dimension tables, the points of the star. Facts hold the measurable, quantitative data about a business while dimensions are descriptive attributes related to fact data. Examples of fact data include sales price, sale quantity, and time, distance, speed, and weight measurements. Related dimension attribute examples include product models, product colors, product sizes, geographic locations, and salesperson names.

The star schema is used when simple and convenient business reporting is required. Star schemas are denormalized and benefits of this include simpler queries (due to simpler join logic), query performance gains and faster aggregation when compared to normalized schemas.

HW 5.1

In the database world What is 3NF? Does machine learning use data in 3NF? If so why?

3NF refers to third normal form. 3NF is a type of normalization used in database design to reduce redudancy and ensure referential integrity. More specifically this means that data must be 2NF and there must exist no transitive functional dependency. Transitive functional dependency occurs when A -> B and B -> C leads to A -> C. For example, if we have a table with StudentID, zip code, state, country, we know that zip code is dependent on StudentID and state and country are dependent on zip code. Therefore state and country are dependent on StudentID. This is not 3NF. To make this data 3NF, a separate table can be created to store the zip code with state/country association and those last two columns can be dropped from the intial table.

Machine learning does use data in 3NF since the data is often in a very structured format and is smaller in size due to the lack of redundancy and duplication.

In what form does ML consume data?

Machine learning algorithms typically consume data from a file in tabular format, reading it line by line where each line represents a different input from the dataset and where each column represents one of its features. Therefore, having normalized data can be useful in order to get only the required data for the dataset.

Why would one use log files that are denormalized?

Using log files that are denormalized can help get the full picture of what is going on. For example, we might want to get information about a customer and his name. If our data were normalized, the log file might only contain the customer ID whereas denormalized data will allow us to also see his name.

HW 5.2

Problem Statement

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

Generating source data

We'll start by running a slightly modified version of the code from HW4 to generate our two sets of source data

In [3]:

```
%%writefile convert_msdata.py
#HW 4.2 (Recycled for 5.2) - Attach customer IDs to page view records

from csv import reader
with open('anonymous-msweb.data','rb') as f:
    data=f.readlines()

for i in reader(data):
    if i[0]=='C':
        visitor_id=i[1] #Store visitor id
        continue
    if i[0]=='V':
        print i[0]+','+i[1]+','+i[2]+',C,'+visitor_id #Append visitor_i
```

Writing convert_msdata.py

In [4]:

```
%%writefile create_urls.py
#HW 4.2 (Recycled for 5.2) - Extract URLs (not explicitly required, but

#Save only results from 'A' rows into their own file for easy URL acces
from csv import reader
with open('anonymous-msweb.data','rb') as f:
    data=f.readlines()

for i in reader(data):
    if i[0]=='A':
        print i[1]+','+i[3]+','+i[4]
```

Writing create_urls.py

In [6]:
```
%%writefile freq_visitor.py
# HW 4.4 (Recycled for 5.2) - MRJob Code

import csv
from collections import Counter
from operator import itemgetter

from mrjob.job import MRJob
from mrjob.step import MRStep


def csv_readline(line):
    """Given a string CSV line, return a list of strings."""
    for row in csv.reader([line]):
        return row

class FreqVisitor(MRJob):

    def mapper_extract_views(self, line_no, line):
        """Extracts the page that was visited and the visitor id"""
        cell = csv_readline(line)
        #Ignore any irrelevant messy data, though hopefully we don't ha
        if cell[0] == 'V':
            yield cell[1],cell[4]

    def reducer_load_urls(self):
        """Load file of page URLs into reducer memory"""
        with open('ms_urls.txt','rb') as f:
            urls=csv.reader(f.readlines())
        self.url_dict={}
        for i in urls:
            #Saving the URLs into a dictionary will make it easy to acc
            self.url_dict[int(i[0])]=i[2]

    def reducer_sum_views_by_visitor(self, vroots, visitor):
        """Summarizes visitor counts for each page,
        yields one record per page with the visitor responsible for
        the most views on that page"""
        visitors=Counter()
        for i in visitor:
            visitors[i]+=1 #Aggregate page views for all visitors
        output= max(visitors.iteritems(), key=itemgetter(1))[0] #Find v
        yield (str(vroots)),(output,visitors[output],self.url_dict[int(

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

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

```
Writing freq_visitor.py
```

In [ ]:

```
#Make files executable, convert data, and view some example results to
#!chmod +x convert_msdata.py create_urls.py
!python convert_msdata.py > clean_msdata.txt
!cat clean_msdata.txt | head -10
!python create_urls.py > ms_urls.txt
```

In [8]:

```
%%writefile freq_visitor_driver.py
#HW 4.4 - Driver Function
from freq_visitor import FreqVisitor
import csv

mr_job = FreqVisitor(args=['clean_msdata.txt','--file','ms_urls.txt'])
with mr_job.make_runner() as runner:
    runner.run()
    for line in runner.stream_output():
        output=mr_job.parse_output_line(line)
        print str(output[0])+'\t'+str(output[1][0])+'\t'+str(output[1][
```

```
Writing freq_visitor_driver.py
```

In [ ]:

```
#Make files executable, convert data, and view some example results to
!chmod +x freq_visitor_driver.py
!python freq_visitor_driver.py > freq_visitor_data.txt
```

HW 5.2 - Setting up the joins

Using MRJob, implement a hashside join (memory-backed map-side) for left,

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

Since we're doing a memory-backed map-side join, we want to load the smaller of the two datasets into memory. Therefore, we'll choose the list of most frequent visitors per page that we generated in 4.4 as our left table and the list of URLs as our right table that we'll load during the mapper_init step.

Please report the number of rows resulting from:

(1) Left joining Table Left with Table Right

In [106]:

```
%%writefile left_join.py
# HW 5.2A - Left join MRJob Code
import csv

from mrjob.job import MRJob
from mrjob.step import MRStep

class LeftJoin(MRJob):

    def mapper_init(self):
        """Load file of page URLs into reducer memory"""
        with open('ms_urls.txt','rb') as f:
            urls=csv.reader(f.readlines())
        self.url_dict={}
        for i in urls:
            #Saving the URLs into a dictionary will make it easy to acc
            self.url_dict[int(i[0])]=i[2]


    def mapper(self, _, line):
        """Extracts the page that was visited and the visitor id"""
        line=line.strip().split('\t')
        page=line[0]
        visitor=line[1]
        #This is the "Left Join" logic that ensures that a row will be
        #every row in the
        try:
            url=self.url_dict[int(page)]
        except KeyError:
            url='NONE'
        yield page,(visitor,url)

    def steps(self):
        return [MRStep(
                mapper_init=self.mapper_init,
                mapper=self.mapper,
                )]

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

```
Overwriting left_join.py
```

In [111]:
```
#HW 5.2 - Left Join Driver Function
from left_join import LeftJoin
import csv

mr_job = LeftJoin(args=['freq_visitor_data.txt','--file','ms_urls.txt']
number_of_rows=0
with mr_job.make_runner() as runner:
    runner.run()
    #print 'Page | Visitor ID | URL'
    for line in runner.stream_output():
        output=mr_job.parse_output_line(line)
        number_of_rows+=1
        #print output[0], output[1][0], output[1][1]

print "Left Join returned {0} results".format(str(number_of_rows))
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
```

```
Page | Visitor ID | URL
Left Join returned 285 results
```

(2) Right joining Table Left with Table Right

In [128]:

```
%%writefile right_join.py
# HW 5.2A - Left join MRJob Code
import csv

from mrjob.job import MRJob
from mrjob.step import MRStep

class RightJoin(MRJob):

    def mapper_init(self):
        """Load file of page URLs into memory"""
        with open('ms_urls.txt','rb') as f:
            urls=csv.reader(f.readlines())
        self.url_dict={}
        for i in urls:
            #Saving the URLs into a dictionary will make it easy to acc
            #the second term here is a flag to see if we've emitted a r
            self.url_dict[int(i[0])]=[i[2],0]


    def mapper(self, _, line):
        """Extracts the page that was visited and the visitor id"""
        line=line.strip().split('\t')
        page=line[0]
        visitor=line[1]
        #This is the "Inner Join" logic that emits a row for every reco
        #tables
        try:
            url=self.url_dict[int(page)][0]
            self.url_dict[int(page)][1]=1 #set flag to indicate we've e
            yield page,(visitor,url)
        except KeyError:
            pass


    def mapper_final(self):
        """emit any records in the right table we haven't seen yet"""
        for i in self.url_dict.iteritems():
            if i[1][1]==0:
                page=i[0]
                url=i[1][0]
                yield page,('NONE',url)

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

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

```
Overwriting right_join.py
```

In [129]:
```
#HW 5.2 - Right Join Driver Function
from right_join import RightJoin
import csv

mr_job = RightJoin(args=['freq_visitor_data.txt','--file','ms_urls.txt'
number_of_rows=0
with mr_job.make_runner() as runner:
    runner.run()
    #print 'Page | Visitor ID | URL'
    for line in runner.stream_output():
        output=mr_job.parse_output_line(line)
        number_of_rows+=1
        #print output[0], output[1][0], output[1][1]

print "Right Join returned {0} results".format(str(number_of_rows))
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:

Right Join returned 294 results
```

(3) Inner joining Table Left with Table Right

In [113]:

```
%%writefile inner_join.py
# HW 5.2A - Inner join MRJob Code
import csv

from mrjob.job import MRJob
from mrjob.step import MRStep

class InnerJoin(MRJob):

    def mapper_init(self):
        """Load file of page URLs into memory"""
        with open('ms_urls.txt','rb') as f:
            urls=csv.reader(f.readlines())
        self.url_dict={}
        for i in urls:
            #Saving the URLs into a dictionary will make it easy to acc
            #the second term here is a flag to see if we've emitted a r
            self.url_dict[int(i[0])]=[i[2],0]


    def mapper(self, _, line):
        """Extracts the page that was visited and the visitor id"""
        line=line.strip().split('\t')
        page=line[0]
        visitor=line[1]
        #This is the "Inner Join" logic that emits a row for every recc
        #tables
        try:
            url=self.url_dict[int(page)][0]
            self.url_dict[int(page)][1]=1 #set flag to indicate we've e
            yield page,(visitor,url)
        except KeyError:
            #Skip records that don't appear in both tables
            pass


    def steps(self):
        return [MRStep(
                mapper_init=self.mapper_init,
                mapper=self.mapper
            )]

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

Overwriting inner_join.py

In [114]:
```
#HW 5.2 - Inner Join Driver Function
from inner_join import InnerJoin
import csv

mr_job = InnerJoin(args=['freq_visitor_data.txt','--file','ms_urls.txt'
number_of_rows=0
with mr_job.make_runner() as runner:
    runner.run()
    #print 'Page | Visitor ID | URL'
    for line in runner.stream_output():
        output=mr_job.parse_output_line(line)
        number_of_rows+=1
        #print output[0], output[1][0], output[1][1]

print "Inner Join returned {0} results".format(str(number_of_rows))
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:

Inner Join returned 285 results
```

HW 5.2 Summary and Discussion of Results

To summarize, our joins yielded the following numbers of rows.

Join Type	Rows Returned
Left Join	285
Right Join	294
Inner Join	285

These results make sense, although the right table (the list of pages with corresponding URLs) turned out to be larger than the left table (the list of pages with most frequent visitors) because presumably some pages were never visited. Knowing this size difference, we might reconsider our choice of table to load into memory, since we'd like to keep the memory footprint as small as possible to enable scaling. In an example of this size, the difference doesn't matter much though. Furthermore, depending on the practical applications of this join, we suspect it might provide more versatility for future analyses to have the URLs stored in memory, since this information could then be recycled easily to answer other questions.

HW 5.3

HW 5.3 - Problem Statement

For the remainder of this assignment you will work with a large subset of the Google n-grams dataset

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

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

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

s3://filtered-5grams/

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 (count), i.e., unigrams
- Most/Least densely appearing words (count/pages_count) sorted in decreasing order of relative frequency (Hint: save to PART-000* and take the head -n 1000)
- Distribution of 5-gram sizes (counts) sorted in decreasing order of relative frequency. (Hint: save to PART-000* and take the head -n 1000)
- OPTIONAL Question: Plot the log-log plot of the frequency distributuion of unigrams. Does it follow power law distribution?

For more background see:

- https://en.wikipedia.org/wiki/Log%E2%80%93log_plot
(https://en.wikipedia.org/wiki/Log%E2%80%93log_plot)
- https://en.wikipedia.org/wiki/Power_law (https://en.wikipedia.org/wiki/Power_law)

HW 5.3 - Notes on our implementation

For jobs running on the complete dataset, we ran our code directly from the shell, rather than from within this notebook. This approach enabled us to continue working in the notebook while the job was running. In these situations, we show the code required to call the job, and then the outputs below.

Also, the problem was updated with the changes to the unit testing data after we had already performed our tests. To do this, we simply stripped the first several n-grams out of the corpus for testing into a file called 'testngrams.txt' (shown below). For 5.3, we show the results of this testing process as well as the overall results.

In [360]:
```
!cat testngrams.txt
```

```
A BILL FOR ESTABLISHING RELIGIOUS	59	59	54
A Biography of General George	92	90	74
A Case Study in Government	102	102	78
A Case Study of Female	447	447	327
A Case Study of Limited	55	55	43
A Child's Christmas in Wales	1099	1061	866
A Circumstantial Narrative of the	62	62	50
A City by the Sea	62	60	49
A Collection of Fairy Tales	123	117	80
A Collection of Forms of	116	103	82
A Commentary on his Apology	110	110	69
A Comparative Study of Juvenile	68	64	44
A Comparison of the Properties	72	72	60
A Conceptual Framework and the	91	91	67
A Conceptual Framework for Life	49	49	40
A Concise Bibliography of the	145	143	122
A Continuation of the Letters	52	51	40
A Critical Review and a	197	194	155
A Critique and a Guide	42	42	42
A Defence of the Royal	153	153	120
A Defence of the Short	245	234	163
A Discovery of the Real	253	251	206
A FURTHER LOOK AT THE	51	50	40
A Festschrift in Honour of	549	540	416
A Funny Dirty Little War	180	154	58
A Game of Cat's Cradle	86	86	71
A Guide to America's Censorship	98	98	76
A HANDBOOK ON THEODOLITE SURVEYING	61	61	61
A HISTORY OF TRAVEL IN	130	130	59
A History of Aerial Navigation	61	61	49
A History of Modern Southeast	169	169	134
A History of Postwar American	172	171	136
A History of Railroads in	125	123	85
A History of and for	58	58	53
A History of the Eurobond	59	58	41
A History of the United	24792	23136	14744
A History of the White	152	152	117
A Joint Report by the	94	89	82
A Journey Through Spain in	59	58	47
A Key to Bibliographical Study	56	56	46
A Key to the Arithmetic	79	79	79
A Lakota Woman Tells Her	51	51	40
A Life and Times of	191	191	174
A Longitudinal Study of Life	58	58	51
A Lovely Way to Spend	113	113	85
A MAN FOR ALL SEASONS	376	365	290
A MATHEMATICAL MODEL FOR THE	85	74	45
A Manual of Historical Literature	558	557	294
A Manual of Instruction in	284	284	257
A Manual on the Manipulation	131	131	129
```

HW 5.3.A - Longest N-Gram (by number of characters)

In [2]:
```
%%writefile longest_ngram.py
#HW 5.3.A - MRJob Definition
import csv

from mrjob.job import MRJob
from mrjob.step import MRStep

class LongestNgram(MRJob):

    def mapper(self, _, line):
        """Emit one record for each ngram length with its corresponding
        line=line.strip().split('\t')
        ngram=line[0]
        #We don't need keys here, since we want the overall max
        yield None,(len(ngram),ngram)

    def reducer(self, _, ngram_and_length):
        """Return only the ngram with the max character length"""
        yield None, max(ngram_and_length)

    def steps(self):
        return [
            MRStep(mapper=self.mapper
                   #Recycle the reducer for the combiner as well
                   ,combiner=self.reducer
                    ,reducer=self.reducer
                  )
        ]

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

```
Overwriting longest_ngram.py
```

In [3]:
```
#HW 5.3.A - Driver Function for Testing
from longest_ngram import LongestNgram

def run_5_3_A():
    mr_job = LongestNgram(args=['testngrams.txt'])
    with mr_job.make_runner() as runner:
        runner.run()
        for line in runner.stream_output():
            print mr_job.parse_output_line(line)

run_5_3_A()
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:

(None, [34, 'A HANDBOOK ON THEODOLITE SURVEYING'])
```

In [ ]:
```
# HW 5.3.A - Shell call for results on full dataset (job output not sho
! python ./longest_ngram.py \
    -r emr s3://filtered-5grams  \
    --conf-path ./mrjob.conf \
    --output-dir=s3://hamlin-mids-261/longest_ngram \
    --no-output \
    --no-strict-protocol
```

In [364]:
```
# HW 5.3.A - Download, extract, and print results
#! mkdir ./longest_ngram_output
#! aws s3 cp --recursive s3://hamlin-mids-261/longest_ngram ./longest_n
!echo "LONGEST NGRAM:"
!cat ./longest_ngram_output/part-* | sort -k2nr | head -1
```

```
LONGEST NGRAM:
null    [159, "ROPLEZIMPREDASTRODONBRASLPKLSON YHROACLMPARCHEYXMMIOU
DAVESAURUS PIOFPILOCOWERSURUASOGETSESNEGCP TYRAVOPSIFENGOQUAPIALLOBO
SKENUO OWINFUYAIOKENECKSASXHYILPOYNUAT"]
```

HW 5.3.B - Top 10 Most Frequent Words

In the job below, we tried several different approaches to optimizing the sorting process in EMR. Our original idea was to use the same approach that we took in HW3 with secondary sorts happening during the shuffle of a second job with identity mappers/reducers that ran after the primary job generated unsorted outputs. This code is included below, but is commented out.

However, this approach didn't work because MRJob applied the custom sorting criteria to ALL steps in the job. Therefore, an advanced shuffle in the second step job would break the partitioning in the first job. In the end, we settled on running the first job on its own, and then sorting the outputs separately afterwards.

With all that said, we finally figured out how to properly assign step-level job conf instructions in MRjob, but only after we'd almost completed the entire rest of the assignment. In the interest of time, we haven't gone back and reimplemented this here, but we'll be using it moving forward in other assignments. Please see our implementation of the jaccard similarity calcluation in HW 5.4 below for an example of this in action.

In [199]:

```
%%writefile most_freq_words.py
#HW 5.3.B - Most Frequent Words MRJob Definition
import csv
import re

from mrjob import conf
from mrjob.job import MRJob
from mrjob.step import MRStep

class MostFreqWords(MRJob):

    def mapper(self, _, line):
        counts = {}
        line.strip()
        #Parse fields from each line
        [ngram,count,pages,books] = re.split("\t",line)
        count = int(count)
        words = re.split(" ",ngram)
        for word in words:

            #We chose to lowercase the words for more meaningful totals
            #though this was before the assignment instructions were up
            counts.setdefault(word.lower(),0)
            counts[word.lower()] += count
        for word in counts.keys():
            yield word,counts[word]

    def combiner(self,word,count):
        yield word,sum(count)

    def reducer(self,word,count):
        yield word,sum(count)

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

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

```
Overwriting most_freq_words.py
```

In [200]:

```
#HW 5.3.B - Test Data Driver Function
from most_freq_words import MostFreqWords

def run_5_3_B():
    mr_job = MostFreqWords(args=['testngrams.txt'])
    with mr_job.make_runner() as runner:
        runner.run()
        for line in runner.stream_output():
            print mr_job.parse_output_line(line)

run_5_3_B()
```

```
WARNING:mrjob.sim:ignoring partitioner keyword arg (requires real Ha
doop): 'org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner'
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.compat:Detected hadoop configuration property names th
at do not match hadoop version 0.20:
The have been translated as follows
 mapreduce.job.output.key.comparator.class: mapred.output.key.compar
ator.class
mapreduce.partition.keypartitioner.options: mapred.text.key.partitio
ner.options
mapreduce.partition.keycomparator.options: mapred.text.key.comparato
r.options
WARNING:mrjob.compat:Detected hadoop configuration property names th
at do not match hadoop version 0.20:
The have been translated as follows
 mapreduce.job.output.key.comparator.class: mapred.output.key.compar
ator.class
mapreduce.partition.keypartitioner.options: mapred.text.key.partitio
ner.options
mapreduce.partition.keycomparator.options: mapred.text.key.comparato
r.options
WARNING:mrjob.compat:Detected hadoop configuration property names th
at do not match hadoop version 0.20:
The have been translated as follows
 mapreduce.job.output.key.comparator.class: mapred.output.key.compar
ator.class
mapreduce.partition.keypartitioner.options: mapred.text.key.partitio
ner.options
mapreduce.partition.keycomparator.options: mapred.text.key.comparato
r.options
WARNING:mrjob.compat:Detected hadoop configuration property names th
at do not match hadoop version 0.20:
The have been translated as follows
 mapreduce.job.output.key.comparator.class: mapred.output.key.compar
ator.class
mapreduce.partition.keypartitioner.options: mapred.text.key.partitio
ner.options
mapreduce.partition.keycomparator.options: mapred.text.key.comparato
r.options
WARNING:mrjob.compat:Detected hadoop configuration property names th
at do not match hadoop version 0.20:
The have been translated as follows
 mapreduce.job.output.key.comparator.class: mapred.output.key.compar
ator.class
mapreduce.partition.keypartitioner.options: mapred.text.key.partitio
ner.options
mapreduce.partition.keycomparator.options: mapred.text.key.comparato
```

```
r.options
WARNING:mrjob.compat:Detected hadoop configuration property names th
at do not match hadoop version 0.20:
The have been translated as follows
 mapreduce.job.output.key.comparator.class: mapred.output.key.compar
ator.class
mapreduce.partition.keypartitioner.options: mapred.text.key.partitio
ner.options
mapreduce.partition.keycomparator.options: mapred.text.key.comparato
r.options
```

```
('a', 32811)
('aerial', 61)
('all', 376)
("america's", 98)
('american', 172)
('and', 579)
('apology', 110)
('arithmetic', 79)
('at', 51)
('bibliographical', 56)
('bibliography', 145)
('bill', 59)
('biography', 92)
('by', 156)
('case', 604)
("cat's", 86)
('censorship', 98)
("child's", 1099)
('christmas', 1099)
('circumstantial', 62)
('city', 62)
('collection', 239)
('commentary', 110)
('comparative', 68)
('comparison', 72)
('conceptual', 140)
('concise', 145)
('continuation', 52)
('cradle', 86)
('critical', 197)
('critique', 42)
('defence', 398)
('dirty', 180)
('discovery', 253)
('establishing', 59)
('eurobond', 59)
('fairy', 123)
('female', 447)
('festschrift', 549)
('for', 627)
('forms', 116)
('framework', 140)
('funny', 180)
('further', 51)
```

```
('game', 86)
('general', 92)
('george', 92)
('government', 102)
('guide', 140)
('handbook', 61)
('her', 51)
('his', 110)
('historical', 558)
('history', 25718)
('honour', 549)
('in', 2348)
('instruction', 284)
('joint', 94)
('journey', 59)
('juvenile', 68)
('key', 135)
('lakota', 51)
('letters', 52)
('life', 298)
('limited', 55)
('literature', 558)
('little', 180)
('longitudinal', 58)
('look', 51)
('lovely', 113)
('man', 376)
('manipulation', 131)
('manual', 973)
('mathematical', 85)
('model', 85)
('modern', 169)
('narrative', 62)
('navigation', 61)
('of', 29443)
('on', 302)
('postwar', 172)
('properties', 72)
('railroads', 125)
('real', 253)
('religious', 59)
('report', 94)
('review', 197)
('royal', 153)
('sea', 62)
('seasons', 376)
('short', 245)
('southeast', 169)
('spain', 59)
('spend', 113)
('study', 786)
('surveying', 61)
('tales', 123)
('tells', 51)
```

```
('the', 26578)
('theodolite', 61)
```

In [ ]:

```
# HW 5.3.B - Shell code to run EMR job on test data (results not shown)
! python ./most_freq_words.py \
    -r emr s3://hamlin-mids-261/testngrams.txt  \
    --conf-path ./mrjob.conf \
    --output-dir=s3://hamlin-mids-261/test \
    --no-output \
    --no-strict-protocol
```

In [28]:

```
# HW 5.3.B - Shell code to run EMR job on full data

# Results are shown here only because this is where we learned that run
# directly in the notebook locks it and prevents work on other problems
# job finishes...which wasn't the most productive way to work!

! python ./most_freq_words.py \
    -r emr s3://filtered-5grams  \
    --conf-path ./mrjob.conf \
    --output-dir=s3://hamlin-mids-261/most_freq_words \
    --no-output \
    --no-strict-protocol
```

```
creating new scratch bucket mrjob-46dd62fe4baf7a13
using s3://mrjob-46dd62fe4baf7a13/tmp/ as our scratch dir on S3
creating tmp directory /var/folders/rz/drh189k95919thyy3gs3tq40000
0gn/T/most_freq_words.nicholashamlin.20160214.182523.935641
writing master bootstrap script to /var/folders/rz/drh189k95919thy
y3gs3tq400000gn/T/most_freq_words.nicholashamlin.20160214.182523.9
35641/b.py
creating S3 bucket 'mrjob-46dd62fe4baf7a13' to use as scratch spac
e
Copying non-input files into s3://mrjob-46dd62fe4baf7a13/tmp/most_
freq_words.nicholashamlin.20160214.182523.935641/files/
Waiting 5.0s for S3 eventual consistency
Creating Elastic MapReduce job flow
Job flow created with ID: j-293AF6MIHM0DD
Created new job flow j-293AF6MIHM0DD
Job launched 30.4s ago, status STARTING: Provisioning Amazon EC2 c
apacity
Job launched 60.7s ago, status STARTING: Provisioning Amazon EC2 c
apacity
```

In [365]:
```
# HW 5.3.B - Download, extract, and display results

#! mkdir ./freq_word_output
#! aws s3 cp --recursive s3://hamlin-mids-261/most_freq_words ./freq_wc
#!cat ./freq_word_output/part-* | sort -k2nr | head -10000 > ./most_fre
!echo "MOST FREQUENT WORDS"
!cat ./most_freq_words.txt | head -10
```

```
MOST FREQUENT WORDS
"the"   5490815394
"of"    3698583299
"to"    2227866570
"in"    1421312776
"a"     1361123022
"and"   1149577477
"that"  802921147
"is"    758328796
"be"    688707130
"as"    492170314
cat: stdout: Broken pipe
```

In [375]:
```
#Make another version of this file for use in 5.4 that excludes "stopwc
!cat ./most_freq_words.txt | head -10000 >most_freq_words_10K.txt
!cat ./most_freq_words_10K.txt | head -10
```

```
"the"   5490815394
"of"    3698583299
"to"    2227866570
"in"    1421312776
"a"     1361123022
"and"   1149577477
"that"  802921147
"is"    758328796
"be"    688707130
"as"    492170314
cat: stdout: Broken pipe
```

HW 5.3.C - Most/Least densely appearing words (count/pages_count) sorted in decreasing order of relative frequency

In [193]:
```
%%writefile word_density.py
#HW 5.3.C - Word Density MRJob Definition
from __future__ import division
import csv

from mrjob.job import MRJob
from mrjob.step import MRStep

class WordDensity(MRJob):

    def mapper(self, _, line):
        """Emit one record per word with corresponding count and page c
        line=line.strip().split('\t')
        ngram=line[0]
        count=line[1]
        page_count=line[2]
        for word in ngram.split(' '):
            yield word,(count,page_count)

    def combiner(self,word,count):
        """Aggregate intermediate word counts and page counts, but don'
        word_count=0
        page_count=0
        for record in count:
            word_count+=int(record[0])
            page_count+=int(record[1])
        yield word,(word_count,page_count)

    def reducer(self,word,count):
        """Final aggregation of word counts and page counts, then divid
        word_count=0
        page_count=0
        for record in count:
            word_count+=int(record[0])
            page_count+=int(record[1])
        yield word,word_count/page_count

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

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

```
Overwriting word_density.py
```

In [192]:

```
#HW 5.3.C - Test Data Driver Function
from word_density import WordDensity

def run_5_3_C():
    mr_job = WordDensity(args=['testngrams.txt'])
    with mr_job.make_runner() as runner:
        runner.run()
        for line in runner.stream_output():
            print mr_job.parse_output_line(line)

run_5_3_C()
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protoc
ols will be strict by default. It's recommended you run your job w
ith --strict-protocols or set up mrjob.conf as described at http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protoco
ls (https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict
-protocols)
WARNING:mrjob.runner:

('A', 1.0588044078925982)
('ALL', 1.0301369863013699)
('AT', 1.02)
('Aerial', 1.0)
("America's", 1.0)
('American', 1.0058479532163742)
('Apology', 1.0)
('Arithmetic', 1.0)
('BILL', 1.0)
('Bibliographical', 1.0)
('Bibliography', 1.013986013986014)
```

In [ ]:

```
#HW 5.3.C - Run full job on EMR (results not shown)
!python ./word_density.py \
    -r emr s3://filtered-5grams  \
    --conf-path ./mrjob.conf \
    --output-dir=s3://hamlin-mids-261/word_density \
    --no-output \
    --no-strict-protocol
```

In [367]:
```
# HW 5.3.C - Download, extract, and display results

#! mkdir ./word_density_output
#! aws s3 cp --recursive s3://hamlin-mids-261/word_density ./word_densi
!echo "HIGHEST DENSITY WORDS"
!cat ./word_density_output/part-* | sort -k2nr | head -10
!echo ""
!echo "LOWEST DENSITY WORDS"
!cat ./word_density_output/part-* | sort -k2nr | tail -10
```

```
HIGHEST DENSITY WORDS
"xxxx"  11.557291666666666
"NA"    10.161726044782885
"blah"  8.0741599073001158
"nnn"   7.5333333333333332
"nd"    6.5611436445056839
"ND"    5.4073642846747196
"oooooooooooooooo"      4.921875
"PIC"   4.7272727272727275
"llll"  4.5116279069767442
"LUTHER"        4.3494983277591972
sort: write failed: standard output: Broken pipe
sort: write error

LOWEST DENSITY WORDS
"zwitterionic"  1.0
"zydeco"        1.0
"zygomaticofacial"      1.0
"zygomaticotemporal"    1.0
"zygosity"      1.0
"zylindrischen" 1.0
"zymogens"      1.0
"zymophore"     1.0
"zymosan"       1.0
"zymosis"       1.0
```

HW 5.3.D - Distribution of 5-gram sizes (character length) sorted in decreasing order of relative frequency. E.g., count (using the count field) up how many times a 5-gram of 50 characters shows up. Plot the data graphically.

In [130]:

```python
%%writefile ngram_distribution.py
#HW 5.3.D - Ngram Distribution MRJob Definition
from __future__ import division
import csv

from mrjob.job import MRJob
from mrjob.step import MRStep

class NgramDistribution(MRJob):

    def mapper_init(self):
        self.count=0

    def mapper(self, _, line):
        """Emit records with ngrams and size"""
        line=line.strip().split('\t')
        ngram=line[0] #The text of the ngram
        size=len(ngram)
        ngram_count=int(line[1]) #The count of the ngram
        self.count+=ngram_count #Add the count to the running total of
        yield size,ngram_count #Yield the ngram and its count

    def mapper_final(self):
        """We needed this for the original statement of the problem, wh
        required relative frequencies.  Though we've left the step in p
        the result is no longer used."""
        yield '*count',self.count #Yield the total for order-inversion

    def reducer_init(self):
        self.total_count=None

    def reducer(self,size,ngram_count):
        total=sum(ngram_count)
        overall_total=None
        #Capture the totals for a relative frequency calcuation (no lon
        if size=='*count':
            overall_total=total
            self.total_count=total
        else:
            #Yield the character length and the number of ngrams with t
            #(relative freq. calculation is commented out)
            yield size,(total)#, total/self.total_count)

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

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

```
Overwriting ngram_distribution.py
```

In [131]:

```
#HW 5.3.D - Test Data Driver Function
from ngram_distribution import NgramDistribution

def run_5_3_D():
    mr_job = NgramDistribution(args=['testngrams.txt'])
    with mr_job.make_runner() as runner:
        runner.run()
        for line in runner.stream_output():
            print mr_job.parse_output_line(line)

run_5_3_D()
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:

(17, 62)
(19, 191)
(20, 58)
(21, 634)
(22, 1255)
(23, 25376)
(24, 347)
(25, 184)
(26, 994)
(27, 233)
(28, 1373)
(29, 630)
(30, 280)
(31, 215)
(33, 679)
(34, 61)
```

In [ ]:

```
#HW 5.3.D - Run test job on EMR (results not shown)
! python ./ngram_distribution.py \
    -r emr s3://hamlin-mids-261/testngrams.txt  \
    --conf-path ./mrjob.conf \
    --output-dir=s3://hamlin-mids-261/test \
    --no-output \
    --no-strict-protocol
```

In [ ]:

```
#HW 5.3.D - Run FULL job on EMR (results not shown)
! python ./ngram_distribution.py \
    -r emr s3://filtered-5grams  \
    --conf-path ./mrjob.conf \
    --output-dir=s3://hamlin-mids-261/ngram_distribution \
    --no-output \
    --no-strict-protocol
```

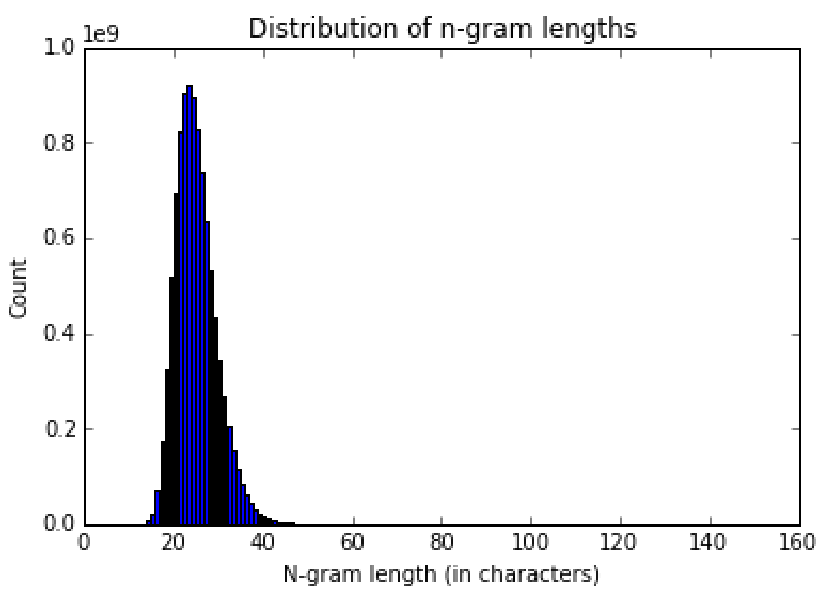
In [135]:

```
#HW 5.3.D - Download results
#! mkdir ./ngram_distribution_output
! aws s3 cp --recursive s3://hamlin-mids-261/ngram_distribution ./ngram
```

```
download: s3://hamlin-mids-261/ngram_distribution/_SUCCESS to ngram_
distribution_output/_SUCCESS
download: s3://hamlin-mids-261/ngram_distribution/part-00003 to ngra
m_distribution_output/part-00003
download: s3://hamlin-mids-261/ngram_distribution/part-00005 to ngra
m_distribution_output/part-00005
download: s3://hamlin-mids-261/ngram_distribution/part-00006 to ngra
m_distribution_output/part-00006
download: s3://hamlin-mids-261/ngram_distribution/part-00000 to ngra
m_distribution_output/part-00000
download: s3://hamlin-mids-261/ngram_distribution/part-00001 to ngra
m_distribution_output/part-00001
download: s3://hamlin-mids-261/ngram_distribution/part-00004 to ngra
m_distribution_output/part-00004
download: s3://hamlin-mids-261/ngram_distribution/part-00002 to ngra
m_distribution_output/part-00002
```

In [182]:

```
%matplotlib inline

#HW 5.3.D - Extract and visualize ngram distribution results

import os
import numpy as np
import matplotlib.pyplot as plt

def run_5_3_D():
    lengths=[]
    totals=[]
    for i in os.listdir('./ngram_distribution_output'):
        if i.startswith('part'):
            #Load results from each output file we downloaded
            with open(i) as f:
                for line in f.readlines():
                    [length,total]=line.strip().split('\t')
                    #Save lengths and totals into separate vectors for
                    lengths.append(int(length))
                    totals.append(int(total))

    fig, chart = plt.subplots()
    #We already know the bar heights, so we can plot them directly rath
    chart.bar(lengths,totals)
    chart.set_ylabel('Count')
    chart.set_xlabel('N-gram length (in characters)')
    chart.set_title('Distribution of n-gram lengths')

    fig = plt.gcf()

run_5_3_D()
```



HW 5.4

HW 5.4 - Problem Statement

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 1001,-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.

HW 5.4 - Implementation Notes:

Here, we continue to use the sample set of test ngrams we created in problem 5.3 to evaluate if our code is working. In addition, when calculating similarity, we also implemented the unit test dataset in the original problem statement to check that the intermediate steps of our jobs were working properly.

HW 5.4 - Building stripes of word co-occurrences

In [207]:

```
%%writefile stripes.py
#HW 5.4 - Stripes MRJob Definition
from __future__ import division
from itertools import combinations
import csv

from mrjob import conf
from mrjob.job import MRJob
from mrjob.step import MRStep

class Stripes(MRJob):

    def jobconf(self):
        orig_jobconf = super(Stripes, self).jobconf()
        # Setting these high enough improves EMR job speed
        custom_jobconf = {
            "mapred.map.tasks":28,
            "mapred.reduce.tasks":28
        }
        return conf.combine_dicts(orig_jobconf, custom_jobconf)


    def mapper_init(self):
        """Load file of words into memory"""
        self.word_dict={}
        #This is the file of words with frequency ranked 9000-10000 tha
        #created in HW 5.3.B
        with open('testwords.txt','rb') as f:
            for row in f.readlines():
                line=row.strip().split('\t')
                self.word_dict[line[0][1:-1]]=line[1]


    def mapper(self, _, line):
        """
        Emit co-occurrence combinations for each pair of relevant words
        """
        line=line.strip().split('\t')
        ngram=line[0].lower() #The full text of the ngram
        count=int(line[1]) #The count associated with it
        potential_words=ngram.split(" ") #List of individual words in c
        output={}
        #Pull out words from ngram that we care about (those that appea
        words=[i for i in potential_words if i in self.word_dict.keys()

        #Update output stripe for each combination of co-occurring, rel
        for word1,word2 in combinations(words,2):

            #This syntax does functionally the same thing as a Counter
            #but they aren't supported in Python 2.6.9, which is the de
            #that comes with the EMR AMIs.  Instead of fighting with AM
            #we decided it was easier to just implement the counter man

            if word1 in output.keys():
```

```
                output[word1][word2]=output[word1].get(word2,0)+count
            else:
                output[word1]={word2:count}

            #This second step ensures we maintain symmetry
            if word2 in output.keys():
                output[word2][word1]=output[word2].get(word1,0)+count
            else:
                output[word2]={word1:count}

        #"cooccurrences" is what I really want to call this second var,
        #but that's too much to type/spell reliably, so I'll settle for
        for word,cos in output.iteritems():
            yield word,cos

    def reducer(self,word,cos):
        """Aggregate stripes based on intermediate results from mapper"
        output_dict={}
        for co in cos:
            # The second_word variable here is so named to distinguish
            # and refers to the words in the co-occurrence stripe
            for second_word,count in co.iteritems():
                output_dict[second_word] = output_dict.get(second_word,
        yield word, output_dict


    def steps(self):
        return [
            MRStep(
                mapper_init=self.mapper_init,
                mapper=self.mapper
                #We can recycle the reducer as combiner here, which is
                ,combiner=self.reducer
                ,reducer=self.reducer
                  )
        ]

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

```
Overwriting stripes.py
```

We'd like to test this job locally on a small test set of data. To do that, we'll use a manually-created sample of the corpus (shown below).

In [205]:

```
! cat testngrams.txt
```

```
A BILL FOR ESTABLISHING RELIGIOUS	59	59	54
A Biography of General George	92	90	74
A Case Study in Government	102	102	78
A Case Study of Female	447	447	327
A Case Study of Limited	55	55	43
A Child's Christmas in Wales	1099	1061	866
A Circumstantial Narrative of the	62	62	50
A City by the Sea	62	60	49
A Collection of Fairy Tales	123	117	80
A Collection of Forms of	116	103	82
A Commentary on his Apology	110	110	69
A Comparative Study of Juvenile	68	64	44
A Comparison of the Properties	72	72	60
A Conceptual Framework and the	91	91	67
A Conceptual Framework for Life	49	49	40
A Concise Bibliography of the	145	143	122
A Continuation of the Letters	52	51	40
A Critical Review and a	197	194	155
A Critique and a Guide	42	42	42
A Defence of the Royal	153	153	120
A Defence of the Short	245	234	163
A Discovery of the Real	253	251	206
A FURTHER LOOK AT THE	51	50	40
A Festschrift in Honour of	549	540	416
A Funny Dirty Little War	180	154	58
A Game of Cat's Cradle	86	86	71
A Guide to America's Censorship	98	98	76
A HANDBOOK ON THEODOLITE SURVEYING	61	61	61
A HISTORY OF TRAVEL IN	130	130	59
A History of Aerial Navigation	61	61	49
A History of Modern Southeast	169	169	134
A History of Postwar American	172	171	136
A History of Railroads in	125	123	85
A History of and for	58	58	53
A History of the Eurobond	59	58	41
A History of the United	24792	23136	14744
A History of the White	152	152	117
A Joint Report by the	94	89	82
A Journey Through Spain in	59	58	47
A Key to Bibliographical Study	56	56	46
A Key to the Arithmetic	79	79	79
A Lakota Woman Tells Her	51	51	40
A Life and Times of	191	191	174
A Longitudinal Study of Life	58	58	51
A Lovely Way to Spend	113	113	85
A MAN FOR ALL SEASONS	376	365	290
A MATHEMATICAL MODEL FOR THE	85	74	45
A Manual of Historical Literature	558	557	294
A Manual of Instruction in	284	284	257
A Manual on the Manipulation	131	131	129
```

It's probably a safe bet that our test dataset doesn't contain too many co-occurrances of words in the 9001-10000 range, so we'll use a comparably simple arbirary test vocabulary to test our stripe creation code (again, shown below).

In [206]:
```
! cat testwords.txt
```

```
"honour"        549
"of"    447
"female"        447
"study" 447
"case"  447
"a"     376
"seasons"       376
"man"   376
"for"   376
"all"   376
"of"    284
"in"    284
"instruction"   284
"manual"        284
"a"     284
"real"  253
"the"   253
"a"     253
"of"    253
"discovery"     253
"theodolite"    61
```

With our test data in hand, we can now see how our stripes code performs.

In [208]:

```
#HW 5.4 - Driver function for local testing of stripes creation on test
from stripes import Stripes

def run_5_4_stripe_test():
    mr_job = Stripes(args=['testngrams.txt','--file','testwords.txt'])
    with mr_job.make_runner() as runner:
        runner.run()
        for line in runner.stream_output():
            print mr_job.parse_output_line(line)

run_5_4_stripe_test()
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protoc
ols will be strict by default. It's recommended you run your job w
ith --strict-protocols or set up mrjob.conf as described at http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protoco
ls (https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict
-protocols)
WARNING:mrjob.runner:

('a', {'a': 478, 'case': 604, 'all': 376, 'in': 2348, 'for': 627,
'of': 29443, 'study': 786, 'instruction': 284, 'manual': 973, 'the
odolite': 61, 'female': 447, 'honour': 549, 'real': 253, 'season
s': 376, 'the': 26578, 'discovery': 253, 'man': 376})
('all', {'a': 376, 'seasons': 376, 'for': 376, 'man': 376})
('case', {'a': 604, 'of': 502, 'study': 604, 'female': 447, 'in':
102})
('discovery', {'a': 253, 'of': 253, 'the': 253, 'real': 253})
('female', {'a': 447, 'case': 447, 'study': 447, 'of': 447})
('for', {'a': 627, 'all': 376, 'of': 58, 'seasons': 376, 'the': 8
5, 'man': 376})
```

This test yields stripes for each of the words appearing in our ngrams that also appears in our test vocabulary. Each stripe contains co-occurrences with other words in our vocabulary. By omitting words outside the vocabulary, we dramatically reduce the computational load on our job. This will be really important when we try to scale our implementation in EMR. Our next test is to run the same test as above, but in EMR to make sure we get the same results

In [ ]:

```
#HW 5.4 - Testing stripes calculation on test dataset in EMR (results n
! python ./stripes.py \
    -r emr s3://hamlin-mids-261/testngrams.txt  \
    --conf-path ./mrjob.conf \
    --file ./testwords.txt \
    --output-dir=s3://hamlin-mids-261/cooccurrence_stripes_TEST \
    --no-output \
    --no-strict-protocol
```

The results for the job above aren't shown since we ran it in the shell rather than the notebook, but examining the output on S3 confirms that we get the same results as our local test. This job runs in about 7 minutes on a single m1.small instance. With these results in hand, we can expand our job to the full dataset using the 9001-10000 word vocabulary.

In [ ]:
```
#HW 5.4 - Running stripes calculation on FULL dataset in EMR (results n
! python ./stripes.py \
    -r emr s3://filtered-5grams  \
    --conf-path ./mrjob.conf \
    --file ./most_freq_words_10K.txt \
    --output-dir=s3://hamlin-mids-261/cooccurrence_stripes_FINAL \
    --ec2-instance-type c1.medium \
    --num-ec2-instances 6 \
    --no-output \
    --no-strict-protocol
```

For our full-scale stripes creation, we're running on a 6-node c1.medium cluster in EMR and the job completes in just over 1 hour. In hindsight, we would've saved some money if we'd added one more to enable to job to finish in just under one hour, but that's life. For comparison purposes, we originally implemented the stripes a little differently (code is included in the Appendix for posterity) such that we created a stripe for all 10000 of the top words. On the same size cluster, this job would've run for over six hours. Clearly, omitting high frequency terms from the vocabularly made a huge difference in the scalability of our job.

HW 5.4 - Using stripes to calculate word similarities

After setting up two test sets, one based on the "unit test" in the original assignment and another based on the output of our stripes EMR test from above, we proceed to implement two versions of word similarity (cosine and jaccard) as the the example on slide 223 in the Week 5 deck.

In [216]:
```
%%writefile test.txt
('docA', {'X': 20, 'Y': 30, 'Z': 5})
('docB', {'X': 100, 'Y': 20})
('docC', {'M': 5, 'N': 20, 'Z': 5})
```

```
Overwriting test.txt
```

In [212]:
```
%%writefile test.txt
'docA'  {'X': 20, 'Y': 30, 'Z': 5}
'docB'  {'X': 100, 'Y': 20}
'docC'  {'M': 5, 'N': 20, 'Z': 5}
```

```
Overwriting test.txt
```

In [130]:
```
%%writefile stripes.txt
('a', {'a': 478, 'case': 604, 'all': 376, 'in': 2348, 'for': 627, 'of':
('all', {'a': 376, 'seasons': 376, 'for': 376, 'man': 376})
('case', {'a': 604, 'of': 502, 'study': 604, 'female': 447, 'in': 102})
('discovery', {'a': 253, 'of': 253, 'the': 253, 'real': 253})
('female', {'a': 447, 'case': 447, 'study': 447, 'of': 447})
('for', {'a': 627, 'all': 376, 'of': 58, 'seasons': 376, 'the': 85, 'ma
('honour', {'a': 549, 'of': 549, 'in': 549})
('in', {'a': 2348, 'case': 102, 'of': 1088, 'study': 102, 'instruction'
('instruction', {'a': 284, 'of': 284, 'manual': 284, 'in': 284})
('man', {'a': 376, 'seasons': 376, 'all': 376, 'for': 376})
('manual', {'a': 973, 'of': 842, 'the': 131, 'instruction': 284, 'in':
('of', {'a': 29443, 'real': 253, 'for': 58, 'of': 232, 'study': 628, 'i
('real', {'a': 253, 'of': 253, 'the': 253, 'discovery': 253})
('seasons', {'a': 376, 'all': 376, 'for': 376, 'man': 376})
('study', {'a': 786, 'case': 604, 'in': 102, 'female': 447, 'of': 628})
('the', {'a': 26578, 'real': 253, 'for': 85, 'of': 25985, 'manual': 131
('theodolite', {'a': 61})
```

```
Writing stripes.txt
```

HW 5.4 - Word similarity using inverted index and cosine similarity

There are some extra "init" and "final" steps in this job that have commented out print statements. By uncommenting these steps when we run the job locally on our unit test data, we can examine the intermediate steps of the calculation and ensure everything's working right. Afterwards, we comment these lines out again when running the job at scale so we don't print an overwhelming amount of intermediate results.

In [215]:

```
%%writefile synonyms.py

#HW 5.4 - Word similarity using inverted index and cosine similarity MR
from __future__ import division
from itertools import combinations
from ast import literal_eval
from math import sqrt
import csv

from mrjob.job import MRJob
from mrjob.step import MRStep

class Synonyms(MRJob):

    def mapper_cosine_inv_index(self, _, line):
        # We're reading these lines in from text files (from the previo
        # them back into objects we can use.
        line=line.strip().split('\t')
        word=literal_eval(line[0])
        cos=literal_eval(line[1]) #The stripes of co-occurrences for th

        #Because our test datasets have slightly different formatting t
        #stripes, we used a slightly different approach to reading the
        #(shown in the three commented lines below)

        #line=eval(line.strip())
        #word=line[0]
        #cos=line[1]

        stripe_length=len(cos) #How many co-occurrences does this word
        for word2, count in cos.iteritems():
            #If we didn't do this last step (of normalizing the length)
            yield word2, (word, 1/sqrt(stripe_length))

    def combiner_inv_index(self, word2, word_1_counts):
        #This relies on this combiner!
        yield word2, dict(word_1_counts)

    def reducer_inv_index_init(self):
        #print "INTERMEDIATE RESULTS - INVERTED INDEX"
        pass

    def reducer_inv_index(self,word,cos):
        """recycled from previous stripe creation job"""
        output_dict={}
        for co in cos:
            for second_word,count in co.iteritems():
                output_dict[second_word] = output_dict.get(second_word,
        #print word,output_dict
        yield word, output_dict

    def reducer_inv_index_final(self):
        #print " "
        pass
```

```
        pass

    def mapper_calculate_distance_init(self):
        #print "INTERMEDIATE RESULTS - PAIRS FROM POSTING LIST"
        pass

    def mapper_calculate_distance(self,word,stripe):
        words=[i for i in stripe.keys()]
        for word1,word2 in combinations(words,2): #CHECK THIS - PERMUTA
            #print (word1,word2),stripe[word1]*stripe[word2]
            yield (word1,word2),stripe[word1]*stripe[word2]

    def mapper_calculate_distance_final(self):
        #print " "
        pass

    def reducer_calculate_distance(self,words,distance):
        yield words,sum(distance)


    def steps(self):
        return [
            #The first step calculates the inverted index
            MRStep(
                mapper=self.mapper_cosine_inv_index
                ,combiner=self.combiner_inv_index
                ,reducer_init=self.reducer_inv_index_init
                ,reducer=self.reducer_inv_index
                ,reducer_final=self.reducer_inv_index_final
                  ),
            #The second step uses the inverted index and the cosine dis
            MRStep(
                mapper_init=self.mapper_calculate_distance_init,
                mapper=self.mapper_calculate_distance
                ,mapper_final=self.mapper_calculate_distance_final
                ,reducer=self.reducer_calculate_distance
                  )
        ]

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

```
Overwriting synonyms.py
```

HW 5.4 - Running cosine similarity jobs on increasingly complex datasets

Our first test is on the unit test data, with all intermediate steps shown so we can make sure things are working right.

In [300]:

```python
#HW 5.4 - Cosine similarity testing driver function
# BASIC UNIT TEST
from synonyms import Synonyms

def run_5_4_cosine():
    mr_job = Synonyms(args=['test.txt']) #Use the unit test data
    with mr_job.make_runner() as runner:
        runner.run()
        print "FINAL RESULTS"
        for line in runner.stream_output():
            print mr_job.parse_output_line(line)

run_5_4_cosine()
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:

INTERMEDIATE RESULTS - INVERTED INDEX
M {'docC': 0.5773502691896258}
N {'docC': 0.5773502691896258}
X {'docB': 0.7071067811865475, 'docA': 0.5773502691896258}
Y {'docB': 0.7071067811865475, 'docA': 0.5773502691896258}
Z {'docC': 0.5773502691896258, 'docA': 0.5773502691896258}

INTERMEDIATE RESULTS - PAIRS FROM POSTING LIST
('docB', 'docA') 0.408248290464
('docB', 'docA') 0.408248290464
('docC', 'docA') 0.333333333333

FINAL RESULTS
(['docB', 'docA'], 0.816496580927726)
(['docC', 'docA'], 0.3333333333333334)
```

Sure enough, these results reproduce exactly the example from the slides, which is what we're hoping for. Next, we can test our cosine distance similarity code on our sample stripes.

In [218]:

```
#HW 5.4 - Cosine similarity testing driver function
# TEST ON SAMPLE STRIPE OUTPUT
from synonyms import Synonyms

mr_job = Synonyms(args=['stripes.txt'])
with mr_job.make_runner() as runner:
    runner.run()
    print "FINAL RESULTS"
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
```

```
FINAL RESULTS
(['a', 'all'], 0.48507125007266594)
(['a', 'case'], 0.5423261445466404)
(['a', 'discovery'], 0.48507125007266594)
(['a', 'female'], 0.48507125007266594)
(['a', 'for'], 0.5940885257860047)
(['a', 'honour'], 0.420084025208403)
(['a', 'in'], 0.6416889479197478)
(['a', 'instruction'], 0.48507125007266594)
(['a', 'man'], 0.48507125007266594)
(['a', 'manual'], 0.5423261445466404)
(['a', 'of'], 0.8744746321952064)
(['a', 'real'], 0.48507125007266594)
(['a', 'seasons'], 0.48507125007266594)
(['a', 'study'], 0.5423261445466404)
(['a', 'the'], 0.5940885257860047)
(['a', 'theodolite'], 0.24253562503633297)
(['all', 'discovery'], 0.25)
(['all', 'female'], 0.25)
(['all', 'for'], 0.6123724356957946)
(['all', 'honour'], 0.2886751345948129)
(['all', 'in'], 0.1889822365046136)
(['all', 'instruction'], 0.25)
(['all', 'man'], 0.75)
(['all', 'manual'], 0.22360679774997896)
(['all', 'of'], 0.2773500981126146)
(['all', 'real'], 0.25)
(['all', 'seasons'], 0.5)
(['all', 'study'], 0.22360679774997896)
(['all', 'the'], 0.4082482904638631)
(['all', 'theodolite'], 0.5)
(['case', 'all'], 0.22360679774997896)
(['case', 'discovery'], 0.4472135954999579)
(['case', 'female'], 0.6708203932499369)
(['case', 'for'], 0.36514837167011077)
(['case', 'honour'], 0.7745966692414834)
(['case', 'in'], 0.50709255283711)
(['case', 'instruction'], 0.6708203932499369)
(['case', 'man'], 0.22360679774997896)
(['case', 'manual'], 0.6)
(['case', 'of'], 0.6201736729460423)
(['case', 'real'], 0.4472135954999579)
(['case', 'seasons'], 0.22360679774997896)
(['case', 'study'], 0.7999999999999999)
(['case', 'the'], 0.36514837167011077)
(['case', 'theodolite'], 0.4472135954999579)
(['discovery', 'man'], 0.25)
(['female', 'discovery'], 0.5)
(['female', 'in'], 0.1889822365046136)
(['female', 'man'], 0.25)
(['female', 'of'], 0.41602514716892186)
(['female', 'real'], 0.5)
(['female', 'seasons'], 0.25)
(['female', 'the'], 0.4082482904638631)
```

```
(['for', 'discovery'], 0.6123724356957946)
(['for', 'female'], 0.4082482904638631)
(['for', 'honour'], 0.4714045207910318)
(['for', 'instruction'], 0.4082482904638631)
(['for', 'man'], 0.6123724356957946)
(['for', 'manual'], 0.5477225575051662)
(['for', 'of'], 0.33968311024337877)
(['for', 'real'], 0.4082482904638631)
(['for', 'seasons'], 0.20412414523193154)
(['for', 'study'], 0.36514837167011077)
(['for', 'the'], 0.33333333333333334)
(['for', 'theodolite'], 0.4082482904638631)
(['honour', 'discovery'], 0.5773502691896258)
(['honour', 'female'], 0.5773502691896258)
(['honour', 'instruction'], 0.5773502691896258)
(['honour', 'man'], 0.2886751345948129)
(['honour', 'manual'], 0.5163977794943223)
(['honour', 'of'], 0.3202563076101743)
(['honour', 'real'], 0.5773502691896258)
(['honour', 'seasons'], 0.2886751345948129)
(['honour', 'study'], 0.5163977794943223)
(['honour', 'the'], 0.4714045207910318)
(['honour', 'theodolite'], 0.5773502691896258)
(['in', 'discovery'], 0.3779644730092272)
(['in', 'female'], 0.5669467095138407)
(['in', 'for'], 0.3086066999241838)
(['in', 'honour'], 0.4364357804719848)
(['in', 'instruction'], 0.3779644730092272)
(['in', 'man'], 0.1889822365046136)
(['in', 'manual'], 0.3380617018914066)
(['in', 'of'], 0.3144854510165755)
(['in', 'real'], 0.3779644730092272)
(['in', 'seasons'], 0.1889822365046136)
(['in', 'study'], 0.3380617018914066)
(['in', 'the'], 0.3086066999241838)
(['in', 'theodolite'], 0.3779644730092272)
(['instruction', 'discovery'], 0.5)
(['instruction', 'female'], 0.5)
(['instruction', 'honour'], 0.2886751345948129)
(['instruction', 'in'], 0.1889822365046136)
(['instruction', 'man'], 0.25)
(['instruction', 'manual'], 0.6708203932499369)
(['instruction', 'of'], 0.2773500981126146)
(['instruction', 'real'], 0.5)
(['instruction', 'seasons'], 0.25)
(['instruction', 'the'], 0.4082482904638631)
(['instruction', 'theodolite'], 0.5)
(['manual', 'discovery'], 0.6708203932499369)
(['manual', 'female'], 0.4472135954999579)
(['manual', 'honour'], 0.2581988897471611 5)
(['manual', 'in'], 0.1690308509457033)
(['manual', 'man'], 0.22360679774997896)
(['manual', 'of'], 0.2480694691784169)
(['manual', 'real'], 0.4472135954999579)
```

```
(['manual', 'seasons'], 0.22360679774997896)
(['manual', 'the'], 0.36514837167011077)
(['manual', 'theodolite'], 0.4472135954999579)
(['of', 'discovery'], 0.5547001962252291)
(['of', 'female'], 0.1386750490563073)
(['of', 'honour'], 0.16012815380508716)
(['of', 'in'], 0.4193139346887673)
(['of', 'instruction'], 0.2773500981126146)
(['of', 'man'], 0.2773500981126146)
(['of', 'manual'], 0.3721042037676254)
(['of', 'real'], 0.41602514716892186)
(['of', 'seasons'], 0.2773500981126146)
(['of', 'study'], 0.2480694691784169)
(['of', 'the'], 0.6793662204867574)
(['real', 'discovery'], 0.75)
(['real', 'for'], 0.20412414523193154)
(['real', 'man'], 0.25)
(['real', 'manual'], 0.22360679774997896)
(['real', 'of'], 0.1386750490563073)
(['real', 'seasons'], 0.25)
(['real', 'the'], 0.4082482904638631)
(['seasons', 'all'], 0.25)
(['seasons', 'discovery'], 0.25)
(['seasons', 'for'], 0.4082482904638631)
(['seasons', 'man'], 0.75)
(['seasons', 'the'], 0.4082482904638631)
(['study', 'discovery'], 0.4472135954999579)
(['study', 'female'], 0.6708203932499369)
(['study', 'honour'], 0.25819888974716115)
(['study', 'in'], 0.1690308509457033)
(['study', 'instruction'], 0.6708203932499369)
(['study', 'man'], 0.22360679774997896)
(['study', 'manual'], 0.6)
(['study', 'of'], 0.3721042037676254)
(['study', 'real'], 0.4472135954999579)
(['study', 'seasons'], 0.22360679774997896)
(['study', 'the'], 0.36514837167011077)
(['study', 'theodolite'], 0.4472135954999579)
(['the', 'discovery'], 0.6123724356957946)
(['the', 'in'], 0.1543033499620919)
(['the', 'instruction'], 0.20412414523193154)
(['the', 'man'], 0.4082482904638631)
(['the', 'real'], 0.20412414523193154)
(['theodolite', 'discovery'], 0.5)
(['theodolite', 'female'], 0.5)
(['theodolite', 'man'], 0.5)
(['theodolite', 'of'], 0.2773500981126146)
(['theodolite', 'real'], 0.5)
(['theodolite', 'seasons'], 0.5)
(['theodolite', 'the'], 0.4082482904638631)
```

We can repeat this same test in EMR to ensure the job scales properly (as before, we haven't shown the results of this test here since we ran it directly from the shell).

In [ ]:
```
# HW 5.4 - Running cosine similarity job on test dataset on EMR
python ./synonyms.py \
    -r emr s3://hamlin-mids-261/stripes.txt  \
    --conf-path ./mrjob.conf \
    --output-dir=s3://hamlin-mids-261/similarity \
    --no-output \
    --no-strict-protocol
```

Next, we do one final test to confirm that our code correctly aggregates results from the separate output files produced by our stripes job.

In [ ]:
```
# HW 5.4 - Running cosine similarity job on test dataset on EMR
python ./synonyms.py \
    -r emr s3://hamlin-mids-261/test_stripes/*  \
    --conf-path ./mrjob.conf \
    --output-dir=s3://hamlin-mids-261/similarity \
    --no-output \
    --no-strict-protocol
```

Finally, after our testing, we can compute cosine similarities on the full dataset.

In [ ]:
```
# HW 5.4 - Running cosine similarity job on FULL dataset on EMR
python ./synonyms.py \
    -r emr s3://hamlin-mids-261/cooccurrence_stripes_NEW/*  \
    --conf-path ./mrjob.conf \
    --output-dir=s3://hamlin-mids-261/similarity_FULL \
    --ec2-instance-type c1.medium \
    --num-ec2-instances 6 \
    --no-output \
    --no-strict-protocol
```

This job runs relatively fast (9 minutes) on the same 6-node c1.medium cluster we used to calculate the stripes (we'll extract the outputs from s3 in problem 5.5).

HW 5.4 - Word similarity using inverted index and jaccard similarity

To avoid accidental cross-pollenation between our two similarity metric implementations, we've repeated all the code from our cosine similarity implementation, including the same process of testing on increasingly complex datasets, but this time set up to calculate jaccard similarity. The process for the creation of the inverted index is the same, but without the normalization step used in cosine similarity. Similarly, the pairs that we output based on the inverted index work slightly differently when using jaccard distance as well.

In [199]:

```
%%writefile synonyms.py
#HW 5.4 - Word similarity using inverted index and jaccard similarity M

from __future__ import division
from itertools import combinations
from math import sqrt
import csv

from mrjob.job import MRJob
from mrjob.step import MRStep

class Synonyms(MRJob):

    def mapper_binarized_inv_index(self, _, line):

        #Extract data from stripe files
        line=line.strip().split('\t')
        word=eval(line[0])
        cos=eval(line[1])

        #Because our test datasets have slightly different formatting t
        #stripes, we used a slightly different approach to reading the
        #(shown in the three commented lines below)

        #line=eval(line.strip())
        #word=line[0]
        #cos=line[1]

        stripe_length=len(cos)
        for word2, count in cos.iteritems():
        #Here, we don't need to normalize, because we only care about i
            yield word2, (word, 1)

    def combiner_inv_index(self, word2, word_1_counts):
        yield word2, dict(word_1_counts)

    def reducer_inv_index_init(self):
        #print "INTERMEDIATE RESULTS - INVERTED INDEX"
        pass

    def reducer_inv_index(self,word,cos):
        """recycled from previous job"""
        output_dict={}
        for co in cos:
            #co=dict(co)
            for second_word,count in co.iteritems():
                output_dict[second_word] = output_dict.get(second_word,
        #print word,output_dict
        yield word, output_dict

    def reducer_inv_index_final(self):
        #print " "
        pass
```

```
    def mapper_generate_pairs_init(self):
        #print "INTERMEDIATE RESULTS - PAIRS FROM POSTING LIST"
        pass

    def mapper_generate_pairs(self, word, cos):
        cos = list(cos)
        number_of_cos = len(cos)
        # This is another key difference between the cosine and jaccard
        # In addition to the intersections between stripes, we also emi
        # use to calculate the unions and, therefore, the jaccard simil
        for i in range(number_of_cos):
            #print ('*',cos[i]), 1
            yield ('*',cos[i]), 1
            for j in range(i+1,number_of_cos):
                #print (cos[i],cos[j]),1
                yield (cos[i],cos[j]),1

    def mapper_generate_pairs_final(self):
        #print " "
        pass

    def reducer_aggregate_distance(self,words,distance):
        yield words,sum(distance)

    def reducer_jaccard_calculation_init(self):
        self.total_dict={}

    def reducer_jaccard_calculation(self, words, values):
        word1,word2 = words
        if word1 == '*':
            self.total_dict[word2]=sum(values)
        else:
            intersection = sum(values)
            #Once again, Python 3 style division makes this expression
            distance = intersection / (self.total_dict[word1] + self.to
            yield (word1,word2), distance

    def steps(self):
        return [
            # First step creates the inverted index
            MRStep(
                mapper=self.mapper_binarized_inv_index
                 ,combiner=self.combiner_inv_index
                 ,reducer_init=self.reducer_inv_index_init
                 ,reducer=self.reducer_inv_index
                 ,reducer_final=self.reducer_inv_index_final
                ,jobconf={"mapred.map.tasks":16,"mapred.reduce.tasks":8
                   ),
            # Second step generates the pairs of co-occurrences in the
             MRStep(
                 mapper_init=self.mapper_generate_pairs_init,
                 mapper=self.mapper_generate_pairs
                 ,mapper_final=self.mapper_generate_pairs_final
                 ,reducer=self.reducer_aggregate_distance
```

```
                ,jobconf={"mapred.map.tasks":8,"mapred.reduce.tasks":4}

                   ),
            # Third step actually performs the final jaccard calculatic
             MRStep(
                reducer_init=self.reducer_jaccard_calculation_init,
                 reducer=self.reducer_jaccard_calculation
                ,jobconf={"mapred.map.tasks":4,"mapred.reduce.tasks":1}
                   )
        ]


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

```
Overwriting synonyms.py
```

HW 5.4 - Running jaccard similarity jobs on increasingly complex datasets

As before, our first test is on the unit test data, with all intermediate steps shown so we can make sure things are working right.

In [350]:

```
#HW 5.4 - Jaccard similarity testing driver function
# BASIC UNIT TEST
from synonyms import Synonyms

mr_job = Synonyms(args=['test.txt'])
with mr_job.make_runner() as runner:
    runner.run()
    print "FINAL RESULTS"
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:

INTERMEDIATE RESULTS - INVERTED INDEX
M {'docC': 1}
N {'docC': 1}
X {'docB': 1, 'docA': 1}
Y {'docB': 1, 'docA': 1}
Z {'docC': 1, 'docA': 1}

INTERMEDIATE RESULTS - PAIRS FROM POSTING LIST
('*', 'docC') 1
('*', 'docC') 1
('*', 'docB') 1
('docB', 'docA') 1
('*', 'docA') 1
('*', 'docB') 1
('docB', 'docA') 1
('*', 'docA') 1
('*', 'docC') 1
('docC', 'docA') 1
('*', 'docA') 1

FINAL RESULTS
(['docB', 'docA'], 0.6666666666666666)
(['docC', 'docA'], 0.2)
```

This matches the design pattern that we discussed during the week 5 lecture, so we can move on to testing locally on our sample stripes.

In [220]:

```
#HW 5.4 - Jaccard similarity testing driver function
# TEST ON SAMPLE STRIPE OUTPUT
from synonyms import Synonyms

mr_job = Synonyms(args=['stripes.txt'])
with mr_job.make_runner() as runner:
    runner.run()
    print "FINAL RESULTS"
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
```

```
FINAL RESULTS
(['a', 'all'], 0.48507125007266594)
(['a', 'case'], 0.5423261445466404)
(['a', 'discovery'], 0.48507125007266594)
(['a', 'female'], 0.48507125007266594)
(['a', 'for'], 0.5940885257860047)
(['a', 'honour'], 0.420084025208403)
(['a', 'in'], 0.6416889479197478)
(['a', 'instruction'], 0.48507125007266594)
(['a', 'man'], 0.48507125007266594)
(['a', 'manual'], 0.5423261445466404)
(['a', 'of'], 0.8744746321952064)
(['a', 'real'], 0.48507125007266594)
(['a', 'seasons'], 0.48507125007266594)
(['a', 'study'], 0.5423261445466404)
(['a', 'the'], 0.5940885257860047)
(['a', 'theodolite'], 0.24253562503633297)
(['all', 'discovery'], 0.25)
(['all', 'female'], 0.25)
(['all', 'for'], 0.6123724356957946)
(['all', 'honour'], 0.2886751345948129)
(['all', 'in'], 0.1889822365046136)
(['all', 'instruction'], 0.25)
(['all', 'man'], 0.75)
(['all', 'manual'], 0.22360679774997896)
(['all', 'of'], 0.2773500981126146)
(['all', 'real'], 0.25)
(['all', 'seasons'], 0.5)
(['all', 'study'], 0.22360679774997896)
(['all', 'the'], 0.4082482904638631)
(['all', 'theodolite'], 0.5)
(['case', 'all'], 0.22360679774997896)
(['case', 'discovery'], 0.4472135954999579)
(['case', 'female'], 0.6708203932499369)
(['case', 'for'], 0.36514837167011077)
(['case', 'honour'], 0.7745966692414834)
(['case', 'in'], 0.50709255283711)
(['case', 'instruction'], 0.6708203932499369)
(['case', 'man'], 0.22360679774997896)
(['case', 'manual'], 0.6)
(['case', 'of'], 0.6201736729460423)
(['case', 'real'], 0.4472135954999579)
(['case', 'seasons'], 0.22360679774997896)
(['case', 'study'], 0.7999999999999999)
(['case', 'the'], 0.36514837167011077)
(['case', 'theodolite'], 0.4472135954999579)
(['discovery', 'man'], 0.25)
(['female', 'discovery'], 0.5)
(['female', 'in'], 0.1889822365046136)
(['female', 'man'], 0.25)
(['female', 'of'], 0.41602514716892186)
(['female', 'real'], 0.5)
(['female', 'seasons'], 0.25)
(['female', 'the'], 0.4082482904638631)
```

```
(['for', 'discovery'], 0.6123724356957946)
(['for', 'female'], 0.4082482904638631)
(['for', 'honour'], 0.4714045207910318)
(['for', 'instruction'], 0.4082482904638631)
(['for', 'man'], 0.6123724356957946)
(['for', 'manual'], 0.5477225575051662)
(['for', 'of'], 0.33968311024337877)
(['for', 'real'], 0.4082482904638631)
(['for', 'seasons'], 0.20412414523193154)
(['for', 'study'], 0.36514837167011077)
(['for', 'the'], 0.33333333333333334)
(['for', 'theodolite'], 0.4082482904638631)
(['honour', 'discovery'], 0.5773502691896258)
(['honour', 'female'], 0.5773502691896258)
(['honour', 'instruction'], 0.5773502691896258)
(['honour', 'man'], 0.2886751345948129)
(['honour', 'manual'], 0.5163977794943223)
(['honour', 'of'], 0.3202563076101743)
(['honour', 'real'], 0.5773502691896258)
(['honour', 'seasons'], 0.2886751345948129)
(['honour', 'study'], 0.5163977794943223)
(['honour', 'the'], 0.4714045207910318)
(['honour', 'theodolite'], 0.5773502691896258)
(['in', 'discovery'], 0.3779644730092272)
(['in', 'female'], 0.5669467095138407)
(['in', 'for'], 0.3086066999241838)
(['in', 'honour'], 0.4364357804719848)
(['in', 'instruction'], 0.3779644730092272)
(['in', 'man'], 0.1889822365046136)
(['in', 'manual'], 0.3380617018914066)
(['in', 'of'], 0.3144854510165755)
(['in', 'real'], 0.3779644730092272)
(['in', 'seasons'], 0.1889822365046136)
(['in', 'study'], 0.3380617018914066)
(['in', 'the'], 0.3086066999241838)
(['in', 'theodolite'], 0.3779644730092272)
(['instruction', 'discovery'], 0.5)
(['instruction', 'female'], 0.5)
(['instruction', 'honour'], 0.2886751345948129)
(['instruction', 'in'], 0.1889822365046136)
(['instruction', 'man'], 0.25)
(['instruction', 'manual'], 0.6708203932499369)
(['instruction', 'of'], 0.2773500981126146)
(['instruction', 'real'], 0.5)
(['instruction', 'seasons'], 0.25)
(['instruction', 'the'], 0.4082482904638631)
(['instruction', 'theodolite'], 0.5)
(['manual', 'discovery'], 0.6708203932499369)
(['manual', 'female'], 0.4472135954999579)
(['manual', 'honour'], 0.25819888974716115)
(['manual', 'in'], 0.1690308509457033)
(['manual', 'man'], 0.22360679774997896)
(['manual', 'of'], 0.2480694691784169)
(['manual', 'real'], 0.4472135954999579)
```

```
(['manual', 'seasons'], 0.22360679774997896)
(['manual', 'the'], 0.36514837167011077)
(['manual', 'theodolite'], 0.4472135954999579)
(['of', 'discovery'], 0.5547001962252291)
(['of', 'female'], 0.1386750490563073)
(['of', 'honour'], 0.16012815380508716)
(['of', 'in'], 0.4193139346887673)
(['of', 'instruction'], 0.2773500981126146)
(['of', 'man'], 0.2773500981126146)
(['of', 'manual'], 0.3721042037676254)
(['of', 'real'], 0.4160251471689218 6)
(['of', 'seasons'], 0.2773500981126146)
(['of', 'study'], 0.2480694691784169)
(['of', 'the'], 0.6793662204867574)
(['real', 'discovery'], 0.75)
(['real', 'for'], 0.20412414523193154)
(['real', 'man'], 0.25)
(['real', 'manual'], 0.22360679774997896)
(['real', 'of'], 0.1386750490563073)
(['real', 'seasons'], 0.25)
(['real', 'the'], 0.4082482904638631)
(['seasons', 'all'], 0.25)
(['seasons', 'discovery'], 0.25)
(['seasons', 'for'], 0.4082482904638631)
(['seasons', 'man'], 0.75)
(['seasons', 'the'], 0.4082482904638631)
(['study', 'discovery'], 0.4472135954999579)
(['study', 'female'], 0.6708203932499369)
(['study', 'honour'], 0.2581988897471611 5)
(['study', 'in'], 0.1690308509457033)
(['study', 'instruction'], 0.6708203932499369)
(['study', 'man'], 0.22360679774997896)
(['study', 'manual'], 0.6)
(['study', 'of'], 0.3721042037676254)
(['study', 'real'], 0.4472135954999579)
(['study', 'seasons'], 0.22360679774997896)
(['study', 'the'], 0.36514837167011077)
(['study', 'theodolite'], 0.4472135954999579)
(['the', 'discovery'], 0.6123724356957946)
(['the', 'in'], 0.1543033499620919)
(['the', 'instruction'], 0.20412414523193154)
(['the', 'man'], 0.4082482904638631)
(['the', 'real'], 0.20412414523193154)
(['theodolite', 'discovery'], 0.5)
(['theodolite', 'female'], 0.5)
(['theodolite', 'man'], 0.5)
(['theodolite', 'of'], 0.2773500981126146)
(['theodolite', 'real'], 0.5)
(['theodolite', 'seasons'], 0.5)
(['theodolite', 'the'], 0.4082482904638631)
```

This gives us similar (though not the same results) as what we saw for similarities when we used cosine distance, which is a good sign. At this point, we'll skip to testing directly on a subset of our production stripes in EMR.

In [ ]:
```
# HW 5.4 - Running jaccard similarity job on test dataset on EMR
python ./synonyms.py \
    -r emr s3://hamlin-mids-261/test_stripes/*  \
    --conf-path ./mrjob.conf \
    --output-dir=s3://hamlin-mids-261/jaccard_similarity \
    --no-output \
    --no-strict-protocol
```

Finally, we compute jaccard similarites for our full dataset

In [ ]:
```
# HW 5.4 - Running jaccard similarity job on FULL dataset on EMR
python ./synonyms.py \
    -r emr s3://hamlin-mids-261/cooccurrence_stripes_NEW/*  \
    --conf-path ./mrjob.conf \
    --output-dir=s3://hamlin-mids-261/jaccard_similarity_FULL \
    --ec2-instance-type c1.medium \
    --num-ec2-instances 6 \
    --no-output \
    --no-strict-protocol
```

This job runs slightly slower than our cosine similarity job (11 minutes vs 9 minutes) on the same cluster (6 c1.medium nodes).

HW 5.5

HW 5.5 Problem Statement

In this part of the assignment you will evaluate the success of you synonym detector. Take the top 1,000 closest/most similar/correlative pairs of words as determined by your measure in (2), and use the synonyms function in the accompanying python code

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.

HW 5.5 Evaluation criteria implementation

Here, we use the NLTK-based synonym detection function provided in the original assignment and wrap it in a driver function. This driver iterates through the output file of our similarity calculation and keeps track of how many correct pairings we've made. In order to properly calculate precision and recall, we need to define both the space of selected elements and the space of relevant elements. To do this, we specify a threshold of calculate similarity above which we'll consider a match to be "matched". By establishing this threshold, we can track both the true and false positives as well as true and false negatives required to compute all the required metrics.

In [181]:

```python
# HW 5.5 - Functions to calculate precision, recall, and f1 scores

import nltk
from nltk.corpus import wordnet as wn
import sys
#print all the synset element of an element
def synonyms(string):
    ''' pass a string to this funciton ( eg 'car') and it will give you
    words which is related to cat, called lemma of CAT. '''
    syndict = {}
    for i,j in enumerate(wn.synsets(string)):
        syns = j.lemma_names()
        for syn in syns:
            syndict.setdefault(syn,1)
    return syndict.keys()


def test_similarity(file,threshold):
    hits = [] #used to store whether the word pair is a hit (0 or 1) as
    preds = [] #used to store a binary prediction value based on the si
    with open(file) as f: #Pass in a file of results and extract data
        for line in f.readlines():
            line=line.strip().split('\t')
            word1 = eval(line[0])[0]
            word2 = eval(line[0])[1]
            score = eval(line[1])

            #if one of the two words is a synonym of the other, than st
            if word1 in synonyms(word2) or word2 in synonyms(word1):
                hits.append(1)
            else:
                hits.append(0)

            #if the similarity score for these pairs is above the thres
            if score > threshold:
                preds.append(1)
            else:
                preds.append(0)

    # Set up variables for our measures of true positives, false positi
    tp = 0 # true positives
    fp = 0 # false positives
    fn = 0 # false negatives
    tn = 0 # true negatives

    #Iterate through our results and total true positives, false positi
    for i in range(len(hits)):
        if hits[i] == 1 and preds[i] == 1:
            tp += 1
        elif hits[i] == 0 and preds[i] == 1:
            fp += 1
        elif hits[i] == 1 and preds[i] == 0:
            fn += 1
        else:
```

```
            tn += 1

    #Finally, calculate precision, recall, and f1 scores
    try:
        precision = float(tp) / float(tp + fp)
    except ZeroDivisionError:
        precision = float("inf")

    try:
        recall = float(tp) / float(tp + fn)
    except ZeroDivisionError:
        recall = float("inf")

    try:
        f1score = 2 * (precision*recall) / (precision + recall)
    except ZeroDivisionError:
        f1score = float("inf")

    print "Precision:\t%s" % precision
    print "Recall:\t\t%s" % recall
    print "F1 Score:\t%s" % f1score
```

HW 5.5 - Testing our evaluation code

Before we use our evaluation code to evaluate our similarity results, we should make sure it does what we want it to do. To do this, we'll create two small test files with one "matching" pair and one "unmatching". By manually setting the associated similarity scores to be "correct" or not, we can test our evaluation process.

In [195]:
```
#HW 5.5 - Evaluation test 1
#Here we've got one pair "correct" and one "incorrect", so we should ha
%%writefile pairs1.txt
["auto", "car"] 0.8
["car", "plane"]    0.8
```

```
Overwriting pairs1.txt
```

In [196]:
```
#HW 5.5 - Evaluation test 2
#Here we've classified both pairs correctly, so we should get perfect r
%%writefile pairs2.txt
["auto", "car"] 0.8
["car", "plane"]    0.01
```

```
Overwriting pairs2.txt
```

In [197]:
```
#HW 5.5 - Test our similarity function on our dummy data
def run_5_5_test():
    print "Pairs1 results"
    test_similarity('pairs1.txt',0.1)
    print ""
    print "Pairs2 results"
    test_similarity('pairs2.txt',0.1)
    print ""

run_5_5_test()
```

```
Pairs1 results
Precision:      0.5
Recall:         1.0
F1 Score:       0.666666666667

Pairs2 results
Precision:      1.0
Recall:         1.0
F1 Score:       1.0
```

These tests look good, so we can move on to evaluating our overall results.

HW 5.5 - Download similarity data from s3 and evaluate results

While we ran our similarity calculations in 5.4 and examined a sample of our results manually, now we need to download our full datasets so we can evaluate how well we did. For the purposes of this analysis, we consider a calculate similarity of above 0.1 to be a "hit".

In [201]:
```
#HW 5.5 - Download results
! mkdir ./cosine_similarity_output
! aws s3 cp --recursive s3://hamlin-mids-261/similarity_FULL ./cosine_s
! cat ./cosine_similarity_output/part-* > cosine_results.txt
! mkdir ./jaccard_similarity_output
! aws s3 cp --recursive s3://hamlin-mids-261/jaccard_similarity_FULL ./
! cat ./jaccard_similarity_output/part-* > jaccard_results.txt
```

```
download: s3://hamlin-mids-261/jaccard_similarity_FULL/_SUCCESS to j
accard_similarity_output/_SUCCESS
download: s3://hamlin-mids-261/jaccard_similarity_FULL/part-00000 to
jaccard_similarity_output/part-00000
```

In [223]:
```
# HW 5.5 - Run final evaluation of full-set similarity results

def run_5_5_final():
    print "Cosine results"
    test_similarity('cosine_results.txt',0.1)
    print ""

    print "Jaccard results"
    test_similarity('jaccard_results.txt',0.1)
    print ""

run_5_5_final()
```

```
Cosine results
Precision:       0.00172228202368
Recall:          0.603773584906
F1 Score:        0.00343476627489

Jaccard results
Precision:       0.00538720538721
Recall:          0.153846153846
F1 Score:        0.0104098893949
```

HW 5.5 - Discussion of results

Overall, our similarity calculation didn't perform exceptionally well. This may be the result of our choice to only include co-occurrences of the 1000 terms in our vocabulary in our stripes, we may have unintentionally excluded too much information for the similarities to calculate accurately. Also of note is the difference in performance of our two distance metrics. Cosine similarity had relatively high recall, but extremely low precision, while jaccard similarity had lower recall, but a slightly higher precision and f1 score. The fact that both of our similarity metrics performed less well than we'd have hoped suggests that the focus of any future troubleshooting should begin with the questions about stripes calculation mentioned above.

Appendix

We didn't end up using any of this code for our final analysis, but wanted to save it here so we didn't lose track of it. This represents one of our intermediate implementations of stripes in which we calculated a stripe for each of the 10000 rows. However, this job would have run for a prohibitively long time, which is why we ended up implementing stripes only for the words in our specified vocabulary.

In [74]:

```
%%writefile stripes.py
#HW 5.4 - Stripes MRJob Definition
from __future__ import division
from itertools import combinations

from mrjob import conf
from mrjob.job import MRJob
from mrjob.step import MRStep

class Stripes(MRJob):

    def jobconf(self):
        orig_jobconf = super(Stripes, self).jobconf()
        # Setting these high enough improves EMR job speed
        custom_jobconf = {
            "mapred.map.tasks":28,
            "mapred.reduce.tasks":28
        }
        return conf.combine_dicts(orig_jobconf, custom_jobconf)


    def mapper_init(self):
        """Load file of words into memory"""
        self.all_words={} #Contains all 10000 words
        self.vocab_words={} #Contains words 9001-10000
        #This is the file of top 10000 words we created in HW 5.3.B
        with open('most_freq_words_10K.txt','rb') as f:
            count=0
            for row in f.readlines():
                line=row.strip().split('\t')
                if count>9000:
                    self.vocab_words[line[0][1:-1]]=line[1]
                self.all_words[line[0][1:-1]]=line[1]
                count+=1

    def mapper(self, _, line):
        """
        Emit co-occurrence combinations for each pair of relevant words
        """
        line=line.strip().split('\t')
        ngram=line[0].lower() #The full text of the ngram
        count=int(line[1]) #The count associated with it
        potential_words=ngram.split(" ") #List of individual words in c
        output={}
        #Pull out words from ngram that we care about (those that appea

        #We only want to include a word in our analysis if it appears i
        #and is also in the 10000 most frequent words
        words=[i for i in potential_words if i in self.all_words.keys()
        #For each word in our ngram in the top 10K, look for co-occurre
        #words ranked from 9001-10000.

        #Update output stripe for each combination of co-occurring, rel
        for word1,word2 in combinations(potential_words,2):
```

```
            #This syntax does functionally the same thing as a Counter
            #but they aren't supported in Python 2.6.9, which is the de
            #that comes with the EMR AMIs.  Instead of fighting with AM
            #we decided it was easier to just implement the counter man

            if word2 in self.vocab_words.keys(): #Ensures vocab is in 9
                if word1 in output.keys():
                    output[word1][word2]=output[word1].get(word2,0)+cou
                else:
                    output[word1]={word2:count}

            #This second step ensures we maintain symmetry
#               if word1 in self.vocab_words.keys():
#                   if word2 in output.keys():
#                       output[word2][word1]=output[word2].get(word1,0)+c
#                   else:
#                       output[word2]={word1:count}

        #"cooccurrences" is what I really want to call this second var,
        #but that's too much to type/spell reliably, so I'll settle for
        for word,cos in output.iteritems():
            yield word,cos

    def reducer(self,word,cos):
        """Aggregate stripes based on intermediate results from mapper"
        output_dict={}
        for co in cos:
            # The second_word variable here is so named to distinguish
            # and refers to the words in the co-occurrence stripe
            for second_word,count in co.iteritems():
                output_dict[second_word] = output_dict.get(second_word,
        yield word, output_dict


    def steps(self):
        return [
            MRStep(
                mapper_init=self.mapper_init,
                mapper=self.mapper
                #We can recycle the reducer as combiner here, which is
                ,combiner=self.reducer
                ,reducer=self.reducer
                  )
        ]

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

```
Overwriting stripes.py
```

End of Submission