MIDS - w261 Machine Learning At Scale

Course Lead: Dr James G. Shanahan (**email** Jimi via James.Shanahan *AT* gmail.com)

Assignment - HW5

Name: *Jason Sanchez*
Class: MIDS w261 (Section *Fall 2016 Group 2*)
Email: *jason.sanchez*@iSchool.Berkeley.edu
Week: 5

Due Time: 2 Phases.

- **HW5 Phase 1** This can be done on a local machine (with a unit test on the cloud such as AltaScale's PaaS or on AWS) and is due Tuesday, Week 6 by 8AM (West coast time). It will primarily focus on building a unit/systems and for pairwise similarity calculations pipeline (for stripe documents)
- **HW5 Phase 2** This will require the AltaScale cluster and will be due Tuesday, Week 7 by 8AM (West coast time). The focus of HW5 Phase 2 will be to scale up the unit/systems tests to the Google 5 gram corpus. This will be a group exercise

Table of Contents

1 Instructions

Back to Table of Contents

MIDS UC Berkeley, Machine Learning at Scale DATSCIW261 ASSIGNMENT #5

Version 2016-09-25

=== INSTRUCTIONS for SUBMISSIONS === Follow the instructions for submissions carefully.

https://docs.google.com/forms/d/1ZOr9RnIe_A06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiis/viewform?usp=send_form
(https://docs.google.com/forms/d/1ZOr9RnIe_A06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiis/viewform?usp=send_form)

IMPORTANT

HW4 can be completed locally on your computer

Documents:

- IPython Notebook, published and viewable online.
- PDF export of IPython Notebook.

2 Useful References

Back to Table of Contents

- See async and live lectures for this week

HW Problems

Back to Table of Contents

3. HW5.0

Back to Table of Contents

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

Data warehouse: Stores a large amount of relational, semi-structured, and unstructured data. Is used for business intelligence and data science.

A star schema has fact tables and many dimension tables that connect to the fact tables. Fact tables record events such as sales or website visits and encodes details of the events as keys (user_id, product_id, store_id, ad_id). The dimension tables store the detailed information about each of these keys.

Star schemas provide simple approached to structuring data warehouses in a relational way.

3. HW5.1

Back to Table of Contents

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

3NF means third normal form. It is used to transform large flat files that have repeated data into a linked collection of smaller tables that can be joined on a set of common keys.

Machine learning does not use data in 3NF. Instead it uses large flat files so the details that are hidden by the keys can be used in the algorithms.

Log files can track specific events of interest. A denormalized log file allows a company to track these events in real time conditioned on specific customer features. Alternatively, a model can be running that triggers appropriate responses based on the next predicted action of a user given the user's latest action.

3. HW5.2

Back to Table of Contents

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

List files in directory

In [1]:
```
!ls
```

```
MIDS-W261-HW-05-Sanchez.ipynb      mostFrequentVisitors.txt
anonymous-msweb-preprocessed.data
```

Count lines in dataset. View the first 10 lines. Rename data to log.data

In [25]:
```
!wc -l anonymous-msweb-preprocessed.data && echo
!head anonymous-msweb-preprocessed.data
!cp anonymous-msweb-preprocessed.data log.txt
```

```
   98654 anonymous-msweb-preprocessed.data

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

Convert the output of 4.4 to be just url and url_id

In [22]:
```
!cat mostFrequentVisitors.txt | cut -f 1,2 -d',' > urls.txt
!wc -l urls.txt && echo
!head urls.txt
```

```
     285 urls.txt

"/regwiz",1000
"/support",1001
"/athome",1002
"/kb",1003
"/search",1004
"/norge",1005
"/misc",1006
"/ie_intl",1007
"/msdownload",1008
"/windows",1009
```

The urls.txt file is much smaller than the log.txt data and should be what is loaded into memory. This means it would be the right-side table in a left join.

Left-side join

In [54]:
```
%%writefile lj.py

from mrjob.job import MRJob
# Avoid broken pipe error
from signal import signal, SIGPIPE, SIG_DFL
signal(SIGPIPE,SIG_DFL)

class LJ(MRJob):
    def mapper_init(self):
        self.urls = {}
        with open("urls.txt") as urls:
            for line in urls:
                url, key = line.strip().replace('"',"").split(",")
                self.urls[key] = url

    def mapper(self, _, lines):
        try:
            yield (lines, self.urls[lines[2:6]])
        except ValueError:
            yield (lines, "")

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

```
Overwriting lj.py
```

In [55]:
```
!cat log.txt | python lj.py --file urls.txt -q | head
```

```
"V,1000,1,C,10001"      "/regwiz"
"V,1001,1,C,10001"      "/support"
"V,1002,1,C,10001"      "/athome"
"V,1001,1,C,10002"      "/support"
"V,1003,1,C,10002"      "/kb"
"V,1001,1,C,10003"      "/support"
"V,1003,1,C,10003"      "/kb"
"V,1004,1,C,10003"      "/search"
"V,1005,1,C,10004"      "/norge"
"V,1006,1,C,10005"      "/misc"
```

In [56]:

```
%%writefile ij.py

from mrjob.job import MRJob
# Avoid broken pipe error
from signal import signal, SIGPIPE, SIG_DFL
signal(SIGPIPE,SIG_DFL)

class IJ(MRJob):
    def mapper_init(self):
        self.urls = {}
        with open("urls.txt") as urls:
            for line in urls:
                url, key = line.strip().replace('"',"").split(",")
                self.urls[key] = url

    def mapper(self, _, lines):
        try:
            yield (lines, self.urls[lines[2:6]])
        except ValueError:
            pass

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

```
Writing ij.py
```

In [57]:

```
!cat log.txt | python ij.py --file urls.txt -q | head
```

```
"V,1000,1,C,10001"      "/regwiz"
"V,1001,1,C,10001"      "/support"
"V,1002,1,C,10001"      "/athome"
"V,1001,1,C,10002"      "/support"
"V,1003,1,C,10002"      "/kb"
"V,1001,1,C,10003"      "/support"
"V,1003,1,C,10003"      "/kb"
"V,1004,1,C,10003"      "/search"
"V,1005,1,C,10004"      "/norge"
"V,1006,1,C,10005"      "/misc"
```

In [73]:

```
%%writefile rj.py

from mrjob.job import MRJob
# Avoid broken pipe error
from signal import signal, SIGPIPE, SIG_DFL
signal(SIGPIPE,SIG_DFL)

class RJ(MRJob):
    def mapper_init(self):
        self.urls_used = set()
        self.urls = {}
        with open("urls.txt") as urls:
            for line in urls:
                url, key = line.strip().replace('"',"").split(",")
                self.urls[key] = url

    def mapper(self, _, lines):
        try:
            url = lines[2:6]
            yield (self.urls[url], lines)
            self.urls_used.add(url)

        except ValueError:
            pass

    def mapper_final(self):
        for key, value in self.urls.items():
            if key not in self.urls_used:
                yield (self.urls[key], "*")



    def reducer(self, url, values):
        quick_stash = 0
        for val in values:
            if val != "*":
                quick_stash += 1
                yield (url, val)
        if quick_stash == 0:
            yield (url, "None")

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

```
Overwriting rj.py
```

To prove it works, we can only use the first 100 log entries. We see that urls without corresponding log entries are listed as "None".

In [77]:
```
!head -n 100 log.txt | python rj.py --file urls.txt -q | head -n 15
```

```
"/access"       "None"
"/accessdev"    "None"
"/activeplatform"       "None"
"/activex"      "None"
"/adc"  "None"
"/ado"  "None"
"/ads"  "None"
"/advtech"      "None"
"/argentina"    "None"
"/atec" "None"
"/athome"       "V,1002,1,C,10001"
"/athome"       "V,1002,1,C,10019"
"/athome"       "V,1002,1,C,10020"
"/athome"       "V,1002,1,C,10031"
"/australia"    "None"
```

3. HW5.3 Systems tests on n-grams dataset (Phase1) and full experiment (Phase 2)

Back to Table of Contents

3. HW5.3.0 Run Systems tests locally (PHASE1)

Back to Table of Contents

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 and on s3:

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

s3://filtered-5grams/

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

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

The next cell shows the first 10 lines of the googlebooks-eng-all-5gram-20090715-0-filtered.txt file.

DISCLAIMER: Each record is already a 5-gram. We should calculate the stripes cooccurrence data from the raw text and not from the 5-gram preprocessed data. Calculatating pairs on this 5-gram is a little corrupt as we will be double counting cooccurences. Having said that this exercise can still pull out some simialr terms.

1: unit/systems first-10-lines

In [79]:
```
%%writefile googlebooks-eng-all-5gram-20090715-0-filtered-first-10-line
s.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
```

```
Writing googlebooks-eng-all-5gram-20090715-0-filtered-first-10-lines.tx
t
```

For *HW 5.4-5.5*, unit test and regression test your code using the followings small test datasets:

- googlebooks-eng-all-5gram-20090715-0-filtered.txt [see above]
- stripe-docs-test [see below]
- atlas-boon-test [see below]

2: unit/systems atlas-boon

In [80]:
```
%%writefile atlas-boon-systems-test.txt
atlas boon      50      50      50
boon cava dipped        10      10      10
atlas dipped    15      15      15
```

```
Writing atlas-boon-systems-test.txt
```

3: unit/systems stripe-docs-test

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

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

In [82]:
```
############################################
# Stripes for systems test 1 (predefined)
############################################

with open("mini_stripes.txt", "w") as f:
    f.writelines([
        '"DocA"\t{"X":20, "Y":30, "Z":5}\n',
        '"DocB"\t{"X":100, "Y":20}\n',
        '"DocC"\t{"M":5, "N":20, "Z":5, "Y":1}\n'
    ])
!cat mini_stripes.txt
```

```
"DocA"	{"X":20, "Y":30, "Z":5}
"DocB"	{"X":100, "Y":20}
"DocC"	{"M":5, "N":20, "Z":5, "Y":1}
```

TASK: Phase 1

Complete 5.4 and 5.5 and systems test them using the above test datasets. Phase 2 will focus on the entire Ngram dataset.

To help you through these tasks please verify that your code gives the following results (for stripes, inverted index, and pairwise similarities).

Make stripes

In [115]:

```
%%writefile MakeStripes.py

from mrjob.job import MRJob
from collections import Counter

class MakeStripes(MRJob):
    def mapper_init(self):
        self.stripes = {}

    def mapper(self, _, lines):
        terms, term_count, page_count, book_count = lines.split("\t")
        terms = terms.split()
        term_count = int(term_count)

        for item in terms:
            yield (item, {val:term_count for val in terms if val != ite
m})

    def combiner(self, keys, values):
        values_sum = Counter()
        for val in values:
            values_sum += Counter(val)
        yield keys, dict(values_sum)

    def reducer(self, keys, values):
        values_sum = Counter()
        for val in values:
            values_sum += Counter(val)
        yield keys, dict(values_sum)

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

```
Overwriting MakeStripes.py
```

In [117]:
```
!cat atlas-boon-systems-test.txt | python MakeStripes.py -q > atlas_stri
pes.txt
!cat atlas_stripes.txt
```

```
"atlas" {"boon": 50, "dipped": 15}
"boon"  {"atlas": 50, "cava": 10, "dipped": 10}
"cava"  {"boon": 10, "dipped": 10}
"dipped"        {"boon": 10, "atlas": 15, "cava": 10}
```

Stripe documents for atlas-boon systems test

In [ ]:
```
"atlas" {"dipped": 15, "boon": 50}
"boon"  {"atlas": 50, "dipped": 10, "cava": 10}
"cava"  {"dipped": 10, "boon": 10}
"dipped"        {"atlas": 15, "boon": 10, "cava": 10}
```

The calculated stripes match the systems test.

Inverted index

In [132]:

```
%%writefile InvertIndex.py

from mrjob.job import MRJob
from mrjob.protocol import JSONProtocol
from collections import Counter

class InvertIndex(MRJob):
    MRJob.input_protocol = JSONProtocol

    def mapper(self, key, words):
        n_words = len(words)

        for word in words:
            yield (word, {key:n_words})

    def combiner(self, keys, values):
        values_sum = Counter()
        for val in values:
            values_sum += Counter(val)
        yield keys, dict(values_sum)

    def reducer(self, keys, values):
        values_sum = Counter()
        for val in values:
            values_sum += Counter(val)
        yield keys, dict(values_sum)

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

```
Overwriting InvertIndex.py
```

In [136]:

```
!cat atlas_stripes.txt | python InvertIndex.py -q > atlas_inverted.txt
!cat atlas_inverted.txt
```

```
"atlas" {"boon": 3, "dipped": 3}
"boon"  {"atlas": 2, "dipped": 3, "cava": 2}
"cava"  {"boon": 3, "dipped": 3}
"dipped"        {"atlas": 2, "boon": 3, "cava": 2}
```

In [137]:

```
!cat mini_stripes.txt | python InvertIndex.py -q > mini_stripes_inverte
d.txt
!cat mini_stripes_inverted.txt
```

```
"M"     {"DocC": 4}
"N"     {"DocC": 4}
"X"     {"DocB": 2, "DocA": 3}
"Y"     {"DocB": 2, "DocC": 4, "DocA": 3}
"Z"     {"DocC": 4, "DocA": 3}
```

```
In [ ]: Systems test mini_stripes - Inverted Index
        ==========================================================================

                    "M" |          DocC 4 |                 |
                    "N" |          DocC 4 |                 |
                    "X" |          DocA 3 |          DocB 2 |
                    "Y" |          DocA 3 |          DocB 2 |          DocC 4
                    "Z" |          DocA 3 |          DocC 4 |

         systems test atlas-boon - Inverted Index
        ==========================================================================

                "atlas" |          boon 3 |        dipped 3 |
               "dipped" |         atlas 2 |          boon 3 |          cava 2
                 "boon" |         atlas 2 |          cava 2 |        dipped 3
                 "cava" |          boon 3 |        dipped 3 |
```

Tests pass

Similarity

In [216]:

```
%%writefile Similarity.py

from mrjob.job import MRJob
from mrjob.protocol import JSONProtocol
from itertools import combinations
from statistics import mean

class Similarity(MRJob):
    MRJob.input_protocol = JSONProtocol

    def mapper(self, key_term, docs):
        doc_names = docs.keys()
        for doc_pairs in combinations(sorted(list(doc_names)), 2):
            yield (doc_pairs, 1)
        for name in doc_names:
            yield (name, 1)

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

    def reducer_init(self):
        self.words = {}
        self.results = []

    def reducer(self, doc_or_docs, count):
        if isinstance(doc_or_docs, str):
            self.words[doc_or_docs] = sum(count)
        else:
            d1, d2 = doc_or_docs
            d1_n_words, d2_n_words = self.words[d1], self.words[d2]
            intersection = sum(count)

            jaccard = round(intersection/(d1_n_words + d2_n_words - inte
rsection), 3)
            cosine = round(intersection/(d1_n_words**.5 * d2_n_words**.
5), 3)
            dice = round(2*intersection/(d1_n_words + d2_n_words), 3)
            overlap = round(intersection/min(d1_n_words, d2_n_words), 3)
            average = round(mean([jaccard, cosine, dice, overlap]), 3)

            self.results.append([doc_or_docs, {"jacc":jaccard, "cos":cos
ine,
                                               "dice":dice, "ol":overla
p, "ave":average}])

    def reducer_final(self):
        for doc, result in sorted(self.results, key=lambda x: x[1]["av
e"], reverse=True):
            yield (doc, result)

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

Overwriting Similarity.py

In [257]:
```
!cat atlas_inverted.txt | python Similarity.py -q --jobconf mapred.reduc
e.tasks=1
```

```
["atlas", "cava"]       {"ave": 1.0, "jacc": 1.0, "ol": 1.0, "cos": 1.
0, "dice": 1.0}
["boon", "dipped"]      {"ave": 0.625, "jacc": 0.5, "ol": 0.667, "cos":
 0.667, "dice": 0.667}
["atlas", "boon"]       {"ave": 0.39, "jacc": 0.25, "ol": 0.5, "cos":
 0.408, "dice": 0.4}
["atlas", "dipped"]     {"ave": 0.39, "jacc": 0.25, "ol": 0.5, "cos":
 0.408, "dice": 0.4}
["boon", "cava"]        {"ave": 0.39, "jacc": 0.25, "ol": 0.5, "cos":
 0.408, "dice": 0.4}
["cava", "dipped"]      {"ave": 0.39, "jacc": 0.25, "ol": 0.5, "cos":
 0.408, "dice": 0.4}
```

In [261]:
```
!cat mini_stripes_inverted.txt | python Similarity.py -q --jobconf mapre
d.reduce.tasks=1
```

```
["DocA", "DocB"]        {"ave": 0.821, "jacc": 0.667, "cos": 0.816, "o
l": 1.0, "dice": 0.8}
["DocA", "DocC"]        {"ave": 0.554, "jacc": 0.4, "cos": 0.577, "ol":
 0.667, "dice": 0.571}
["DocB", "DocC"]        {"ave": 0.347, "jacc": 0.2, "cos": 0.354, "ol":
 0.5, "dice": 0.333}
```

average	pair	cosine	jaccard	overlap	dice

Systems test mini_stripes - Similarity measures

average	pair	cosine	jaccard	overlap	dice
0.741582	DocA - DocB	0.816497	0.666667	1.000000	0.800000
0.488675	DocA - DocC	0.577350	0.400000	0.666667	0.571429
0.276777	DocB - DocC	0.353553	0.200000	0.500000	0.333333

Systems test atlas-boon 2 - Similarity measures

average	pair	cosine	jaccard	overlap	dice
1.000000	atlas - cava	1.000000	1.000000	1.000000	1.000000
0.625000	boon - dipped	0.666667	0.500000	0.666667	0.666667
0.389562	cava - dipped	0.408248	0.250000	0.500000	0.400000
0.389562	boon - cava	0.408248	0.250000	0.500000	0.400000
0.389562	atlas - dipped	0.408248	0.250000	0.500000	0.400000
0.389562	atlas - boon	0.408248	0.250000	0.500000	0.400000

The numbers I calculated exactly match the systems test except for the average calculations of the mini_stripes set. In this instance, the systems test calculations are not correct.

Test code on AWS

In [274]:
```
%%time
!source ../private/aws_creds.sh && python MakeStripes.py -r emr atlas-bo
on-systems-test.txt
```

```
No configs found; falling back on auto-configuration
Using s3://mrjob-3d3e189cec521ef3/tmp/ as our temp dir on S3
Creating temp directory /var/folders/sz/4k2bbjts7x5fmg9sn7kh6hlw0000gn/
T/MakeStripes.Jason.20161003.090134.937573
Copying local files to s3://mrjob-3d3e189cec521ef3/tmp/MakeStripes.Jaso
n.20161003.090134.937573/files/...
Created new cluster j-5ZRHGZTUSQSW
Waiting for step 1 of 1 (s-25AWSL4OHYR4F) to complete...
  PENDING (cluster is STARTING)
  PENDING (cluster is STARTING)
  PENDING (cluster is STARTING)
  PENDING (cluster is STARTING)
  PENDING (cluster is STARTING)
  PENDING (cluster is STARTING)
  PENDING (cluster is STARTING)
  PENDING (cluster is STARTING)
  PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)
  PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)
  PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)
  PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)
  PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)
  PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)
  PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)
  PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)
  RUNNING for 31.3s
  RUNNING for 62.9s
  RUNNING for 93.7s
  RUNNING for 125.0s
  COMPLETED
Attempting to fetch counters from logs...
Waiting for cluster (j-5ZRHGZTUSQSW) to terminate...
  TERMINATING
  TERMINATING
  TERMINATED
Looking for step log in s3://mrjob-3d3e189cec521ef3/tmp/logs/j-5ZRHGZTU
SQSW/steps/s-25AWSL4OHYR4F...
  Parsing step log: s3://mrjob-3d3e189cec521ef3/tmp/logs/j-5ZRHGZTUSQS
W/steps/s-25AWSL4OHYR4F/syslog.gz
Counters: 54
        File Input Format Counters
                Bytes Read=101
        File Output Format Counters
                Bytes Written=163
        File System Counters
                FILE: Number of bytes read=151
                FILE: Number of bytes written=312061
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=316
                HDFS: Number of bytes written=0
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=2
                HDFS: Number of write operations=0
                S3: Number of bytes read=101
                S3: Number of bytes written=163
                S3: Number of large read operations=0
```

```
                S3: Number of read operations=0
                S3: Number of write operations=0
        Job Counters
                Data-local map tasks=2
                Launched map tasks=2
                Launched reduce tasks=1
                Total megabyte-seconds taken by all map tasks=26241024
                Total megabyte-seconds taken by all reduce tasks=213555
20
                Total time spent by all map tasks (ms)=34168
                Total time spent by all maps in occupied slots (ms)=102
504
                Total time spent by all reduce tasks (ms)=20855
                Total time spent by all reduces in occupied slots (ms)=
83420
                Total vcore-seconds taken by all map tasks=34168
                Total vcore-seconds taken by all reduce tasks=20855
        Map-Reduce Framework
                CPU time spent (ms)=3500
                Combine input records=7
                Combine output records=6
                Failed Shuffles=0
                GC time elapsed (ms)=1229
                Input split bytes=316
                Map input records=3
                Map output bytes=190
                Map output materialized bytes=184
                Map output records=7
                Merged Map outputs=2
                Physical memory (bytes) snapshot=906825728
                Reduce input groups=4
                Reduce input records=6
                Reduce output records=4
                Reduce shuffle bytes=184
                Shuffled Maps =2
                Spilled Records=12
                Total committed heap usage (bytes)=598155264
                Virtual memory (bytes) snapshot=3938279424
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Streaming final output from s3://mrjob-3d3e189cec521ef3/tmp/MakeStripe
s.Jason.20161003.090134.937573/output/...
"atlas" {"boon": 50, "dipped": 15}
"boon"  {"cava": 10, "atlas": 50, "dipped": 10}
"cava"  {"boon": 10, "dipped": 10}
"dipped"        {"cava": 10, "boon": 10, "atlas": 15}
Removing s3 temp directory s3://mrjob-3d3e189cec521ef3/tmp/MakeStripes.
Jason.20161003.090134.937573/...
Removing temp directory /var/folders/sz/4k2bbjts7x5fmg9sn7kh6hlw0000gn/
T/MakeStripes.Jason.20161003.090134.937573...
Removing log files in s3://mrjob-3d3e189cec521ef3/tmp/logs/j-5ZRHGZTUSQ
SW/...
```

```
Terminating cluster: j-5ZRHGZTUSQSW
CPU times: user 8.31 s, sys: 2.53 s, total: 10.8 s
Wall time: 13min 55s
```

PHASE 2: Full-scale experiment on Google N-gram data

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

3. HW5.3.2 Full-scale experiment: EDA of Google n-grams dataset (PHASE 2)

Back to Table of Contents

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

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

In [ ]:

3. HW5.3.4 OPTIONAL Question: log-log plots (PHASE 2)

Back to Table of Contents

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)

In [ ]:

In [ ]:

3. HW5.4 Synonym detection over 2Gig of Data

Back to Table of Contents

For the remainder of this assignment please feel free to eliminate stop words from your analysis

> There is also a corpus of stopwords, that is, high-frequency words like "the", "to" and "also" that we sometimes want to filter out of a document before further processing. Stopwords usually have little lexical content, and their presence in a text fails to distinguish it from other texts. Python's nltk comes with a prebuilt list of stopwords (see below). Using this stopword list filter out these tokens from your analysis and rerun the experiments in 5.5 and disucuss the results of using a stopword list and without using a stopword list.
>
> from nltk.corpus import stopwords stopwords.words('english') ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now']

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

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

In this part of the assignment we will focus on developing methods for detecting synonyms, using the Google 5-grams dataset. At a high level:

1. remove stopwords
2. get 10,0000 most frequent
3. get 1000 (9001-10000) features
4. build stripes

To accomplish this you must script two main tasks using MRJob:

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

TASK (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 TASK (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 TASK (2).

Design notes for *TASK (2)*

For this task you will have to determine a method of comparison. Here are a few that you might consider:

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

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

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

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

In [ ]:

```

```

In [ ]:

```
print "\nTop/Bottom 20 results - Similarity measures - sorted by cosine"
print "(From the entire data set)"
print '—'*117
print "{0:>30} |{1:>15} |{2:>15} |{3:>15} |{4:>15} |{5:>15}".format(
        "pair", "cosine", "jaccard", "overlap", "dice", "average")
print '-'*117

for stripe in sortedSims[:20]:
    print "{0:>30} |{1:>15f} |{2:>15f} |{3:>15f} |{4:>15f} |{5:>15f}".fo
rmat(
        stripe[0], float(stripe[1]), float(stripe[2]), float(stripe[3]),
 float(stripe[4]), float(stripe[5]) )

print '—'*117

for stripe in sortedSims[-20:]:
    print "{0:>30} |{1:>15f} |{2:>15f} |{3:>15f} |{4:>15f} |{5:>15f}".fo
rmat(
        stripe[0], float(stripe[1]), float(stripe[2]), float(stripe[3]),
 float(stripe[4]), float(stripe[5]) )
```

In [ ]:

Top/Bottom 20 results - Similarity measures - sorted by cosine
(From the entire data set)

pair	cosine	jaccard	overlap	dice	average
cons - pros	0.894427	0.800000	1.000000	0.888889	0.895829
forties - twenties	0.816497	0.666667	1.000000	0.800000	0.820791
own - time	0.809510	0.670563	0.921168	0.802799	0.801010
little - time	0.784197	0.630621	0.926101	0.773473	0.778598
found - time	0.783434	0.636364	0.883788	0.777778	0.770341
nova - scotia	0.774597	0.600000	1.000000	0.750000	0.781149
hong - kong	0.769800	0.615385	0.888889	0.761905	0.758995
life - time	0.769666	0.608789	0.925081	0.756829	0.765091
time - world	0.755476	0.585049	0.937500	0.738209	0.754058
means - time	0.752181	0.587117	0.902597	0.739854	0.745437
form - time	0.749943	0.588418	0.876733	0.740885	0.738995
infarction - myocardial	0.748331	0.560000	1.000000	0.717949	0.756570
people - time	0.745788	0.573577	0.923875	0.729010	0.743063
angeles - los	0.745499	0.586207	0.850000	0.739130	0.730209
little - own	0.739343	0.585834	0.767296	0.738834	0.707827
life - own	0.737053	0.582217	0.778502	0.735951	0.708430
anterior - posterior	0.733388	0.576471	0.790323	0.731343	0.707881
power - time	0.719611	0.533623	0.933586	0.695898	0.720680
dearly - install	0.707107	0.500000	1.000000	0.666667	0.718443
found - own	0.704802	0.544134	0.710949	0.704776	0.666165

arrival - essential	0.008258	0.004098	0.009615	0.008163	0.007534
governments - surface	0.008251	0.003534	0.014706	0.007042	0.008383
king - lesions	0.008178	0.003106	0.017857	0.006192	0.008833
clinical - stood	0.008178	0.003831			

```
0.011905 |        0.007634 |        0.007887
               till - validity |        0.008172 |        0.003367 |
0.015625 |        0.006711 |        0.008469
            evidence - started |        0.008159 |        0.003802 |
0.012048 |        0.007576 |        0.007896
               forces - record |        0.008152 |        0.003876 |
0.011364 |        0.007722 |        0.007778
               primary - stone |        0.008146 |        0.004065 |
0.009091 |        0.008097 |        0.007350
             beneath - federal |        0.008134 |        0.004082 |
0.008403 |        0.008130 |        0.007187
                factors - rose |        0.008113 |        0.004032 |
0.009346 |        0.008032 |        0.007381
           evening - functions |        0.008069 |        0.004049 |
0.008333 |        0.008065 |        0.007129
                   bone - told |        0.008061 |        0.003704 |
0.012346 |        0.007380 |        0.007873
             building - occurs |        0.008002 |        0.003891 |
0.010309 |        0.007752 |        0.007489
                 company - fig |        0.007913 |        0.003257 |
0.015152 |        0.006494 |        0.008204
               chronic - north |        0.007803 |        0.003268 |
0.014493 |        0.006515 |        0.008020
             evaluation - king |        0.007650 |        0.003030 |
0.015625 |        0.006042 |        0.008087
             resulting - stood |        0.007650 |        0.003663 |
0.010417 |        0.007299 |        0.007257
                 agent - round |        0.007515 |        0.003289 |
0.012821 |        0.006557 |        0.007546
         afterwards - analysis |        0.007387 |        0.003521 |
0.010204 |        0.007018 |        0.007032
            posterior - spirit |        0.007156 |        0.002660 |
0.016129 |        0.005305 |        0.007812
```

3. HW5.5 Evaluation of synonyms that your discovered

Back to Table of Contents

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

nltk_synonyms.py

Note: This will require installing the python nltk package:

http://www.nltk.org/install.html (http://www.nltk.org/install.html)

and downloading its data with nltk.download().

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

Calculate performance measures:

$$Precision(P) = \frac{TP}{TP + FP}$$

$$Recall(R) = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 * (precision * recall)}{precision + recall}$$

We calculate Precision by counting the number of hits and dividing by the number of occurances in our top1000 (opportunities)
We calculate Recall by counting the number of hits, and dividing by the number of synonyms in wordnet (syns)

Other diagnostic measures not implemented here: [https://en.wikipedia.org/wiki/F1_score#Diagnostic_Testing (https://en.wikipedia.org/wiki/F1_score#Diagnostic_Testing)](https://en.wikipedia.org/wiki/F1_score#Diagnostic_Testing)

In [ ]:

```
''' Performance measures '''
from __future__ import division
import numpy as np
import json
import nltk
from nltk.corpus import wordnet as wn
import sys
#print all the synset element of an element
def synonyms(string):
    syndict = {}
    for i,j in enumerate(wn.synsets(string)):
        syns = j.lemma_names()
        for syn in syns:
            syndict.setdefault(syn,1)
    return syndict.keys()
hits = []

TP = 0
FP = 0

TOTAL = 0
flag = False # so we don't double count, but at the same time don't miss
 hits

## For this part we can use one of three outputs. They are all the same,
 but were generated differently
# 1. the top 1000 from the full sorted dataset -> sortedSims[:1000]
# 2. the top 1000 from the partial sort aggragate file -> sims2/top1000s
ims
# 3. the top 1000 from the total order sort file -> head -1000 sims_part
s/part-00004

top1000sims = []
with open("sims2/top1000sims","r") as f:
    for line in f.readlines():

        line = line.strip()
        avg,lisst = line.split("\t")
        lisst = json.loads(lisst)
        lisst.append(avg)
        top1000sims.append(lisst)


measures = {}
not_in_wordnet = []

for line in top1000sims:
    TOTAL += 1

    pair = line[0]
    words = pair.split(" - ")

    for word in words:
        if word not in measures:
            measures[word] = {"syns":0,"opps": 0,"hits":0}
        measures[word]["opps"] += 1
```

```
    syns0 = synonyms(words[0])
    measures[words[1]]["syns"] = len(syns0)
    if len(syns0) == 0:
        not_in_wordnet.append(words[0])

    if words[1] in syns0:
        TP += 1
        hits.append(line)
        flag = True
        measures[words[1]]["hits"] += 1



    syns1 = synonyms(words[1])
    measures[words[0]]["syns"] = len(syns1)
    if len(syns1) == 0:
        not_in_wordnet.append(words[1])

    if words[0] in syns1:
        if flag == False:
            TP += 1
            hits.append(line)
            measures[words[0]]["hits"] += 1

    flag = False

precision = []
recall = []
f1 = []

for key in measures:
    p,r,f = 0,0,0
    if measures[key]["hits"] > 0 and measures[key]["syns"] > 0:
        p = measures[key]["hits"]/measures[key]["opps"]
        r = measures[key]["hits"]/measures[key]["syns"]
        f = 2 * (p*r)/(p+r)

    # For calculating measures, only take into account words that have s
ynonyms in wordnet
    if measures[key]["syns"] > 0:
        precision.append(p)
        recall.append(r)
        f1.append(f)


# Take the mean of each measure
print "—"*110
print "Number of Hits:",TP, "out of top",TOTAL
print "Number of words without synonyms:",len(not_in_wordnet)
print "—"*110
print "Precision\t", np.mean(precision)
print "Recall\t\t", np.mean(recall)
print "F1\t\t", np.mean(f1)
print "—"*110

print "Words without synonyms:"
print "-"*100
```

```
for word in not_in_wordnet:
    print synonyms(word),word
```

Sample output

In [ ]:

```
————————————————————————————————————————————————————————————————————————

Number of Hits: 31 out of top 1000
Number of words without synonyms: 67
————————————————————————————————————————————————————————————————————————

Precision       0.0280214404967
Recall          0.0178598869579
F1              0.013965517619
————————————————————————————————————————————————————————————————————————

Words without synonyms:
-------------------------------------------------------------------------
----------------------------
[] scotia
[] hong
[] kong
[] angeles
[] los
[] nor
[] themselves
[]
.......
```

3. HW5.6 OPTIONAL: using different vocabulary subsets

Back to Table of Contents

Repeat HW5 using vocabulary words ranked from 8001,-10,000; 7001,-10,000; 6001,-10,000; 5001,-10,000; 3001,-10,000; and 1001,-10,000; Dont forget to report you Cluster configuration.

Generate the following graphs: -- vocabulary size (X-Axis) versus CPU time for indexing -- vocabulary size (X-Axis) versus number of pairs processed -- vocabulary size (X-Axis) versus F1 measure, Precision, Recall

In [ ]:

```

```

3. HW5.7 OPTIONAL: filter stopwords

Back to Table of Contents

There is also a corpus of stopwords, that is, high-frequency words like "the", "to" and "also" that we sometimes want to filter out of a document before further processing. Stopwords usually have little lexical content, and their presence in a text fails to distinguish it from other texts. Python's nltk comes with a prebuilt list of stopwords (see below). Using this stopword list filter out these tokens from your analysis and rerun the experiments in 5.5 and disucuss the results of using a stopword list and without using a stopword list.

> from nltk.corpus import stopwords
>
> > stopwords.words('english') ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now']

In [ ]:

3. HW5.8 OPTIONAL

Back to Table of Contents

There are many good ways to build our synonym detectors, so for this optional homework, measure co-occurrence by (left/right/all) consecutive words only, or make stripes according to word co-occurrences with the accompanying 2-, 3-, or 4-grams (note here that your output will no longer be interpretable as a network) inside of the 5-grams.

In [ ]:

3. HW5.9 OPTIONAL

Back to Table of Contents

Once again, benchmark your top 10,000 associations (as in 5.5), this time for your results from 5.6. Has your detector improved?

```
In [ ]:
```