Finding Similarity between words across documents

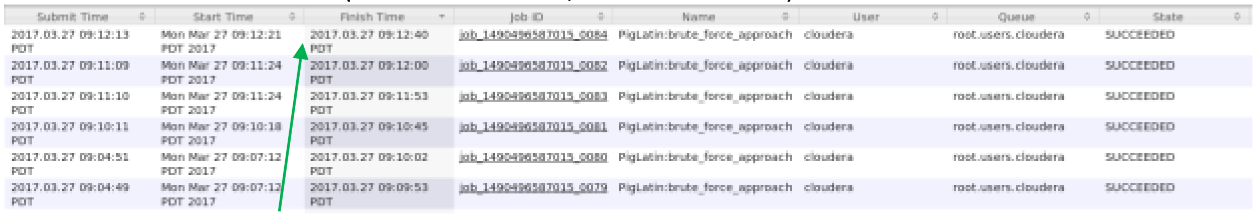
Referring to this article online: http://ucrel.lancs.ac.uk/llwizard.html

System Stats:

Framework: Hadoop Framework
Setup: Cloudera Distribution of Hadoop (Virtual Machine) (cloudera-quickstart-vm-5.8.0-0-vmware)
System: 2 Cores, 12GB Ram
Languages used: Apache Pig, Shell scripts

Approach 1: Brute Force Method

- This approach did not even generate correct results
- Created just to compare the running time between Approach1 and Approach2
- Here, calculated the word count using Linux Command: ‘`wc -w <file_name>`’
- Upload two files into HDFS and hard-code their paths inside the Pig script
- Pig script will read two input files, crunch numbers for a, b, c, d
- Join two files using Cartesian product
- For this joined result, compute similarity score (G2) using the following two equations:
  E1 = c*(a+b) / (c+d) and E2 = d*(a+b) / (c+d)
  G2 = 2*((a*ln (a/E1)) + (b*ln (b/E2)))
- Store the result in a file on HDFS
- Run-time: ~5 Minutes (Started: 9.07 am, Ended: 9.12 am)



Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State
2017.03.27 09:12:13 PDT	Mon Mar 27 09:12:21 PDT 2017	2017.03.27 09:12:40 PDT	job_1490496587015_0084	PigLatin:brute_force_approach	cloudera	root.users.cloudera	SUCCEEDED
2017.03.27 09:11:09 PDT	Mon Mar 27 09:11:24 PDT 2017	2017.03.27 09:12:00 PDT	job_1490496587015_0082	PigLatin:brute_force_approach	cloudera	root.users.cloudera	SUCCEEDED
2017.03.27 09:11:10 PDT	Mon Mar 27 09:11:24 PDT 2017	2017.03.27 09:11:53 PDT	job_1490496587015_0083	PigLatin:brute_force_approach	cloudera	root.users.cloudera	SUCCEEDED
2017.03.27 09:10:11 PDT	Mon Mar 27 09:10:18 PDT 2017	2017.03.27 09:10:45 PDT	job_1490496587015_0081	PigLatin:brute_force_approach	cloudera	root.users.cloudera	SUCCEEDED
2017.03.27 09:04:51 PDT	Mon Mar 27 09:07:12 PDT 2017	2017.03.27 09:10:02 PDT	job_1490496587015_0080	PigLatin:brute_force_approach	cloudera	root.users.cloudera	SUCCEEDED
2017.03.27 09:04:49 PDT	Mon Mar 27 09:07:12 PDT 2017	2017.03.27 09:09:53 PDT	job_1490496587015_0079	PigLatin:brute_force_approach	cloudera	root.users.cloudera	SUCCEEDED

Figure 1 Run-time for brute force approach

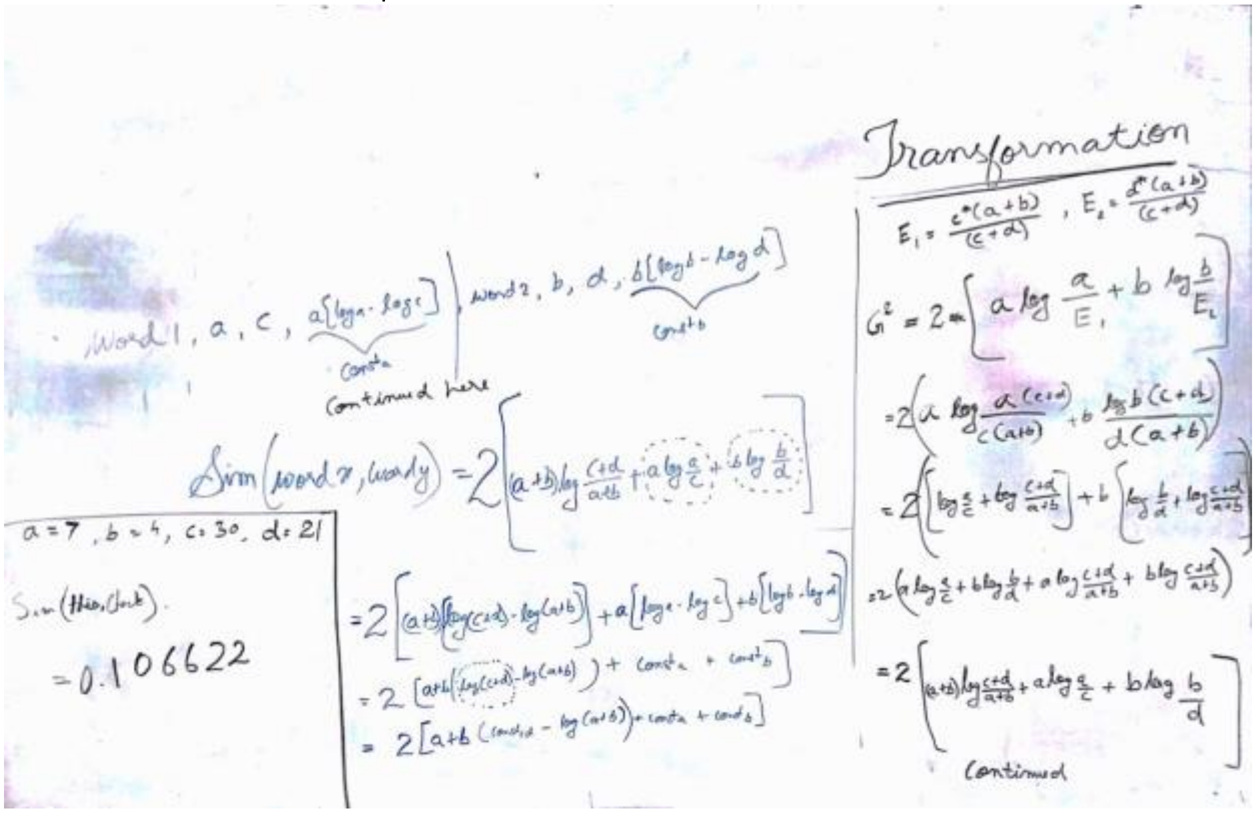
Approach 2: Stage by Stage

Stage 1: Here, we pre-process each document computing values that will be used in the equation for finding the similarity between words. Steps involved here:

1. Read document1 as plain text.
2. Separate each word and remove any control characters and punctuation marks, also convert each word into lower case, trim off white spaces.
3. Calculate the following values for each word:
   a, which denotes the occurrences of this word in document.
   c, total number of words in document.
   $const_{ac}$ = a[log(a) - log(c)], which will be used in calculating similarity score.

4. For the above variables, we calculate c and log(c) once per document. This is to reduce time by avoiding calculating same metric for each record. These two values will stay same for the entire document.
5. Repeat the above steps 1 – 4 for document2 as well. Denote the variables as ‘b’ and ‘d’.

Stage 2: Using the pre-processed files, we join two documents (corpus) using Cartesian Product. Next, we filter out the records in this joined result where word1 = word2. Compute the Similarity score using the transformed and reduced equation using the pre-computed values.
Transformed and Reduced Equation:



Transformation

$$E_1 = \frac{c*(a+b)}{(c+d)}, \quad E_2 = \frac{d*(a+b)}{(c+d)}$$

$$G^2 = 2*\left[a \log \frac{a}{E_1} + b \log \frac{b}{E_2}\right]$$

$$= 2\left(a \log \frac{a(c+d)}{c(a+b)} + b \log \frac{b(c+d)}{d(a+b)}\right)$$

$$= 2\left(a\left[\log \frac{a}{c} + \log \frac{c+d}{a+b}\right] + b\left[\log \frac{b}{d} + \log \frac{c+d}{a+b}\right]\right)$$

$$= 2\left(a \log \frac{a}{c} + b \log \frac{b}{d} + a \log \frac{c+d}{a+b} + b \log \frac{c+d}{a+b}\right)$$

$$= 2\left[(a+b) \log \frac{c+d}{a+b} + a \log \frac{a}{c} + b \log \frac{b}{d}\right]$$

Continued

word1, a, c, $\underbrace{a[\log a - \log c]}_{const_a}$ | word2, b, d, $\underbrace{b[\log b - \log d]}_{const_b}$

Continued here

$$Sim(word\,x, word\,y) = 2\left[(a+b) \log \frac{c+d}{a+b} + a \log \frac{a}{c} + b \log \frac{b}{d}\right]$$

$$= 2\left[(a+b)[\log(c+d) - \log(a+b)] + a[\log a - \log c] + b[\log b - \log d]\right]$$

$$= 2\left[a+b(\log(c+d) - \log(a+b)) + const_a + const_b\right]$$

$$= 2\left[a+b(const_{cd} - \log(a+b)) + const_a + const_b\right]$$

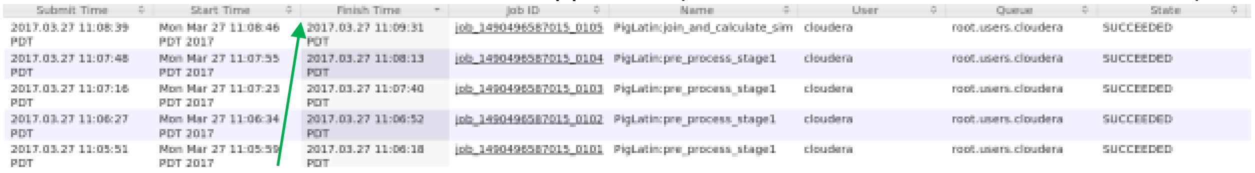
$a = 7,\ b = 4,\ c = 30,\ d = 21$

Sim(this, Clock)

$= 0.106622$

Steps involved here:
1. Read two pre-processed documents into two separate aliases
2. Join them using Cartesian product
3. Filter out the records where word1 = word2 as we need two distinct words to calculate the similarity score
4. Substitute values into the following equation:
   Sim(word1, word2) = 2*[(a+b){log(c+d) - log(a+b)} + $const_{ac}$ + $const_{bd}$]
   In the above image the terms inside dotted circle are reduced into simpler terms.
5. For the above equation, the only metric calculated while processing each record is (a+b) and log(a+b). All other variables are from pre-processed files. Log(c+d) is calculated only once for the entire process.
6. Output the final result and store in a file on HDFS. The columns will be *word1, word2, similarity_score.*

Run-time: The total run-time of both stages together was ~3min 30 seconds for the same test documents that we used for Brute-Force approach. (Started: 11:06 am, Ended: 11:09:31 am)



Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State
2017.03.27 11:08:39 PDT	Mon Mar 27 11:08:46 PDT 2017	2017.03.27 11:09:31 PDT	job_1490496587015_0105	PigLatin:join_and_calculate_sim	cloudera	root.users.cloudera	SUCCEEDED
2017.03.27 11:07:48 PDT	Mon Mar 27 11:07:55 PDT 2017	2017.03.27 11:08:13 PDT	job_1490496587015_0104	PigLatin:pre_process_stage1	cloudera	root.users.cloudera	SUCCEEDED
2017.03.27 11:07:16 PDT	Mon Mar 27 11:07:23 PDT 2017	2017.03.27 11:07:40 PDT	job_1490496587015_0103	PigLatin:pre_process_stage1	cloudera	root.users.cloudera	SUCCEEDED
2017.03.27 11:06:27 PDT	Mon Mar 27 11:06:34 PDT 2017	2017.03.27 11:06:52 PDT	job_1490496587015_0102	PigLatin:pre_process_stage1	cloudera	root.users.cloudera	SUCCEEDED
2017.03.27 11:05:51 PDT	Mon Mar 27 11:05:59 PDT 2017	2017.03.27 11:06:18 PDT	job_1490496587015_0101	PigLatin:pre_process_stage1	cloudera	root.users.cloudera	SUCCEEDED

Figure 2 Run time for Stage-by-Stage approach

Proof of correctness of algorithm and implementation:

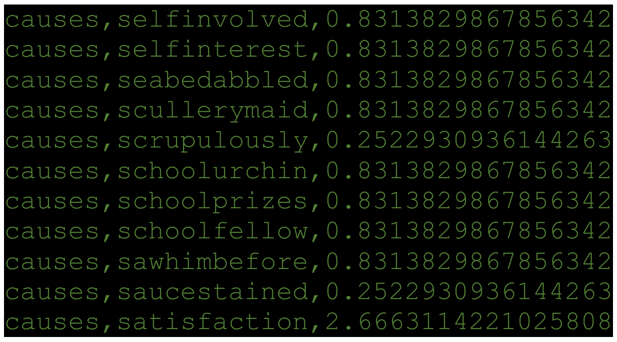
1. Prepared two documents of size 30, 21 words respectively (present in data folder data1, data2)
2. Word 'this' occurs 7 times in document1, word 'check' occurs 4 times in document2
3. Therefore, a = 7, b = 4, c = 30 and d = 21 for these two documents
4. Used the same data-set across three different procedures:
   - One procedure was online here: http://ucrel.lancs.ac.uk/llwizard.html Using the above values a, b, c, d gave G2 score as **0.11**
   - Next procedure was on white-board and hand-held scientific calculator. This was the transformed and reduced equation. Using the same 4 values, G2 score came out to be **0.106622**
   - Last Procedure was the implemented Code on Hadoop Framework. Ran the code on same two input documents. The corresponding line from output of result file reads:

```
this,check,0.10662207028751425
```

Running this across two books from Project Gutenberg:

- Document 1: https://www.gutenberg.org/ebooks/2817.txt.utf-8 This contained 70523 words
- Document 2: https://www.gutenberg.org/files/4300/4300-0.txt This contained 267875 words
- Total number of records processed: 18891348625 (18 Billion +) This was result of Cartesian product of two processed files.
- Time taken: ~57 minutes
- Result File: https://www.dropbox.com/s/pr5655wsb0eulx6/auto_out.zip?dl=0 (Huge file ~ 1.54 GB compressed, raw is ~9 GB)
- Sample Output:

```
[cloudera@quickstart Similarity]$ hadoop fs -cat
auto_dry_run_result/part-m-00007 | head -20
causes,shamewounded,0.8313829867856342
causes,shameclosing,0.8313829867856342
causes,shakespeares,0.8313829867856342
causes,seventyseven,0.2522930936144263
causes,semiluminous,0.8313829867856342
causes,semidetached,0.8313829867856342
causes,semicircular,0.8313829867856342
causes,selfweighing,0.8313829867856342
causes,selfpretence,0.8313829867856342
```



```
causes,selfinvolved,0.8313829867856342
causes,selfinterest,0.8313829867856342
causes,seabedabbled,0.8313829867856342
causes,scullerymaid,0.8313829867856342
causes,scrupulously,0.2522930936144263
causes,schoolurchin,0.8313829867856342
causes,schoolprizes,0.8313829867856342
causes,schoolfellow,0.8313829867856342
causes,sawhimbefore,0.8313829867856342
causes,saucestained,0.2522930936144263
causes,satisfaction,2.6663114221025808
```

Q. Results should be computed both over the whole corpus, as well as across different groups present in the data.

Computation across whole corpus

1. For the whole corpus with multiple documents, we will calculate the ${}^{n}C_{2}$ combinations without repeating the same pair for n documents.
2. Use the pre-processed files from Stage-1 above, two at a time and calculate similarity between words across these two documents
3. Output one result file per pair of documents. Total number of result files = ${}^{n}C_{2}$
4. This is implemented in script named “automate_script.sh”.

Computation across different groups

1. We already have the pre-processed file for each independent document.
2. We will group these pre-processed documents in groups (by author, or any other criteria). These will just be the names of the pre-processed documents in HDFS.
3. Re-calculate a, c by treating these set of documents as a corpus1 from the pre-processed documents instead of using the raw files. This helps in avoiding repeating many steps.
4. New ‘a’ will be sum of occurrences of all ‘a’ s after grouping by words.
5. New ‘c’ will be sum of ‘c’ s for the word having max value of ‘a’ across this new grouping.
6. We can get the record with max value of ‘a’ by sorting the records in descending order and picking the top record.
7. Do the same for second group of documents, treat them as corpus2.
8. Next, use Stage-2 of the above approach and use corpus1 and corpus2 as pre-processed input documents.

Additional questions: 1. Please provide an evaluation of the scalability of your solution (in terms of actual algorithm, infrastructure, or both)?

Improvements for scalability

Algorithm Wise:

1. Do not use the formulae as is for calculating G2.
2. Transform the equation and simplify it as shown in the image
3. Multiplication and Division are expensive operations for any system. Avoid as many of these as possible.
4. Pre-Calculate values which will stay constant throughout the execution. E.g only ‘a’ and ‘b’ keep changing for each record of the joined_file. Except for these two variables, use pre-computed values for all others.

Infrastructure Wise:

1. Design the solution for this problem in stages.
2. Pre-process each corpus once and calculate the metrics.
3. Store these pre-processed files in HDFS.
4. To find the similarity score of words between documents, pick the required two documents from the pre-processed list and run only Stage-2 of the above approach.
5. This saves considerable time and effort as we avoid calculating the same metrics multiple times for each document from raw.

2. Suppose working in a production environment where new documents are constantly added to the corpus. Discuss the applicability of your solution and outline potential improvements to the core algorithm - if any.

One of the approaches to work with environments where documents are constantly updated are is to employ IncMR: Incremental Data Processing based on MapReduce (Link)