Chinese Document Similarity

Analysis and Ranking based on Keyword Extraction

Name	USC ID	USC email	Division
Tian Lu	4156843753	tlu554@usc.edu	Algorithm Theory, Report Writing
Yunjie Zhao	3687586919	yunjiezh@usc.edu	Standard-Simhash, Evaluation, Visualization
Jian Chang	3693026821	jianchan@usc.edu	TF-IDF, Keywords Extraction, Modified-Simhash
Word count			2144

1 Introduction

The main purpose of our project is to explore the relationship between documents written in Chinese. At present there are millions of online articles of all fields, and chances are that when we are reading one of these articles, we actually want to get more information about the same topic or read more related essays. It's natural to come up with the idea to compare essays word by word to make sure whether they are related. However, it would be very costly when we have large amount of essays.

Under this consideration, the basic idea to find the most related essays in a large essay pool is that first we extract the keyword for every essay, and then use a Sim-Hash function in the keywords pool to get the most related essays for each given document.

2 Method

2.1 Material

We choose the following corpus to realize our project: Sogou Lab Data (SLD, http://www.sogou.com/labs/dl/c.html). SLD is comprised of a wide variety of corpus. In our project, we analyzed the relationship between about 18,000 pieces of Chinese essays with around 1,000 Chinese character in each essay, and most of them are news reports.

2.2 Procedure

2.2.1 Chinese words segmentation

Unlike English, Chinese words have no space between each other in a sentence. In order to extract the keywords of each document, the first thing we should do is to parse the sentence into separate words.

Here we use a third party Chinese words segmentation module called “Jieba” to segment the text.

2.2.2 Keywords extraction

In our plan, we decided to use the TF-IDF algorithm to extract 16 keywords for each document. Detailed procedures are as follows.

2.2.2.1 Stop words design

To make the result more accurate, we have to rule out as many noisy words as possible. We store all the unnecessary words into the file *Stop_Words.txt* with one word in each line.

The stop words including basically 4 types:

a) Punctuation.
b) Common Chinese words which mean nothing. Like 的(mean ‘s), 是(be), etc.
c) Single number or single English character.
d) Noise from other sources. For example, “ ” and “nbsp” appears many times in the essays, and it’s a space in html which is a completely noise.

In the following calculation, we read in the stop list and ignore the word in essay every time when it’s in the list.

2.2.2.2 IDF calculation

IDF means Inverse Document Frequency, and it’s a parameter to present how often a word appears in the corpus. Generally, IDF gives out the weight of a word. The more common a word appears in all kinds of documents, the less important it is.

To calculate IDF:

$$IDF(t, D) = \log(N/ nt) ;$$

Where t is a given word, D is the union of all documents, N is the number of documents in D, and nt is the number of documents that word t appears in it.

We can see from the expression that, the more times a word appears in documents, the lower IDF it will get.

And consider the case that in the target document we may get a word never appears in the corpus, we should do some smoothing:

$$IDF(t, D) = \log(N/ (nt+1));$$

And it's the final expression we use in our project.

We record the IDF result in the file called *IDFfile.txt*. The first line is the number of all documents, and from the second line, every line contains a word and the number of documents it appears. The detailed IDF will be calculated in the next step for considering accuracy.

2.2.2.3 TF-IDF calculation and keywords extraction

TF stands for Term Frequency, it means the frequency of a word in a specific document.

$$TF(t, d) = Ct / C ;$$

Where t is a given word, d is a given document, Ct is the count of word t appears in document d, C is the count of all words in d.

To calculate TF-IDF score:

$$TF\text{-}IDF(t, d, D) = TF(t, d) * IDF(t, D);$$

From the expression we know that, the more frequent a word appears in a given document, the higher score it can get; the less frequent a word appear in corpus documents, the higher score it can get. Thus, the higher score one word gets, the more likely it is to be the keyword.

And for each file, we get the top 16 words who get the highest score and extract them as keywords.

The results are stored in file *Demo_Corpus_KeyWords.txt*, with each line represent a document.

2.2.3 Similarity calculation

We calculate the similarity and relationship between documents by considering their keywords which are stored in *Demo_Corpus_KeyWords.txt* in previous steps.

2.2.3.1 Standard Sim-Hash function

The standard Sim-Hash function are as follows [1]:

a) Use hash function to hash all words into 128 bits 0-1 string;
b) For each bit of the hash code, if it's 0, then make it –wi; if it's 1, make it +wi; where wi is the assigned weight of the word, and it can be calculated by IDF;
c) Given all the keywords hash code of a certain document, for the 128 bits get the sum of each bit;
d) For each bit, if it's greater than 0, make it 1; else, make it 0, and make the 128 bit 0-1 string to be a binary number ***bi*** represent the document;
e) To compare two documents ***i*** and ***j***, do XOR function between ***bi*** and ***bj***. The number of 1 in the result is called humming distance. The smaller it is, the similar they are.

We tried this function, but the result is quite disappointing. Here is one of the typical example. Our target file is *C14/1824.txt*, call it File1, keywords as follows:

```
刘翔/田径/大阪/栏/大奖赛/锦标赛/比赛/室内/伤脚/夺冠/肌肉/刘翔脚/踝关节/训练/上周二/八九月份

Liuxiang/tianjing/daban/lan/dajiangsai/jinbiaosai/bisai/shinei/shangjiao/duoguan/jirou/liuxiangjiao/huaiguanjie/xunlian/
shangzhouer/bajiuyuefen

Liu-Xiang(athlete)/track-and-field/Osaka/ hurdles/championship(presentation1)/championship(presentation2)/race/indoor/
foot-injury/win-the-champion/muscle/Liuxiang's food/ankle/training/last Tuesday/Aug. and Sep.
```

File 1

The news talks about athlete Liu Xiang's race after his injure in Osaka, and our idea related file is *C14/1452.txt*, call it File2, which talks about the same thing, and given its keywords:

```
刘翔/大阪/大奖赛/比赛/阿兰/秒/三连冠/田径/约翰逊/锦标赛/成绩/室内/运动员/栏驾/夺冠/维格诺尔

Liuxiang/daban/bisai/alan/miao/sanlianguan/tianjing/yuehanxun/jinbiaosai/chengji/shinei/yundongyuan/lanjia/duoguan/weigenuoer

Liu Xiang(an athlete)/ Osaka/ championship(presentation1)/race/Alan(a-name)/second/ three-successive-championships/
track-and-field/Johansson/championship(presentation2)/score/indoor/athelete/column/win the champion/Wignall
```

File 2

These two keywords set get a humming distance of 41.

However, by using the Sim-Hash function, we get file *C22/1582.txt* as the most related one with a humming distance of 39, keywords as follows:

```
李娜/工作/不想/家庭/页/比起/女性/谢小姐/第/养儿育/胜/看看书/她/过起/上/上网/什麼

Lina/gongzuo/buxiang/jiating/ye/biqi/nvxing/xiexiaojie/di/yangeryu/sheng/kankanshu/ta/guoqi/shang/shangwang/shenme

Lina/work/don't-want-to/family/page/compare/female/miss-Xie/first/children/beat/reading/she/live/up/internet/what
```

File 3

It's obvious that they are totally different, and this situation appears in many testing files.

We have tried different hash function to generate the result, though every time our idea file File2 get a good score, there are always some totally unrelated file comes out with a lower humming distance.

To explain this, we think that it's because when comparing each file with other 18,000 files, there may exist some irrelevant files whose characters are offset when doing hash, which will cause a short hamming distance.

2.2.3.2 Modified Sim-Hash function

To make the result more accurate, we use a modified Sim-Hash function. Instead hash a word into 0-1 string, we take each word as a bit of 0-1.

When comparing two sets of keywords, we first get a union of the two sets, then for each file, consider each word in the union, if the word is in the file, then this bit is 1, else it's 0. The final score is decided by the hamming distance of two file divided by the length of the union:

FinalScore = 1 - (HammingDistance/UnionLength);

The larger it is, the more similar the documents are.

Consider the previous File1, File2 and File3. When compare File1 and File2:

```
Union Sequence:

Liuxiang/tianjing/daban/lan/dajiangsai/jinbiaosai/bisai/shinei/shangjiao/duoguan/jirou/
liuxiangjiao/huaiguanjie/xunlian/shangzhouer/bajiuyuefen/alan/miao/sanlianguan/yuehanxun/
chengji/yundongyuan/lanjia/weigenuoer

Liu-Xiang(athlete)/track-and-field/Osaka/hurdles/championship(presentation1)/
championship(presentation2)/race/indoor/foot-injury/win-the-champion/muscle/Liuxiang's-
food/ankle/training/last-Tuesday/Aug-and-Sep/Alan(a-name)/second/three-successive-
championships/Johansson/score/athlete/column/Wignall

File1: 111111111111111100000000
File2: 111011110100000011111111
```

File1 and File2 have a hamming distance of 16, and the length of union is 24, and final score is 1-(16/24)=0.33. For File1 and File3, final score is 1-32/32=0. Thus File2 is much better than File 3.

Also we tried many other files and this function gets a good result.

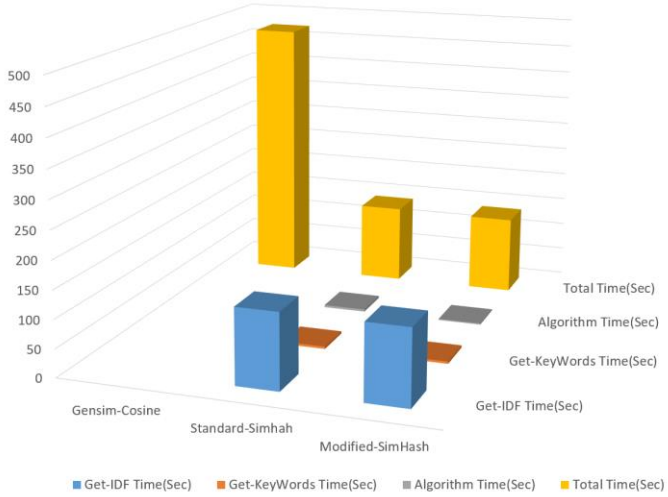
2.2.4 Visualization tool

Our output is a list about each document name and its top-10 similarity documents names. To make our data visualization more intuitive and concise, we use the Force-Directed Graph of D3.js library to draw it. We visualized two graph, one is from the modified-simhash similarity output, and another is from the evaluation standard (Gensim cosine similarity) output. To reduce the visual complexity, we only choose the best similar match for each document. You can see these graphs in Part 3 "Result" of this report. Simply introduce our Force-Directed Graph is that each node represent a document and an edge between two node means one is another's best similar match. You can drag a node and its connected nodes using mouse to click on it. We also achieve a fisheye distortion feature on our graphs, it based on mouseover to distort and zoom the nodes and edges. We also use the JQuery to achieve a paging table from the same demo modified-simhash output, it will be convenient for you to check together with the force-directed graph.

2.3 Evaluation

We are mainly concerned with evaluating the similar accuracy and the speed of our algorithm. To perform this evaluation, we randomly choose about 500 .txt files. Our evaluation standard is a python library – Gensim. Gensim is a robust, efficient and hassle-free piece of software to realize unsupervised semantic modelling from plain text to generate text similarity. It's a powerful library and contains many ways to generate similarity. Here we choose its cosine similarity function by TF-IDF and Latent Semantic Indexing (LSI) to be the standard [2]. The test data is our Standard-SimHash Similarity output and Modified-SimHash Similarity output.

2.3.1 Evaluation of Algorithm Speed



The Gensim Cosine total run time is 7min 52sec. Because each txt file contains about 1,000 Chinese character. The Cosine Similarity will construct many vectors for each file. The square calculation for each vector will greatly slow down algorithm speed. It cannot imagine if we increase the demo corpus size to thousands and millions.

Both our two SimHash functions are based on the same TF-IDF and Keywords modeling data. The training data of TF-IDF calculation is our 17,000 txt files, and this step spends 2min 14sec. Keywords abstraction only calculates the 500 txt files, it spends 4sec. The Standard-SimHash spends 4sec, and Modified-SimHash spends 2sec. Both of them are much faster than Cosine way.

2.3.2 Evaluation of Algorithm Accuracy

To compare standard output and our output, calculating each file score can shows us the accuracy within them. The key point is how to define the weight of each one of Top-N list.

$$score(File_i) = \sum_{i=1}^{N}(a_i \sum_{j=1}^{M} \frac{C_{ij}}{M})$$

a) **N** is the length of Top-N, in this test, our output is a top-10 similarity list of each txt file, so the value of N is 10;

b) **M** is the length of Demo-Corpus size, in this test, we tested about 500 txt files, so the value is 500;

c) The value of a_i is 0 or 1. 0 means the cosine similarity ranking list[i] of Top-10 is not found in the test list, and 1 means has found;

d) $Cij\,(0 \le Cij \le 1)$ is the cosine similarity between a txt file and its Top-10 files.

Here is our output format:

```
Modified-SimHash,Standard-SimHash,Gensim-Cosine TopN Output Format
    file(1) List[1] List[2] List[3] list[4] ... List[N]
    file(2) ...     ...     ...     ...     ...     ...
      .     ...     ...     ...     ...     ...     ...
      .     ...     ...     ...     ...     ...     ...
    file(M) ...     ...     ...     ...     ...     ...

Gensim-Cosine Similarity Value Format
    file(1) C[1,1] C[2,1] C[3,1] C[4,1] ... C[N,1]
    file(2) C[1,2] C[2,2] C[3,2] C[4,2] ... C[N,2]
      .     ...    ...    ...    ...    ...    ...
      .     ...    ...    ...    ...    ...    ...
    file(M) ...    ...    ...    ...    ... C[N,M]
```

Then, we get can get the accuracy for our algorithm. It's a float value from 0 to 1.

$$accuracy = \sum_{i=1}^{M} \frac{score(File_i)}{max(score(File)) \times M}$$

3 Result

The final Top-N results are stored in *Similarity.txt (Modified-SimHash Output)* and *Similarity_SimHash.txt (Standard-SimHash Output)*. In each line, the first file name is the target file name, and it is followed by the names of the most related files in the corpus.

The accuracy scores are stored in *Evaluation_Score.txt (Modified-SimHash Output)* and *Evaluation_Score_SimHash.txt (Standard-SimHash Output)*.

Here is the accuracy of our algorithm comparing with Gensim Cosine Similarity:

Random 500 txt Files	Top 10 Accuracy
Standard SimHash	11.01%
Modified SimHash	67.21%

Here is Hamming Search Accuracy by Deepak Ravichandran, Patrick Pantel, and Eduard Hovy [3]:

Random permutations q	Top 1	Top 5	Top 10	Top 25	Top 50	Top 100
25	9.2%	9.5%	7.9%	6.4%	5.8%	4.7%
50	15.4%	17.7%	14.6%	12.0%	10.9%	9.0%
100	27.8%	27.2%	23.5%	19.4%	17.9%	16.3%
500	73.1%	67.0%	60.7%	55.2%	53.0%	50.5%
1000	87.6%	84.4%	82.1%	78.9%	75.8%	72.8%

First, documents similarity ranking judgement is not an either-or thing. To achieve a high evaluation score is almost impossible. The difference of training data and the algorithm statistical parameter decided this. Compare with the result of Deepak Ravichandran, we get a similar accuracy score, 60%, in 500 documents size and Top-10 list. It's a not-bad result.

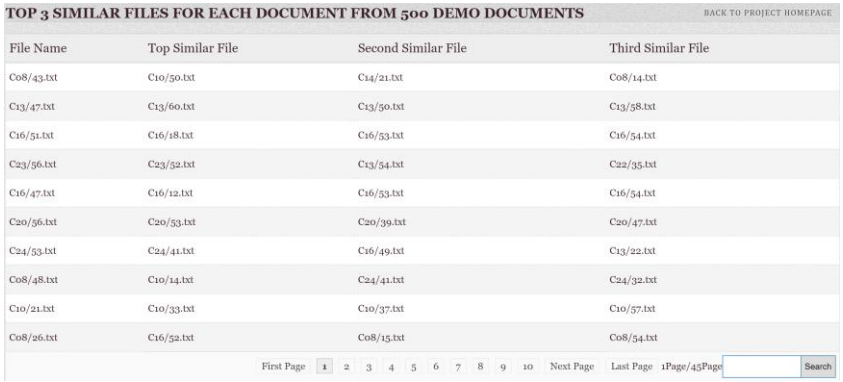
Second, our algorithm used the 17,000 txt files to training our TF-IDF value, but Gensim only used the 500 demo txt files to generate TF-IDF. According to the different keywords, we found Gensim reduced the influence of words frequency to IDF. It means a word only appears a few times in a corpus, and this word should be a very important word, but it may not have a high IDF score. Gensim has considered the words internal relations within a document using the LSI. So, Gensim has a more similarity accuracy than our modified-simhash. These are the crucial differences so that Gensim has a high score.

Why the Standard-Simhash has such a pathetic score? Standard-Simhash will add the product for each keyword and its weight. It will greatly increase the influence of weight, and ignore the important of keywords set.

```
Document:   word1           word2           word3
Hashcode:   1001            0110            0101
Weight  :   9               2               3
            [9][-9][-9][9] + [-2][2][2][-2] + [-3][3][-3][3]
            [4][-4][-10][10]
SimHashCode:1001
```

Likes the above example, word1 has a very high weight, so the simhash code of document almost equal to the hash code of word1. It reduced from keywords comparison to one keyword comparison. If two document is easy to distinguish similar or not, it's very efficient. But if there are two ambiguous document and weights are very not balanced. It's will produce unsatisfied result.

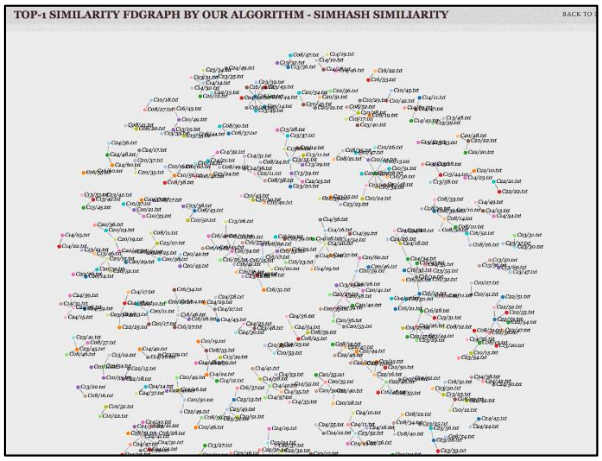
Also, we have put our result in http://www-scf.usc.edu/~yunjiezh/csci544/projecthome.html and realized visualization.

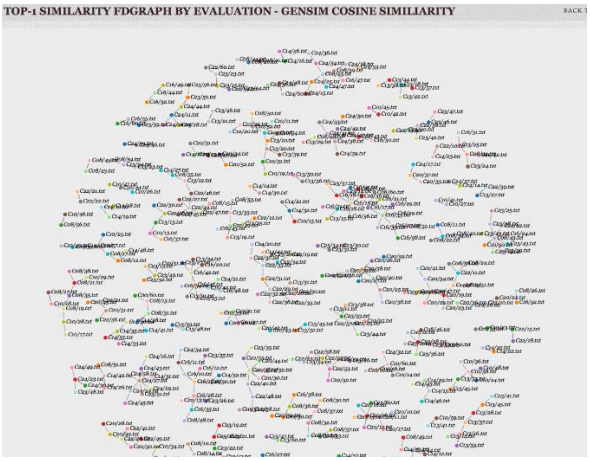

TOP 3 SIMILAR FILES FOR EACH DOCUMENT FROM 500 DEMO DOCUMENTS

BACK TO PROJECT HOMEPAGE

File Name	Top Similar File	Second Similar File	Third Similar File
C08/43.txt	C10/50.txt	C14/21.txt	C08/14.txt
C13/47.txt	C13/60.txt	C13/50.txt	C13/58.txt
C16/51.txt	C16/18.txt	C16/53.txt	C16/54.txt
C23/56.txt	C23/52.txt	C13/54.txt	C22/35.txt
C16/47.txt	C16/12.txt	C16/53.txt	C16/54.txt
C20/56.txt	C20/53.txt	C20/39.txt	C20/47.txt
C24/53.txt	C24/41.txt	C16/49.txt	C13/22.txt
C08/48.txt	C10/14.txt	C24/41.txt	C24/32.txt
C10/21.txt	C10/33.txt	C10/37.txt	C10/57.txt
C08/26.txt	C16/52.txt	C08/15.txt	C08/54.txt

First Page 1 2 3 4 5 6 7 8 9 10 Next Page Last Page 1Page/45Page Search





You can test our code from GitHub: https://github.com/WinterCJ/Chinese-Document-Similarity-Analysis-and-Ranking-based-on-Key-Words-Extraction.git

4 Discussion

From the result we can see that by extracting keywords and using TF-IDF algorithm, we successfully measured the relationship between corpus documents in acceptable time. And the research can be applied in providing readers more related essays especially in the area of academic research.

During our discussion and research, we found that one of the reasons that may cause the inaccuracy is Chinese words segmentation. Many proper none, like company names set up by simple Chinese characters are not able to be recognized. In the future research we can study more about the words segmentation and make the final result more accurate.

In NLP similarity research field, to get a better time complexity, dimensionality reduction is the main thinking, but it must be produce an effect on accuracy. Our TF-IDF and LSH model has a lots of room for improvement. Self-adaption parameter processing, POS preprocessing and more data to train are very important. NLP in Chinese is still at an early stage. Web similarity algorithm in Baidu is much worse than Google. We hope that our research will make a contribution to Chinese corpus similarity and will make feasible at the terascale. Data internal connection also closely associated with similarity. We believe that document similarity will play a more important role in future.

Reference

[1] *Moses S. Charikar. 2002. Similarity estimation techniques from rounding algorithms. In Proceedings of the 34th annual ACM symposium on Theory of computing, pages 380-388.*

[2] *Pantel, Patrick and Dekang Lin 2002. Discovering Word Senses from Text. In Proceedings of SIGKDD-02, pp. 613–619.*

[3] *Deepak Ravichandran, Patrick Pantel and Eduard Hovy 2005. Randomized Algorithms and NLP: Using Locality Sensitive Hash Functions For High Speed Noun Clustering. In Proceedings of Annual Meeting of the Association of Computational Linguistics, P05-1077*