Homework 2

Jiachi Liu

October 2, 2013

Problem 1

Overview of Plagiarism Detection System

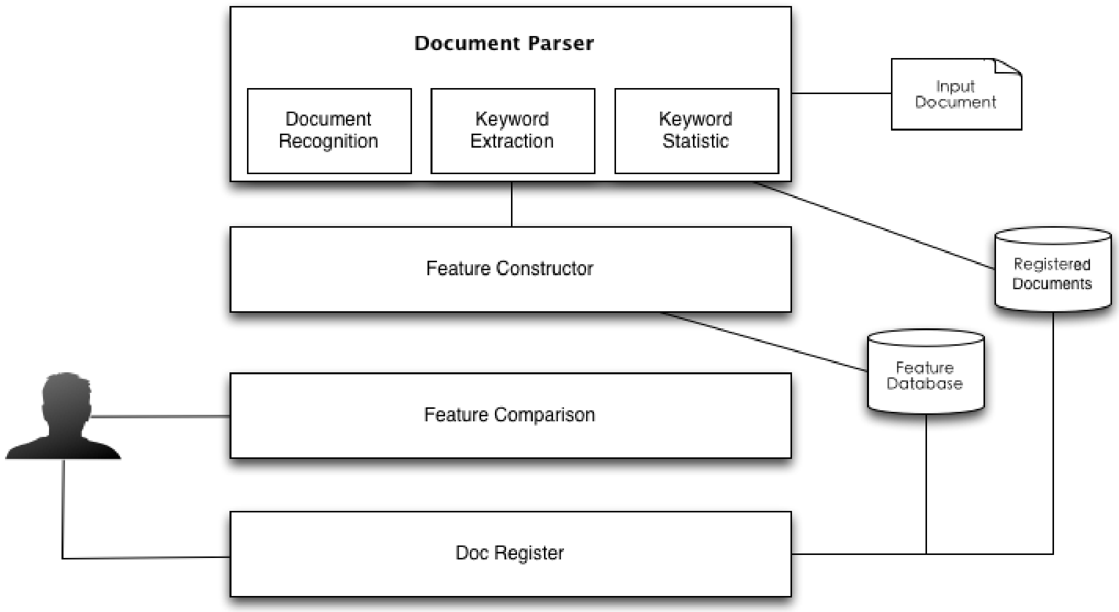

Figure 1: Plagiarism Detection System

The Plagiarism Detection System takes a input document and compare it to all the documents in registered document database. For each document in database as well as input document, the system will created fingerprint based on passage and compare each passage to find similarity in documents. For those documents that fail the detection, it will return to user for human justification. And for those file that pass the detection they will be added to registered database as original file that would be compared with future undecided document. Figure 1 shows the architecture of plagiarism detection system. It contains the following components:

- Document Parser: The document parser component parses the raw input document into plain text, extracts the keywords in the unit of passage and makes a keyword statistic to assign the weight for each keyword.
- Feature Constructor: The feature constructor component takes the result of keyword statistic and using simhash to create fingerprint for each passage in document.
- Feature Comparison: The feature comparison component will compare each passage in two given documents based on the passage's fingerprint and returns whether it is similar.
- Doc Register: the doc register component will add the document that considered to be original to registered document.

Document Parser

Document Parser has three sub-module: Document Recognition, Keyword Extraction and Keyword Statistic.

- Document Recognition: this module is responsible to convert the different format of input document into plain text. It should support the major format of documents such as txt, pdf, html or latex. After converted, the result plain text will be passed to Keyword Extraction module for further processing.
- Keyword Extraction: this module takes the input plain text for a document, it will eliminate the stop words such as "the", "and", "or",etc. and preserve other words as keywords which can represent the document content. Furthermore, in html file, the markdown like "¡a¿" also will be eliminate.
- Keyword Statistic: after getting the keyword collection of document, for each passage in document, this module will statistic the frequency, position and other useful information of each keyword. And it will create a inverse index list for each passage that shows the information above mentioned. Thus for each document, there will be a list of inverse index lists after statistic. Those lists will be saved into feature database for further feature creation.

Feature Constructor

The feature constructor component is responsible to calculate the fingerprint for each passage in document based on the keyword statistic result. It uses simhash algorithm to calculate the fingerprint for each passages in document as the following steps:

- Read statistic results of current passage in feature database.
- For each word in results, assigned weight to each keywords. The weight could be the frequency statistic.
- Generate hash value in b bits for each word where b should be picked to ensure that the hash value should be unique for all words.
- Caculate the b-dimensional vector V. Updating each component of V by adding or subtracting the weight of each words.
- Reduce V to 0-1 vector.

The final 0-1 vector is the fingerprint of current passage. Save it to feature database so that it can be used for future comparison.

Feature Comparison

After processing all the registered documents and get the features of them, we can compare new input document with the records in feature database. When a new document input to the system, first using the Document Parser and Feature Constructor to parse and get the feature of the document. Then for each passages in the new document, compare the fingerprint with all passages' fingerprint in feature database.
The comparison is using Hamming distance to test whether two fingerprint is similar. The Hamming distance is the total number of different bit in two fingerprints. Normally, if the hamming distance is less than 3, then the two fingerprints could be considered as similar.
After finishing comparison all the passages in new document, if their are any similarity was found, the result will be showed to user for human justification. Otherwise, there is no similarity.

Doc Register

For those documents that is not considered as plagiarism, the user could use doc register component to save these documents into register document database and save their fingerprints and statistic result in feature database.

Reference

- Charikar, Moses S. "Similarity estimation techniques from rounding algorithms." Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. ACM, 2002.
- Croft, W. Bruce, Donald Metzler, and Trevor Strohman. Search engines: Information retrieval in practice. Reading: Addison-Wesley, 2010.
- Si, Antonio, Hong Va Leong, and Rynson WH Lau. "CHECK: a document plagiarism detection system." Proceedings of the 1997 ACM symposium on Applied computing. ACM, 1997.
- Manku, Gurmeet Singh, Arvind Jain, and Anish Das Sarma. "Detecting near-duplicates for web crawling." Proceedings of the 16th international conference on World Wide Web. ACM, 2007.

Problem 2

a)

The top 25 words in document is:
format: [rank,word,frequency,probability of occurrence,product of probability and rank,product of frequency and rank]

```
[ 1, the, 1644, 0.0615891806841, 0.0615891806841, 1644 ]
[ 2, and, 872, 0.0326677406062, 0.0653354812123, 1744 ]
[ 3, to, 729, 0.0273105308508, 0.0819315925524, 2187 ]
[ 4, a, 632, 0.0236766193384, 0.0947064773536, 2528 ]
[ 5, she, 541, 0.0202674858577, 0.101337429289, 2705 ]
[ 6, it, 530, 0.0198553927996, 0.119132356798, 3180 ]
[ 7, of, 514, 0.0192559847151, 0.134791893006, 3598 ]
[ 8, said, 462, 0.0173079084404, 0.138463267523, 3696 ]
[ 9, i, 410, 0.0153598321657, 0.138238489492, 3690 ]
[ 10, alice, 386, 0.014460720039, 0.14460720039, 3860 ]
[ 11, in, 369, 0.0138238489492, 0.152062338441, 4059 ]
[ 12, you, 365, 0.013673996928, 0.164087963136, 4380 ]
[ 13, was, 357, 0.0133742928858, 0.173865807515, 4641 ]
[ 14, that, 280, 0.010489641479, 0.146854980707, 3920 ]
[ 15, as, 263, 0.00985277038924, 0.147791555839, 3945 ]
[ 16, her, 248, 0.00929082531001, 0.14865320496, 3968 ]
[ 17, at, 212, 0.00794215711984, 0.135016671037, 3604 ]
[ 18, on, 193, 0.00723036001948, 0.130146480351, 3474 ]
[ 19, all, 182, 0.00681826696138, 0.129547072266, 3458 ]
[ 20, with, 181, 0.00678080395609, 0.135616079122, 3620 ]
[ 21, had, 178, 0.00666841494025, 0.140036713745, 3738 ]
[ 22, but, 170, 0.00636871089799, 0.140111639756, 3740 ]
[ 23, for, 153, 0.00573183980819, 0.131832315588, 3519 ]
[ 24, so, 151, 0.00565691379762, 0.135765931143, 3624 ]
[ 25, be, 148, 0.00554452478178, 0.138613119544, 3700 ]
```

Next 25 words start with 'f':

```
[ 91, first, 51, 0.0019106132694, 0.173865807515, 4641 ]
[ 127, from, 36, 0.00134866819016, 0.171280860151, 4572 ]
[ 141, found, 32, 0.00119881616903, 0.169033079834, 4512 ]
[ 187, felt, 23, 0.000861649121493, 0.161128385719, 4301 ]
[ 197, find, 21, 0.000786723110928, 0.154984452853, 4137 ]
[ 210, feet, 19, 0.000711797100363, 0.149477391076, 3990 ]
[ 251, face, 15, 0.000561945079234, 0.141048214888, 3765 ]
[ 297, footman, 13, 0.00048701906867, 0.144644663395, 3861 ]
[ 305, far, 13, 0.00048701906867, 0.148540815944, 3965 ]
[ 324, finished, 12, 0.000449556063387, 0.145656164538, 3888 ]
[ 371, fan, 10, 0.000374630052823, 0.138987749597, 3710 ]
[ 375, foot, 10, 0.000374630052823, 0.140486269809, 3750 ]
[ 416, few, 9, 0.000337167047541, 0.140261491777, 3744 ]
[ 422, four, 8, 0.000299704042258, 0.126475105833, 3376 ]
[ 452, followed, 8, 0.000299704042258, 0.135466227101, 3616 ]
[ 455, fish, 8, 0.000299704042258, 0.136365339228, 3640 ]
[ 457, feel, 8, 0.000299704042258, 0.136964747312, 3656 ]
[ 471, fact, 8, 0.000299704042258, 0.141160603904, 3768 ]
[ 473, five, 8, 0.000299704042258, 0.141760011988, 3784 ]
[ 482, feeling, 7, 0.000262241036976, 0.126400179822, 3374 ]
[ 488, frightened, 7, 0.000262241036976, 0.127973626044, 3416 ]
[ 511, fall, 7, 0.000262241036976, 0.134005169895, 3577 ]
[ 521, fancy, 7, 0.000262241036976, 0.136627580264, 3647 ]
[ 535, fetch, 7, 0.000262241036976, 0.140298954782, 3745 ]
[ 560, fell, 6, 0.000224778031694, 0.125875697748, 3360 ]
```

Total words is 26693 and Vocabulary Size is 2632.

Using matlab to plot. Using the rank as x-coordinate and frequency as y-coordinate, the result is showed as following:

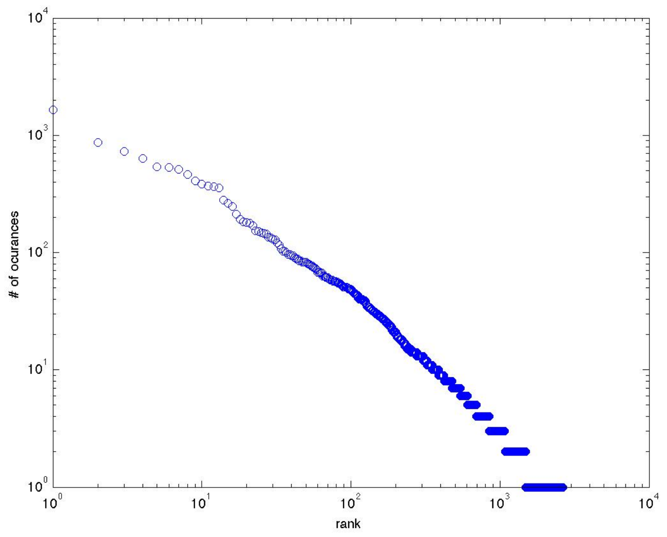

Figure 2: Matlab Plot of Alice in Wonderland

According to the image, although the beginning and ending of text is not follow the Zipf's Law due to its high or low frequency, the middle part still satisfies. Thus the text in Alice in Wonderland satisfies Zipf's Law.

b)

Words that omitted:

```
Total omit vocabulary (occur == 1 ): 1145
Actual omit proportion: 0.435030395137
Estimate omit proportion: 0.5

Total omit vocabulary (occur == 2 ): 408
Actual omit proportion: 0.155015197568
Estimate omit proportion: 0.166666666667

Total omit vocabulary (occur == 3 ): 233
Actual omit proportion: 0.0885258358663
Estimate omit proportion: 0.0833333333333

Total omit vocabulary (occur == 4 ): 149
Actual omit proportion: 0.0566109422492
Estimate omit proportion: 0.05

Total actual omit proportion:  0.735182370821
Total estimated omit proportion:  0.8
```

According to Zipf's law, using the formula $p = 1/n * (n + 1)$ where n is the occurrence of word to calculate the total proportion of omitted words. $1/(1 * 2) + 1/(2 * 3) + 1/(3 * 4) + 1/(4 * 5) = 0.8$. Thus 80% words would be omitted. In *Alice in Wonderland* text, 73.5% words is actually omitted.

Problem 3

a)

According to Heap's Law we have $v = k * n^{\beta}$, so we have $0.9 * v = k * (xn)^{\beta}$ where $\beta = 0.5$. Solve these two equation we have $x = 0.81$. Thus about 81% of a collection of text must be read before 90% of its vocabulary has been encountered.

b)

The "q3.py" is using numpy and scipy library to calculate the value of k and β in function $v = k * n^{\beta}$. It uses curve_fit function that based on least square method in scipy.optimization. By reading the document, each time we meet a words we add current total words that already read and vocabulary size as a pair (x,y). And after finishing read document, pass the list of (x,y) and function $v = k * n^{\beta}$ to curve_fit method to get the result k and β.
The result is: $[k, \beta] = [8.98631362, 0.55792255]$

Problem 4

a)

The two query in Google has the same result that the estimated result size is about half of the actual returns from search engine. Google may have other algorithms to estimate result size which is more accurate. And since the passed values in formula $f_{ac} * f_{bc}/f_c$ is also estimated value, it may cause incorrect estimation.
However, in Bing, the estimation is approximately equal to the actual returns. My guess is that Bing is using the same formula to calculate the result size.

Table 1: Query "fantasy japanese animation" in Google and Bing

Search Engine	Google
f_{ac}	87,100,000
f_{bc}	104,000,000
f_c	579,000,000
actual	34,300,000
$estimate = f_{ac} * f_{bc}/f_c$	15,644,905
f_a	731,000,000
$N = f_a * f_c/f_{ac}$	4,859,345,580
Search Engine	Bing
f_{ac}	43,000,000
f_{bc}	57,500,000
f_c	168,000,000
actual	16,000,000
$estimate = f_{ac} * f_{bc}/f_c$	14,717,262
f_a	126,000,000
$N = f_a * f_c/f_{ac}$	492,279,070

Table 2: Query "data mining tools" in Google and Bing

Search Engine	Google
f_{ac}	671,000,000
f_{bc}	126,000,000
f_c	2,710,000,000
actual	74,000,000
$estimate = f_{ac} * f_{bc}/f_c$	31,197,786
f_a	5,280,000,000
$N = f_a * f_c/f_{ac}$	21,324,590,164
Search Engine	Bing
f_{ac}	121,000,000
f_{bc}	31,700,000
f_c	222,000,000
actual	17,200,000
$estimate = f_{ac} * f_{bc}/f_c$	17,277,928
f_a	298,000,000
$N = f_a * f_c/f_{ac}$	546,743,802

b)

Choosing two words, "java" and "Gandhi", which have less probability of dependency as control group to estimate the document collection for Google and Bing.
Compared with N in table 1 and table 2, in google, using words "fantasy animation" to estimate

Table 3: Query "java" and "Gandhi" in Google and Bing

Search Engine	Google
f_{java}	778,000,000
f_{Gandhi}	99,500,000
$f_{java \cap Gandhi}$	4,060,000
N	19,066,748,768
Search Engine	Bing
f_{java}	107,000,000
f_{Gandhi}	9,960,000
$f_{java \cap Gandhi}$	248,000
N	4,297,258,065

N=4,859,345,580 which is far more less than 19,066,748,768 in control group, whereas using "data tools" to estimate we have N=21,324,590,164 which is near the control group. The reason is probably that "fantasy animation" is more likely appears together which cause to a large overlap between $f_{fantasy}$,$f_{animation}$ and $f_{fantasy \cap animation}$ and thus lower estimated results of N.

On the other hand, Using Bing as search engine, the estimation is very similar but far more less than control group. I think the reason maybe is that the chosen two words has the save dependency in Bing but still less independent which cause the over-small result in estimation.