Selecting the Most Highly Correlated Pairs within a Large Vocabulary

Kyoji Umemura
Department of Computer Science
Toyoahshi University of Technology
umemura@tutics.tut.ac.jp

Abstract

Occurence patterns of words in documents can be expressed as binary vectors. When two vectors are similar, the two words corresponding to the vectors may have some implicit relationship with each other. We call these two words a correlated pair. This report describes a method for obtaining the most highly correlated pairs of a given size. In practice, the method requires $O(N \times \log(N))$ computation time, and $O(N)$ memory space, where N is the number of documents or records. Since this does not depend on the size of the vocabulary under analysis, it is possible to compute correlations between all the words in a corpus.

1 Introduction

In order to find relationships between words in a large corpus or between labels in a large database, we may use a distance measure between the binary vectors of N dimensions, where N is the number of documents or records, and the ith element is 1 if the ith document/record contains the word or the label, or 0 otherwise.

There are several distance measures suitable for this purpose, such as the mutual information(Church and Hanks, 1990), the dice coefficient(Manning and Schueutze 8.5, 1999), the phi coefficient(Manning and Schuetze 5.3.3, 1999), the cosine measure(Manning and Schueutze 8.5, 1999) and the confidence(Arrawal and Srikant, 1995). There are also special functions for certain applications, such as then complimentary similarity measure (CSM)(Hagita and Sawaki, 1995) which is known as to be suitable for cases with a noisy pattern.

All of these five measures can be obtained from a simple contingency table. This table has four numbers for each word/label x and word/label y. The first number is the number of documents/records that have both x and y. We define this number as $df_a(x, y)$. The second number is the number of documents/records that have x but not y. We define this number as $df_b(x, y)$. The third number is the number of documents/records that do not have x but do have y. We define this number as $df_c(x, y)$. The fourth and the last number is the number of documents/records that have neither x nor y. We define this number as $df_d(x, y)$.

An obvious method to obtain the most highly related pairs is to calculate df_a, df_b, df_c, df_d for all pairs of words/labels, compute the similarity for all pairs and then select pairs of the highest values. Let l be the number of possible words/labels, and N be the total number of documents/records in a corpus/database. This method requires $O(l^2)$ memory space and $O(l^2 \times N)$ computation time. However, its use is only feasible if l is smaller than 10^4. When l is larger than ten thousand, execution of this procedure becomes difficult.

The method described here is based on the observation that there is an upper boundary to the number of different words in one document. The assumption of such a boundary could even made of a large scale

corpus. For example, a collective corpus of a newspaper may become larger and larger, but the length of each article is stable. It is not likely that one article would contain thousands of different words.

In view of this observation and the assumption, this method is effective for obtaining the most highly correlated pairs in a large corpus, and uses $O(N)$ memory space, and $O(N \times \log(N))$ computation time.

2 Notations

Several notations are introduced in this section to describe the method. Assuming a corpus C, which is a set of sets of words, values are assigned as follows.

- d : documents (elements of the corpus).

$$d \in C$$

- x, y, z : label (elements of a document).

$$x \in d, y \in d, z \in d$$

- $x < y$: y is placed after x in the alphabetical order.
- N : the total number of documents.

$$N = \mid C \mid$$

- $df(x)$: the number of documents that contain x.

$$df(x) = \mid \{d | x \in d\} \mid$$

- $df_a(x, y)$: the number of documents that contain x and contain y.

$$df_a(x, y) = \mid \{d | x \in d \wedge y \in d\} \mid$$

- $df_b(x, y)$: the number of documents that contain x but not y.

$$df_b(x, y) = \mid \{d | x \in d \wedge y \notin d\} \mid$$

- $df_c(x, y)$: the number of documents that contains y but not x.

$$df_c(x, y) = \mid \{d | x \notin d \wedge y \in d\} \mid$$

- $df_d(x, y)$: the number of documents that conetain neither x nor y.

$$df_d(x, y) = \mid \{d | x \notin d \wedge y \notin d\} \mid$$

3 Problem Definition

When the corpus of a set of sets of labels is provided, and the function $rel(x, y)$ of a pair of labels to the number in the following form is also provided, we will obtain P: the set of pairs of a given size that satisfies the following condition.

$$\forall (x_2, y_2) \notin P; \forall (x_1, y_1) \in P; r(x_1, y_1) \geq r(x_2, y_2)$$

where

$$\begin{aligned} & r(x, y) \\ = \; & f(df_a(x, y), df_b(x, y), df_c(x, y), df_d(x, y)) \end{aligned}$$

The following are examples of $f(a, b, c, d)$.

- cosine function

$$\frac{a}{\sqrt{(a+b) \times (a+c)}}$$

- dice coefficient

$$\frac{2 \times a}{(a+b) + (a+c)}$$

- confidence

$$\frac{a}{a+c}$$

- pairwise mutual information

$$\frac{a}{N} \times \log \frac{a \times N}{(a+b) \times (a+c)}$$

- phi coefficient

$$\frac{a \times d - b \times c}{\sqrt{(a+b) \times (a+c) \times (b+d) \times (c+d)}}$$

- complementary similarity measure

$$\frac{a \times d - b \times c}{\sqrt{(a+c) \times (b+d))}}$$

Implementation of a program that requires $4 \times l^2$ memory space and $4 \times l^2 \times N$ computation time is easy. A program of this type could be used to calculate df_a, df_b, df_c, and df_d for all pairs of x and y, and could then provide the most highly correlated pairs. However, compuation with this method is not feasible when l is large.

For example, in order to calculate the most highly correlated words within a newspaper over several years of publication, l becomes roughly 10^5, and N becomes 10^8. The amount of computation time is then increased to 10^{18}.

4 Approach

In terms of the actual data, the number of correlated pairs is usually much smaller than the number of uncorrelated pairs. Moreover, most of the uncorrelated pairs usually satisfy the condition: $df_a(x, y) = 0$, and are not of interest. This method takes this fact into account. Moreover, it also uses the relationship between $\{N, df_a\}$ and $\{df, df_b, df_c, df_d\}$ to make the computation feasible.

5 Relationship between $\{N, df_a\}$ and $\{df, df_b, df_c, df_d\}$

Proofs of the following equations are provided below.

$$\begin{aligned} df(x) &= df_a(x, x) \\ df_b(x, y) &= df_a(x, x) - df_a(x, y) \\ df_c(x, y) &= df_a(y, y) - df_a(x, y) \\ df_d(x, y) &= N - df_a(x, x) - df_a(y, y) + df_a(x, y) \end{aligned}$$

Proof:

1. $P \wedge P$ is equivalent to P.

$$\begin{aligned} & df_a(x, x) \\ = \; & |\{d | x \in d \wedge x \in d\}| \\ = \; & |\{d | x \in d\}| \\ = \; & df(x) \end{aligned}$$

2. By definition, the sum of df_a, df_b, df_c, and df_d always represents the total number of documents.

$$\begin{aligned} & df_a(x, y) + df_b(x, y) \\ & + df_c(x, y) + df_d(x, y) \\ = \; & N \end{aligned}$$

3. Similarly, the sum of $df_a(x, y)$ and $df_b(x, y)$ is the number of documents that contain x. This equals $df(x)$.

$$\begin{aligned} & df_a(x, y) + df_b(x, y) \\ = \; & |\{d | x \in d \wedge y \in d\}| \\ & + |\{d | x \in d \wedge y \notin d\}| \\ = \; & |\{d | x \in d\}| \\ = \; & df(x) \end{aligned}$$

4. Similarly, the sum of $df_a(x, y)$ and $df_c(x, y)$ is the number of documents that contain y. This equals $df(y)$.

$$\begin{aligned} & f_a(x, y) + df_c(x, y) \\ = \; & |\{d | x \in d \wedge y \in d\}| \\ & + |\{d | x \notin d \wedge y \in d\}| \\ = \; & |\{d | y \in d\}| \\ = \; & df(y) \end{aligned}$$

5. These four equations make it possible to express df, df_b, df_c and df_d by df_a and N.

These formulas indicate that the number of required two-dimensional tables is not four, but just one. In other words, if we create a table of $df_a(x, y)$ and one variable for N, we can obtain $df(x)$, $df_b(x, y)$, $df_c(x, y)$, and $df_d(x, y)$.

6 The memory requirement for df_a

Let k be the maximum number of different words/labels in one document. The following property exists in $df_a(x, y)$.

$$\sum_x \sum_y df_a(x, y) \leq k^2 \times N$$

The left side of the formula equals the total number of all pairs of words/labels. This cannot exceed $k^2 \times N$.

This relationship indicates that if the table is stored using tuples of $(x, y, df_a(x, y))$ where $df_a(x, y) > 0$, the required memory space is $O(N)$. Tuples where $df_a(x, y) = 0$ are not necessary because we know that $df_a(x, y) = 0$ when the tuple for $(x, y, df_a(x, y))$ does not exist in memory.

This estimation is pessimistic. The actual size of the tuples will be smaller than $k^2 \times N$, since not all documents will have k different words/labels.

7 Obtaining df_a, and N

The algorithm to obtain $df_a(x, y)$, and N is straightforward. First, the corpus must be trasformed into a set of sets of words/labels. Since this is a set form, there are no duplications of the words/labels of one document. In the following program, the hashtable returns 0 for a non-existent item.

```
(01) Let DFA be empty hashtable.
(02) Let DF be empty hashtable
(03) Let N be 0

(04) For each document, assign it to D
(05) | N = N + 1
(06) | For each word in D
(07) | assign the word to X
(08) | |  For each word in D
(09) | |  assign the word to Y
(10) | |  | DFA(X, Y)=DFA(X,Y)+1
(11) | |  end of loop
(12) | end of loop
(13) end of loop
```

The computation time for this program is less than $k^2 \times N$. Since k is independent from N, the computation time is $O(N)$. Again, $k^2 \times N$ is a pessimistic estimation, since not all documents will have k different words/labels.

8 Selecting Pairs

Even though df_a, df_b, df_c, and df_c can be obtained in constant time after $O(N)$ preprocessing, there are l^2 values to consider to obtain the best N correlated pairs. Fortunately, many of the functions that are usable as indicators of correlation and, at least, all five functions, return a lower value than the known threshold if $df_a(x, y) = 0$.

The cosine measure, the dice coefficient, and pairwise mutual information have property 1 and property 2 as defined below. This implies that the value for (x, y) where $df_a(x, y) = 0$ is actually the minimum value of all (x, y). Therefore, the first part of the total ordered sequence of (x, y) is the sorted list of (x, y) where $df(x, y) > 0$. The rest is an arbitary order of pairs where $df(x, y) = 0$.

Property 1: the value is not negative.

Property 2: when $df_a(x, y) = 0$, the value is 0.

The phi coefficient and the complementary similarity measure have the following properties 1, 2 and 3. Therefore, the first part of the total ordered sequence where the value is positive, is equal to first part of the sorted list where $df(x, y) > 0$ and the value is positive. Moreover, this list contains all pairs that have a positive correlation. This list is long enough for the actual application.

Property 1: when $df_a(x, y) = 0$, the value is negative.

Property 2: when x and y are not correlated, the estimated value is 0.

Property 3: when x and y tend to appear at the same time, the estimated value is positive.

It should be recalled that the number of pairs where $df_a(x, y) > 0$ is less than $k^2 \times N$. The sorted list is obtained in $O(k^2 \times N \times \log(k^2 \times N))$ computation time, where k is the maximum number of different words/labels in one document. Since k is constant, it becomes $O(N \times \log(N))$, even if the size ofvocabulary is very large.

It is true that for the given some fixed vocabulary of size l, $k^2 \times N$ might be larger than l^2 as we increase the size of corpus. Fortunately, the actual memory consumption of this procedure also have the upper bound of $O(l^2)$, and we will not loose any memory space. When l is not fixed and l may become very large compare to N as is the case for proper nouns, $k^2 \times N$ is smaller than l^2.

9 Case study of a Newspaper Corpus

The computation time of the baseline system is $l^2 \times N$ where l is the distinct number of labels in the

N	time(sec.)	speed(sec./doc)
1000	2.4	2.4×10^{-3}
3000	7.8	2.6×10^{-3}
10000	21.1	2.1×10^{-3}
30000	60.9	2.0×10^{-3}

Table 1: The actual execution time shows a linear relationship to the size of input data.

corpus. When we analyzed labels of names of places in a newspaper over the course of one year, this corpus consisted of about 60,000 documents. The place names totalled 1902 after morphological analysis. The maximum number of names in one document was 142, and the average in one document was 4.02. In this case, the method described here, was much more efficient than the baseline system.

Table 1 shows the actual execution time of the program in the appendix, changing the length of the corpus. This program computes similarity values for all pairs of words where $df_a > 0$. It indicates that the execution time is linear.

Our observation shows that even if the corpus were extended year by year, k which is the maximum number of different words in one document is stable, even though the total number of words would increase with the ongoing addition of proper nouns and new concepts.

10 For a large corpus

Although the program in the appendix cannot be applied to a corpus larger than memory size, we can obtain a table of df_a using sequential access to file. The program in the appendix stores every pair in memory. The space requirement of $k^2 \times N$ may seem too great to hold in memory. However, sequential file can be used to obtain the df_a table, as follows. Although the computation time for df_a is $O(N \times \log(N))$ rather than $O(N)$, the total computation time remains the same because computation of $O(N \times \log(N))$ is required to select pairs in both cases.

Consider the following data. Each line corresponds to one document.

```
a b
a c
x y
x
x y z
a b c
```

When the pairs of words in each document are recorded, the following file is obtained. Note that since $df_a(x, y) = df_a(y, x)$, it is not necessary to record pairs where $x > y$. This reduces the memory requirement.

```
a       a
a       b
b       b
a       a
a       c
c       c
x       x
x       y
y       y
x       x
x       x
x       y
x       z
y       y
y       z
z       z
a       a
a       b
a       c
b       b
b       c
c       c
```

Using the merge sort algorithm which can sort a large file using sequential access only, the file can be sorted in $O(N \times \log(N))$ computation time. After sorting in alphabetical order, same pairs come together. Then, the pairs can be counted with sequential access, thereby providing the df_a table. An example of this table fllows:

```
a       a       3
a       b       2
a       c       2
b       b       2
b       c       1
c       c       2
x       x       3
x       y       2
x       z       1
y       y       2
y       z       1
z       z       1
```

It should be noted that the df table can be obtained easily by extracting lines in which letter of the first column and that of the second column are the same, since $df(x) = df_a(x, x)$. The df table can usually be stored in memory since it is a one dimensional array. After storing df in memory, similarity can be computed line by line. The following example uses the

phi coefficient. The first column is the coefficient, followed by df_a, df_b, df_c, df_d, x and y. Since the phi coefficient is reflective, the (x, y) value where $x > y$ is not required. When the function is not symmetric, (x, y) and (df_b, df_c) can be exchanged at the same time.

```
0.544705  3  0  0  3  a  a
0.384900  2  1  0  3  a  b
0.384900  2  1  0  3  a  c
0.624695  2  0  0  4  b  b
0.156174  1  1  1  3  b  c
0.624695  2  0  0  4  c  c
0.544705  3  0  0  3  x  x
0.384900  2  1  0  3  x  y
0.242536  1  2  0  3  x  z
0.624695  2  0  0  4  y  y
0.392232  1  1  0  4  y  z
0.674200  1  0  0  5  z  z
```

The ordered list can be obtained by sorting this table with the first column. This example shows that pairs where $df_a(x, y) = 0$, such as $(\mathtt{a}, \mathtt{x})$ or $(\mathtt{a}, \mathtt{y})$, do not add any overhead to either memory or computation time.

11 Comparison with Apriori

There is a well known algorithm for forming a list of related items termed Apriori(Arrawal and Srikant, 1995). Apriori lists all relationship using confidence, where $df_a(x, y)$ is larger than a specified value. Using Apriori, the df_a threshold can be specified in order to reduce computation, whereas with the proposed method, there is no way to adjust this threshold. This implies that Apriori may be faster than our algorithm in terms of confidence. However, since Apriori uses the property of confidence to reduce computation, it cannot be used for other functions, unlike the proposed method which can employ many standard functions, at least the five measures used here including confidence.

12 Correlation of All Substrings

When computing correlations of all substrings in a corpus, l can be as large as $N \times (N-1)/2$. Since the memory space requirement and computation time does not depend on l, this method can be used to generate a list of the most hightly correlated substrings of any length. In fact, in some cases, k may be too large to compute.

The Yamamoto-Church method(Yamamoto and Church, 2001) allows for the creation of a $df(x)$ table using $O(N)$ memory space and $O(N \times \log(N))$ computation time, where x represents all substrings in a given corpus. Yamamoto's method shows that although there may be $N \times (N-1)/2$ kinds of substrings in a corpus, there is $2 \times N$ occurence patterns (or sets of substrings which have same occurence pattern) at most. The computational cost is greatly reduced if we deal with each pattern instead of each substring. Although the order of computional complexity does not depend on l, k differs whether the pattern is used or not. We have also developed a system using the pattern which actually reduces the cost of computation. Although the number of k is still problematic even using the Yamamoto-Church method, and although the computation cost is much larger than using words, the program runs much faster than the simple method.

13 Conclusion

This paper describes a method for selecting correlated pairs in $O(N)$ memory space and $O(N \times \log(N))$ computation time, where N is the number of documents in a corpus, provided that there is an upper boundary in the number of different words/labels in one document/record. We have observed that a corpus usually has this kind of upper boundary, and have shown that we can uses a sequential file for most of our memory requirements. This method is useful not only for confidence but also for other functions whose values are decided by df_a, df_b, df_c, df_d. Examples of these functions are mutual information, the dice coefficient, the confidence measure, the phi coefficient and the complimentary similarity measure.

References

K. W. Church and P. Hanks 1990 *Word association norms, mutual information and lexicography* Computational Linguistics, 16(1):22–29

R. Agrawal and R. Srikant 1995 *Mining of association rules between sets of items in large databases.* In Proceedings of the ACM SIGMOD Conference on Management of Data:94–105

N. Hagita and M. Sawaki: 1995 *Robust recognition of degraded machine-printedcharacters using complimentary similarity measure and error-correction learning* Proceedings of the SPIE - The International Society for Optical Engineering 2442:234–244

U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth *The KDD Process for Extracting Useful Knowledge from Volumes of Data* Communications of the ACM, 39(11):27–34,

Christopher D. Manning and Hinrich Schuetze, 1999 *Chapter 8.5, Semantic Similarity* Foundations of statistical natural language processing:294–303, The MIT Press

Christopher D. Manning and Hinrich Schuetze, 1999 *Chapter 5.3.3, Pearson's chi-square test* Foundations of statistical natural language processing:169–172, The MIT Press

Mikio Yamamoto and Kenneth W. Church 2001 *Using Suffix Arrrays to Compute Term Frequency and Document Frequency for All Substring in a Corpus* Computational Linguistics, 27(1):1–30, The MIT Press.

Appendix

Sample of DATA

```
1992.01.01.00000043 Takarazuka Tokyo
1992.01.01.00000046 Okinawa Yatushiro
1992.01.01.00000048 Hiroshima Kurihara Onomichi Yokohama
1992.01.01.00000049 Tokyo
1992.01.01.00000050 Ichihara Tokyo
1992.01.01.00000051 Aizuwakamatu Fukushima Tokyo
1992.01.01.00000056 Matumoto Sahara Utsunomiya
1992.01.01.00000065 Tokyo
1992.01.01.00000066 Aomori Shimokita
```

Sample Program(csm.awk)

```
 1  # Definition of similarity -
 2  #   Complimentary Similarity Measure.
 3  function f(a, b, c, d) {
 4      return (a * d - b * c) / sqrt((a + c) * (b + d));
 5  }
 6  # For each line, count up both df(x) and dfa(x, y).
 7  { for(i=2; i<= NF; i++) {
 8      df[$i]++;
 9      for(j=i;j<= NF; j++) {
10          dfa[$i, $j]++;
11      }
12    }
13  }
14  # For all (x, y) where dfa(x, y)<>0, get the value.
15  END{
16      for(k in dfa) {
17          split(k, x, SUBSEP);
18          if(x[1] != x[2]) {
19              a = dfa[k];
20              b = df[x[1]] - a;
21              c = df[x[2]] - a;
22              d = NR - a - b - c;
23              r = f(a, b, c, d);
24              printf("%10.6f\t%s\t%s\n",
25                     r, x[1], x[2]);
26              r = f(a, c, b, d);
27              printf("%10.6f\t%s\t%s\n",
28                     r, x[2], x[1]);
29          }
30      }
31  }
```

Usage:

```
$ awk -f csm.awk < mai.txt | sort -nr | head  -20
```