インターフェイスの街角– 類似ファイル検索

増井 俊之

内容にもとづく検索

前号で紹介した〝近傍検索システム〟では、ファイルの置き場所（ディレクトリ）や作成時間をもとに、その〝近く〟にあるファイルを検索する方法を使いました。一方、ファイルの内容の近さにもとづいて検索する方法もきわめて重要です。

検索文字列や既存の文書に内容が似ている文書の検索法については、長年にわたってさまざまな研究がおこなわれてきました。計算機を利用する場合、文書の内容を理解したうえでの比較は難しいので、同じような単語を含むファイルは内容も近いだろうという考えにもとづいて比較するのが一般的です。

同じような単語を含む文書を探す方法といっても、以下のようにいろいろなレベルがあります。

- 検索文字列と共通の単語を含む文書を検索
  Google や Namazu は、このような検索手法を用いています。
- 重要な共通単語を含む文書を検索
  なんらかの方法で単語に重みづけができれば、〝重要な単語〟を共通に含む文書を優先して検索できます。
- 同じ意味の単語を含む文書を検索
  〝インターフェイス〟と〝インタフェース〟〝インターフェース〟のように、1 つの単語に複数の表記がある場合もあれば、〝計算機〟と〝コンピュータ〟のように、意味が同じでも表記がまったく異なることもあります。あるいは、〝時間〟と〝時刻〟のように、似てはいても意味がすこし違うものもあります。このような表記の揺れや単語の意味まで考慮しが検索が可能であれば、さらに効果的でしょう。

共通単語を含む文書を検索する手法は、全文検索システムや Web 上のサーチエンジンなどでひろく使われていますが、検索文字列を上手に選択する必要があります。Google などを利用する場合、一般的によく使われる単語を指定すると意図どおりの結果が得られないので、検索したい文書だけに含まれるであろうキーワードを工夫して指定しなければなりません。

たとえば、以下のような内容の 4 つのファイルの類似度を計算してみましょう。

text1：文書検索関連の記事です
text2：文書検索技術に関する文書
text3：文字入力関連の記事です
text4：政治経済関連の記事です

text1 と text2 に共通なのは〝文書〟と〝検索〟の 2 つの単語だけですが、text1 と text3 では〝関連〟〝の〟〝記事〟〝です〟の 4 つの単語が共通しています。すべての単語が同じように重要であるという前提で計算すると、text1 は text2 よりも text3 や text4 に近いことになってしまいます。この例のような場合は、〝記事〟〝資料〟〝関する〟などの一般的な単語より、〝検索〟や〝入力〟のような単語を重視したいところです。

意味の近い単語を同等に扱ったり、文章の内容も考慮して類似度を判断するのが最善の方法だと思いますが、それには高度な自然言語処理やシソーラスが必要です。したがって、普通のシステムでは、重要な単語が共通に使われているかを基準としてファイルの類似度を計算するのが実際的でしょう。

図 1　文書中の単語の出現頻度表

	文書	検索	関連	の	記事	です	技術	に	関する	文字	入力	政治	経済
文書検索関連の記事です	1	1	1	1	1	1	0	0	0	0	0	0	0
文書検索技術に関する文書	2	1	0	0	0	0	1	1	1	0	0	0	0
文字入力関連の記事です	0	0	1	1	1	1	0	0	0	1	1	0	0
政治経済関連の記事です	0	0	1	1	1	1	0	0	0	0	0	1	1

単語の重要度の計算

ある単語が重要かどうかは、文章の内容から判断しなければ分かりません。しかし、これを計算機で自動的に計算するのはひどく難しいので、とりあえず簡便に計算するために、

●1 つの文書中に何回も出現する単語を重要単語とみなす

という方法が考えられます。この記事を例にとると、〝検索〟や〝類似〟といった単語が何回も出てくるので、これらを重要単語とみなすわけです。

一方、この記事には〝使用〟や〝重要〟などの一般的な単語も頻出しますが、これらは文章の内容と深い関係があるとはいえないので、重要単語から除外しなければなりません。それには、

●いろいろな文書に出現する単語は重要単語とみなさない

といった工夫が必要になるでしょう。

これらの指標に従い、単語がファイルに出現する頻度を計算するのは比較的簡単です。

前者の場合は、文書中の単語の出現頻度でほぼ計算できます。後者の場合も、すべての文書のうち、その単語が含まれる文書の割合を調べれば計算できます。

文書中の単語の出現頻度を〝tf (Term Frequency)〟と呼びます。また、全文書数 N のうち、該当する単語を含む文書が n 個あるとき、その比の対数 $(\log(N/n))$ を〝idf (Inverse Document Frequency)〟と呼びます。tf と idf の積は〝tf・idf 値〟といい、これをもとに単語の重要度を計算する tf・idf 法がひろく使われています[1]。

idf は、ある単語が多くの文書に出現する一般的なキーワードの場合には小さくなり、特定の文書に限って出現する場合は大きくなります。

ここでは idf 値の計算に logを使っていますが、これとは異なる基準も使えます。

一連の文書に含まれるあらゆる単語について、各文書にいくつ含まれているかを示す図 1 のような表を用意すれば、さきほどの tf や idf などの値を簡単に計算できます。tf や idf の計算式自体は比較的単純ですが、大量の文書を対象とする場合は巨大な表を扱うことになります。したがって、効率的に計算するには、かなりうまく実装する必要があります。

国立情報学研究所の高野明彦氏、日立製作所の西岡真吾氏らは、tf や idf などを計算するための大きな表を操作したり、その情報を用いて文書間の類似度を計算するためのライブラリ「GETA」を公開しています。これを利用すれば、類似度を計算するシステムを比較的簡単に構築できます。

汎用連想検索エンジン GETA

汎用連想計算エンジン GETA (Generic Engine for Transposable Association)[1]は、情報処理振興事業協会 (IPA)[2]が実施した「独創的情報技術育成事業」[3]の一環として、日立製作所などのグループにより開発されたシステムです。GETA のソースコードははすべて公開されており、IPA の使用許諾条件に従って使用することができます。

GETA は、**WAM** (Word-Article Matrix)[4]というデータ構造を扱う WAM アクセス・ライブラリ (libwam)、WAM を用いて tf・idf のような連想計算をおこなう Association Engine (libae)、文書のクラスタリングをサポートするクラスタリング・ライブラリ (libcs)、その他のユーティリティ・ライブラリおよびコマンド群から構成されています。

GETA を利用して類似文書やキーワードを検索するための手順は、おおよそ次のようになります。

1. 事前処理 (検索前の計算)
   - 検索対象の全文書の形態素解析をおこなって単語に分割

1 http://geta.ex.nii.ac.jp/
2 http://www.ipa.go.jp/
3 http://www.ipa.go.jp/STC/dokusou.html
4 図 1 のような、文書中の単語の出現頻度を示す疎行列のうち、大規模なものを表現するためのデータ構造です。

図 2　頻度ファイル (freqfile)

```
@./text1
1 です
1 の
1 関連
1 記事
1 検索
1 文書
@./text2
1 に関する
1 技術
1 検索
2 文書
@./text3
  ……
```

- 各単語の各文書における出現頻度を計算して WAM に格納

2. 検索処理 (検索実行時の処理)
   - 検索文字列の形態素解析をおこなって単語に分割
   - tf・idf のような重みづけをおこないつつ、検索対象の全文書と検索文字列との単語のマッチングを実行し、スコアの高いものから順に表示する

全文書の形態素解析をおこなって WAM を生成する処理にはある程度の時間がかかりますが、GETA を使うと WAM データに対する計算が高速におこなえるので、ハードディスク上の文書の検索程度なら処理は一瞬で終了します。

事前処理

事前処理では、文書に含まれる単語の出現頻度を表現する WAM を作成します。文書名と単語の出現頻度を記述したテキストファイルをもとに、バイナリファイルで表現される WAM を生成します。

さきほどの例であれば、文書ごとの単語の出現頻度を表現する図 2 のようなテキストファイルを用意します。

このテキストデータに対して GETA パッケージに含まれる mkw コマンドを実行すると、以下のようなメッセージが表示され、WAM を表現するバイナリファイルが生成されます。

```
% mkw testdata freqfile
ocf: info: asc:[freqfile]
mkw: counting number of keys in freqfiles:
 cnrr: info: f=0x0, *s=0, p=0x0
*8192/6 lt-level: 0
ht-level: 0, usage=0.000000
rrvq: info: turn over (pos=180)

mkw: time=0.006
mkw: rnum=4
mkw: total_elem_num=22
  ……
mkw: time=0.022785, ht_usage=0.001465, num
_lword=0, rehash=1 nhts=8192
%
```

mkw の引数の ˋtestdata´は WAM を識別するハンドル名で、ci.conf という定義ファイルで指定します。ci.conf では、検索されるデータ集合ごとに各種の属性を定義します。mkw で生成した WAM のバイナリファイルは、dataroot として定義したディレクトリに格納されます。

```
% cat /usr/local/geta/etc/ci.conf
handle: unix.magazine.testdata
short-name: testdata
dataroot: /usr/local/geta/data/testdata/
jma:p: japanese.sh
% ls /usr/local/geta/data/testdata/
cw.c  cw.r  xr.c  xr.r
%
```

WAM が生成されたかどうかは、dumpwam コマンドで確認できます。

```
% dumpwam testdata cw_col
nrsyms = 12
です
の
関連
記事
文書
検索
に関する
技術
経済
政治
入力
文字
%
```

図 2 のような頻度ファイルは、形態素解析プログラムを用いて作成します。

この種のプログラムとしては、奈良先端科学技術大学院大学の松本研究室で開発された「茶筌」[5]が有名ですが、同研究室の工藤 拓さんがアルゴリズムを改良し、さらに高速化した「MeCab」[6]を公開しています。

5 http://chasen.aist-nara.ac.jp/
6 http://cl.aist-nara.ac.jp/~taku-ku/software/mecab/

図 3　日本語テキストを単語単位に分割

```
% cat text1
文書検索関連の記事です
% mecab text1
文書    名詞,一般,*,*,*,*,文書,ブンショ,ブンショ
検索    名詞,サ変接続,*,*,*,*,検索,ケンサク,ケンサク
関連    名詞,サ変接続,*,*,*,*,関連,カンレン,カンレン
の      助詞,連体化,*,*,*,*,の,ノ,ノ
記事    名詞,一般,*,*,*,*,記事,キジ,キジ
です    助動詞,*,*,*,特殊・デス,基本形,です,デス,デス
EOS
%
```

図 4　頻度ファイル生成スクリプト（makefreqfile）

```
#!/bin/sh
while read f
do
    echo "@$f"
    cat $f |
    mecab |
    grep -v '^EOS$' |
    perl -pe "s/\\s.*$//" |
    sort |
    uniq -c |
    perl -pe "s/\\t/ /" |
    perl -pe "s/^[ \\t]*//"
done
```

MeCab を使うと、日本語テキストを図 3 のような単語単位に簡単に分割できます。図 4 のようなスクリプトでこれを処理すると、頻度ファイルが生成されます。

```
% find . -print | grep "text.$$" | \
./makefreqfile > freqfile
%
```

検索処理

作成した WAM を使い、tf・idf などによる検索処理ができます。

GETA では連想検索ライブラリ libae を利用することにより、tf・idf などの指標を使って文書の類似度を計算させることができます。

検索文字列を指定し、それに近い文書をリストアップしたいときは、以下のような処理をおこないます。

1. 指定された検索文字列を形態素解析
2. tf・idf などの指標にもとづいて各文書との距離を計算
3. 結果をスコア順に表示

指標として、tf・idf ではなく tf のみを使って検索するプログラムの例を図 5 に示します。

形態素解析のためのプログラムは、ci.conf で指定しておきます。さきほど例に挙げた ci.conf では、形態素解析に下記の japanese.sh というスクリプトを使うと定義してあります。

```
% cat /usr/local/geta/data/testdata/japanese.sh
#!/bin/sh
/usr/local/bin/mecab | perl -pe "s/\\s.*$//"
%
```

ハンドル名を指定して WAM をオープンしたあと、wstem() 関数で検索文字列を形態素解析して単語に分割し、wsh() 関数で検索対象の文書と比較します。ここでは第 5 引数に WT_TF を指定しているので、文書と検索文に共通に出現する単語の頻度 (tf) だけを考慮した検索結果が得られます。

text1 を検索文字列として search_tf を実行すると、以下のような結果になります。

```
% cat text1 | ./search_tf testdata 3
1 1.000000 ./text1
2 0.666667 ./text2
3 0.666667 ./text3
%
```

text1 と text3、text1 と text4 は 6 個の単語のうち 4 個の共通単語（関連、の、記事、です）を含んでいるため、類似度が 0.666 になっています。text2 は、実際には text1 と内容が近いにもかかわらず、ランク外になってしまっています。

図 5 の WT_TF を WT_TFIDF に変更したプログラム search_tfidf では、tf・idf を基準としてファイルの類似度が計算されます。実行結果は、以下のようになります。

```
% cat text1 | ./search_tfidf testdata 3
1 0.422837 ./text1
4 0.415888 ./text4
2 0.191788 ./text2
%
```

〝関連〟や〝の〟などの単語は多くの文書に含まれるので idf 値が小さくなりますが、〝検索〟は text1 と text2 のみに含まれ、tf 値も idf 値も大きいため、text1 と text2 の類似度が大きくなっています。

WT_TF や WT_TFIDF は GETA で定義されている計算指標ですが、GETA では類似度を計算するための指標をユーザーが自由に再定義できます。

図 5　tf のみによる検索

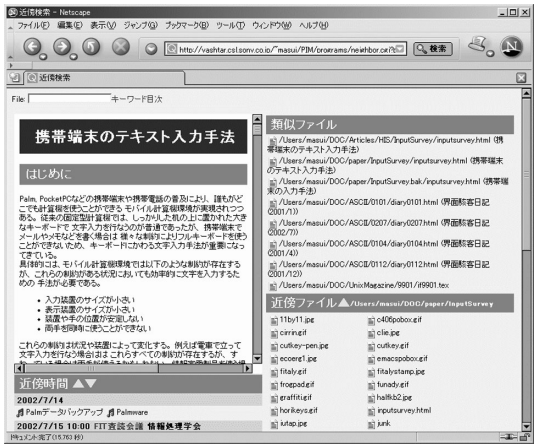
```
/*  search_tf.c  */
#include <stdio.h>
#include <sys/types.h>

#include <geta/srvmu.h>
#include <geta/wam.h>
#include <geta/ae.h>

main(int argc, char **argv)
{
    char *jma, *handle, *query;
    WAM *wam;
    syminfo *q, *r;
    int i, qlen, rlen;

    if(argc!=3) exit(1);
    handle = argv[1];
    rlen = atoi(argv[2]);

    wam_init(NULL);
    wam = wam_open(handle, 0);
    jma = ci_value(handle, "jma");
    if(!jma) exit(1);

    query = readin(stdin);

    q = wstem(query, wam, WAM_COL, jma, &qlen);

    r = wsh(q, qlen, wam, WAM_COL, WT_TF, &rlen,
        NULL, NULL, NULL);

    sort_w_syminfo_list(r, rlen);

    for (i=0; i<rlen; i++) {
        printf("%d\t%lf\t%s\n", r[i].id, r[i].weight,
                wam_id2name(wam, WAM_ROW, r[i].id));
    }

    wam_close(wam);
    return(0);
}
```

近傍検索システムへの組込み

前回紹介した近傍検索システムに、GETA を用いた類似文書検索機構を組み込めば、より多様な検索が可能になります。

図 6 は、類似文書検索機能を追加した近傍検索システムの表示例です。左上のウィンドウでは入力手法に関する 7 月中旬作成の文書を閲覧していますが、そのころに作成したファイルや予定が左下に表示され、その文書ファイルを含むディレクトリの内容が右下に表示されています。また、GETA を用いてこの文書に内容の近い文書を検索し、右上に表示しています。入力手法については何度かサーベイや紹介記事を書いているので、それらのファイルが右上に一覧表示されていることが分かります。

図 6　類似ファイル検索を追加した近傍検索システム



おわりに

tf・idf などを利用した類似ファイル検索手法は、情報検索の研究としてはポピュラーですが、効率のよい実装には手間がかかるため、実用性という面ではいま一歩でした。しかし、GETA や MeCab で行列計算や形態素解析が手軽におこなえるようになり、今回紹介したような検索システムを誰でも簡単に実験できるようになりました。

内容が雑多で巨大なデータベースを検索する場合は、表記の揺れを吸収したり、単語や文書の意味まで考慮した高度な検索が必要とされるかもしれません。しかし、個人が蓄積したデータベースを検索するような場合は、使われる単語も比較的限られていることが多いので、tf・idf のような手法も十分に役立つはずです。また、文書の置き場所や作成時刻も考慮した近傍検索と組み合わせれば、より柔軟で強力な検索が可能になるでしょう。

（ますい・としゆき　ソニー CSL）

[参考文献]

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto, *Modern Information Retrieval*, Addison-Wesley, May 1999