インターフェイスの街角 (6)
POBox の予測手法と辞書の作成

増井俊之

先月号では、予測型テキスト入力システム POBox（図1）の概要を紹介しました。今回は、POBox の単語予測手法、辞書の作成方法などについて解説します。

POBox の予測方式

POBox は、単語辞書と例文辞書の 2 つの辞書を使用して次に入力する単語の予測をおこないます。

単語辞書

単語辞書は単語を使用頻度順に並べたもので、図 2 のような読みと単語を並べた単純なテキストファイルです。ユーザーが次に入力する単語の予測に現在のコンテクストを利用できない場合は、この単語辞書が使用されます。

ユーザーがまだ何も入力していない段階では、「する」「システム」のような、出現頻度の高い単語が上から順に候補として表示されます。ユーザーがソフトキーボードで読みを指定すると、その読みに合った単語のみが表示されます。たとえば、〝し〟を入力すると、〝システム〟〝して〟〝した〟のように〝し〟で始まる単語だけが候補として表示され、さらに〝て〟と入力すると〝して〟〝指定〟〝しても〟などの単語が表示されます。

例文辞書

単語を並べて文書を作成している途中や、単語を文書中に挿入する場合には、先行する文字列から次の入力単語をある程度予測できます。このために、図 3 のような例文辞書を使用します。たとえば、〝…… する〟という文字列の次には〝こと〟という単語がつながる可能性がもっとも高いことを示しています。

図 1　POBox の使用例

図 2　単語辞書

読み	単語
する	する
しすてむ	システム
ますい	増井
して	して
した	した
ばあい	場合
です	です
おこな	行な
…	…

〝…… という〟という文字列がすでに入力されており、その後に続く単語を予測するような場合は、例文辞書を上から順番に調べ、先行文字列とユーザーの入力した読みがマッチする単語を候補として提示します。ユーザーが読みを入力していない状態では、〝と/いう〟〝と/して〟〝と/思い〟〝すると/いう〟などのエントリの先行文字列が〝すると〟にマッチするので、〝いう〟〝して〟〝思い〟などを候補単語とします。ここでユーザーが〝し〟を入力すると、〝と/いう〟などは読みがマッチしないため、〝して〟のみが表示されます。

先行文字列がある場合は、POBox はまず例文辞書を使用して次単語を予測しますが、十分な候補単語が得られないときは単語辞書も使います。

図 3　例文辞書

先行文字列	読み	単語
と	いう	いう
こと	か	が
で	は	は
する	こと	こと
…	…	…
ユーザ	が	が
こと	は	は
思い	ます	ます
…	…	…
と	おもい	思い
ため	の	の
の	か	が
使用	する	する
増井	としゆき	俊之
こと	を	を
場合	は	は
…	…	…
すると	いう	いう
…	…	…
よろしく	おねがい	お願い
…	…	…

単語/例文の学習

ユーザーが候補から単語を選択すると、その単語のエントリは単語辞書の先頭に移動します。また、連続して選択された単語列は新たな例文として例文辞書の先頭に追加されます。同様な例文がすでに登録されているときは、そのエントリが先頭に移動します。このような辞書の学習機能により、以前に選択した単語をもう 1 度入力しようとするとその単語が候補リストの先頭近くに表示されるため、選択しやすくなります。

単語選択を繰り返していくことにより、単語辞書や例文辞書は自動的に現在のコンテクストを反映するようになっていくと考えられます。このように、ユーザーが計算機を使っているうちに計算機がそのユーザーに徐々に適応し、使いやすくなっていくようなシステムを**適応型インターフェイス** (adaptive interface) システムと呼びます。適応型インターフェイスに関しては古くから多くの研究がおこなわれていますが、実用に耐える手法はさほどありません。かな漢字変換辞書や POBox の辞書学習機能は、手法が簡単であるにもかかわらず有効な適応インターフェイスの例といえます。

単語辞書/例文辞書の作成

POBox の有効性は、単語辞書/例文辞書の品質に大きく左右されます。質の高い単語/例文辞書は手作業でも作れますが、自動的に作成できればそれに越したことはありません。さいわい、最近はインターネットなどから大量の文章データを例文として辞書作成に活用できるので、比較的簡単に良質な単語/例文辞書が作れます。これらのデータを用いて、単語の出現頻度や、先行する文字列に対してどのような単語が続きやすいかという頻度を計算すればよいわけです。

通常、日本語の文章では単語間に区切りがありません。したがって、単語の出現頻度を計算するには意味のある部分で文章を適当に分割する必要があります。たとえば〝単語間に区切りがない〟という文字列は、〝単語/間/に/区切り/が/ない〟と分割すべきであり、〝単語間/に/区切/り/が/な/い〟のように単純に文字種で区切ってはいけません。前者のように文章を文法的に正しく分割するためには、辞書や文法知識が不可欠です。こういった分割の処理を**形態素解析**といいます。

形態素解析は、自動翻訳や情報検索など、各種の自然言語処理に必要とされる重要な技術です。たとえば形態素解析の不具合により、情報検索対象となる文書中の〝大地震〟という単語を〝大地/震〟などと分割して索引づけしてしまった場合は、〝地震〟というキーワードではその文書を検索できなくなる可能性があります[1]。

〝大地震です〟を正しく〝大/地震/です〟と分割するためには、

- 〝地震〟という名詞がある
- 〝大〟という接頭語が名詞に接続可能
- 名詞の後ろに〝です〟が接続可能

といった文法知識が使われます。一方、

- 〝大地〟という名詞がある
- 〝震〟という固有名詞がある
- 名詞には名詞が接続可能
- 名詞の後ろに〝です〟が接続可能

1 単語への分割をおこなわず、あらゆる文字の並びに索引をつければこのような誤りを防ぐことができますが、検索効率が悪くなってしまうので、これらの折衷としてさまざまな手法が考案されています [3]。

図 4　茶筌による ｀かもしれません´の解析結果

```
か      か      か      終助詞
も      も      も      副助詞
しれ    しれ    しれる  動詞            母音動詞        基本連用形
ませ    ませ    ます    動詞性接尾辞    動詞性接尾辞ます型  未然形
ん      ん      ぬ      助動詞          助動詞ぬ型      基本形
```

という解釈も可能です。けっきょく、どの解釈がもっとも妥当かを計算して判定する作業が必要になります。

このような計算をおこなうには、良質な辞書、文法知識、アルゴリズムが必要ですが、「茶筌」という形態素解析プログラムを利用することができます。

茶筌

茶筌は、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形態素解析プログラムで、1997 年より UNIX 版と Windows 版がフリーで配布されています[2]。奈良先端科学技術大学院大学は、大阪府に隣接した奈良県生駒市の山中の高山町にあります。高山町は日本有数の茶筌の生産地でもあり、それにあやかってこのように命名されたそうです。

茶筌の実行例

｀大地震です´という文字列を茶筌で形態素解析すると、以下のような結果が得られます。

```
% echo '大地震です' | chasen
大      だい    大      名詞接頭辞
地震    じしん  地震    普通名詞
です    です    だ      判定詞      判定詞      デス列基本形
EOS
%
```

一方、｀大地震える´の場合は以下のように解析されます。

```
% echo '大地震える' | chasen
大地    だいち  大地    普通名詞
震える  ふるえる  震える  動詞      母音動詞    基本形
EOS
%
```

2 http://cactus.aist-nara.ac.jp/lab/nlt/chasen.html。松本研究室では、以前から「JUMAN」と呼ばれる形態素解析プログラムを開発/配布していましたが、茶筌はその後継プログラムです。辞書にパトリシア木（トライ構造の一種）を使うなど、各種の改良が施されています。JUMAN では、辞書として ndbm を使っていたため、システムによっては巨大なディスク領域が必要でしたが、茶筌ではそのような問題がなく処理速度も向上しています。

このように、茶筌では｀地震´の後ろに｀える´は接続されにくいといった文法知識を用いて、文章を意味のある要素に分割することができます。辞書および文法知識は独立した定義ファイルとなっており、ユーザーは簡単に新しい固有名詞を追加したり、文法知識を修正したりすることができます。

茶筌からの POBox 辞書生成

大量の例文に対して茶筌で形態素解析をおこない、単語の出現頻度や接続頻度を計算して POBox 用の単語辞書/例文辞書を作ることができます。

茶筌は文法知識にもとづいて日本語の文章を分割しますが、POBox を用いた文書の作成において、その分割がかならずしも適切とはかぎりません。たとえば、｀かもしれません´という文字列を茶筌で形態素解析すると、図 4 のようにばらばらになってしまいますが、POBox での使用を考えると、｀かも/しれ/ません´または｀かも/しれません´程度に分割するほうが好ましいでしょう。

POBox 用の文法知識を作成して茶筌で使うことも可能ですが、かなりの文法変更が必要になってしまいます。そこで、POBox で扱いやすいように茶筌の出力を適当にグループ化して辞書を作ることにします。リスト 1 のプログラム **cscount** を利用すれば、以下のようにして POBox の単語辞書が作れます。入力として今回の記事の原稿を使うと、｀辞書´や｀単語´などの単語が上位にきます。例文辞書も、同様の手法で作れます。

```
% nkf -e unimaga.tex | chasen | cscount | \
sort | uniq -c | sort -r -n
 220 の/の
 149 が/が
 138 を/を
 134 に/に
 108 と/と
  95 は/は
  86 ます/ます
  82 たんご/単語
  77 する/する
  67 じしょ/辞書
```

リスト 1　茶筌の出力から POBox 単語辞書を作る cscount

```
#!/usr/local/bin/perl
while(<>){
    chop;
    ($w,$y) = split;
    next if ! $w || ! $y;
    if($w =~ /^[\x00-\x7f]*$/ ||
       ord($y)<=0xa3 || ord($y)>=0xa5 ||
       ord($w)<=0xa3){
       $cont = 0; next;
    }
    print "$y/$w\n";
    print "$y1$y/$w1$w\n" if $cont;
    $y1 = $y; $w1 = $w;
    $cont++;
}
```

図 5　地名辞書

先行文字列	読み	単語
東京都	しながわく	品川区
東京都	しんじゅく	新宿区
東京都	めぐろく	目黒区
…	…	…

図 6　時間辞書

先行文字列	読み	単語
朝	はちじ	8 時
朝	くじ	9 時
…	…	…
夜	じゅうじ	10 時
夜	じゅういちじ	11 時
分	から	から
分	まで	まで
時	から	から
時	まで	まで
月	まで	まで
今度	の	の
来週	の	の
…	…	…

```
   …
%
```

特殊な例文辞書

自然言語解析を使わなくても、特殊なコンテクストでは有用な例文辞書が作れる場合もあります。たとえば、都道府県名を入力した場合は、後ろに続けて市区町村名が入力される確率がきわめて高いので、図 5 のような地名辞書を例文辞書として利用できるでしょう。

また、図 6 のような〝時間辞書〟を作成すれば、「来週の月曜朝 9 時 30 分から 11 時まで……」といった文を簡単に入力できますし、図 7 のような〝会議辞書〟を使えば、「大会議室で大統領と打合せ」といった文が簡単に入力できます。

図 7　会議辞書

先行文字列	読み	単語
増井氏	と	と
大統領	と	と
と	うちあわせ	打合せ
と	かいぎ	会議
小会議室	で	で
大会議室	で	で
…	…	…

図 8　Newton 版 POBox “TFM”

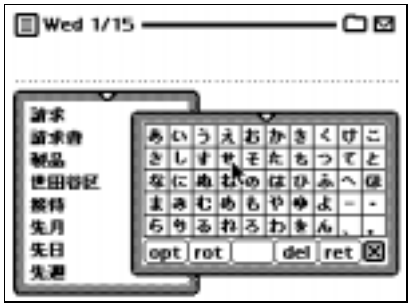


図 9　Pilot 版 POBox

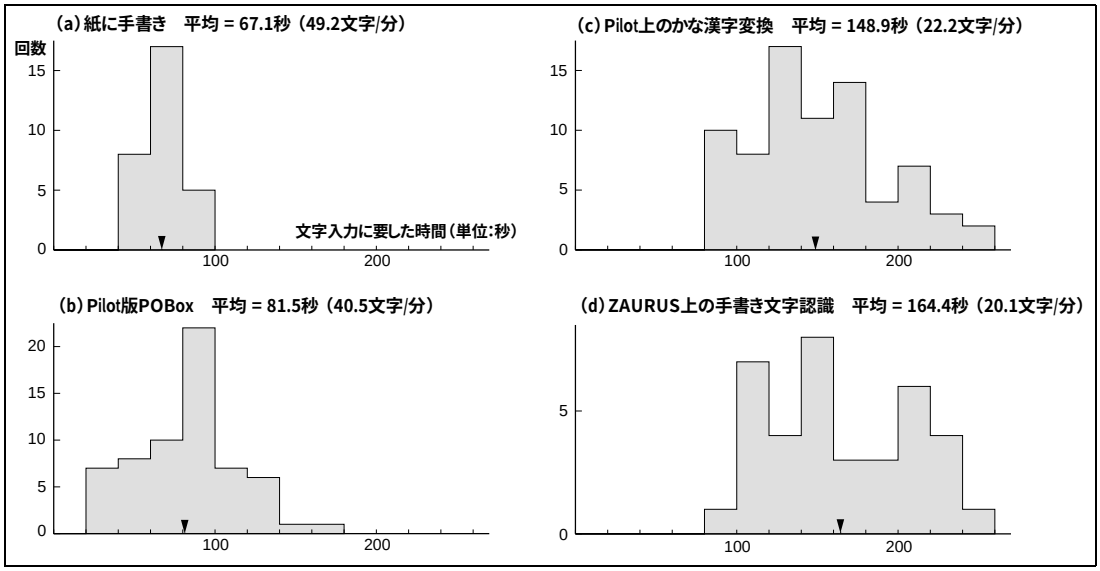
POBox に関するその他の話題

POBox は UNIX 版のほか、3Com の Pilot、Windows、Java[3]、Newton[4]など各種のプラットホームに移植されています。とくに、Pilot 版（図 9）は数多くのユー

3 日本サン・マイクロシステムズの戸松さんが移植したもの。
http://www.n-sun.com/~tomatsu/java/pobox/readme.html

4 Apple の広瀬さんによる〝TFM〟。
http://www.t3.rim.or.jp/~hirose1/tfm/tfm.html

図 10　入力方式による所要時間の分布



ザーに使われています。ユーザーを対象としたアンケートや解析の結果、興味深い性質が明らかになったので、ここで簡単に紹介しておきます。

テキスト入力速度の比較

約 30 名の Pilot 版 POBox のユーザーに依頼して、文字入力方式ごとの入力速度を比較してみました。この実験では、55 文字からなる例文を 4 種類の文字入力方式を用いて入力し、それぞれの所要時間の分布を比較しています。結果を図 10 に示します [2]。

人により各方式を実験した回数が異なっているため、実験回数が方式ごとに異なっていますが、すべて集計すると、普通の紙に文字を書くのに 40〜60 秒かかった場合が 8 回、60〜80 秒が 17 回、80〜100 秒が 5 回あったことを示しています。日本語入力において、手書き文字認識では 1 分間に平均 20 文字程度しか入力できませんが、POBox の場合は平均 40 文字程度となり、慣れればさらに高速な入力が可能と思われます。

単語の頻度分布と予測のヒット率

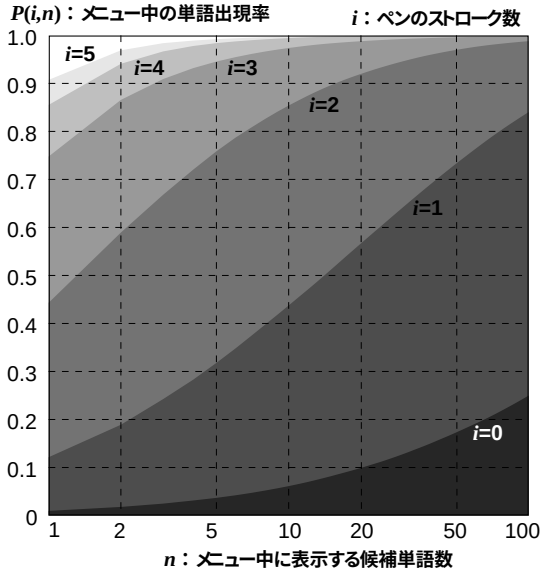
自然言語の単語の出現頻度 fは一様ではなく、単語の出現ランク r（何番目に多く出現するか）にほぼ反比例する $(f(r) \propto 1/r)$ ことが Zipf の法則として経験的に知られています[5]。このような性質は、POBox の候補単語リストにおいてもおよそ成立すると思われます。n 番目の候補がユーザーの求める単語である確率が $1/n$ に比例するとみなすと、多くの場合において、必要な単語は最初の数個の候補のなかに現れるか、候補中にまったく現れないかのどちらかになると考えられます。

ユーザーが読みを何文字か入力したとき、求める単語が候補のなかに出現する確率を実際の文章データを用いて計算してグラフ化すると図 11 のようになります。たとえば i=2 の線は、読みを 2 文字入力した時点において目的の単語が候補単語中に含まれる確率が、表示候補数を増やすことによりどの程度変化するかを示しています。また、n=10 の軸に注目し、表示する候補単語が 10 個に固定されていた場合について調べた場合、最初の 1 文字だけ読みを指定した場合に目的の単語が候補リスト中に表示される確率が 44%、2 文字指定した場合 85%、3 文字指定すると 97%以上となることが分かります。

このように、表示する候補の数が少ない場合でも，読み

5　この性質は言語によらずほぼ成立しますし、都市の人口の分布や会社の売上高の分布など、世の中のさまざまな分布において同様の性質がみられます。単語の出現頻度になぜこのような性質がみられるかについてはまだ本質的に解明されていませんが、さらに広範な現象について Zipf の法則を拡張した冪乗則 $(f \propto (r+a)^b)$ が成り立つことが知られており、このような性質をカオスやフラクタルで説明する試みがさかんにおこなわれています [1]。

図 11　候補リスト中に目的の単語が含まれる確率



を 1〜2 文字指定するだけでほとんどの単語が選択できることが分かります。逆に、表示する候補の数を倍にしても、候補が出現する確率はそれほど増えません。

表示候補数の最適値

アンケートの結果、POBox に慣れない人は、目的の単語が候補リスト中に見つからないときにフラストレーションを感じることが多いと分かりました。とくに、使用頻度の低い単語を入力する場合、まず 1 文字入力したあとで候補中にその単語が出現していないことを確認し、2 文字目を入力したあとで候補中にその単語が出現していないことを確認し、…… という具合に何度も候補リストを確認しているうちに疲れて嫌になってしまうようです。このような場合は読みを 3 文字程度入力してから候補を探したほうが効率的なのですが、慣れないうちはコツがのみこめないのでしょう。

読みを入力するたびに候補単語を確認する場合、候補をいくつ表示するのがもっとも効果的かは読みの入力速度 V_k と候補単語列からの選択速度 V_s の比によって決まります。$V_k \gg V_s$ の場合は、候補の確認に時間をかけるよりも読みを手早く入力したほうが目的単語に早く到達しますから、表示する候補数を少なくして確認の手間を省くほうが有利です。普通のキーボードを使う場合はこのようになります。一方、$V_k \ll V_s$ の場合は、入力の手間はなるべく減らして候補リストからの選択を重視するほうが効率的です。目での確認は高速にできるが、キーボードやポインティング・デバイスの操作は難しいといった状況がこの場合に相当します。ペン入力で携帯端末を扱う場合はこれらの中間の条件になるので、入力装置の操作しやすさに応じて数個から 10 個程度の候補を表示する方法がよいと思われます。

図 12　〝入〟の最初のストロークを書いた状態

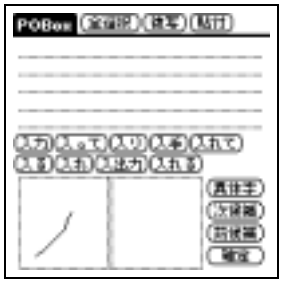
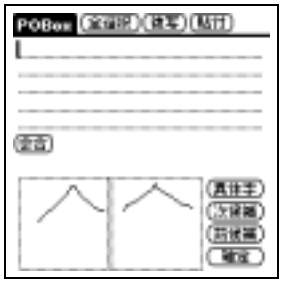


図 13　1 文字目と 2 文字目の両方を部分指定した状態



読み以外による条件指定

POBox では、単語の読みの先頭を指定して候補単語を絞り込むようにしていますが、読み以外の条件を指定して絞り込んでも同様の効果が期待できます。読み以外の条件として最初に考えられるのは、文字の形（文字を構成するストローク）です。文字のストロークを与えるたびに部分的な文字認識をおこない、マッチする文字/単語を検索して候補単語を表示すれば、文字認識による入力も効率化できます [4]。

図 12 は、〝入力〟と入力しようとして最初のストロークを書き始めたところです。図 13 は、〝会合〟と入力するために 1 文字目と 2 文字目の一部だけを手書きで指定したものです。この手法を応用すれば、文字だけでなく絵や画像の入力にも POBox を使えるかもしれません。

おわりに

今回は、予測型文章入力システム POBox の予測手法や、それに関連した話題をいくつか紹介しました。UNIX 版 POBox のソースおよび茶筌を用いた辞書作成ツールのソースは私の Web ページ (http://www.csl.sony.co.jp/person/masui/POBox/) で公開していますのでご利用ください。

（ますい・としゆき　ソニー CSL）

[参考文献]

[1] Murray Gell-Mann, *The Quark And The Jaguar*, W. H. Freeman, 1994（邦訳：野本陽子訳『クォークとジャガー』、草思社、1997 年）

[2] Toshiyuki Masui, "An efficient text input method for pen-based computers", *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'98)*, Addison-Wesley, April 1998

[3] Yasushi Ogawa and Toru Matsuda, "Overlapping statistical word indexing: a new indexing method for japanese text", *Proceeding of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.226–234, July 1997

[4] 増井俊之「動的パタンマッチを用いた高速文章入力手法」、尾内理紀夫（編）『インタラクティブシステムとソフトウェア V』日本ソフトウェア科学会 WISS'97、pp.81–86、近代科学社、1997 年 12 月

おまけプログラム

硬い話が続いたので、茶筌を使った冗談プログラムを〝おまけ〟として紹介します。ニュース記事を HTTP で取得し、固有名詞だけを適当に入れ替えてウソ記事を作ります。

下記のプログラム **asciinews** でアスキーの Web ページから速報記事のリストを取得し、茶筌で形態素解析をしたあと、固有名詞を **fake** プログラムで適当に置き換えるとニセのニュースを楽しむことができます。fake のプログラム名を **quiz** に変更すると、固有名詞の部分を〝○○〟で表したクイズになります。

```
% asciinews | nkf -e | chasen | fake
米ドンデ・インスツルメンツ社、ADSLチップセット3製品を発表
ヴァシュタが情報通信システム事業推進体制を強化
…
%
```

• asciinews プログラム

```
#!/usr/local/bin/perl
require 'chat2.pl';

# プロキシとポートは適当に修正
$proxy = "proxy.foo.co.jp";
$port = 8080;
$url = "www.ascii.co.jp/ascii24/";

sub get {
  local($server,$port,$url) = @_;
  local($s,$r,$status);
   &chat'open_port($server,$port);
   &chat'print("GET http://$url\n");
   $status = '';
   for(;;){
     $r = &chat'expect(1000,'[^\n]+\n','$&',
          'TIMEOUT','$status = 2',
          'EOF','$status = 1');
     if($status == 1){ return $s; }
     elsif($status == 2){ return '';}
     else { $s .= $r; }
  }
}
for(split(/[\r\n]+/,&get($proxy,$port,$url))){
  if (/\x81\x9c(.*)<BR>/){
    print "$1\n";
  }
}
```

• fake/quiz プログラム

```
#!/usr/local/bin/perl
$program = $0;
sub proper{ # でたらめな固有名詞(?)を並べる
  @a = ('ドンデ', 'コモエスタ', 'ファブラ',
   'キテパ', 'ヴァシュタ', 'ホイテ',
   'スレドニ', 'チョリチョリ', 'モンボ',);
  $a[rand($#a)];
}
while(<>){
  chop;
  if(/^EOS/){
    print join('',@s),"\n";
    @s = ();
    next;
  }
  @a = split;
  push(@s, $a[3] ne '固有名詞' ? $a[0] :
    $program =~ /quiz$/ ? '○○' : &proper);
}
```