# Suffix arrayを用いた言語モデリング

伊東秀夫
リコー情報通信研究所
hideo@ic.rdc.ricoh.co.jp

## 1 はじめに

統計的言語モデルとして最もよく用いられるn-gramモデルを次のように分類する。

- 静的n-gramモデル
  事前に次数または推定すべきパラメタ数を限定し、それらのパラメタ値を計算しておくモデル。

- 動的n-gramモデル
  入力に応じて次数を動的に決定し、ランタイムにパラメタ値を計算するモデル。

前者は音声認識など、off-lineでの言語モデル構築が可能な状況においてよく利用され、事前の訓練または学習を重要視する。後者はテキスト圧縮などon-lineでの言語モデル構築が必要な状況で用いられ、その時の入力に対する適応性を重要視する。

テキスト圧縮においてよく用いられる動的n-gramモデルとしてPPM (Predict by Partial Matching) がある [1]。このモデルは予め次数の上限を定めておくが、実際に確率推定に用いられる次数または文脈は入力に応じて決定される。PPMにはPPM* と呼ばれる変種がある [2] [3]。PPMは、モデルの次数n (文脈長) に上限を設けない点を特徴とする。確率推定に用いる文脈を入力に応じて選択する際の自由度が大きい点で様々な発展の可能性を秘めているといえよう。しかしPPM*には次の問題点がある。

- 文脈管理の計算コスト
  過去の文脈をコンパクトに格納し、柔軟かつ高速にそれらを参照および追加削除できる文字列索引が必要になる。この索引構造としてはon-line構築が可能であるtrieやsuffix tree等が用いられる。しかし文脈の規模が増大するにつれ、特に記憶量の面で限界がある。

- 文脈選択の戦略
  現状のPPM*では比較的簡単な文脈選択の戦略を取っており、その可能性を十分に引き出しているとはいえない。実際、多少高次のPPMに対してはテキスト圧縮率で劣る場合が多い。

これらの問題点をテキスト圧縮の分野ではon-line処理という制約下で解決しなければならない。これに対し自然言語処理への応用の多くはoff-lineでの処理を許容する。そこに我々はまず着目し、上記の文脈管理の計算コストの問題を、文字列索引としてSuffix arrayを用いることで解消する。Suffix arrayはoff-line構築しかできないが、無限長の文脈を扱える文字列索引として最もコンパクトであり、検索速度も非常に高速である。これにより大規模な動的言語モデルの構築が初めて可能になる。

そして次に第2の問題である文脈選択の問題について、日本語文字を単位とするn-gramモデルを対象として取り組み、従来のPPM*および静的言語モデルとの性能比較をＥＤＲコーパスを用いて行う。

なお、本モデルの応用系としては、オンライン手書き文字認識における次入力文字の予測および音素認識などを想定している。

## 2 ＰＰＭとＰＰＭ*

有限のアルファベット集合をAとし、Aの要素を記号と呼ぶ。言語モデルへの入力は記号列であり、ある時点までの入力を $x_0, x_1, \ldots x_{i-1}$ とする。ここで各 $x_k (k = 0, 1, \ldots, i-1)$ は記号を表す。n-gramモデルにおいて中心となるのは以下の条件付確率の推定である

$$P(x_i|x_0, x_1, \ldots x_{i-1}) \quad (1)$$

上記で $x_i$ は次の入力として予測される記号であり、$\sum_{x_i \in A} P(x_i|x_0, x_1, \ldots x_{i-1}) = 1$ が要請される。

長さnの記号列 $c_n$ 及び、$c_n$ の直後に記号 $x$ が出現する頻度を与える関数 $c_n(x)$ との対を要素とする集合C（文脈集合と呼ぶ）を用意する。この時 $c_n$ を $n$ 次の文脈と呼ぶ。上記の確率を推定するために文脈集合Cの要素と条件部の記号列を右端を起点として照合して部分マッチする要素 $c_n (n = -1, 0, 1, 2, .., m)$ を取り出す。これらの要素からなる集合を一致文脈集合と呼

ぶ。0次の文脈は空列であり uni-gram に相当する。また仮想的な文脈として $-1$ 次の文脈を設け 0 次の文脈の直後に出現しなかった記号（つまり未知記号）に対する文脈として機能させる。0 次および $-1$ 次の文脈は任意の記号列と部分マッチするものとする。

一致文脈集合の要素の内、最大次数 m を持つ文脈 $c_m$ の頻度関数を用い以下のように上記確率を推定する。

$$p_m = p(x_i|x_0, x_1, \ldots x_{i-1}) = \frac{c_m(x_i)}{C_m + t} \quad (2)$$

ただし $C_m = \sum_{x \in A} c_m(x)$ であり $t$ は $c_m(x) > 0$ となる記号の異なり数である。上記確率値が 0 の時は文脈 $c_m$ にとって $x_i$ は未知であったということに相当する。文脈 $c_m$ における未知記号全体に対しては、次のエスケープ確率 $e_m$ を割り当てる。

$$e_m = \frac{t}{C_m + t} \quad (3)$$

上記のエスケープ確率は未知記号全体に対するものであるから、個々の未知記号に対する確率は、次に大きい次数 $m'$ の文脈を用いて得られた確率との積をとることにより配分される。以上説明したエスケープ機構は、最も高次の文脈をまず用い、その文脈で未知の記号については、一つ低次の文脈を用いるという点で back-off 法と類似している。しかしテキスト圧縮では上記の確率推定をランタイムに行わなければならないため、よりシンプルである。

PPM は文脈集合の要素の次数に上限 (3 が普通) を設けることで文脈管理の計算コストを軽減する。PPM* はこの上限を取り払った PPM であり、より高次の文脈が用いられる。高次の文脈ほどエスケープが発生する割合が高くなり、かつ $C_m$ が小さいので未知記号のために消費する確率が多くなる。この弊害を抑制するため、上記の t が 1 である文脈（決定性文脈）については最短のものを用い、$C_m$ を増大させエスケープ確率を小さくする工夫がなされている。

## 3 Suffix array による実装

PPM*の文脈集合を Trie を用いて実装するとノード数が文脈長の総計をNとするとNの２乗のオーダで増加してしまう。Suffix tree を用いればNのオーダになるが、やはり記憶量の問題は残る。標準的な実装法では 17 Nバイトを要し、現状の計算機環境では文脈集合が 100MB 以上の構築は困難である。

Suffix array を用いることで、前述した文脈集合をコンパクトに表現し、かつ、部分マッチによる文脈選択を高速に行うことができる。さらに Suffix array はアルファベットのサイズによらないので、日本語文字、タグ列、状態遷移列など適用可能な対象範囲が広い。Suffix array を用いた PPM*の実装は以下に説明するように Trie 等に比べ非常にシンプルである。

言語モデル構築は、予めリバース（反転）されたたコーパス T に対して Suffix array $S$ を生成することに相当する [4]。事前の訓練、パラメタの選定、およびパラメタ値の推定は全く行わない。

コーパス中の文字数を N とすると $S$ は長さ N の整数配列、各要素 $S[i]$ の値はコーパス中の文字位置であり、その位置を起点とする Suffix に一対一対応する。配列 $S$ の要素は対応する Suffix をキーとして辞書順にソートされている。検索キーとなる文字列 $K$ に対し配列 $S$ を介してコーパスを二分探索することで、$K$ のコーパス中の全出現位置を配列 $S$ のある連続領域 $S[i,j] = S[i], S[i+1], \ldots, S[j]$ 上に得る。

以下に Suffix array を用いた PPM*の動作アルゴリズムを示す。言語モデルへの入力は文字列 $I = x_0, x_1, \ldots, x_n$ であり、文字列中の各位置で、次に位置する文字の生起確率を求めるものである。入力文字列の各位置 $i = 0, 1, \ldots, n$ において以下を行う。

1. $L \leftarrow x_{i-1}, x_{i-2}, \ldots, x_2, x_0$
2. $L$ をキーとして Suffix array によりコーパスを二分探索し最長一致文字列 $c = x_{i-1}, x_{i-2}, \ldots, x_m$ を求める。$c$ は次数 m の文脈である。
3. 文脈 $c$ のコーパス中の全出現位置は、Suffix array $S$ 上の或る連続領域 $R$ に格納されている。この領域を再び二分探索で求める。得られた領域のサイズは次数 m の文脈 $C$ の出現頻度であり $Cm$ と表現する。
4. 領域 $R$ 中の各配列要素 $S[i]$ について、コーパス中の先頭から $S[i]-1$ の位置にある文字 $x_k$ を検索し、その異なり数 $t$ を求める。また、$x_i = x_k$ が出現した場合はその頻度も求める。この値は前述した頻度関数 $c_m(x_i)$ の値に相当する。
5. $c_m(x_i) > 0$ ならば式 2 の確率 $p_m$ を返して終了。
6. そうでなければ、$L \leftarrow x_{i-1}, x_{i-2}, \ldots, x_{m-1}$ のように次数を一つ下げて上記ステップ 2 に戻り確率 $p_{m-1}$ を求める。そして $e_m \times p_{m-1}$ を返して終了。このエスケープ確率 $e_m$ は式 3 で与えられる。

この再帰的なエスケープ処理 (back-off) は、次文字 $x_i$ の出現がコーパス中で得られるまで行う。次数 0 (つまり uni-gram) でも得られない未知文字に対しては次数 $-1$ の文脈として確率 $p_{-1} = 1/|A|$ を与えて停止す

る。$|A|$はアルファベットサイズである。なお我々の現在の実装では上記に加え、前述の決定性文脈の処理および排除処理 (Exclusion) も導入している。

処理ステップ 4 では、各文脈 $c$ の直後に出現する文字の異なり数 $t$ と頻度を求めるため、Suffix array を介しコーパスをランダムアクセスする。この処理は、次数 0,1,2 では処理速度を非常に低下させる。

この問題を解消するために、これら次数 0,1,2 の文脈については直後に出現する文字の頻度を事前にカウントし記憶しておく。日本語文字の場合、次数 0,1 の文脈集合の記憶は高々数 MB を要するのみだが、次数 2 (つまり tri-gram) の場合は数 10 MB の記憶量になる (例えば新聞 3 年分では 76MB)。この記憶コストを軽減するためには、高頻度の tri-gram のみを記憶すればよい。高速化には十分効果がある。

また、このように低次の文脈を Suffix array とは別に記憶することで動的なタスク適応 (頻度情報の逐次更新など) の手段を提供できる。

# 4 文脈選択のための戦略

PPM では利用文脈な次数が、ある値を過ぎて高くなるほど逆に性能が低下する。つまり、高次になるほど確率を推定できる文字の範囲が少なくなるためエスケープが頻繁に起こる。そしてエスケープ確率の推定の粗さによる性能低下が顕著になる。Cleary らによる決定性文脈処理の導入は、この問題を軽減するものの解消はしないことが事前の実験によりわかった。この問題を解決するには決定性文脈処理よりも、さらに木目の細かい文脈選択の戦略が必要と考えられる。

我々は Method–1 と Method–2 と呼ぶ文脈選択の戦略を提案する。Method–1 は汎用的な戦略だが、Method–2 は日本語文字の n–gram に固有の方法である。

## 4.1 文脈選択 Method-1

この方法の基本的アイデアは、ある文字の生起確率を推定するための文脈を選択する際に、その直前の推定で選択した文脈の情報を利用する点にある。具体的には以下である。

> 次に用いる文脈長の上限をその直前に用いた文脈長＋１とする。

決定性文脈の処理を行わず、常に最大一致の文脈を最初に用いるならば上記の要請は何もしなくても常に満たされる。しかし決定性文脈の導入により、上記の要請が破られる場合が多く起こる。Method–1 により選択された文脈の左端が単語の途中で終わる場合が減少する。

## 4.2 文脈選択 Method-2

この方法の基本的なアイデアは、文脈選択において潜在的にエスケープの可能性が高い状況では高次文脈の選択を抑制する点にある。ここでは以下のように日本語文字の字種情報を利用する。

> 次に用いる文脈長の上限を、現在の文脈の右端が記号類の場合は１、平仮名の場合は３とする。それ以外については上限を設けない。

上記の要請の効果は、記号類および平仮名の次の文字の予測は非常に困難であり、そのような状況での高次の文脈選択が抑制される点にある。平仮名の次文字が結果的に平仮名であった場合は上記の要請は不要だが、結果的に漢字であった場合に得られる性能向上は大きいであろう。

上述の両文脈選択のための戦略を、決定性処理と共に導入した PPM*をここでは PPM++とここでは呼ぶ。

# 5 基本性能の評価

森らは形態素–gram による静的言語モデルを用いて日本語の情報量を求める際の参考となる上限値を求めている [5]。その実験には以下の EDR コーパスが用いられた。我々も表 1 のデータ[1]を用いて動的 n–gram の

| 用途 | 文数 | 形態素数 | 文字数 |
|---|---|---|---|
| 学習用 | 187,022 | 4,595,786 | 7,252,558 |
| 評価用 | 20,780 | 509,261 | 802,576 |

表 1: EDR Corpus

基本性能を調査した。

ただし我々のモデルは文字単位の n–gram モデルなので、文や形態素の情報は一切用いない。また森らにならって学習用と評価用というように各データを名付

[1]EDR コーパスを先頭から走査し 10 の剰余類により 9 対 1 の割合で分割すると上記が得られる。

けたが我々のモデルでは、学習用データを用いた事前訓練等は行わない。単に Suffix array で索引づけされる文字列として用られる。

まず、統計的言語モデルの性能評価としてよく用いられるクロスエントロピー（文字当たりの）と文脈次数の関係を３つの言語モデル（PPM, PPM*, PPM++）について計測した。

各言語モデルの性能値を図１に示す。このグラフは言語モデルの次数の上限を増加させていったときに、クロスエントロピーがどのように変化するかを示している。PPM*と PPM++は事前にモデルの次数の上限を定めない枠組みだが、この実験目的のために、最長一致文脈の次数が上限を超える場合は、上限になるまで文脈を短くするように動作ステップを加えて実験値を得た。グラフから、我々の PPM++の文脈選択の戦略

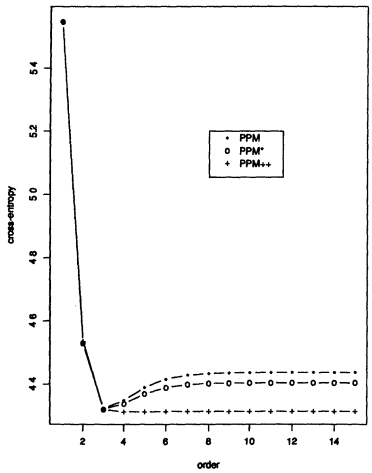


図 1: 次数とクロスエントロピーの関係

の効果がわかる。高次文脈の弊害問題はほどんど解消されている。またクロスエントロピーの面でも総じて PPM*,PPM に比べて優れている。

## 6 静的モデルと動的モデルの比較

森らが前述した EDR コーパスに対して見積もったクロスエントロピーと、我々が実験で得た性能値を用いて、静的言語モデルと動的言語モデルの性能性能比較を以下に行う。

森らのは形態素（単語表記と品詞）を単位とする n-gram であるが性能値として文字当たりのクロスエントロピーを報告している。形態素を単位とするため未知語モデルが必要となり独自の方法を提案しているが、削除補間法による事前訓練を行う静的言語モデルの典型例と位置づけてよいであろう。

森らの実験では、文区切りおよび単語の区切り情報を所与のものとして用いている。つまり可能な全ての記号列の生起確率の和が１になる推定ではない。そこで、単語および文区切り情報つきの文字列に対して、我々の言語モデルを構築し、文字当たりのクロスエントロピーも計測した。具体的には日本語文字に単語および文区切りのための特殊文字を加えたものをアルファベットとし、EDR コーパスの文および単語区切りに沿って文字列を構成した。ただしクロスエントロピーの文字当たりの平均を取る際には、これら拡張文字に関する分は除外してある。アルファベットサイズは全て 6878 である。

結果を以下に示す。上段が単語区切り情報なし、下

| PPM | PPM* | PPM++ | 森ら |
|---|---|---|---|
| 4.43693 | 4,40414 | 4.31289 | – |
| – | – | 3.98183 | 4.30330 |

表 2: cross-entropy of EDR corpus

段が単語区切り情報ありの評価値である。これより事前訓練なしでも PPM++は静的モデル以上の性能を実現できることがわかる。なお、PPM++の一文字当たりの確率推定速度は Ultra-30 上で 0.1 msec であり実用上の問題がないこともわかった。

## 参考文献

[1] I. H. Witten T. C. Bell, J. G. Cleary. *Text Compression.* Prentice Hall, 1990.

[2] J. G. Cleary and W. J. Teahan. Unbounded length contexts for ppm. volume 40, pages 67–75, 1997.

[3] 小田 祐樹 and 北 研二. Ppm* モデルによる日本語単語分割. ＮＬ研報告, 128(2), 1999.

[4] 伊東 秀夫. 大規模テキストに対する suffix array の効率的な構築法. ＮＬ研報告, 129(5), 1999.

[5] 森 信介 and 山地 治. 日本語の情報量の上限の推定. volume 38, pages 2191–2199, 1997.