修士学位論文

統計的手法による形態素解析の過分割検出とその応用

平成 12 年度

茨城大学大学院理工学研究科

システム工学専攻

池谷 昌紀

要旨

本論文では統計的手法を用いた形態素解析の分割誤りの検出と修正の方法を提案し，それらを応用した2つの研究を行った。1つは複合語の単語分割であり，もう1つは未知語の検出である．

1つ目の複合語単語分割は，形態素解析が誤りやすい局所的な部分を文字ベースHMMを用いて修正する．複合語の単語分割は言語処理の基礎技術として重要であるほかに全文検索におけるインデックス作成や検索式の解析を行う際にも必要とされる．通常，複合語の単語分割には形態素解析が使われるが未知語の処理に問題がある．これに対処する手法として文字ベースHMMを使う方法がある．文字ベースHMMでは文字ベースのため，未知語の問題は生じにくい．しかし，文字ベースで処理を行うため長い文字列が1単語になる場合に単語分割が正確でない可能性が多くなる．形態素解析では，その仕組から，長い文字列を1単語としてみなすのは容易である．そこで，両者の利点をうまく利用することにより複合語の単語分割を行うことにする．具体的には，通常は形態素解析の結果を採用し，あるパターンになったときだけHMMの結果を優先させるという方法を考えた．形態素解析による単語分割の誤りは文字列の局所的な部分で発生する．文字ベースのHMMではこのような場合に有効である．そこで，形態素解析が/a/bcd.../　と分割した場合HMMの結果を優先させるパターンとした．これを使って，8543種類の複合語の単語分割を行った結果形態素解析の単語分割の誤りの178種類を正しく修正できた．

2つ目の未知語の検出については，未知語のタイプに応じて2種類の研究を行った．ここでは未知語のタイプを2あるいは3文字からなる漢字列(TYPE1)と一般的な文字列(TYPE2)に分類した．まずTYPE1に対しては，文字列が単語になる確率を用いた過分割検出の尺度を利用する．次にTYPE2に対しては，既存の過分割検出尺度を改良することにより実

現する．未知語の検出は形態素解析の一つの課題である未知語処理に対して有効である．未知語処理とは未知語を含む文を形態素解析にかけると単語分割が誤ってしまうという問題である．形態素解析の結果から未知語を検出できれば形態素解析の精度向上につながる．また情報抽出の前処理としての固有表現抽出を行う際にも人名，地名，組織名などは未知語として出ることが多いためここでも未知語を検出する技術が望まれている．一般的に未知語を形態素解析にかけると過分割が生じる．このことから，本研究では未知語を検出を行うために過分割を判定する尺度を定義した．この尺度が大きいほど未知語になる可能性が大きいといえる．実験の結果，TYPE1 の方は，F 値 0.68(2 文字列)，0.182(3 文字列) を得ることができ，TYPE2 に対しては，118 種類の未知語のうち 43 種類の未知語を抽出できた．従来手法では，文字列が過分割かどうかを判定する際その文字列のコーパス中での頻度を利用する．このため，低頻度のものに対して過分割判定が出来ない可能性がある．本手法で利用した 2 つの尺度は対象となる文字列の頻度を直接利用していない．このため低頻度の未知語に対しても有効であるといえる．

Abstract

In this thesis, I argue about the statistical method to detect and correct word segmentation errors produced by Japanese morphological analysis, and apply it to two problems. One is word segmentation of compounds, and another is detection of unknown words in the next.

In the first method, word segmentation of compound is conducted by correcting word segmentation errors produced by morphological analysis. I use Hiden Maukov Model (HMM) for this correction. Word segmentation for a compound is an important technology because it is required not only in conventional natural language systems but also in indexing of full text search and retrieval and analysis of retrieval keys. In general, word segmentation can be conducted by morphological analysis. However, morphological analysis has the problem of unknown words besides incomplete accuracy. Thus, methods other than morphological analysis have been proposed for word segmentation, such as character-based HMM. Character-based HMM does not suffer the problem of unknown words because it is a character-based method. However, character-based HMM cannot recognize a word composed of many characters as one word. Morphological analysis does not suffer this problem of character-based HMM. It is rather easy for morphological analysis to recognize a word composed of many characters as one word. Accordingly, morphological analysis and character-based HMM are complementary to each other. Concretely, I correct word segmentation errors produced by morphological analysis through character-based HMM. Errors of morphological analysis occur in a local part in a compound, and character-based HMM can avoid such errors. Then, I import error patterns of morphological analysis. If

word segmentation by morphological analysis includes /a/bcd.../, I compare it with the corresponding part of word segmentation by character-based HMM. If they are different, I correct the former to the latter. In experiments, I applied this method to 8,543 kinds of compounds. As a result, this method corrected 178 kinds of incorrect word segmentation produced by morphological analysis.

In the second method, I conduct two kinds of research which are extraction of unknown word. One is for two or three kanji characters (TYPE1), and another is for the general characters (TYPE2). Extraction of unknown words is conducted by the measure for detecting over segmentations using probablity of accepting the kanji character sequence as one word in TYPE1 and by improving existing scale in TYPE2. A problem of morphological analysis is the detection of unknown words which are not in the dictionary. In a morphological analysis of a Japanese sentence, an unknown word causes long word segmentation, therefore, the detection of unknown words in the text improves the accuracy of morphological analysis. Moreover, the importance of detecting proper nouns in the text for information extraction (IE) has recently been recognized. Most proper nouns such as person names, place names and company names are unknown words, so we need a method to detect unknown words also for IE. Generally, unknown words are segmented into some words by the morphological analysis. By this characteristic, I define the measure for detecting over segmentations in order to extract unknown words. And, a string rated high by the measure is likely to be unknown word. In experiments, in TYPE1, the F-measure for extraction unknown words was about 0.7 (two characters) and 0.4 (three characters). In TYPE2, I applied to 118 kinds of unknown words. As a result, this method extracted 43 kinds of unknown words. The conventional methods cannot extract unknown words which have low frequencies in the trainning corpus because they need the frequency of these words. This method does not need to use the frequency of the unknown word in the training corpus to judge whether the word is the unknown word or not. Therefore, this method has the advantage that low frequency unknown words are extracted.

目 次

図目次

表目次

第1章　序論

1.1　はじめに

自然言語処理の重要な課題として高精度の単語分割の実現がある．現在の日本語単語分割は十分高精度であり，実用の域に達しているが，複合語の分割と未知語の検出がまだ十分とは言えない．複合語の単語分割とは，”歩行者優先道路” を ”歩行者/優先/道路” のように複合語をそこに含まれる複数の単語に分割することである．また未知語の検出とは，特殊な人名や地名のような辞書に登録されていない単語（未知語）を文書から検出することである．これらを精度良く行うことで，高精度の単語分割が実現できる．

ここでは，上記の課題に取り組み，大きく３つの研究を行った．それらはみな統計的尺度を用いて形態素解析の過分割を検出するというアプローチをとっている．１つ目の研究はHMMと形態素解析を相補的に利用した複合語の単語分割の研究である．２つ目の研究は2あるいは3文字の漢字からなる未知語に限定した未知語の検出の研究である．３つ目の研究は，２つ目の研究のような制限をおかない一般の未知語の検出である．

３つの研究を通して，統計的尺度を用いて形態素解析の過分割を修正するアプローチにより効果的に複合語の単語分割や未知語の検出が行えることが確認できた．本報告では各研究の概要と結果，それらを通した考察を述べ，最後にまとめる．

1.2　本論文の構成

まず第2章で，単語分割を行なう際に必要となる形態素解析とその問題点について説明する第3章で，本研究を進めるにあたって，必要な自然言語の統計的性質を説明する．以降研究成果として，第4章で，複合語における形態素解析の分割誤りをHMMにより修正

し，複合語単語分割を実現する方法について解説する．第5章では，文字列が単語になる確率を使い文字列が未知語になる尺度を設定し2あるいは3文字からなる漢字列からなる未知語を検出する方法を説明する．第6章では，2つの文字列が分割されるかどうかの尺度を求めそれを使い形態素解析結果から未知語を検出する方法を解説する．

第2章　形態素解析の過分割

2.1　形態素解析とは

形態素解析とは，文を構成する形態素を認定し（単語分割），その品詞を与える作業のことをいう．

形態素とは，正確に言えば，言語学的に意味がある (文を構成する) 最小言語単位の事であるが，以下では，辞書に登録されている単語 (単に語とも呼ぶこともある) を形態素と呼ぶこともある．自然言語解析の通常の流れは，文の形態素解析を行ってから次のステップに進む．文中の形態素が認定できなければ次の処理が不可能なため，形態素解析は自然言語処理の要素技術と言える．

英語の形態素解析では，語は空白で区切られているため，形態素を認定する作業は容易である．しかし，英語では，殆んどの名詞が動詞としても使うことができるため品詞の曖昧性が存在する．このため，英語の形態素解析では，それぞれの単語の品詞を同定することが困難とされている．一方日本語の形態素解析では，語を区切る空白というものが存在せず，日本語における形態素の定義もそれほど明確でないため，単語分割の方が難しいとされている．分割されてできれば品詞の付与はそれほど難しい問題ではない．

次節では既存の形態素解析の手法を紹介する．

2.2　形態素解析の方法

ここでは2語の連接可能性をチェックするもっとも基本的な形態素解析について説明し，次に優先規則によってもっともらしい解を選択する方法について述べる．

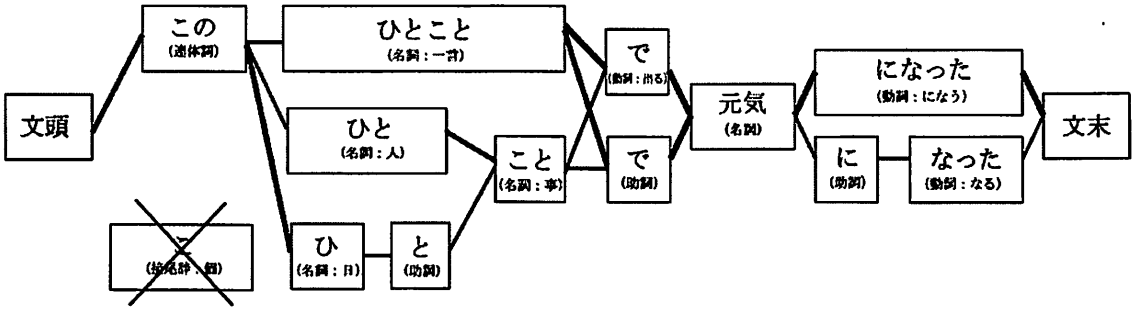

図 2.1: 連接規則による形態素解析

2.2.1　2単語間の連接規則

語の並びとして日本語文を見た場合，どのようなタイプの2語が連続して文中に現れ得るかという条件は明確である．そこで，2語の連接可能性を規則(制約)として与え，それを参照しながら入力文を語のならびに変換するという方法がもっとも基本的な形態素解析の方法である．この処理は，

1. 辞書を参照して入力文中の各位置から始まる語を取り出す

2. 連接可能性をチェックしながら取り出された語をつないでいく

という二つの処理を繰り返し行うことによって実現される．この処理によって得られる解析結果の例を図 2.1 に示す．このように解析結果は語をノードとするラティス構造となる．枝によって連結されている2語は連接可能であったことを示している．このような解析を行うために，「単語辞書」，「連接可能性辞書」の2つの辞書を用意する．

2.2.2　コストの導入

前項の規則を使うことにより，連接可能性を考慮した解析結果を得ることができる．しかし，この結果では不適切な解も多く含まれている．そこで，何らかの優先規則によってもっともらしい解だけを選択することが必要になる．

この問題に対してこれまでに用いられてきた種々の優先規則は次のようになる．

- 最長一致法：文頭から長い形態素を優先して縦形探索で解を抽出する
- 2文節最長一致法：文頭から，2文節ごとの長さが長い解を優先して縦形探索で解を抽出する
- 形態素数最小法：形態素数が少ない解を優先する
- 文節数最小法：文節数の少ない解を優先する
- コスト最小法：語や語の連接にコストを与えて，総コストの少ない解を優先する

これらの方法に共通する考え方は，できるだけ長い語によって構成される解，あるいはできるだけ少ない語数の解を優先するというものである．このような考え方に理論的な根拠は認められないが，直感的にはその良さを認めることができる．

最長一致，2文節最長一致では，文全体のラティス解を求める必要がない．例えば最長一致法では文頭からもっとも長い語を取り出し，次にその語の終る位置からその語に連接可能な語の中で再びもっとも長い語を取り出すという事を繰り返して行う．この方法は明らかに高速であるが，文全体を見た場合には必ずしも長い語の並びを取り出せるわけではない．このような方法は計算機の処理能力の低い時代によく使われていた方法であるが現在でも仮名漢字変換のように非常に高速な処理が要求される状況では用いられることがある．また形態素数最小法，文節数最小法，コスト最小法は一旦連接規則を用いた方法で連接可能性を考慮した解析結果を求めておいてから種々の基準でもっともらしい解を選択する．これらの方法でもっともよく使われてきたのは文節数最小法で，これは得られたラティス解の中から文節数が最小である解を選択する．一方，現在使われている形態素解析では主にコスト最小法が使われている．

コスト最小法は，ラティス解のノードとリンクに適当なコストを与え，コスト最小のパスを優先解として選択する方法である．形態素数最小法は，語のコストを1，連接のコストを0にした場合のコスト最小法に対応する．また文節数最小法は，自立語のコストを1，それ以外の語のコストと連接コストを0とした場合のコスト最小法に対応する．コストの与え方をさらに詳細にして，各品詞や個々の語に個別のコストを与えたり，種々のタイプ

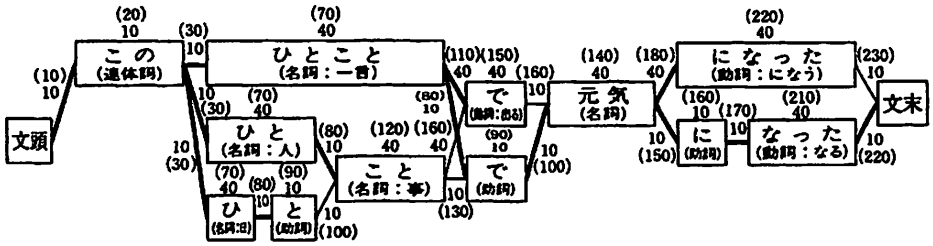

図 2.2: コスト最小法による形態素解析

の語の連接に異なったコストを与えることによって，よりきめの細かい優先規則を実現することが可能である．

ノードとリンクに適当なコストを与えて前述の例に対してコスト最小法を実行した結果を図 2.2 に示す．この図では，() 内の数字が各ノード，リンクまでの部分最小コストで，太線のリンクがマーク付けされたリンク（部分最適解を示すリンク）である．文末から太線のリンクでつながれたノードをたどることによってコスト最小解を得ることができる．

2.3　問題点

2.3.1　複合語の問題

コストによって優先解を求めるという方法では，できるだけ長い語が優先的に取り出されることになる．そのため，広い範囲でみた場合に確実な語の並びがあれば，それらをひとまとまりで辞書登録することによって解析精度を向上させることができる．ひとまとまりにする方が良いものとしては以下のようなものがある．

- 広く認知されている (1 語と認識されている) 複合語や，専門用語である複合語.「本+棚」,「計算+機」,「自然+言語+処理」など.
- 助詞として働く複合語.「に+対して」,「に+関する」など.

- 助動詞として働く複合語.「なければ+なら+ない」など.

このような語を辞書登録する場合には，その内部構造の情報を付加したかたちで登録し，必要な時にその情報が簡単に参照できるようにする必要がある.

2.3.2　未知語の問題

形態素解析では未知語 (形態素解析システムの単語辞書に登録されていない語) に遭遇することは避けられない．未知語は多くの場合，人名，地名，会社名などの固有名詞である．そこで，ある位置から辞書引きを行っても何も語が得られない場合には，連続する漢字列，カタカナ列，記号列などが名詞であるとして解析を進めることによって多くの場合うまく対処することができる．しかし，名詞以外の未定義語 (例えば「スタンばる」などの新語) にまで対処しようとすると問題は極端にむずかしくなり，今のところ有効な解決策は存在しない.

コストの自動調整

英語の品詞付けの処理は品詞付きテキストコーパスを利用して確率的に行う方法が主流となっている．このような方法は，規則，コストを人手で与える際の手間，均一性，精度などの問題をある程度解決することができる．一方，日本語文の形態素解析の場合はコストを人手で与えることの方が一般的である．これは，これまで日本語の品詞付きコーパスが存在しなかったこと，単語の区切りを発見するという問題を確率的に解決するのはかなりむずかしいことなどが理由である．しかし，最近になって日本語コーパスもある程度整備されてきたので，日本語形態素解析においてもコストの自動調整，確率的モデルの適用などが出来るようになった.

2.4　分割誤り

2.4.1　分類

普通，ある文字列を分割した場合，分割誤りというのは以下の3つになる．

- 本来分割しない場所で分割してしまう過分割 (例えば，/abc/ が正解であるところを /a/bc/ のようにしてしまう)
- 分割しなければならない場所で分割をしない分割不足 (例えば，/a/bc/ が正解であるところを /abc/としてしまう)
- 分割点がずれている，語境界交差型 (例えば正解が /a/bc/であるにも関わらず/ab/c/ のようにしてしまう)

形態素解析においてこの3つを考える時，2番目の分割不足は，誤りとするのはむずかしい．なぜなら，形態素解析で文字列 α ， β が/$\alpha\beta$/ となると判断されたとき，辞書に $\alpha\beta$ という単語が登録されているからである．例えば仮に “池谷昌紀” という文字列が形態素解析によって分割されなかった場合このシステムに池谷昌紀という単語が登録されていたということになる．つまりこのシステムで，池谷昌紀を分割しないと判断したことは分割不足ということにはならない．

3番目の分割点がずれているという問題は過分割と分割不足が同時に起こっていると考えることができる．上でも書いたように分割不足というのは起こりにくいため殆んどの場合，原因は過分割にあると考えることができる．つまり，/a/bc/ において bc の過分割さえなければ正しく分割できる可能性が高くなる．

2.4.2　過分割誤り

一般的に形態素解析の分割誤りというのは過分割である．そして，茶筌において過分割の原因の大半が未知語によるものである．もちろん，未知語が原因でない過分割も存在するが現在の茶筌ではほとんど発生することがない．

未知語による形態素解析の過分割の一番単純な例を下に示す．a,b という文字からなる単語 /ab/ が形態素解析にとって未知語であった場合，次に /a/ と /b/ という単語があるかを探す．普通，1文字は何かしら単語になるので，結果的に /a/b/ という過分割が発生する．

/a/bc/という単語において，/ab/ が既知，/bc/ が未知，/abc/が未知という場合，上で説明した /ab/c/ という結果が得られる場合もある．

2.5　茶筌

茶筌 は，計算機による日本語の解析の研究を目指す多くの研究者に共通に使える形態素解析ツールを提供するために開発され，原形は京都大学長尾研究室，奈良先端科学技術大学院大学松本研究室において開発された JUMAN(2.0) である [10].

解析を行なう際，使用者によって文法の定義，単語間の接続関係の定義などを容易に変更できるように配慮されている．

2.5.1　使用している辞書

茶筌 で用いられている辞書は，wnn かな漢字変換システムの辞書，および，ICOT から公開された日本語辞書を利用し，開発者が独自に修正を加えている．

2.5.2　形態素解析アルゴリズム

茶筌は，EUC あるいは JIS(ISO-2022-JP) コードの日本語文字列を標準入力から一行ごとに読み込んで入力とし，コスト最小 (それぞれの形態素の区切りで最小コストとの差が許容されるコスト幅いない) の解を求め，結果をオプションに従って表示する．出力時のコードは日本語 EUC である．

茶筌の解析アルゴリズムは，入力として与えられた日本語の文字列に対する次の基本動作よりなる．改行をもって一つの入力文字列の終了とする．

- ある特定の位置からはじまるすべての可能な形態素を辞書引きによって得る.
- 辞書引きによって得られた個々の形態素に対して，その直前の位置に存在するすべての形態素との連接可能性のチェック，および，コストの計算を行なう.

2.5.3　未知語の取り扱い

Chasen の前身である JUMAN では，入力文字列中のあらゆる位置で未知語が存在する可能性を考慮している．平仮名および漢字については一文字ずつを一語の未定義語として切り出す．それ以外の文字については，同種の文字 (カタカナ，アルファベット，数字 等) の終りまでをまとまった一語の未定義語とする．そして，カタカナ文字列には「カタカナ」，アルファベット文字列には「アルファベット」，それ以外には「その他」という品詞細分類を与える.

未定義語という品詞と，その細分類である「カタカナ」,「アルファベット」,「その他」は形態品詞辞書で定義しておかなければならない．連接関係を連接規則辞書で定義すること，コストをリソースファイルの「品詞コスト」欄で定義することは通常の品詞と同様である.

以上のことに加え茶筌 では未定義語 (未知語) 連接コストの導入により，未定義語の出力を減らすことができるようになった.

2.5.4　茶筌 を用いた形態素解析の例

茶筌を用いた形態素解析の一例を示す．次の様なデータに形態素解析をかけてみた.

```
==================================
マイケルジャクソン
豊田君
豊田市
老子
私は昨日学校へ行きました.
==================================
```

結果は次のようになった.

```
==================================
マイケル マイケル マイケル 名詞-固有名詞-人名-名
ジャクソン ジャクソン ジャクソン 名詞-固有名詞-人名-姓
EOS
豊田 トヨダ 豊田 名詞-固有名詞-人名-姓
君 クン 君 名詞-接尾-一般
EOS
豊田市 トヨタシ 豊田市 名詞-固有名詞-一般
EOS
老 ロウ 老 接頭詞-名詞接続
子 コ 子 名詞-一般
EOS
私 ワタシ 私 名詞-代名詞-一般
は ハ は 助詞-係助詞
昨日 キノウ 昨日 名詞-副詞可能
学校 ガッコウ 学校 名詞-一般
へ へ へ 助詞-格助詞-一般
行き イキ 行く 動詞-自立 五段・カ行促音便 連用形
まし マシ ます 助動詞 特殊・マス 連用形
た タ た 助動詞 特殊・タ 基本形
. . . 記号-句点
EOS
==================================
```

第3章　統計的にみた言語処理

3.1　自然言語の統計的性質

3.1.1　文字・単語の使用頻度

文字や単語の仕様頻度については，かなり古くから調べられてきている．特に印刷所では，各文字の使用頻度に応じて活字を用意しておかなければならず，どの文字がよく使われるかということが経験則として知られていた．印刷所での経験則では，英語のアルファベットの場合，”ETAOINSHRDLU”の順番でよく用いられるとされていた．その後，英語アルファベットの頻度順は”ETAOINSHRDLU”ではなく，”ETAONIRSHDLC”であるとか，あるいは”ETANOISRHLDC”であるなどの報告もなされている．もちろん，どのような種類の文献を調査したかによって，調査結果が異なってくるのは当然であろう．

このような文字の使用頻度に関する経験則が有効に使われている例として，モールス符号をあげることができる．モールス符号は，モールス (Morse,S.) とベイル (Vail,A.) により考案された，文字伝送のための符号化方式であり，各文字は空白 (スペース) とトンとツーの組み合せによって表現される．空白は電流が流れない状態に，またトンは電流が短時間流れることに，ツーは電流が長時間流れることに対応している．モールスらは，伝送効率を高めるために，頻繁に使われる文字に短い符号を割り当て，滅多に使われない文字には長い符号を割り当てるということを考えた．実際には，最も使用頻度の高いアルファベットである”E”に最短の符号である「トン」を割り当て，次に使用頻度の高い”T”に「ツー」を割り当てている．

電子化されたコーパスが利用可能になるに連れ，プログラムにより自動的に頻度調査を行なうことができるようになり，各種のコーパスを対象に調査が行なわれている．単なる文字の頻度だけではなく，連続した文字列に関する頻度調査も行なわれている．表 3.1 は，

ブラウン・コーパス中の頻度文字および文字列を示している．なお，表中の"*"は空白(スペース)を表している．

単語についても各種の調査が行なわれており，英語の場合，単語の平均長は約4.5文字であると一般に認められている．単語の使用頻度調査についてもいろいろ行なわれているが，ここで問題となるのは単語に対する定義である．例えば，"letter","letters","lettering"などは表層上は異なった単語であるが，これらはすべて"letter"から派生した単語であり，これらを同じ単語として数えるのかあるいは異なった単語として数えるのかという問題がある．同じ問題が同形異義語(例:助動詞の"can"と名詞の"can")や異表記(例:"can not"，"cannot","can't")などの場合にも存在する．したがって,，単語の使用頻度を調査する場合には，単語とは何かという定義を明確にしておく必要がある．

表3.2は，ブラウン・コーパス中の頻出単語および単語列についての調査結果の一部を示したものである．表から明らかなように，"the"や"of"などの短い機能語は名詞や動詞などの実質語よりも使用頻度が高いことがわかる．なお，ブラウン・コーパスに出現する最も出現頻度の高い5文字単語は"which"であり，最も出現頻度の高い6文字単語は"should"，7文字単語は"through"，8文字単語は"American"，9文字単語は"something"，10文字単語は"individual"，11文字単語は"development"である．また，使用頻度の高い上位100個の単語がコーパス中の述べ単語数の42それらは異なり単語数(100,237語)のわずか0.1コーパス中に一度しか現れなかった単語は，異なり単語数の58延べ単語数という点からすると5.7また，この調査では，コーパス中には辞書に載っていない単語(派生語，固有名詞，ハイフンで連結された語など)も非常にに多いということが明らかにされている．LOBコーパス中のタグ(品詞)や単語の頻度調査も行なわれており，調査結果は2分冊(1巻はタグおよび単語の頻度，2巻はタグ列および単語列の頻度)となって刊行されている．ブラウン・コーパスやLOBコーパスは，さまざまな分野のテキストから成るコーパスであるが，特定の分野での頻度調査も行なわれている．文献には，香港理工科大学(Hong Kong University of Science and Technology;HKUST)が中心になって収集した計算機科学関係の約100万語のテキストの頻度調査結果が掲載されている．

表 3.1: ブラウンコーパス中の頻出文字・文字列

	文字	確率	2文字組	確率	3文字組	確率	4文字組	確率
	*	17.41	e*	3.05	*th	1.62	*the	1.25
	e	9.76	*t	2.40	the	1.36	the*	1.04
	t	7.01	th	2.03	he*	1.32	*of*	0.60
	a	6.15	he	1.97	*of	0.63	and*	0.48
	o	5.90	*a	1.75	of*	0.60	*and	0.46
	i	5.51	s*	1.75	ed*	0.60	*to*	0.42
	n	5.50	d*	1.56	*an	0.59	ing*	0.40
	s	4.97	in	1.44	nd*	0.57	*in*	0.32
	r	4.74	t*	1.38	and	0.55	tion	0.29
	h	4.15	n*	1.28	*in	0.51	n*th	0.23
	l	3.19	er	1.26	ing	0.50	f*th	0.21
	d	3.05	an	1.18	*to	0.50	of*t	0.21
	c	2.30	*o	1.14	to*	0.46	hat*	0.20
	u	2.10	re	1.10	ng*	0.44	*tha	0.20
	m	1.87	on	1.00	er*	0.39	***	0.20
	f	1.76	*s	0.99	in*	0.38	his*	0.19
	p	1.50	,*	0.96	is*	0.37	*for	0.19
	g	1.47	*i	0.93	ion	0.36	ion*	0.18
	w	1.38	*w	0.92	*a*	0.36	that	0.17
	y	1.33	at	0.87	on*	0.35	*was	0.17
	b	1.10	en	0.86	as*	0.33	d*th	0.16
	,	0.98	r*	0.83	*co	0.32	*is*	0.16
	.	0.83	y*	0.82	re*	0.32	was*	0.16
	v	0.77	nd	0.81	at*	0.31	t*th	0.16
	k	0.49	.*	0.81	ent	0.30	atio	0.15
	T	0.30	*h	0.78	e*t	0.30	*The	0.15
	"	0.29	ed	0.77	tio	0.29	e*he	0.15
	...	...	...	...	...	...	...	...
文字組数	94		3410		30249		131517	
エントロピー	4.47		3.59		2.92		2.33	

表 3.2: ブラウンコーパス中の頻出単語・単語列

	単語	確率	2単語組	確率	3単語組	確率
	the	6.15	of the	0.95	one of the	0.03
	of	3.54	in the	0.55	as well as	0.02
	and	2.70	to the	0.33	the United States	0.02
	to	2.51	on the	0.23	out of the	0.02
	a	2.14	and the	0.21	some of the	0.02
	in	1.90	for the	0.17	the end of	0.01
	that	0.97	to be	0.16	the fact that	0.01
	is	0.95	at the	0.15	part of the	0.01
	was	0.94	with the	0.14	to be a	0.01
	for	0.86	of a	0.14	of the United	0.01
	with	0.68	that the	0.13	a number of	0.01
	as	0.65	from the	0.13	end of the	0.01
	he	0.65	by the	0.13	members of the	0.01
	The	0.64	in a	0.13	in order to	0.01
	his	0.63	as a	0.09	the user of	0.01
	be	0.61	with a	0.09	that he had	0.01
	on	0.61	is a	0.08	the number of	0.01
	it	0.54	it is	0.08	most of the	0.01
	had	0.50	of his	0.08	side of the	0.01
	by	0.49	was a	0.08	that he was	0.01
	at	0.49	is the	0.08	in front of	0.01
	I	0.44	had been	0.07	and in the	0.01
	not	0.41	for a	0.07	thre is a	0.01
	are	0.41	it was	0.07	of the most	0.01
	from	0.41	he was	0.07	It was a	0.01
	or	0.40	into the	0.07	One of the	0.01
	have	0.38	as the	0.07	thre was a	0.01
	...	...	...	...	...	...
単語組数	100237		539929		884371	
エントロピー (bit/word)	11.47		6.06		2.01	
エントロピー (bit/char)	1.94		1.03		0.34	

3.1.2　文字・単語の出現頻度分布

単語の出現順位と出現確率の間には，ジップの法則 (Zipf's law) と呼ばれる経験則が成り立つことが知られている．この経験則は，単語の出現順位と出現頻度 (あるいは出現確率) は反比例の関係にあるというもので，横軸を単語の出現順位の対数，縦軸を出現頻度の対数とした座標平面上に単語を出現頻度の大きい順に並べると右下がりの直線になる．ブラウン・コーパス中の単語を，頻度順に対数座標平面上にプロットしたものが図 3.1 であるが，単語が頻度順にきれいに直線上に並んでいる．

ジップの法則を式で表すと，出現頻度 n 番目の単語の出現確率は，

$$P_n = \frac{C}{n}$$

となる．C は定数であり，調査対象となったコーパスから求めることができる．ジップは，ジェームズ・ジョイス (James Joyce) の「ユリシーズ (Ulysses)」中の単語を調べることにより，C の値はほぼ 0.1 であると推定している．

ジップの法則は出現頻度が中程度の単語にはよく当てはまるが，出現頻度の小さい単語や大きい単語に対しては，現実の頻度分布を必ずしも反映していない．マンデルブロー (Mandelbot,B.) は，単語の出現頻度分布のより精密な近似式として，次のようなものを提案している．

$$P_n = \frac{C}{(n+A)^B}$$

マンデルブローによると，B の値は 1 よりも大きく $1/B$ によって語彙の豊富さを測ることができるとしている．すなわち，$1/B$ の値が大きいほど，調査した文献中に他数のいろいろな単語が使われていることになる．

ジップの法則は単語の出現頻度分布に非常によく当てはまるが，文字の出現頻度分布に対してはあまりいいモデルであるとはいえない．文字の場合には次のような式がよく当てはまるとされている．

$$P_n = \frac{1}{n}\sum_{i=0}^{N-n}\frac{1}{N-i}$$

N は文字数，P_n は出現順位が n 番目の文字の確率である．なお，この式は文字だけでなく，音韻の頻度分布に対してもよい近似となっている．

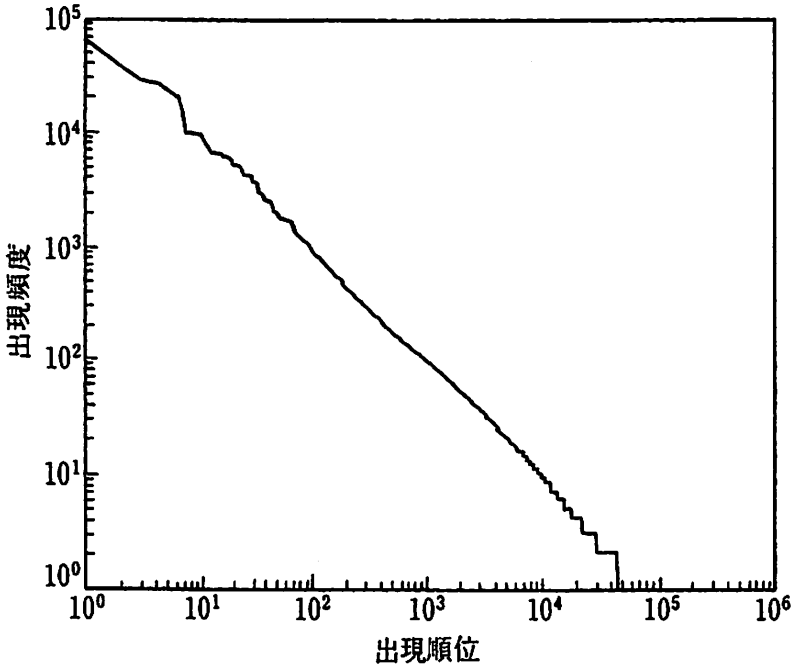

図 3.1: ジップの法則

3.2 N-gramモデル

3.2.1 N-gramモデルの定義

n単語からなる単語列 $w_1...w_n$ が与えられたとき，単語列 $w_1...w_n$ の生成確率は次式により与えられる [5].

$$P(w_1...w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_1...w_{n-1}) = \prod_{i=1}^{n} P(w_i|w_1...w_{i-1})$$

しかし，さまざまな単語の組み合せに対し，条件付確率 $P(w_i|w_1...w_{i-1})$ を求めるのは実際上不可能である．単語数 L の場合，$P(w_i|w_1...w_{i-1})$ を完全に指定するためには，L^i 個もの値を求めなければならない，現実的なモデルとするためには，単語数の履歴 $w_1...w_{i-1}$ を同値類に分割し，モデルのパラメータ数を削減する必要がある．

一般にある時点で生起する事象の確率が，その直前の N 個の時点で生起した事象だけの影響を受けるとき，これを N 重マルコフ過程というが，単語の生起を $N-1$ 重マルコフ過程で近似したモデルを N-gram モデルと呼んでいる．すなわち，N-gram モデルでは，ある時点での単語の生起は直前の $N-1$ 単語にのみ依存すると考えている．したがって，N-gram モデルにおいては，

$$P(w_n|w_1...w_{n-1}) = P(w_n|w_{n-N+1}...w_{n-1})$$

となる．

いま，単語列 $w_1...w_n$ がコーパス中に出現する回数を $C(w_1...w_n)$ で表すことにする．N-gram の確率は，コーパス中に出現する単語の N 個組と $(N-1)$ 個組の出現回数から，次のように推定することができる．

$$P(w_n|w_{n-N+1}...w_{n-1}) = \frac{C(w_{n-N+1}...w_n)}{C(w_{n-N+1}...w_{n-1})}$$

Nの値が大きいほど，学習データから信頼性の高い N-gram の値を推定するのが難しくなる．通常音声言語処理でよく使われているのは，$N=2$ あるいは $N=3$ の場合であり，それぞれ bigram，trigram と呼ばれている．また，N=1 の場合は単語の生起確率となるが，これを特に unigram と呼んでいる．

N-gram モデルの学習では，学習データ中に出現しない単語列に対しては確率値として 0 を推定しまうという問題がある．また，たとえ学習データ中に出現しても，出現頻度の小さな単語列に対しては，統計的に信頼性のある確率値を推定するのが難しい．このような問題は，スパースネスの問題 (sparseness problem) と呼ばれている．スパースネスの問題に対処するために，さまざまなスムージング (smoothing) の手法が提案されている．以下で，代表的なスムージングの手法であるバックオフ・スムージングについて説明する．

3.2.2　バックオフ・スムージング

バックオフ・スムージング (back-off smoothing) は，コーパス中に出現しない N-gram の値を，低次の (N-1)-gram の値から推定する方法である．この方法は，グッド・チューリングの推定 (Good-Turing estimator) と呼ばれるものを基礎にしている．なお，グッド・チューリングの推定では，語彙数が有限で，かつ，単語の出現が二項分布に従うという仮定をおいている．

グッド・チューリングの推定

学習データのスパースネスに対処する一つの方法として，単純に出現回数 r を使うのではなく，出現回数を補正した数 r^* を使うということが考えられる．このように，出現回数 r を r^* に補正することを ディスカウンティング (discounting) といい，比 $d_r = r^*/r$ をディスカウント係数 (discount coefficient) と呼ぶ．ディスカウンティングを使った場合，語彙数 N のコーパス中に r 回出現する単語 w の生起確率は $P(w) = r^*/N$ と推定される．いま，コーパス中に r 回のみ出現する単語の数を n_r で表す．ディスカウンティングにおいて，

$$r^* = (r+1)\frac{n_{r+1}}{n_r}$$

を使った場合の推定をグッド・チューリングの推定と呼ぶ．

コーパス中の単語 w の出現回数を $C(w)$ で表すことにし，コーパス中に一回以上出現した (すなわち $C(w) > 0$) すべての単語に対し，グッド・チューリングによる単語の生起確

率の和を求めることを考えよう．

$$\sum_{w:C(w)>0} P(w) = \sum_{r\geq 1} \frac{n_r r^*}{N} = \sum_{r\geq 1} \frac{(r+1)n_{r+1}}{N}$$
$$= \sum_{r\geq 1} \frac{r n_r}{N} - \frac{n_1}{N}$$

一方，コーパス中に延べ単語数 N については，次の式が成り立つ．

$$N = \sum_{r\geq 1} r n_r$$

したがって，

$$\sum_{w:C(w)>0} P(w) = 1 - \frac{n_1}{N}$$

上式は，グッド・チューリングの推定を用いた場合には，コーパス中に出現するすべての単語に対する確率値の合計が 1 にならない (1 に n_1/N だけ足りない) ことを示している．つまり，未知単語 (コーパスに出現しない単語) に対する確率値の合計を n_1/N を推定していることになる．すなわち，

$$\sum_{w:C(w)=0} P(w) = \frac{n_1}{N}$$

バックオフ・スムージング

ここでは，バックオフ・スムージングを用いた単語の bigram の推定について説明する．コーパス中に単語列 w_1w_2 が出現する場合には，グッド・チューリングの推定値を用いて，条件付き確率 $P(w_2|w_1)$ を求める．

$$P(w_2|w_1) = \frac{C^*(w_1w_2)}{C(w_1)} = \frac{C^*(w_1w_2)}{C(w_1w_2)} \frac{C(w_1w_2)}{C(w_1)}$$

次にコーパス中に単語列 w_1w_2 が出現しない場合について考えよう．グッド・チューリングの推定のところで延べたように，コーパス中に出現するすべての bigram に対し，上式を用いて推定した確率値の合計は 1 にはならない．1 からの不足分を，$C(w_1w_2) = 0$ となるような単語列に対して分配することを考える．ここで次のような関数 β を定義する．

$$\beta(w_1) = 1 - \sum_{w_2:C(w_1w_2)>0} P(w_2|w_1)$$

右辺の $P(w_2|w_1)$ は上式より求められたものである．関数 $\beta(w_1)$ は単語 w_1 に対し，$C(w_1w_2) = 0$ となるようなすべての単語 w_2 に対する条件付き確率の和を与えている．$\beta(w_1)$ を単語 w_2 の出現確率 $P(w_2)$ に従って，$C(w_1w_2) = 0$ となるような単語列に分配することにより，$P(w_2|w_1)$ を求める．このような分配を行なう関数を α とすると，$C(w_1w_2) = 0$ の場合には，

$$P(w_2|w_1) = \alpha P(w_2)$$

となる．α は，単語 w_1 に依存する関数であり，次のように定義される．

$$\alpha = \alpha(w_1) = \frac{\beta(w_1)}{\sum_{w_2:C(w_1w_2)=0} P(w_2)}$$

実際にバックオフ・スムージングを用いる場合，出現回数の大きい単語列に対しては単純に相対頻度を用い，出現回数の小さな単語列に対してだけグッド・チューリングの推定値を用いるという方法がとられることがある．すなわち，適当な定数 k に対し，$r > k$ であるときは，$d_r = 1(r^* = r)$ とし，$r \leq k$ であるときのみグッド・チューリングの推定値を用いる．カッツ (Katz, S. M.) は，$k = 5$ 程度が適当な値であると報告している．また，彼は $d_1 = 0$ としても，実用上問題ないと報告している．これにより，言語モデルの記憶空間の節約を図ることができる．

第4章　複合語単語分割への応用

4.1　背景

本章では複合語の単語分割を行うために，通常の形態素解析と文字ベースの HMM とを相補的に利用する方法を提案する [1].

複合語の単語分割は従来の言語処理システムの一要素技術として重要であるだけでなく，全文検索で生じるインデックス作成や検索式の解析などにも必要とされその重要性は高い．複合語の単語分割は一般に形態素解析によって行えるが，分割精度の他に未知語の扱いも問題となるため，形態素解析を用いない手法も提案されている [4] [8] [9]．その中の一つとして文字ベースの HMM もある．文字ベースの HMM では，文字ベースなので未知語の問題は生じない．ただし文字ベースの HMM において，状態 i から 状態 j に遷移するときに文字 a が出力されるコスト B を uni-gram や bi-gram などから得ると，長い文字列が一単語となる場合に正しく単語分割が行えないことが多い．そのためにプラスアルファの何らかの工夫が必要となる．山本は単純な文字ではなく，拡張文字として品詞などの情報も文字に付加している．また Tsuji は対訳辞書から形態素数を得る工夫を行っている．また小田は PPM モデルを利用して出力シンボル可変長 $n-gram$ にしている．

ただし上記の問題は形態素解析では生じない．形態素解析による単語分割では，長い文字列からなる一単語を正しく認識することはむしろ容易である．一方，形態素解析による単語分割の誤りは文字列の局所的な部分であり，あるパターンが存在する．このような文字列の局所的な部分の単語分割に対しては，文字ベースの HMM が有効である．

つまり単語分割に対して文字ベースの HMM と一般の形態素解析は相補的に利用できる．そこで本論文では，形態素解析で生じた誤りを文字ベースの HMM から修正することで単語分割を行う．形態素解析の誤りをどのように判断するかが問題であるが，ここでは形態

素解析の誤りパターンに注目し，該当パターン部分を文字ベースのHMMによる単語分割結果と比較することで，誤りかどうかを判断する．

また本論文で利用した形態素解析システムはJUMAN 3.5であることを注記しておく．

4.2 HMM

HMM（Hidden Markov Model, 隠れマルコフモデル）は，出力シンボルによって一意に状態遷移が決まらないという意味での非決定有限状態オートマトンとして定義される（一般に，マルコフモデルは最終状態の概念はないが，今回の実験での単語分割に用いたマルコフモデルは初期状態，最終状態を設定する）[3] [6] [7]．定義から当然HMMはユニフィラーではない．すなわち，出力シンボルが与えられても状態遷移系列は唯一に決まらない．観測できるのはシンボル系列だけであることから隠れ(hidden)マルコフモデルと呼ばれている．簡単な例を図2.4に示す．図中のアーク上のスカラ値は状態遷移確率を，アーク上のベクトル値はシンボル a, b の状態遷移による条件付きの出力確率を示している．すなわち，ベクトルの第一要素が a の，第二要素が b の出力確率を示している．この場合シンボル系列が abb であった場合可能性のある状態遷移系列は $S_1S_2S_3$ と $S_1S_2S_2S_3$ の二通りがある．今回は出力シンボル系列は最終状態に達するものとして扱う．通常HMMは全状態が最終状態となりうるが，複合語の単語分割では一部分の状態のみが最終状態となる．一般性を失うことなく，状態遷移にナル遷移（シンボルを何も出力しない）は存在せず二つの状態間での遷移はたかだか一つと仮定できる．前者の場合は，ナル遷移のときは空記号 λ が出力されたと考えればよい．また，ナル遷移を除去することもできる．

これによって後者が満たされなくなるが，二つの状態遷移 $S_2 \to S_3$ に対して，新たな状態 $\dot{S}_3$ を設けて，一方の遷移を $S_2 \to \dot{S}_3$ に変換し，状態 $\dot{S}_3$ の遷移先を状態 S_3 の遷移先に等しくすればよい．また，y, x をそれぞれ出力と状態の確率変数値の系列としよう．今考えているHMMを $\boldsymbol{M}$ とすれば，マルコフモデルの仮定から

$$P(x_i|x_i^{i-1}, \boldsymbol{M}) = P(x_i|x_{i-1}, \boldsymbol{M})$$

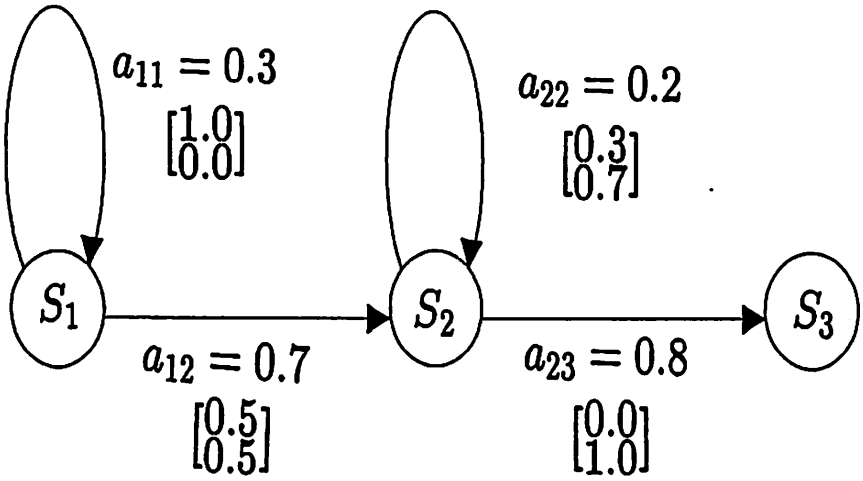
$a_{11} = 0.3$
$\begin{bmatrix}1.0\\0.0\end{bmatrix}$
$a_{22} = 0.2$
$\begin{bmatrix}0.3\\0.7\end{bmatrix}$
S_1
S_2
S_3
$a_{12} = 0.7$
$\begin{bmatrix}0.5\\0.5\end{bmatrix}$
$a_{23} = 0.8$
$\begin{bmatrix}0.0\\1.0\end{bmatrix}$

図 4.1: HMM の例

また，非決定性（hidden）の仮定から，

$$P(y_i|\boldsymbol{x}_1^i, \boldsymbol{y}_1^{i-1}, \boldsymbol{M}) = P(y_i|x_{i-1}, x_i, \boldsymbol{M}) = P(y_i|x_{i-1}, \boldsymbol{M})$$

となる．式は状態遷移によって y_i の出力確率が定まるとするもので，式は遷移元の状態によって定まるとするものである．混同のない限りHMMを示す $\boldsymbol{M}$ の条件項を省くことにする．単語 w_i に対するHMMを上述の $\boldsymbol{M}$ とすれば，式の $\boldsymbol{Y}_{t_{i-1}}^{t_i}$ をあらためて $\boldsymbol{y}_1^T$ とおけば，$P(\boldsymbol{y}_1^T|M)$ は，

$$P(\boldsymbol{y}_1^T) = \sum_{\boldsymbol{x}} P(\boldsymbol{y}_1^T|\boldsymbol{x})P(\boldsymbol{x})$$

となる．また，

$$P(\boldsymbol{x}) = \prod_i P(x_i|\boldsymbol{x}_1^{i-1}) = \prod_i P(x_i|x_{i-1})$$

から，

$$P(\boldsymbol{y}) = \sum_{\boldsymbol{x}} \prod_i P(x_i|x_{i-1}) \cdot P(y_i|x_{i-1}, x_i)$$

となる．図2.4の例で $P(abb)$ の値を式を用いて求めてみよう．先にも述べたとおり $\prod_i P(y_i|x_{i-1}, x_i)$ が 0 にならないのは二つの状態系列だけであったので，この各々の系列による値を $P_1(abb)$, $P_2(abb)$ とすれば，

$$P_1(abb) = 0.3 \times 1.0 \times 0.7 \times 0.5 \times 0.8 \times 1.0 = 0.084$$

$$P_2(abb) = 0.7 \times 0.5 \times 0.2 \times 0.7 \times 0.8 \times 1.0 = 0.0392$$

ゆえに，$P(abb)$ は，

$$P(abb) = P_1(abb) + P_2(abb) = 0.084 + 0.0392 = 0.1232$$

となる．以上から，HMM $\boldsymbol{M}$は次の六つの組
で定義される．

- $\boldsymbol{S}$：状態の有限集合；$\boldsymbol{S} = \{s_i\}$
- $\boldsymbol{Y}$：出力シンボルの集合

- $\boldsymbol{A}$：状態遷移確率の集合；$\boldsymbol{A} = \{a_{ij}\}$；$a_{ij}$ は状態 s_i から状態 s_j への遷移確率，ここで

  $\sum_j a_{ij} = 1.$

- $\boldsymbol{B}$：出力確率の集合；$\boldsymbol{B} = \{b_{ij}(k)\}$；$b_{ij}(k)$ は状態 s_i から状態 s_j への遷移の際にシンボル k を出力する確率. ここで，

$$\sum_k b_{ij}(k) = 1(\text{離散 HMM})$$

$$\int_{-\infty}^{\infty} b_{ij}(k)dk = 1(\text{連続 HMM})$$

- π：初期状態確率の集合；$\pi = \{\pi_i\}$ ；π_i は初期状態が s_i である確率.

$$\sum_j \pi_j = 1$$

- $\boldsymbol{F}$：最終状態の集合

ここで注意を要するのは，出力 $\boldsymbol{Y}$ が有限集合の場合と無限集合の場合とがあることである．各々の場合について $\boldsymbol{B}$ の表現方法が異なってくる．前者の場合は，出力確率分布は離散的でノンパラメトリックである．シンボルは有限集合だからあらかじめ $\boldsymbol{B}$ を求めてテーブル化しておける．一方，後者の場合のそれは，ガウス分布のような(連続分布)で与えられる．連続分布の時は，シンボルは無限集合となり，あらかじめテーブル化しておけないので，出力シンボルが観測されるごとに $b_{ij}(k)$ を求めなければならない.

また y_i は時間的に等間隔に観測される場合と可変時間間隔ごとに観測される場合とがある.

4.3 Viterbi Algorithm

二つの単語 w_1 と w_2 に対応する HMM $\boldsymbol{M}_1$ と $\boldsymbol{M}_2$ を連結させて一つのHMM $\boldsymbol{M}_3$ を構成したとしよう. この時，$\boldsymbol{M}_1$ の最終状態から $\boldsymbol{M}_2$ の初期状態へナル遷移を設けて連結するものとしよう（あるいは，$\boldsymbol{M}_1$ の最終状態を $\boldsymbol{M}_2$ の初期状態に縮退させても良い）.

M_3 と入力文字列に対応する出力シンボル系列 $y_1, y_2, \cdots, y_T$ に対して前述のアルゴリズムで $P(y_1, y_2, \cdots, y_T|M)_3$ を求めることができる．これは一般に，

$$P(y_1, y_2, \cdots, y_T)|M_3 = \sum_t P(y_1, y_2, \cdots, y_t)|M_1 \times P(y_{t+1}, y_{t+2}, \cdots, y_T)|M_2$$

と表現できる．これは，M_1 の最終状態からほかの状態への遷移先は M_2 の初期状態にかぎられる事から明らかである．上式の関係から，$P(y_1, y_2, \cdots, y_T|M_3)$ の算出の際には，$w_1(M_1)$ にもっともよく対応するシンボル系列区間 $y_1, y_2, \cdots, y_t$ と $w_2(M_2)$ にもっともよく対応するシンボル系列区間 $y_1, y_2, \cdots, y_T$ を決定する事はできない．ところが複合語の単語分割のためにこの最適な単語境界時点 t を知りたい．これは次のような定義

$$P'(y_1, y_2, \cdots, y_T|M_3)$$

から求められる．

$$P'(y_1, y_2, \cdots, y_T|M_3) = \max_t P(y_1, y_2, \cdots, y_t|M_1) \times P(y_{t+1}, y_{t+2}, \cdots, y_T|M_2)$$

上式の定義は近似的ではあるが妥当な定義とも考えられる．言い換えれば y_t をHMM M_3 の唯一の状態遷移に対応させたことになる．この考えをすべての y_t に拡張すれば，式

$$P(y_1, y_2, \cdots, y_T) = \sum_x \prod_i P(x_i|x_{i-1}) \cdot P(y_i|x_{i-1}, x_i)$$

の代わりに，

$$P''(y_1, y_2, \cdots, y_T) = \max_x \{\prod_i P(x_i|x_{i-1}) \cdot P(Y_i|x_{i-1}, x_i)\}$$

を求めることになる．

これは式の {} 内の項を最大にする状態遷移系列上での確率で $P(y_1, y_2, \cdots, y_T)$ を近似するものである．これによって，任意の y_t は唯一の状態遷移に対応付けられる．この状態遷移系列を最適パスと呼ぶ．

最適パスとこのパス上での確率を求めるためには，動的計画法を用いる．このアルゴリズムは最初Viterbiによって提案された事から，Viterbiアルゴリズムと呼ばれている．$Q(i,t)$ を $P''(y_1, y_2, \cdots, y_t) = P(y_1, y_2, \cdots, y_t, \{x\})$ を最大にするパス（状態遷移系列），$f(i,t)$ をその確率と定義するとViterbiアルゴリズムは以下で与えられる．

Viterbiアルゴリズム

1. 初期化

   すべての状態 i に対して $\boldsymbol{f}(i,0) = \pi_i$

2. $t = 1, 2, \cdots, T$ に対して (3), (4) を実行.

3. すべての状態 i に対して (4) を実行.

4. $\hat{i} = \underset{j}{\operatorname{argmax}}\, \boldsymbol{f}(k,t)a_{ki}$

   $\boldsymbol{f}(i,t) = \max\{\boldsymbol{f}(\hat{j},t-1)a_{\hat{j}i} \cdot b_{\hat{j}i}(y_t)\}$
   $\boldsymbol{Q}(i,t) = \boldsymbol{Q}(\hat{j},t-1) \otimes \hat{j}$

5. $i = \underset{i \in F}{\operatorname{argmax}}\, \boldsymbol{f}(i,T)$

   $P''(y_1, y_2, \cdots, y_T) = P(y_1, y_2, \cdots, y_T, \boldsymbol{Q}(\hat{i},T)) = \boldsymbol{f}(i,T)$

ここで，$\otimes$ は状態遷移系列と状態を連結し，新たな状態遷移系列をつくるオペレータである．最適状態遷移系列は（最適パス）は $Q(\hat{i},T)$ で与えられる．HMM法による認識では，$P(y)$ をもちいる代わりに，この $f(\hat{i},T)$ を用いて最大値を与えるモデルのカテゴリーを認識結果とする方法もあり，同等の認識精度が得られている．

4.4　HMMを用いた単語分割

複合語の単語分割は文字間に単語の境界が存在するかしないかのどちらかの記号を割り当てる問題に一般かできる．

ここでもう一度HMMの定義を示す．

HMMは次の六つで定義できる．

- $\boldsymbol{S}$：状態の有限集合；$\boldsymbol{S} = \{s_i\}$

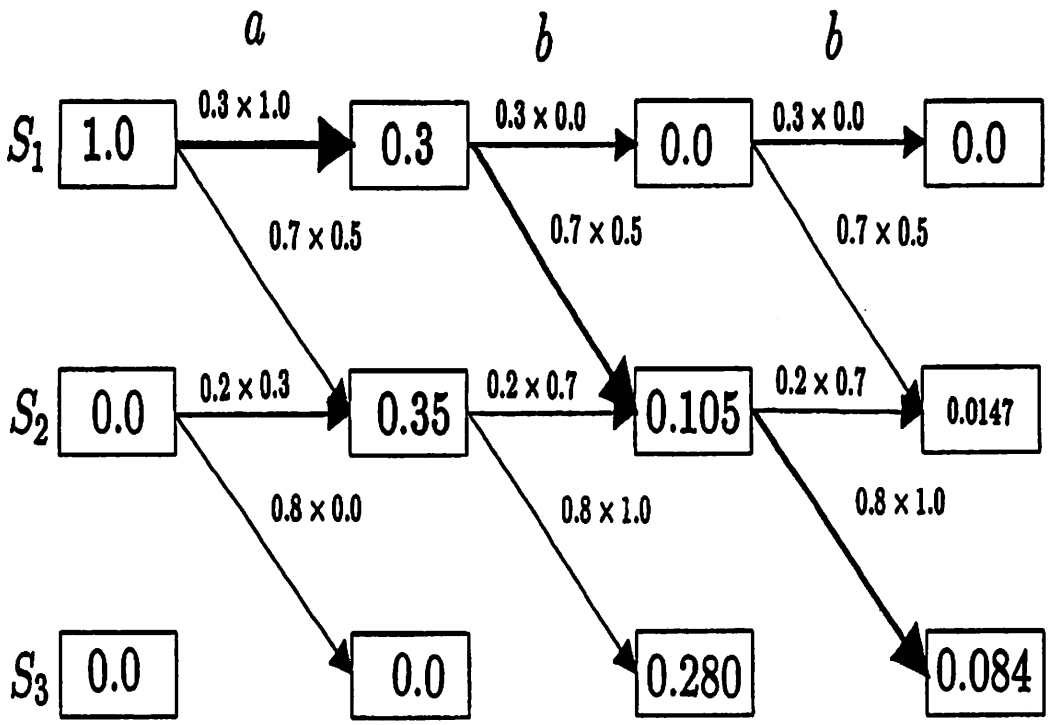

図 4.2: Viterbi アルゴリズム

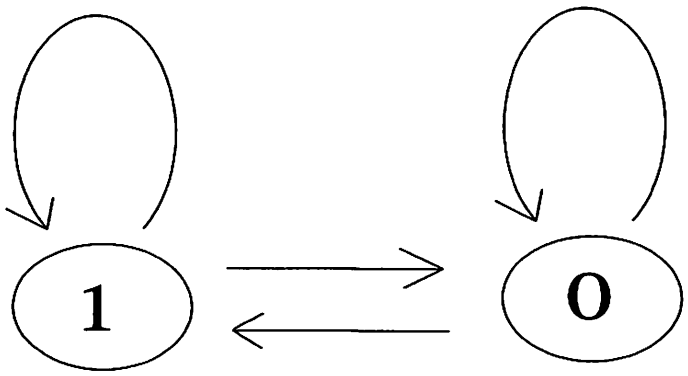

図 4.3: 実験で使った HMM

- $\boldsymbol{Y}$：出力シンボルの集合
- $\boldsymbol{A}$：状態遷移確率の集合；$\boldsymbol{A} = \{a_{ij}\}$；$a_{ij}$ は状態 s_i から状態 s_j への遷移確率，ここで

  $\sum_j a_{ij} = 1.$
- $\boldsymbol{B}$：出力確率の集合；$\boldsymbol{B} = \{b_{ij}(k)\}$；$b_ij(k)$ は状態 s_i から状態 s_j への遷移の際にシンボル k を出力する確率. ここで，

  $$\sum_k b_{ij}(k) = 1(\text{離散 HMM})$$

  $$\int_{-\infty}^{\infty} b_{ij}(k)dk = 1(\text{連続 HMM})$$
- π：初期状態確率の集合；$\pi = \{\pi_i\}$ ；π_i は初期状態が s_i である確率.

  $$\sum_j \pi_j = 1$$
- $\boldsymbol{F}$：最終状態の集合

図 4.3 に今回単語分割に利用した HMM M を示す．また図 4.4 に今回のモデルでの単語分割の例を示す.

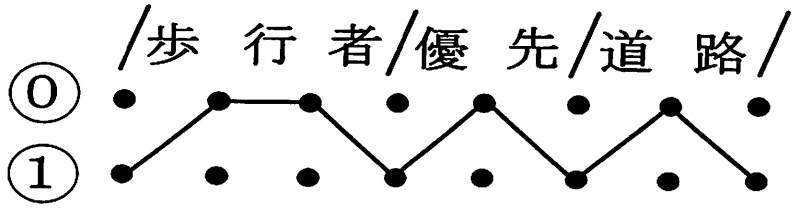

図 4.4: HMMによる単語分割

定義に従って示すと，状態の集合Sは単語の境界が存在する(1)としない(0)の2つの状態とした．また，出力シンボルの集合 Y として文字を考えた．ここで言う文字とは，普段使っている文字の事である．状態遷移確率Aは考慮せず，初期状態確率 π と最終状態の集合 F は { 1 } とおいた．あとは出力コストの集合 B を設定すればHMM Mが構築できる．

HMMでは出力シンボル系列がどのような状態をたどってきたかを動的計画法の一種であるViterbiアルゴリズムにより推定することができる．すなわち文字間に単語の境界が存在するかしないかが推定でき，単語分割が行える．

また，本当の定義では出力コストではなく出力確率なのだが，状態遷移の尤もらしさを比較できればよいので，得点やコストの形でも不都合はないので出力コストとした．

4.5　文字の出現確率の算出

B を状態 i から 状態 j へ移るとき文字 a が出現するコストとする．この場合，各文字 a に対する $B_{11}(a), B_{10}(a), B_{01}(a), B_{11}(a)$ を求めることが B の構築に対応する．

$B_{ij}(a)$ を求めるために，まず日経新聞CD-ROM'90の1年分の新聞記事を形態素解析し，複合語を取り出した．取り出した複合語の各文字を見ると，その文字の左右に単語境界があるかどうか，つまり状態がわかる．今，文字 a の左の状態が i，右の状態が j，である

ような個数を$C(i,j)$とする．例えば「自然言語」という単語は形態素解析により，/自然/言語/と分割されるので，$C(1,0,自),C(0,1,然),C(1,0,言),C(0,1,語)$にそれぞれ1が加えられる．$B_{ij}(a)$は以下の式で示した．

$$B_{ij}(a) = \log_2 \frac{C(i,\bar{j},a)+1}{C(i,j,a)+1}$$

ここで$\bar{j}$は状態jではない状態とする．つまり$j=0$なら$\bar{j}=1$であり，$j=1$なら$\bar{j}=0$である．

上記の形だけでも$\boldsymbol{B}$は構築できるが，精度をあげるためにbigramを利用する．まず$C(i,j,ab)$を考える．これは文字aの左の状態がi，右の状態がjで更に，文字aの次の文字がbである個数を示す．この値は先に取り出した複合語から得られる．また$B_{ij}(ab)$を状態iから状態jへ移るときに文字aが出現し，しかも文字aの次の文字がbであるときのコストとする．$B_{ij}(ab)$は以下の式で計算した．

$$B_{ij}(ab) = \log_2 \frac{C(i,\bar{j},ab)+1}{C(i,j,ab)+1}$$

$B_{ij}(a)$と$B_{ij}(ab)$から線形和をとり新たな$B'_{ij}(a)$を以下の様に定義した．

$$B'_{ij}(a) = \alpha \cdot B_{ij}(a) + (1-\alpha) \cdot B_{ij}(ab)$$

ここでは$\alpha = 0.3$とした．

文字の出現コストの計算例

以下に今回用いたデータから得られた文字の出現コストの計算例を示す．
ただし，Bi-gramでは，文字列の最初と最後に“＊”が出力されているものとする．また表4.1に文字の出現コストの例を示す．

A 00はAというパターンでありA 01はA/でありA 10は/Aというパターン．またA 11は/A/となっているパターンである．Cはデータの中でそれぞれのパターンが出力された回数．ここで“期”について計算例を示す．

$$B_{00}(\text{期}) = \log_2 \frac{931+1}{41907+1} = \log_2 \frac{932}{41908} = \log_2 0.022239191 = 5.49075$$

$$B_{01}(\text{期}) = \log_2 \frac{41907+1}{931+1} = -\log_2 \frac{932}{41908} = -B_{00} = -5.49075$$

$$B_{10}(\text{期}) = \log_2 \frac{12064+1}{17177+1} = 0.509734$$

$$B_{11}(\text{期}) = \log_2 \frac{17177+1}{12064+1} = -0.509734$$

表4.1:文字の出現コスト

a,i,j	C(i,j,a)	Bij(a)	a,i,j	C(i,j,a)	Bij(a)
期00	931	5.49075	有田00	15	0.169925
期01	41907	-5.49075	有田01	17	-0.169925
期10	12064	0.509734	有田10	122	-5.35755
期11	17177	-0.509734	有田11	2	5.35755
欠00	3	5.45121	伯＊00	0	6.61471
欠01	174	-5.45121	伯＊01	97	-6.61471
欠10	455	-3.97490	伯＊10	0	4.16993
欠11	28	3.97490	伯＊11	17	-4.16993
奏00	247	1.46786	回落00	0	3.45943
奏01	685	-1.46786	回落01	10	-3.45943
奏10	79	-6.32192	回落10	0	1.58496
奏11	0	6.32192	回落11	2	-1.58496
偕00	0	0.0	居西00	3	-2.0
偕01	0	0.0	居西01	0	2.0
偕10	17	-4.08746	居西10	0	0.0
偕11	0	4.08746	居西11	0	0.0

4.6　文字ベースHMMによる複合語単語分割の誤り修正

4.7　形態素解析が誤るパターン

一般に形態素解析の誤りは未知語によって生じる．日本語の単語はほとんどの場合，2文字あるいは3文字から構成される．このため未知語部分はほとんど以下のように過分割される．

2文字の未知語　○○　→　/○/○/

3文字の未知語　○○○　→　/○/○/○/ OR /○/○○/

また未知語ではないが，JUMAN の場合，以下の誤りのケースに共通する特徴として，先頭が1文字で分割されている点がある．つまり形態素解析で1文字単語と認識去れている部分は，誤りである可能性がある．

形態素解析　　正しい分割/○/○○～　　/○○/○～

一方，文字ベースの HMM は，このような局所的な単語分割に有効である．

例で示すと，「鈴木健四郎」の単語分割は，/鈴木/健四郎/であるはずだが，形態素解析では登録されていない「健四郎」が/健/四郎/と過剰に分割される．これは局所的な誤りといえる．

文字ベースの HMM では，この場合，“健” の前に単語境界があるという仮定の元で “健” と “四” あるいは “健” と “四郎” の接続の強さを測ることになる．“健” と “四” の間の接続は弱そうだが，/健/○/という単語分割のパターンより/健○ (健康，健次など) という単語分割のパターンが多いので，結果的に “健” と “四” の間には単語境界を置かずに正解が得られる．表 4.1 は 形態素解析の誤りの例である．

4.8　HMMが誤るパターン

文字ベースの HMM では長い文字列が一単語となるような場合に過剰に分割を行い単語分割が誤る．これは文字ベースが基本的に局所的な部分から，単語の境界があるかどうかを判断するために生じている．例えば，「リレハンメル五輪」の単語分割は/リレハンメル/五輪/ が正解であるが，HMM では，/リレハン/メル/五輪/と過剰に分割してしまう．これはリレハンメルという単語を局所的に単語境界を判断しているためである．形態素解析では辞書に登録されているので正しく分割できる．

表 4.1: 形態素解析の誤りの例

正解	形態素解析の誤り
/鈴木/健四郎/	/鈴木/健/四郎/
/延岡/市/	/延/岡市/
/苑田/	/苑/田/
/河口/利加/さん/	/河/口利/加さん/
/京都府/出身/	/京/都府/出身/
/小泉/順一郎/郵政相/	/小泉/順/一郎/郵政相/
/織田/大次郎/店長/	/織田/大/次郎/店長
/清涼/感/	/清/涼感/
/積極/度/	/積/極度/
/総合/力/	/総/合力/
/罰金/額/	/罰/金額/

またJUMANでは「一次産品共通基金」という文字列が一単語となっている．この文字列が言語的に一単語かどうかは問題ではない．想定しているアプリケーションにおいて一単語として扱いたいと考え，辞書に一単語として登録してあれば，一単語として解析すべきであろう．

さてこの「一次産品共通基金」を単語分割する場合は一単語とするのが正解である．一方「一次産品」を単語分割する場合は，/1/次産品/ と3つの単語列として解析するのが正解である．つまり “1” と “次” の間に単語境界が存在するかどうかを判断するには，“1” と “次” だけの局所的な関係では判断できない．さらにこれは “1” と “次産” の関係あるいは “1” と “次産品” の関係などに拡張しても単語境界が存在するかどうかは判断できない．

一方，一般の形態素解析は「一次産品共通基金」 を一単語,「一次産品」を /1/次/産品/ と解析するのは容易である．表 4.2 にHMMの誤りの一例を示す．

表 4.2: HMM での誤りの例

正解	HMM での誤った分割
/引き分け/	/引き/分け/
/やじうま/写真/大賞/	/やじ/うま/写真/大賞/
/高規格幹線道路/	/高/規格/幹線/道路/
/死者/数/	/死/者/数/
/三役/経験者/	/三役/経験/者/
/缶詰め/状態/	/缶/詰め/状態/
/振りそで/姿/	/振り/そで/姿/
/道交法/違反/	/道/交法/違反/

表 4.3: 修正するパターン

形態素解析	HMM
/○/○～/	/○○～/

4.9　相補的利用方法

前述したように，形態素解析と文字ベースの HMM のそれぞれの欠点は，互いに補えることが分かる．

本論文では，形態素解析と文字ベースの HMM を相補的に利用した複合語の単語分割を行う．まず形態素解析による単語分割と文字ベースの HMM による単語分割を並行して行い，両者の結果を比較する．基本的には形態素解析の結果を採用するが，以下のパターン部分は，その部分を HMM の結果に修正する．

注意として，/○/○～/ と/○○～/ の文字列の長さは等しく，しかも，単語分割のマークである “/” が一致している部分は，最初と最後の部分の 2 個所しか存在しない．例えば表 4.4 の解析結果の下線部分が上記のパターンに当てはまる．

表 4.4: パターンの例

HMM	形態素解析
/鈴木/健四郎/	/鈴木/健/四郎/
/延岡/市/	/延/岡市/
/苑田/	/苑/田/
/河口/利加/さん/	/河/口利/加さん/
/京都府/出身/	/京/都府/出身/
/永島/帝二/さん/	/永島/帝/二/さん/
/積極/度/	/積/極度/

4.10　実験

4.10.1　実験の手順

1. 毎日新聞の新聞記事 ('94 年度版) 1 年分から，複合語を取り出す
2. 形態素解析を用いて単語分割を行う
3. その結果を用いて文字の出現コストを算出する
4. 文字ベースの HMM で単語分割を行う
5. 形態素解析と文字ベースの HMM での単語分割の結果を比較し一致した物としなかった物にわける
6. 一致しなかった物の中で修正が生じた物としなかった物にわける
7. 修正が生じた物のなかで有効な修正だったものと悪影響だった物にわける.

表 4.5: 分割の修正

形態素解析	HMM
/延岡/市/	/延/岡市/
/奥谷/番/司/	/奥谷/番司/
/追/加点/	/追加/点/
/病/人食/	/病人/食/
/島/内/監督/	/島内/監督/
/若/田光/一/さん	/若田/光一/さん
/若/貴兄/弟/	/若貴/兄弟/
⋮	⋮
/小泉/順一郎/郵政相/	/小泉/順/一郎/郵政相/
/織田/大次郎/店長/	/織田/大/次郎/店長

4.10.2　実験の結果

8543 種類の複合語に対して実験を行い，形態素解析の結果と文字ベースの HMM の結果が一致した物が 7760 種類 (90.8 %)，一致しなかった物は 783 種類 (9.2 %) であった．一致しなかった物の中で修正が生じた物は 212 種類 (2.5 %) であった．一部を表 6.1 に示す．また修正が生じなかった物は 571 種類であった．この 571 種類に対しては，形態素解析の結果を採用する．

修正が生じた 212 種類について調べると，128 種類 (84.0 %) は正解であったが，34 種類 (16.0 %) は不正解であった．ただし，34 種類の中には，形態素解析による単語分割でも誤りがある場合や，正解が曖昧な物が 15 種類あった．純粋に形態素解析の方が正しかった物は 34 種類中 19 種類である．つまり修正による悪影響は，修正した物全体に対して実質 $\frac{19}{212} = 9.0\,\%$ であった．

図 4.5 は実験の結果を図にした物である．

「若田光一さん」,「若貴兄弟」の例は少し特殊である．一見，今回のパターンでないようだが次の段階で修正出きる

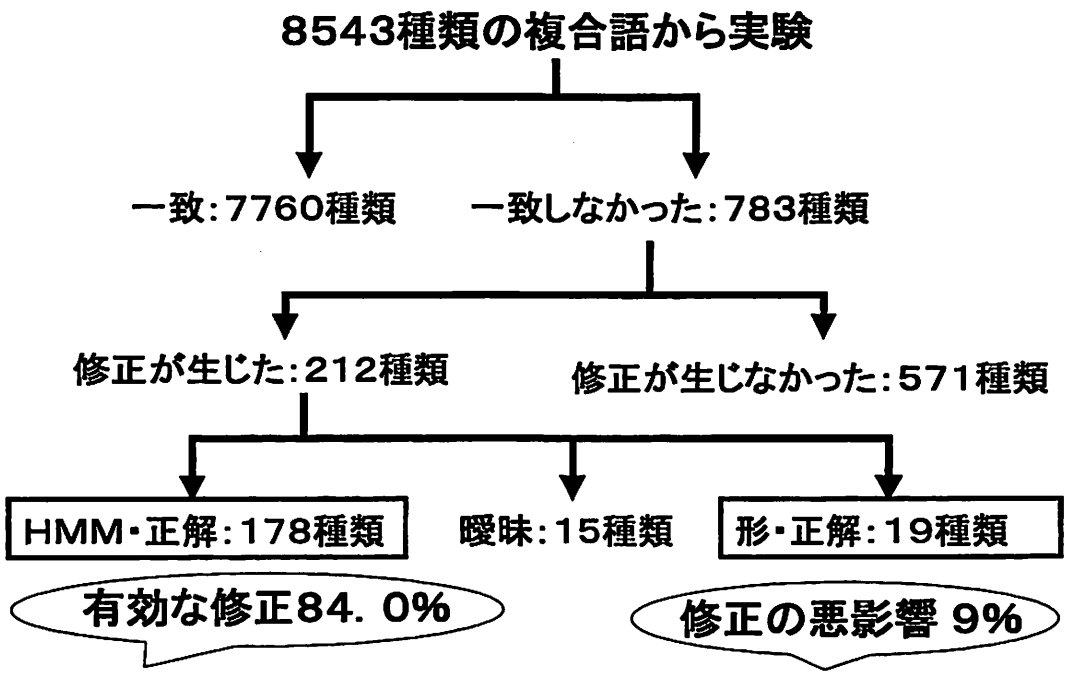

図 4.5: 複合語単語分割の実験結果

修正の段階（正解：/若貴/兄弟/）

/若/貴兄/弟/

↓

/若貴/兄/弟/

↓

/若貴/兄弟/

図

4.11　考察

ここで提案した手法は未知語に対処することが主な目的である．未知語に対処するための形態素解析を用いない従来手法では，ここで問題とした長い文字列からなる単語を過剰に分割するという欠点が残っている．本手法はこの欠点を克服している点で優れている．

また誤りのカバー率は次の様になった．ランダムに選んだ783種類の中で誤りは28種類あり，そのうち26種類(92.8 %)が本研究の相補敵利用で注目した誤りとなっていた．実験では84.0 %を修正できているので，結果的に誤りの78.0 %を修正できると考えられる．

問題点としては本研究のHMMでは形態素解析の結果をトレーニングデータとして利用してパラメータを得ているため，トレーニングデータの形態素解析の結果の誤りがHMMの精度を悪くしている原因の一つになっていると考えられる．

このことから今後は，今回の手法で得られた結果を用いて，形態素解析の結果を修正することを課題とする．

第5章　未知語抽出(その1)

ここでは，形態素解析の結果から過分割を検出する尺度を導入し，それを利用して未知語を自動抽出する方法を説明する．

本章では抽出する対象を2あるいは3文字からなる漢字とした．また，過分割を検出する尺度は文字列が単語になる確率から求めた．

5.1　背景

形態素解析の1つの課題として未知語処理がある．日本語の場合，未知語によって形態素解析の単語分割が誤り，形態素解析の精度悪化の一因となっている．そのためテキスト中の未知語を検出することは，形態素解析の精度向上に貢献する．また近年，情報抽出の前処理として固有表現抽出の重要性が指摘されている．固有表現抽出を行なう場合，人名，地名，会社名などは未知語としてあらわれやすいため，そこでも未知語を検出する技術が望まれている．

テキスト中のある文字を含むどのような部分文字列をテキストから取り出しても，その文字列が辞書に記載されていない場合，その文字を含む文字列が未知語になっていることがわかる．この種の未知語は記号やカタカナ文字で構成されることが多く，字種の変わり目を確認することで，ある程度，抽出は可能である．未知語抽出の困難な点は既知語から構成される文字列が未知語となる場合である．この種の未知語を検出するには，その単語列が単語列として正しいのか，全体として未知語となっているかの判定が鍵となる．

この判定を行なう場合に，複合語と未知語の区別にまず留意すべきである．例えば，「自然言語」という文字列を考えてみる．これは，「自然」と「言語」という2単語から構成される句だと考えれば，複合語である．一方，「自然言語」は1つの単語と考え，こ

の単語が辞書に記載されていなければ，未知語となる．1単語とみなせる複合語を取り出すことも未知語抽出と捉えられるが，ここではそのように捉えないことにする．1単語として扱える複合語は熟語や連語として捉え，未知語とは考えない．

この立場をとると，既知語から構成される未知語の文字列長は比較的短く，特に，漢字から構成される未知語の長さは2あるいは3であると考えることができる．本稿では漢字2文字あるいは3文字から構成される未知語を抽出の対象とする．当然，それ以外の字種から構成される未知語の検出も重要であるが，前述したように，カタカナや記号で構成される未知語は字種の変わり目からある程度判断できるので，ここでは対象としない．また平仮名を含む未知語の抽出も重要な課題ではあるが，新聞記事のような文書では，平仮名で表記される名詞の割合は低いと考え，ここでは対象としない．

ここで提案する手法は，まず形態素解析を行ない，ある単語列が設定したパターンになった場合に，その単語列が未知語の可能性があると考えて，未知語候補として取り出す．次にその単語列を構成する各単語の文字列が1単語としてあらわれる確率を定義する．今，単語 w の文字列が α のとき，$w = |\alpha|$ と書くことにする．次に，文字列 α が単語としてあらわれる確率を $P(\alpha)$ と書くことにする．そして単語列 $|\alpha||\beta|$ が未知語となる尺度 $m(\alpha\beta)$ を以下によって定義する．

$$m(\alpha\beta) = 1 - P(\alpha)P(\beta)$$

この値がある閾値を越えた場合に，未知語として検出する．$P(\alpha)$ の計算は，コーパス（トレーニングデータ）から，以下の式で求めておく．

$$P(\alpha) = \frac{c(|\alpha|)}{c(\alpha)}$$

ここで，$c(|\alpha|)$ は単語 $|\alpha|$ のコーパス中の頻度であり，$c(\alpha)$ は文字列 α のコーパス中の頻度である．

本手法は文字列 $\alpha\beta$ が未知語かどうかの判断に，コーパス中の $\alpha\beta$ の頻度を利用していない．このため，コーパス中で低頻度の未知語も検出可能という長所がある．

なお，本稿で用いた形態素解析は「茶筌 (version 2.0b)」であることを注記しておく．

5.2　パターンの分類

本章での未知語抽出の対象は2文字あるいは3文字の漢字である．まず始めにここで以下の仮定をおく．

仮定 漢字列 α が未知語であった場合，α を含む文を形態素解析すると α の前と後に単語分割の切れ目が生じる

この仮定に反する例は存在するが，ほとんどの場合この仮定は成立している．

上記の仮定が成立しているとすると，漢字文字 a と b の 2 文字列 ab が未知語であった場合，形態素解析による ab の単語分割は以下のようになる．

- パターン0： $|ab| \rightarrow |a|b|$

また漢字文字 a , b , c からなる 漢字列 abc が未知語であった場合，形態素解析による単語分割の結果は以下の 3 つの場合のいずれかになる．

- パターン 1 ： $|abc| \rightarrow |a|b|c|$
- パターン 2 ： $|abc| \rightarrow |a|bc|$
- パターン 3 ： $|abc| \rightarrow |ab|c|$

これらのことから，まずはじめに形態素解析を行ない，単語分割の結果が上記のパターンにあてはまった漢字列を 未知語の候補として取り出すことができる．

5.3　未知語を検出する尺度

5.3.1　過分割を判定する尺度

文字列 α が 1 単語になる確率を次のように定義する．

$$P(\alpha) = \frac{c(|\alpha|)}{c(\alpha)}$$

ただし，$c(\alpha)$ はコーパス中に文字列 α が出現した回数であり，$c(|\alpha|)$ は文字列 α が1単語として出現した回数である．これは，文字列 α があらわれたときに α が1単語としてあらわれる確率を表している．

辞書に未登録である α が1単語になりやすいということは α が未知語になりやすいということである．このことから，文字列 α が未知語かどうかは $P(\alpha)$ の大きさから判断することができる．しかし，α が未知語である場合，文字列 α が1単語として出現する回数 $c(|\alpha|)$ が0となるため，α は未知語でないと判断されてしまう．

そこで，本稿では未知語判定の方法として，確率 $P(\alpha)$ の代わりに，文字列 α が未知語になる尺度を用いる．

文字列 α が 形態素解析により $|\beta|\gamma|$ と分割されたとする．文字列が単語となる確率が前後の文字列に依存しないと考えれば文字列 α が 文字列 $|\beta|\gamma|$ に分割される確率は $P(\beta)P(\gamma)$ と近似することができる．

このことから，単語列 α が1単語となる尺度 $m(\alpha)$ を以下のように決めた．

$$m(\alpha) = 1 - P(\beta)P(\gamma)$$

この尺度は値が大きいほど，その文字列が分割されない可能性が大きい．つまり未知語である可能性が大きい．

またパターン3の場合については，m を以下のように定める．

$$m(abc) = \max\{(1 - P(a)) * m(bc), (1 - P(c)) * m(ab)\}$$

また，2文字列 ab が単語になる確率 $P(ab)$ は

$$P(ab) = \begin{cases} 1 & (c(ab) = 0 \text{ のとき}) \\ \dfrac{c(|ab|)}{c(ab)} & (c(ab) \neq 0 \text{ のとき}) \end{cases}$$

とする．

5.3.2　適用する順序

前述したパターンについて尺度 m を適用することで，未知語を抽出できる．しかし，尺度の適用には曖昧性が存在する．例えば，漢字列 $abcdefg$ の形態素解析による単語分割の

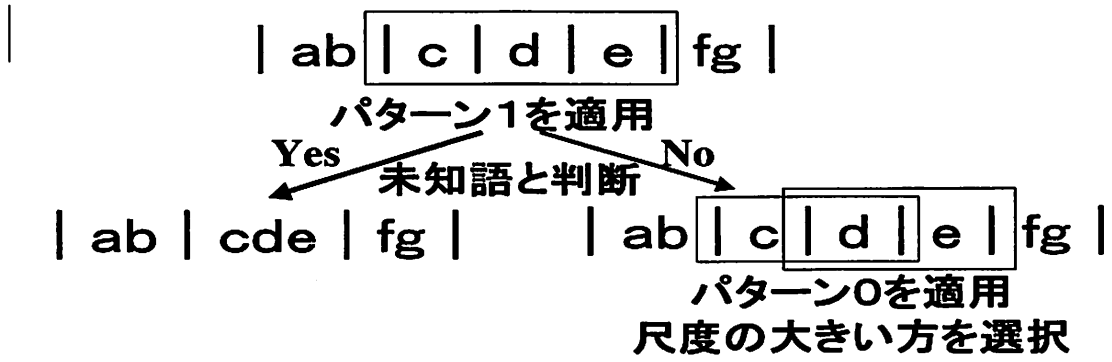

図 5.1: 尺度を適用する順序

結果が $|ab|c|d|e|fg|$ だとしよう．この場合，$|ab|c|$，$|c|d|e|$，$|c|d|$，$|d|e|$，$|e|fg|$ が未知語の候補となる．ここでもし $|c|d|e|$ が未知語だと判断されれば，その他の $|ab|c|$，$|c|d|$，$|d|e|$，$|e|fg|$ は未知語になることはない．

このような曖昧性に対して，ここでは，パターンによる尺度の適用順序を設けた．具体的には以下の順序を設定した．

パターン 1 --> パターン 0 -->

パターン 2 --> パターン 3

上記の例でいえば，まずパターン 1 の $|c|d|e|$ が未知語かどうかを試す．未知語と判断できれば，単語分割は $|ab|cde|fg|$ に変更する．そして，パターン 0，パターン 2，パターン 3 を順次試す．(図 5.1)

また，同じパターンでも適用できる場所が複数存在する場合も考えられる．上の例では $|c|d|$ と $|d|e|$ がそれにあたる．このような場合は，尺度 m の大きい方を選択する．

5.4　実験

毎日新聞 '94 年度の新聞記事 1 年分を形態素解析にかけ単語分割を行なう．これをもとに 2 文字あるいは 3 文字の漢字列 α が単語になる確率 $P(\alpha)$ を算出しておく．

表 5.1: 未知語

有煙，和魂，硬材，和慶，彌弋，作陶
廣場，星島，彩挺，艶麗，羅湖，老子
祥造，学燈，画期，靖英，普選，深銘
仕梅，女衆，三連星，杉乃井，伊那谷

表 5.2: 本手法で抽出できた未知語

和魂，硬材，和慶，作陶
星島，彩挺，老子，祥造
画期，普選，深銘，仕梅
女衆，杉乃井

5.4.1　新聞記事からの未知語抽出

毎日新聞 '95 年度の最初の 1,000 文を対象に 未知語の抽出実験を行なった．最初に手作業により 2 文字列 20 種類，3 文字列 3 種類の未知語をあらかじめ発見しておいた．発見したものを表 5.1 に示す．

また，あらかじめ実験により適当な閾値を調べておいた．その結果ここで決定した閾値は，パターン 1 が 0.90，パターン 0，2，3 が 0.98 になった．

表 5.1 に挙げられている未知語で，本手法で抽出できたものを表 5.2 に示す

2 文字の漢字からなる未知語において，本手法により抽出した種類数 18 種類のうち，有効な抽出は 13 種類であったので，正解率は $\frac{13}{18} = 0.722$，再現率は $\frac{13}{20} = 0.650$，F 値は

表 5.3: 未抽出の未知語

有煙，廣場，彌弋
艶麗 羅湖，学燈，靖英
三連星，伊那谷

表 5.4: 誤抽出の未知語

千件，対中，一本
六百社，二十代，朗読術

表 5.5: 実験結果

	抽出数	正解数	正解率	再現率	F 値
2 文字	18	13	0.722	0.650	0.684
3 文字	8	1	0.125	0.333	0.182

$\frac{2 \times 0.722 \times 0.650}{0.722 + 0.650} = 0.684$ となった．また 3 文字の漢字からなる未知語では，本手法により抽出した未知語 8 種類のうち，有効な抽出は 1 種類であったので，正解率は $\frac{1}{8} = 0.125$，再現率は $\frac{1}{3} = 0.333$，F 値は $\frac{2 \times 0.125 \times 0.333}{0.125 + 0.333} = 0.182$ となった．

5.4.2　未知語の作成による未知語の抽出

前述した実験のように既存の文書から未知語を抽出する場合，3 文字の漢字からなる未知語の出現確率が低いために，再現率を図ることが困難になる．そこで本稿では 1 つの工夫として，3 文字の漢字からなる未知語を強制的に作成し，その未知語が抽出できるかどうかを調べる実験を行なった．

そのために，まず毎日新聞 '94 年度の 1 年間分の新聞記事から 3 文字漢字の単語のうち，頻度が 1 であるものを収集し，それらからランダムに 100 種類を選んだ．同時に，選ばれた単語を含む文も取り出しておいた．

次に上記の 100 単語を「茶筌」の辞書から取り除いて，「茶筌」を再構築した．再構築された「茶筌」では，これらの 100 単語は未知語となる．この再構築された「茶筌」と本システムを用いて，取り出しておいた文から，上記 100 個の未知語が抽出できるかどうかを調べた．パターン別に分けた結果を表 5.5 に示す．ただし，表中パターンの欄で「その他」は，3 章で述べた仮定に反しているものを示している．

表 5.6: 未知語の作成による未知語の抽出

パターン	正解の個数	抽出数	正しい抽出数	正解率	再現率	F 値
パターン1	21	14	11	0.786	0.523	0.628
パターン2	30	12	7	0.583	0.233	0.333
パターン3	45	12	10	0.833	0.222	0.351
その他	13	—	—	—	—	—

表 5.7: 未知語のコーパス中での頻度

硬材：0，彩挺：0，深銘：0
仕梅：0，祥造：1，女衆：2
杉乃井：2，老子：3，普選：5
和魂：7，和慶：14，星島：15
作陶：17，画期：245

これよりパターン1が最も良い結果となった．

5.4.3　低頻度の未知語抽出

本手法の長所として低頻度の未知語を抽出できる点がある．そこで，どの程度，低頻度の未知語を抽出できるか検証するため，4.1章 で抽出できた未知語の トレーニングコーパス中での頻度を調べた．その結果を表 5.7 に示す．

このことから，本手法でコーパス中の出現頻度が低い未知語が抽出可能であることが確認できた．

5.5　考察

本章では 2 文字あるいは 3 文字の漢字からなる未知語を抽出の対象とした．パターン 0 とパターン 1 は比較的良い結果が得られたが，パターン 2 とパターン 3 は良い値を得られなかった．これらの精度が悪かった理由として，パターン 2 と 3 で使われている，2 文字の漢字列が単語になる確率の扱いや，尺度の決め方に問題があると思われる．本手法では 2 文字 の漢字列 ab が単語になる確率を $P(ab) = \frac{c(|ab|)}{c(ab)}$ として与えたが，$\frac{c(|ab|)}{c(ab)} \simeq 1$ となっている場合が多い．実験 4.2 の結果では

- /放浪/癖/ : P(放浪) $= 70/71 = 0.986$
- /下/山田/ : P(山田) $= 871/957 = 0.910$
- /計算/尺/ : P(計算) $= 1082/1114 = 0.971$
- /大/番組/ : P(番組) $= 2669/2681 = 0.996$

などがこれにあたる．

これは，ab が出現しているとき，それらがほとんど 1 単語として出現していることを示している．これを $|ab|c|$ の例で尺度に適用した場合，$P(ab)$ が 1 に近い値なので

$$m(abc) \simeq 1 - P(c)$$

となる．これは c が 1 単語にならない確率のことであり，ab のことは考慮せずに未知語かどうかを判断している．

しかし，上の例を見ればわかるように，本手法で未知語と判定しなかったパターン 2 とパターン 3 の漢字列は複合語と考えることができるものが多い．つまり，パターン 2 とパターン 3 において，本手法が未知語ではないという判定は妥当だと思われる．

第6章　未知語抽出(その2)

本章では，一般的な文字列を対象とした未知語抽出の方法を説明する．過分割を検出する尺度は既存の尺度を改良したものを用いた．

6.1　背景

ある文字列が過分割であるかが分かれば，未知語を検出できるということは前章で説明した．文字列が過分割かどうかを判定する尺度として他に文献[11]の手法がある．この手法では，与えられた文字列が過分割される場合と分割されない場合とで確率を比較し，分割されない確率が高いほど大きな値を取る尺度を利用して，過分割かどうかの判定をしている．

具体的には，与えられた文字列を $S = a_1, a_2, ..., a_k, b_1, b_2, ..., b_l$ とし，S の二つの部分文字列(形態素)を $A = a_1, ..., a_k$ と $B = b_1, b_2, ..., b_l$ とするとき，

$$L(A, B)$$

は以下で述べるように，文字列 S の形態素 A と B への切れにくさを表現している．ただし，$< w >$ と $< /w >$ は，それぞれ，形態素の前後に付ける区切り記号であり，$P_r(...)$ は，文字列の生起確率である．この尺度 $L(A, B)$ は，文字列 $S = AB$ の形態素 A と B への切れにくさを表している．すなわち，$L(A, B)$ が大きいときには，S は A と B には分割されがたい．すなわち，a_K と b_1 の間の分割は過分割である可能性が高い．

基本的に本手法もこのような方法を取るが，尺度の定義に違いがある．内山さんの尺度では，文字列Aと Bの切れにくさに文字列 AB の頻度を必要としている．このため，トレーニングコーパス中にABが出てこなかった場合について，対応していない．

未知語というものは，元来低頻度なものであるため，トレーニングコーパス中に出てこない単語が未知語としてテストデータ中に出てくる可能性は高い．本手法では，トレーニングコーパス中の AB の頻度は使っていない．このため，トレーニングデータの中で低頻度なものに対しても検出が可能であるため，有効な手法といえる．

また，前章で紹介した方法も過分割検出という点では同じだが対象となるものが2あるいは3文字からなる漢字列と限定されている．未知語というものが大半が漢字列であり，なおかつその長さは2あるいは3文字であるものが殆んどであるというのは確かであるが，未知語になりやすい固有表現で出てくる物のなかには平仮名やカタカナだけのものや漢字や仮名が混ざっている物も存在する．本手法では，対象となる文字列の種類や長さは限定しない．その点で本手法のほうが優れているといえる．また，$\alpha\beta$ が未知語であった場合その頻度は極端に低くなる．これを補うため前章の方法では文字列 $\alpha\beta$ が未知語になるかどうかを $P(\alpha) * P(\beta)$ で近似した．しかしこの式は α ， β が続けて出てくるということを何も考慮していない．本手法では，α の最後の文字の字種を使うことによりその欠点を補った．

6.2　尺度の定義

ある文字列が分割されるかどうかの尺度 L を

$$L = \frac{P(/\alpha\beta/)}{P(/\alpha/) * P(/\beta/)}$$

と定義する．これは，与えられた文字列が分割される場合と分割されない場合とで確率を比較し，分割されない確率が高いほど大きな値をとる尺度である．そのため，この尺度の値が大きいような分割をされている文字列は，誤った分割 (過分割) をされている可能性が高い．

ここで問題となるのは分子の $P(/\alpha\beta/)$ である．文字列 α ， β がそれぞれ $\alpha = a_1, ..., a_k$ ， $\beta = b_1, ..., b_l$ の文字で構成されているとすると，この式は

$$P(/\alpha\beta/) = P(a_1, ..., a_k, b_1, ..., b_l)$$

と表すことができる．この式を計算する場合普通は n-gram(通常2から3グラム) を用い

る．つまり，$P(c_1,...,c_k)$ の値は

$$P(c_1,...,c_k) = P(c_1)\prod_{i=1}^{k+1} P(c_i|c_0,...,c_{i-1})$$

$$= P(c_1)\prod_{i=1}^{k+1} P(c_i|c_i-n+1,...,c_{i-1})$$

となり，対象となる文字列の n-gram の頻度を必要とする．しかし，$\alpha\beta$ が未知語である場合その頻度は小さくなる．つまり a_k, b_1 周辺での n-gram の頻度が 0 に近くなってしまう．これでは正確な値を得ることが出来ないため．何らかの近似を行う必要がある．本手法では $P(/\alpha\beta/)$ を次で説明する方法を用い近似した．

まず，$P(/\alpha\beta/)$ は以下のように変形できる．

$$P(/\alpha\beta/) = P(/\alpha) * P(\beta/|/\alpha)$$

この式で未知語の際の頻度が問題となるのは $P(\beta/|/\alpha)$ である．これを詳しくみていくと，

$$P(\beta/|/\alpha) = P(b_1,...,b_l,/|/,a_1,...,a_k)$$

ここで $/,a_1,...,a_k$ の情報を α の最後の文字 a_k の字種 (J_n) の情報で近似する．$P(\beta/|/\alpha)$ は以下のようになる．

$$P(\beta/|/\alpha) \simeq P(b_1,...,b_l,/|J_n) = P(\beta|J_n)$$

また，

$$P(y|x) = \frac{P(x \bigcap y)}{P(x)}$$

であり，一般的に文字列 x，y において $x \bigcap y = xy$ これを利用すると先ほどの式は

$$P(\beta|J_n) = \frac{P(J_n\beta/)}{P(J_n)}$$

と表すことができる．この式を近似して

$$\frac{P(J_n\beta/)}{P(J_n)} \simeq \frac{P(J_n) * P(b_1|J_n) * P(b_2,b_3,...,b_l,/|b_1)}{P(J_n)}$$

$$= P(b_1|J_n) * P(b_2,b_3,...,b_l,/|b_1)$$

を得られる．もう一度，条件付き確率の式を利用するとこの式は

$$P(\beta/|/\alpha) = P(b_1|J_n) * P(b_2, b_3, ..., b_l, /|b_1) = P(b_1|J_n)\frac{P(\beta/)}{P(b_1)}$$

と表せる．この値を $P(/\alpha\beta/) = P(/\alpha) * P(\beta/|/\alpha)$ に代入すると

$$P(/\alpha\beta/) = P(/\alpha) * P(b_1|J_n)\frac{P(\beta/)}{P(b_1)}$$

となり，これを尺度 L に適用すると

$$L = \frac{P(/\alpha\beta/)}{P(/\alpha/) * P(/\beta/)} = \frac{P(/\alpha) * P(\beta/)}{P(/\alpha/)P(/\beta/)} * \frac{P(b_1|J_n)}{P(b_1)}$$

が求められる．この式は $\alpha\beta$ における， a_k, b_l の頻度を利用していない．このため，この尺度を使えば未知語などコーパス中での頻度が低い単語の過分割を検出ができる．

6.3　実験

予め毎日新聞 94 年分を形態素解析にかけ，unigram bigram trigram 字種 unigram 字種 bigram の頻度を求めておく．

下に unigram ， bigram ， 字種 unigram ， 字種 bigram についての例を示す．

/池谷/です/ というデータがあった場合，

*****unigram	****字種 unigram
/	{切れ目}
池	{漢字}
谷	{漢字}
/	{切れ目}
/	{切れ目}
で	{平仮名}
す	{平仮名}
/	{切れ目}

表 6.1: 未知語

老子，有煙，がんじがらめ，普選法
和魂，硬材，和慶，彌弋，作陶
廣場，星島，彩挺，艶麗，羅湖，
祥造，円高，円安，けんさん，たけくらべ
学燈，画期，靖英，普選法，深銘
仕梅，女衆，三連星，杉乃井，伊那谷

*****bigram	****字種 bigram
/池	{切れ目}池
池谷	{漢字}谷
谷/	{漢字}/
/で	{切れ目}で
です	{平仮名}す
す/	{平仮名}/

本実験は毎日新聞 95 年の記事 1000 文を過分割検出を用いて行なった．

あらかじめ，手作業で過分割だと思われるものを列挙しておいた．表 6.1 に一部を示す．

実験では，尺度の値が 25 以上の値のものを未知語として検出するようにした．実験の結果本システムが正解であったものが 43 種類，間違いであったものが 87 種類であった．間違いのうち，誤抽出であったものが 33 種類，未抽出のものは 42 種類であった．それらの結果の一部を表 6.2 ，表 6.3，表 6.4 に示す．その他に複合語との区別が曖昧であるものが 10 種類存在した．それらの例を表 6.5 に示す．また，誤抽出の内/二/種類/ など数字を含んでいたものが 10 種類も含まれていた．

表 6.2: 正解

老子，祥造，伊那谷，一部，三十代，
英国，普選法，杉乃井，白々，有煙
和魂，あきやま，シェーンベルク，選前
選後，硬材，星島，廣場，よこお，楠部

表 6.3: 誤抽出

りつつ，のつじつま，なしたたか
十八度，一度，なみなみとつい，的画期
わしら，おぼろげ，果たそう

表 6.4: 未抽出

数年，えと，靖英，円高，円安，ざんげ，
和慶，交響楽団，艶麗，作陶，羅湖，
たけくらべ，彩挺，けんさん，ノースヨーク

表 6.5: 曖昧な結果

社会党，音楽界，朗読術，駐在員
香港島，経済界，顧問業，観測子

6.4　考察

6.4.1　近似の正当性

本手法では $P(b_1,...,b_l,/|/,a_1,...,a_k)$ を $P(b_1,...,b_l,/|J_k)$ (ただし，J_k は文字列 a の k 番目の文字の字種) として近似した．ここでは，その近似がどれくらい信憑性があるのか検討してみた．

形態素解析をかけた新聞記事のなかから n-gram を使って $P(/\alpha\beta/)$ を計算をする．トレーニングデータもこの形態素解析を使い構築した物であるため a_k, b_l が連続して現れる頻度は殆んど 0 である．このなかから値が 0 でないものをランダムに 19 個取り出す．

次にこの 19 個を本手法の近似である

$$P(/\alpha\beta/) = P(/\alpha) * P(b_1|J_n)\frac{P(\beta/)}{P(b_1)}$$

を用い計算する．それぞれの値をグラフにしたのが図 6.1, 表 6.2 である．

これを見ると，本手法の近似のグラフと n-gram による計算のグラフはほぼ同じ形を描いている．このことから，本手法の近似は間違っていないことがわかる．

6.4.2　未知語への対応

本手法と，関連研究のとの違いは，テストコーパス中で低頻度のものが抽出できるかどうかである．関連研究の方法では

$$P(/\alpha\beta/) = P(a_1,...,a_k,b_1,...,b_l)$$

の計算に直接 n-gram を用いていた．つまり a_k, b_1 の周辺の n-gram の頻度を使用する．そのため，テストコーパス中で低頻度な単語を正しく判定することが保証されない．例えば本実験での「硬材 (頻度:0)」も

$$P(/\text{硬材}/) = P(/) * P(\text{硬}\,|/) * P(\text{材}\,|/\text{硬}) * P(/|\,\text{硬材})$$

となり，「硬材」という単語自身の頻度を使用しているため，該当部分での P の値が 0 となり，過分割の判定が正確でなくなる．

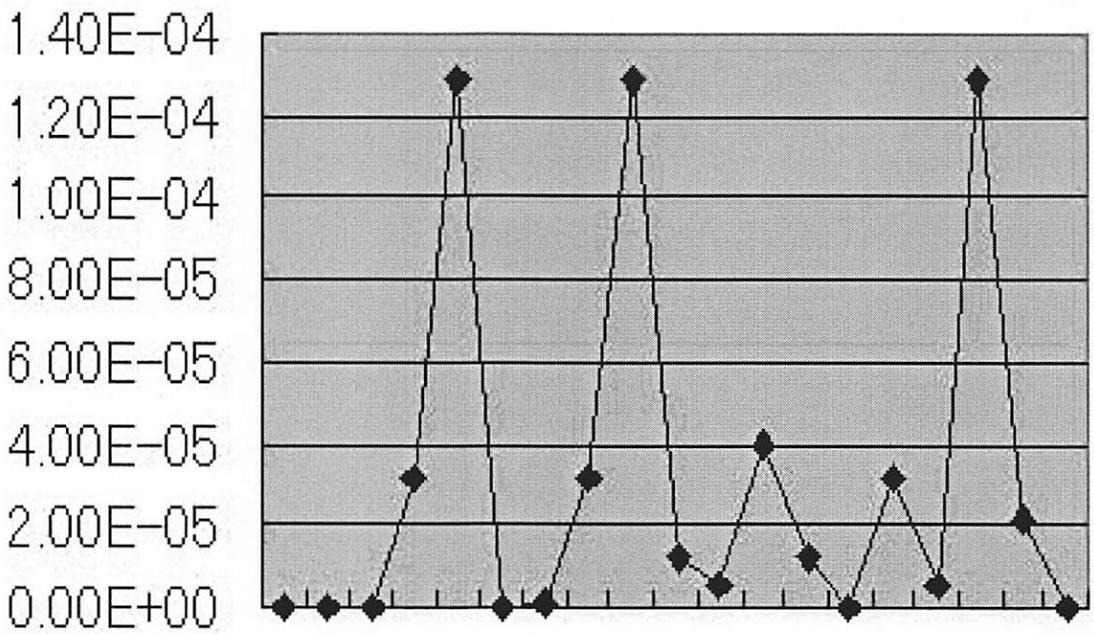

図 6.1: 実際の値

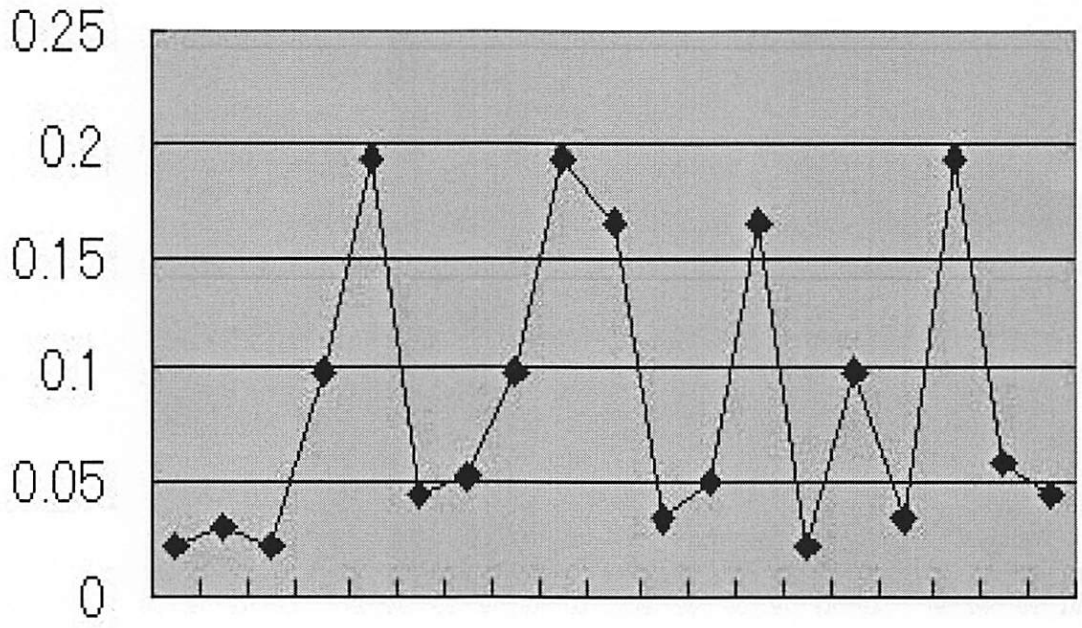

図 6.2: 近似による計算結果

本手法では，$P(/\text{硬材}/)$ の計算は

$$P(/\text{硬材}/) = P(/\text{硬}) * P(\text{材}/|\,\text{漢字})$$

としているため,「硬材」の頻度は使っていない．このため，頻度がたとえ0であっても抽出可能という利点を持つ．

第7章　まとめ

本論文では，統計的手法を使い形態素解析の分割誤りを検出する手法を導入しその技術を応用して複合語の単語分割，未知語の検出を行う方法を紹介した．

複合語の単語分割をおこなうために未知語に強い文字ベースHMMと長い文字列を1単語として認識するのが得意な形態素解析を相補的に利用した．その結果，形態素解析単独で複合語単語分割を行うよりも高精度の複合語単語分割システムを構築することができた．

未知語の検出を行うために過分割を検出する尺度を2つ導入した．一つは文字列が単語になる確率を用いたものであり，対象は2あるいは3文字の漢字列からなる未知語であり，もう一つは従来からある尺度の欠点を取り除いたもので，対象は限定されない．それぞれ，従来手法では難しかった低頻度の未知語に対応することができた．

複合語の単語分割では形態素解析の誤りを検知する仕組みに誤りのパターンを用いたが，当然今回注目したパターン以外の誤りも存在する．実験で利用した複合語からランダムに783種類取り出し，それらに対する形態素解析による単語分割結果を調査した．結果，単語分割の誤りは28種類(92.8％)あり，そのうち26種類が本論文で注目したパターンになっていた．実験では84.0％を正しく修正できているので，結果的に誤りの78.0％を本手法で修正できると考えられる．

2あるいは3文字の漢字からなる未知語の検出に関しては，3文字からなる未知語の再現率が低いかった．未検出の未知語を調査した結果，その多くは1単語とみなすか，複合語とみなすかが微妙な単語が多かった．これは単語の定義の問題にも依存する．

一般的な未知語の検出に関しては，文字列 $\alpha\beta$ の頻度を (α の最後の字種)β で近似した．単語2gramをランダムに19個取り出して，各々の場合の分布の類似性をみることにより，妥当性も確認できた．

2つの未知語検出の研究を通して長さが2文字までの未知語において良い結果が得られ

たが3文字以上では検出精度は低くなってしまった.

3文字以上の未知語の検出精度が悪かった原因は，複合語を未知語として扱ってしまったためである.「大金融」という複合語がその一例である．この単語分割は「/大/金融/」が正解であるが，これを未知語としてしまった．この判断だと「大金融」は1単語となってしまう.

この問題は，複合語の単語分割システムを組み合わせることで改善できると考えられる.具体的には，文字ベースHMMと形態素解析で単語分割が一致したものは複合語である可能性が高いためそれを未知語判定の対象から外すということで実現する.

謝辞

本研究の遂行及び論文の作成において多大な御助言及び御指導を賜った新納 浩幸 教官 (茨城大学工学部システム工学科) に深い感謝の意を表します.

また，その他様々な助言を頂いた主査の 石黒美佐子 教官，実験データの整理や精神面など色々な部分でで支えてくれた荒井亮一君 (茨城大学大学院修士課程，堤研究室)，河野靖明君 (98 年度卒業),「なべの輪」のみなさん，プログラムや論文の画像作成を手伝って頂いた高橋 篤史君，研究室でお世話になった星 秀明君，徳永 崇君 (2000 年度)，甲田 英史君，野沢 洋一君，萩原 裕一君 (99 年度)，その他研究の機会を与えてくれたすべての人々に感謝致します.

また，本研究で利用したコーパスおよび評価文は，日本経済新聞 CD-ROM '90 版 と毎日新聞 '94 および '95 版から得ています. 利用を許可していただいた日本経済新聞社と毎日新聞社に深く感謝致します.

参考文献

[1] 池谷昌紀, 新納浩幸 : "文字ベースの HMM による複合語単語分割の誤り修正", 言語処理学会, 第 5 回年次大会発表論文集,pp.494-497 (1999).

[2] 池谷昌紀, 新納浩幸 : "文字列が単語になる確率を用いた未知語抽出", 情報処理学会自然言語処理研究会, 135-7,pp.49-54 (2000).

[3] 大内和弘 : "既存知識を用いた HMM のパラメータ推定", 茨城大学工学部システム工学科 '96 年度卒業論文 (1997).

[4] 小田裕樹, 北研二 : "PPM モデルによる日本語単語分割.Technical Report NL-128-2", 情報処理学会自然言語処理研究会 (1998).

[5] 北 研二, 中村 哲, 永田昌明 : "音声言語処理-コーパスに基づくアプローチ", 森北出版株式会社 (1996).

[6] 坂本慶行, 石黒真木夫, 北川源四郎 : "情報量統計学", 共立出版株式会社 (1993).

[7] 中川聖一郎 : "確率モデルによる音声認識", 電子情報通信学会 (1988).

[8] 森脇 敏, 河部 恒, 辻井潤一 : "辞書を使わない日本語専門用語の自動分割", 言語処理学会, 第 2 回年次大会発表論文集, pp.273–276 (1996).

[9] 山本幹雄, 増山正和 : "品詞区切り情報を含む拡張文字の連鎖確率を用いた日本語形態素解析", 言語処理学会, 第 3 回年次大会,pp.421-424 (1997).

[10] 松本裕治, 他 : "日本語形態素解析システム「茶筌」version 2.0 使用説明書 第二版", 奈良先端科学技術大学院大学 松本研究室 (1999).

[11] 内山将夫 : "形態素解析結果から過分割を検出する統計的尺度", 言語処理学会,Vol.6,No7,pp-3-28 (1999).

[12] 永田昌明 : "確率モデルによる日本語処理に関する研究", 京都大学博士論文 (1999).

[13] 久光 徹, 丹羽 芳樹 : "書き換え規則と文脈情報を用いた形態素解析後処理", 情報処理学会自然言語処理研究会,98NL-126 (1998).

付 録A　プログラムソースリスト

```
/*=========================================================

/*************************/
/****     shaku.h   *******/
/*************************/
#ifndef __SHAKU_H__
#define __SHAKU_H__


#define UNIGRAM 89734488
#define BIGRAM 65102698
#define TRIGRAM 40470908

//Include Header Files
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include<fcntl.h>
#include<gdbm/ndbm.h>
```

```
//Function Prototype
double kakuritu(char *w1);
double kakuritu2(char *w2);

int seg1(char *moto,char *alpha1,char *beta2);
int seg2(char *moto,char *alpha2,char *beta2);
char *lastchar(char *alpha1);
char *kind(char *moji);
char *firstchar(char *beta1);
int hantei(char *word);
void chop(char *p);
int word_dbm_r(char *word);
int mix_dbm_r(char *mix);
double P_a(char *a,char *b);

#endif


/*********************************************/
/*  形態素解析の結果から過分割を検出する尺度   */
/*********************************************/

#include "shaku.h"

int main(int argc,char *argv[],char *envp[])
{
  FILE *f1;
```

```
char buf[256];
double P[10];
char a1[100],a2[100];
char b1[100],b2[100];
char jisyu[10],first[10];
double L=0.0;


if(argc !=2) {
  printf("error argc=%d \n",argc); exit(1);
}

if((f1 = fopen(argv[1],"r"))==NULL) {
  printf("\"%s\" cannot open!! \n",argv[1]); exit(1);
}

while(fgets(buf,256,f1)!=NULL){
  if(strstr(buf,"EOS")!=NULL) {printf("EOS\n"); continue;}
  chop(buf);
  seg1(buf,a1,b1); // /池谷/昌紀/から /池谷/  /昌紀/ にする.

  //printf("buf = %s\n",buf);
  //printf("UNIGRAM=%d\n",UNIGRAM);
  //printf("BIGRAM=%d\n",BIGRAM);
  //printf("TRIGRAM=%d\n",TRIGRAM);

  //printf("a1 = %s %d\n",a1,word_dbm_r("池谷"));
```

```
//printf("b1 = %s\n",b1);

P[0] = kakuritu(a1);
P[1] = kakuritu(b1);

//printf("P[%s] = %e \nP[%s] = %e\n",a1,P[0],b1,P[1]);
//printf("P[0] = %e \nP[1] = %e\n",P[0],P[1]);

seg2(buf,a2,b2); // /池谷/昌紀/から /池谷 昌紀/ にする.
//printf("a2 = %s\n",a2);
//printf("b2= %s\n",b2);

P[2] = kakuritu(a2);   P[3] = kakuritu(b2);

//printf("P[%s] = %e \nP[%s] = %e\n",a2,P[2],b2,P[3]);
//printf("P[2] = %e \nP[3] = %e\n",P[2],P[3]);

strcpy(jisyu,kind(lastchar(a1))); // /池谷/ の谷の字種を jisyuにコピーす
る
//printf("lastchar = %s\n",lastchar(a1));
//printf("jisyu = %s\n",jisyu);

//P[4] = kakuritu2(jisyu);
P[4] = kakuritu(jisyu);
//printf("P[%s] = %e \n",jisyu,P[4]);
//printf("P[4] = %e \n",P[4]);
```

```
        strcpy(first,firstchar(b1)); // /昌紀/の 昌の文字をfirstにコピーする
        //printf("first = %s\n",first);
        //printf("b2= %s\n",b2);

        P[5] = kakuritu(first);
        //printf("P[%s] = %e \n",first,P[5]);
        //printf("P[5] = %e \n",P[5]);

        strcat(jisyu,first);      //jisyu=谷の字種 と first = 昌 を連結する.
        //printf("New JIsyu = %s\n",jisyu);

        //P[6] = kakuritu2(jisyu); // 具体的には jisyu = ka昌    となる.
        P[6] = kakuritu(jisyu); // 具体的には jisyu = ka昌    となる.
        //printf("P[%s] = %e \n",jisyu,P[6]);
        //printf("P[6] = %e \n",P[6]);

        //尺度を計算する
        L = (P[2] * P[3] * P[6]) / (P[0] * P[1] * P[4] * P[5]);

        printf("%s\t%f\n",buf,L);
    }

    if(fclose(f1)==-1) exit(1);

    return(0);
}

/*****************************************/
```

```
/***  moto = se池谷sese昌紀se    から    ***/
/***alpha1 = se池谷se, beta1 = se昌紀se***/
/***  にする関数                          ***/
/*****************************************/
int seg1(char *moto,char *alpha1,char *beta1)
{
 int i,l;

  for(i=0;moto[i]!='\0';i++){
    if((moto[i]=='s') && (i!=0)){
      if(moto[i-1]=='e'){
alpha1[i]='\0';
break;
      }
      else
alpha1[i]=moto[i];
    }
    else
      alpha1[i]=moto[i];
  }
  l = i;

  for( ;moto[i]!='\0';i++){
    beta1[i-l]=moto[i];
  }
  beta1[i-l]='\0';

//  if(失敗) return(-1);
```

```
//  else return(0);
return(0);
}



/*****************************************/
/***  moto = se池谷sese昌紀se    から    ***/
/***  alpha2 = se池谷  beta2 = 昌紀se   ***/
/***  にする関数                        ***/
/*****************************************/
int seg2(char *moto,char *alpha2,char *beta2)
{
  int i,l;

    for(i=0;moto[i]!='\0';i++){
      if((moto[i]=='s')&&(i != 0)){
alpha2[i]='\0';
break;
      }
      else
alpha2[i]=moto[i];
      alpha2[i]=moto[i];
    }

    /*l=i;*/
    l=0;
    for( ;moto[i]!='\0';i++){
```

```
      if((moto[i]!='s') && (moto[i] != 'e')){
beta2[l]=moto[i]; l++;
      }

    }
    beta2[l]='\0';
    strcat(beta2,"se");
//  if(失敗) return(-1);
//  else return(0);
  return(0);
}



/*****************************************/
/***  alpha1 = se池谷 se から            ***/
/***  最後の文字 谷 をreturnする関数     ***/
/*****************************************/
char *lastchar(char *alpha1)
{

  int i,l=0;
  char *b;

  b = (char *)malloc(10);

  for(i=0;alpha1[i]!='\0';i++){
    if((i!=0) && (alpha1[i]=='s')){
```

```
        i-=2;
        break;
      }
    }
    l=i;
    for( ; alpha1[i] != 's' ;i++){
      b[i-l]=alpha1[i];
      if(i-l==2) break;
    }
    b[i-l] = '\0';
    free(b);
    return(b);
}


/*****************************************/
/***  2バイト文字 moji の                ***/
/***  字種を判定してreturnする関数       ***/
/***  hantei(moji)で数値が返り           ***/
/***  それを，文字に直す                 ***/
/*** 2   >>  kk                         ***/
/*** 1   >>  hi                         ***/
/*** 6   >>  ks                         ***/
/*** 3   >>kn                          ***/
/*** 4   >>an                          ***/
/*** 0   >>ki                          ***/
/*** 9   >>se                          ***/
/*** 5   >>e1                          ***/
```

```
/*** 10  >>e2                          ***/
/*** 11  >>kn                          ***/
/*** 12  >>hi                          ***/
/*** 12  >>hi                          ***/
/*** 13  >>e6                          ***/
/*****************************************/
char *kind(char *moji)
{
  int h;
  h = hantei(moji);
  if (h==2) return("kk");
  else if (h==1) return("hi");
  else if (h==6) return("ks");
  else if (h==3) return("kn");
  else if (h==4) return("an");
  else if (h==0) return("ki");
  else if (h==9) return("se");
  else if (h==5) return("e1");
  else if (h==10) return("e2");
  else if (h==11) return("kn");
  else if (h==12) return("hi");
  else if (h==13) return("e6");
  else if (h==14) return("e7");
  else if (h==15) return("e8");
  else if (h==16) return("e9");
  else if (h==99) return("er");
//  else exit(-1);
}
```

```
/****************************************/
/***  beta1 = se昌紀se から              ***/
/***  最初の文字 昌 をreturnする関数    ***/
/****************************************/
char *firstchar(char *beta1)
{

  int i,l=0;
  char *c;

  c = (char *)malloc(10);

  for(i=0;beta1[i]!='\0';i++)
    if(beta1[i]=='e'){
      i+=1;
      break;
    }

  l=i;
  for( ;beta1[i]!='s';i++){
    c[i-l]=beta1[i];
    if(i-l==2) break;
  }
  c[i-l] = '\0';
  free(c);
  return(c);
}
```

```
/*****************************************/
/***  2バイト文字1文字の字種を判定する ***/
/***結果は数字で戻す．字種は下を参考    ***/
/*****************************************/
int hantei(char *word)
{
  int i;

  char a[100]="一二三四五六七八九十〇壱弐参拾零百千万萬億兆数．・";

  for(i=0;a[i]!='\0';i+=2){  /*漢字数字*/
    if((word[0]==a[i]) && (word[1]==a[i+1])) return(6);
  }

  if((word[0]==-95) && (word[1]==-68)) return(5);       /*  ー  */
  else if((word[0]==-91) && (word[1]==-79)) return(10); /*  ケ  */
  else if((word[0]==-95) && (word[1]==-71)) return(11); /*  々  */
  else if((word[0]==-95) && (word[1]==-74)) return(12); /*  ゞ  */
  else if((word[0]==-95) && (word[1]==-75)) return(12); /*  ヽ  */
  else if((word[0]==-91) && (word[1]==-10)) return(13); /*  ヶ  */
  else if((word[0]==-91) && (word[1]==-11)) return(14); /*  カ  */
  else if((word[0]==-95) && (word[1]==-77)) return(15); /*  ヽ  */
  else if((word[0]==-95) && (word[1]==-76)) return(16); /*  ヾ  */

  else if(word[0]==-91){ /* カタカナ*/
    if((word[1] > -96) && (word[1] < -11)) return(2);
  }
```

```
else if(word[0]==-92){ /*ひらがな*/
  if((word[1] > -96) && (word[1] < -12)) return(1);
}

/*******第壱水準********/
else if(word[0]==-80){  /*上限*/
  if(word[1] > -96)    return(3);
}
else if(word[0]==-49){ /*下限*/
  if(word[1]<-44)
  return(3);
}

else if((word[0] > -80) && (word[0] < -49)){ /*中*/
  return(3);
}
/******第壱おわり********/


/*******第弐水準********/
else if(word[0]==-48){  /*上限*/
  if(word[1] > -96)    return(3);
}
else if(word[0]==-12){ /*下限*/
  if(word[1]<-91)
  return(3);
}
```

```
  else if((word[0] > -48) && (word[0] < -12)){ /*中*/
    return(3);
  }
  /******第弐おわり********/


  else if(word[0]==-93){ /*英数--弐バイト数字*/
    if((word[1] > -81) && (word[1] < -70)) return(4);

    else if((word[1] > -64) && (word[1] < -37))
      return(4);   /*英数--弐バイトアルファベット大*/

    else if((word[1] > -32) && (word[1] < -5))
      return(4);   /*英数--弐バイトアルファベット小*/
  }

//  else if(word[0]==-95){ /*記号*/
//    if((word[1] > -92) && (word[1] < -4)) return(0);
//  }

  else if(word[0]==-95){ /*記号*/
    return(0);
  }
  else if(word[0]==-94){ /*記号*/
    return(0);
  }
```

```
  else if((word[0]==115) && (word[1]==101)) return(9); /*セグメント*/

  //  else exit(-1);

  else return(99);
```

/*下記を戻してください.
　谷なら return(3)
漢字コードは EUC コードでお願いします.

カタカナ 2
ひらがな 1
漢字数字 6
（数億千万百十一二三四五六七八九兆〇など）
普通の漢字 3
英数 4
記号 0
セグメント (se) 9
"ー" 5
"ケ" 10
"々" 11
"ゝ" 12
"ゞ" 12
"ヶ" 13
"カ"　　14
"ヽ"　　15
"ヾ"　　16

```
*/


}


/****************************************/
/***文字列の出現確率の計算                ***/
/***2グラムから求める                     ***/
/****************************************/
double kakuritu(char *w1){
  double P=(double)1.0;
  int i,j;
  int len;
  char temp1[10],temp2[10];

  len = strlen(w1);

  /****一文字の場合************************/
  for(i=0;i<2;i++) temp1[i] = w1[i];
  temp1[i] = '\0';
  P = (double)word_dbm_r(temp1)/UNIGRAM ;
  //printf("Count[%s] = %ld\n",temp1,word_dbm_r(temp1));
  if(len == 2) return(P);
  /****************************************/

  /****2文字の場合*************************/
  j=i;
```

```
for(;i<4;i++) temp2[i-j] = w1[i];
temp2[i-j] = '\0';
P  = P * P_a(temp1,temp2);
if(len == 4) return(P);
/****2文字の場合**************************/

/*****3文字以上の場合ここから ***********/
/*
for(i=2;i<strlen(w1)-2;i+=2){
  for(j = 0;j < 2;j++) temp1[j] = w1[j+i-2];
  temp1[j] = '\0';
  for(j=0;j<4;j++) temp2[j] = w1[j+i];
  temp2[j] = '\0';
  P = P* P_a(temp1,temp2);
}*/

for(i=0;i<strlen(w1)-4;i+=2){
  for(j = 0;j < 4;j++)
    temp1[j] = w1[j+i];
  temp1[j] = '\0';
  for(j=0;j<2;j++)
    temp2[j] = w1[j+i+4];
  temp2[j] = '\0';
  P = P* P_a(temp1,temp2);
}

return(P);
```

```
}


/*****************************************/
/***  最後にある改行文字を切り落とす    ***/
/*****************************************/
void chop(char *p)
{
  int i;
  for(i=0;p[i] != '\0';i++);
  if (i > 0) p[i-1] = '\0';
}


/*****************************************/
/***  dbm形式のファイルから 頻度を引く  ***/
/***  対象は字種を含まない文字列        ***/
/*****************************************/
int word_dbm_r(char *word)
{
  datum key_d;
  datum data_d;
  DBM *dbm_ptr1;
  int i,leng;
  char num[10];

  dbm_ptr1 = dbm_open("TD_DBM", O_RDONLY , 0666 );
```

```
  if( !dbm_ptr1 ) {
    exit(EXIT_FAILURE);
  }

  key_d.dptr = word;
  key_d.dsize = strlen(word);

  data_d = dbm_fetch(dbm_ptr1,key_d);
  leng = data_d.dsize;
  for(i=0;i<leng;i++){
    num[i] = data_d.dptr[i];
  }
  num[leng] = '\0';
  dbm_close(dbm_ptr1) ;

  if(leng == 0) return(1);
  else return(atoi(num));

}

double P_a(char *A,char *B)
{
  char C[20];
  int t1,t2;
  double x,y;

  strcpy(C,A);
```

```
    strcat(C,B);

    t1 = word_dbm_r(A);
    //printf("Count[%s]  = %ld\n",A,(long)t1);
    t2 = word_dbm_r(C);
    //printf("Count[%s]  = %ld\n",C,(long)t2);
    x = (double)t1/UNIGRAM;

    if(strlen(B) == 2)
      y = (double)t2/BIGRAM;
    else if(strlen(B) == 4)
      y = (double)t2/TRIGRAM;
    else exit(-1);

    return(y/x);

}
```