平成21年度茨城大学工学部情報工学科卒業研究論文

# 名詞の主要語義の推定と語義識別への応用

平成21年度2月10日

工学部情報工学科

執筆者：江口晃（05T4007T）

指導教員：新納浩幸 准教授

平成21年　茨城大学等学部情報工学科　卒業論文

# 名詞の主要語義の推定と語義識別への応用

著者：江口晃（05T4007T）

指導教員：新納浩幸

—論文要旨—

本論文では、多義語の名詞の主要語義を推定し、それを語義識別問題に応用した。

自然言語処理では、多義語の語義を識別することが必要となる。文脈中の多義語の識別は、近年需要の上昇している機械翻訳などでも利用される。

文脈中の多義語を識別する際には、文脈をベクトル化する必要がある。ベクトルの素性として、多義語の直前後の単語と、多義語の前後方の自立語を利用する。また、利用する素性は文脈中の単語だけではなく、文脈中の単語の語義コードも利用するのが一般的である。語義コードとは、概念辞書において各単語に割り当てられているコードである。語義コードを素性として利用することにより、異なる単語でも同義ならば類似度を持つことができる。

文脈中の単語は複数の語義コードを持つ場合が多い。素性に利用する単語が複数の語義コードを持つ場合、語義コードをすべて素性に利用することが通常行われている。しかしこの方法では、文脈中の単語の語義として使われていない語義コードも利用することになる。文脈中で使われていない語義が、語義識別において誤った類似度を高める危険性がある。

本研究では、名詞の語義コードから主要な語義コードの推定を行い、主要な語義コードのみを素性として用いて、精度の高い語義識別を目指した。コーパスより出てくる名詞の頻度を調べ、各名詞のすべての語義コードに名詞の頻度を加算していき、語義コードと頻度を加算した値のリストをつくる。この値が大きさを語義コードの順位とし、値が大きい語義コードを主要な語義とした。

実験ではSENSEVAL2の辞書タスクの動詞50単語の語義識別問題を題材に、Naive Bayes分類器を用いて本手法の効果を調べた。実験では"語義コードなし"、"全語義コード"、"主要語義コードのみ"を比較として行った。語義識別問題に語義を使う効果は認められたが、その語義を主要語義に限定することに効果は見られなかった。

今後の課題は、語義コードからの主要語義の推定部分の改良である。また今回は名詞の語義コードのみを素性として利用したが、動詞や形容詞などへの拡張も必要である。

# 目次

# 図目次

# 表目次

# 第1章　序論

## 1．1　背景と目的

自然言語処理では、多義語の語義を識別することが必要となる。文脈中の多義語の識別は、近年需要の上昇している機械翻訳などでも利用される。

文脈中の多義語を識別する際には、文脈をベクトル化する必要がある。ベクトルの素性として、多義語の直前後の単語と、多義語の前後方の自立語を利用する。また、利用する素性は文脈中の単語だけではなく、文脈中の単語の語義コードも利用するのが一般的である。語義コードとは、概念辞書において各単語に割り当てられている同義語で分類されたコードである。語義コードを素性として利用することにより、異なる単語でも同義ならば類似度を持つことができる。

文脈中の単語は複数の語義コードを持つ場合が多い。素性に利用する単語が複数の語義コードを持つ場合、語義コードをすべて素性に利用することが通常行われている。しかしこの方法では、文脈中の単語の語義として本当は使われていない語義コードも素性として利用することになる。文脈中で本当は使われていない語義が、語義識別において誤った類似度を高める危険性があるのだ。文脈のベクトル化の際に、文脈中に多義語存在した場合、その多義語を識別できることが理想である。しかし現実には、多義語の語義を識別するのは困難である。

本研究では、名詞の語義コードから主要な語義コードの推定を行い、主要な語義コードの上位のみを素性として用いてベクトル化し、精度の高い語義識別を目指した。コーパスより出てくる名詞の頻度を調べ、各名詞のすべての語義コードに名詞の頻度を加算していき、語義コードと頻度を加算した値のリストをつくる。この値が大きさを語義コードの順位とし、値が大きい語義コードを主要な語義とした。主要な語義コードの上位3つのみを素性として利用してベクトル化し、Naive Bayes 分類器を用いて語義識別の実験を行った。

## １．２　本論文の構成

第２章では、語義識別の説明と、文脈のベクトル化の基本的な手法について述べる。

第３章では、分類に利用したNaive Bayes分類器についての概要と、Naive Bayesに使われているベイズの定理について説明する。

第４章では、日本語WordNetの説明と、全ての語義コードを素性として利用した手法を説明する。また、語義コード群から主要語義を推定する方法と、語義識別への応用について説明する。

第５章では、Naive Bayesを用いて、文脈のベクトル化に推定語義の語義コードを素性として用いた語義識別の実験結果を報告する。

第６章では、実験結果をふまえての考察を行う。

第７章では、結論と今後の課題について述べる。

# 第２章　語義識別

## ２．１　義語の曖昧性

日本語に限らず様々な言語では、文中で複数の語義を持つ単語がある。例えば『核』も多義語の一つである。『核』には核爆弾や核戦争などで使われる「核兵器」という意味や、核となる人物のような「中心」の意味もある。その他にも数学で使われる「カーネル」など、複数の語義が存在する。

これら多義性のある単語を、文中ではどの例で使われているか機械的に判別することを語義識別という。

多語義には、違いは少ないが、辞書では異なる語義に分類されているような単語もある。こういった単語は語義の識別が難しく、このタスクの難しさでもある。通常、人間にとって識別困難な多義語は機械にとっても識別困難である。また、人間にとっては識別容易な多義語であっても、機械学習により得られた規則では識別困難な多義語もある。例としては、「開発」、「核」、「精神」、「乗る」、「生まれる」、「かかる」などが挙げられる。

語義識別はいろいろなところで利用されている。機械翻訳に応用すると、適切な訳語の選択が可能になる。意味での要約に応用すれば、異なる言い回しの同じ意味の単語が分かり、同じトピックを抽出することが可能になる。コンテンツへ識別した意味タグを付与すれば、同じ意味のコンテンツを検索することが容易なる。教師なし学習の識別では、どの既存の語義にも分類されない新しい語義の発見ができるかもしれない。

## ２．２　文書のベクトル化

機械学習によって語義識別の規則を得るには、語義の特徴を見つけるために、その素性として対象単語の周辺文脈を利用してベクトル化するのが一般的である。多義語をある語義で用いる場合、その語義に関連する単語や、文法的に使われやすい単語が、前後の文脈にあることが多い。例えば『核』を「核兵器」の語義で使う場合、その文脈の前後では「戦争」、「平和」、「廃止」などの単語が使われることが多い。このような規則をベクトルを用いて作ることができる。

ベクトル化するための素性としては、文脈で使われている多義語の前後の単語を利用する。素性 e1 は直前の単語、素性 e2 は直後の単語とする。また、素性 e3 は前方の自立語を５つまで、素性 e4 は後方の自立語を５つまでとする。文脈のベクトル化の概要を表した図は図 2.1 に示す。

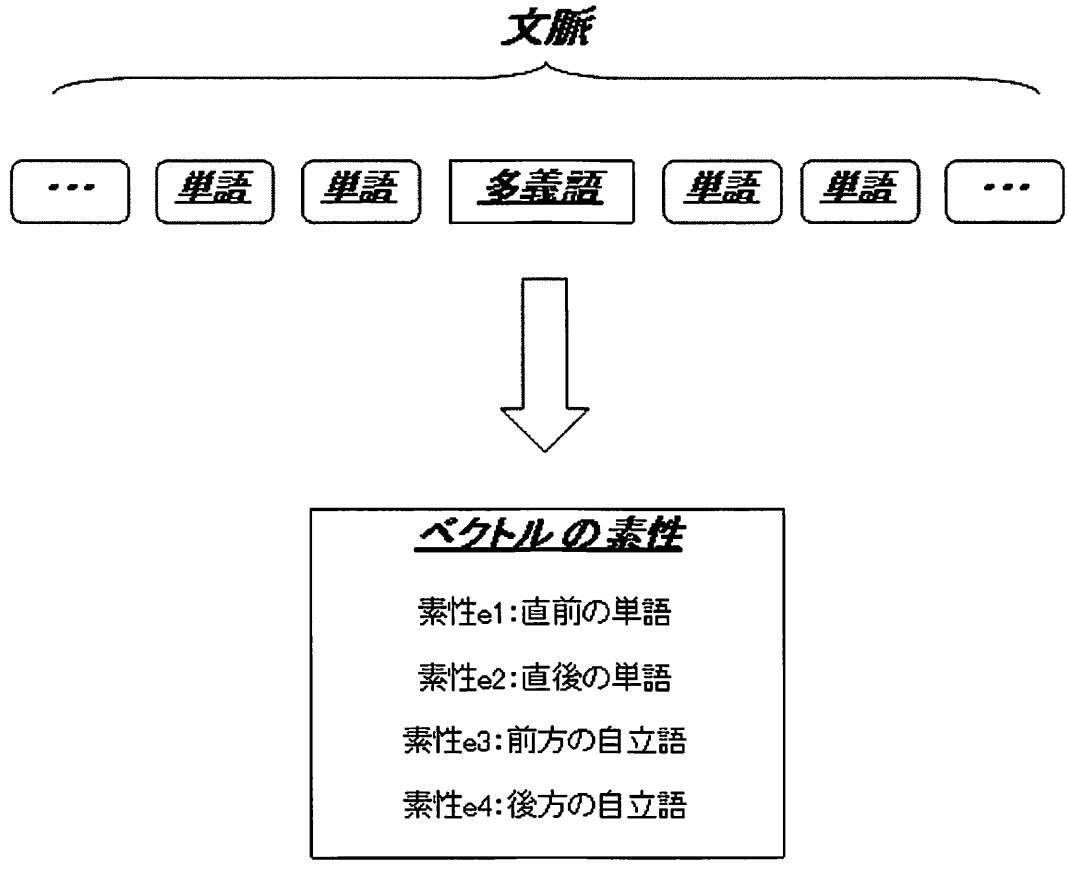


図 2.1:文脈のベクトル化

例として、語義識別の対象単語を「核」として、以下の文をベクトル化してみる。

学校/を/核/と/し/た/地域/活動/。

この場合、「核」の直前、直後の単語から、e1が「を」、e2が「と」となる。次に「核」の前方の自立語から近い順に取り、e3は「家族」になる。「核」の後方の自立語から近い順に取り、e4は「する」「地域」「活動」「。」となる。ここで「し」はサ変動詞「する」の連体形なので、原型の「する」にする。句読点も内用語として設定していくので、「。」も素性として含める。結果として以下の7個が素性となる。

表2.1:語義コードを用いない素性

| | |
|---|---|
| e1 | を |
| e2 | と |
| e3 | 家族 |
| e4 | する、地域、活動、。 |

## ２．３　教師あり学習

機械学習の手法として、教師あり学習がある。事前に与えられたデータをいわば「例題（＝先生からの助言）」とみなして、それをもとに学習を行うところからこの名がついている。教師あり学習は分類問題や回帰問題などに用いられる。

例えば最も簡単な分類問題である2値分類問題では、訓練データ(例題)が典型的にはベクトルとラベルの組として、(ｘ1,ｙ1),(ｘ2,ｙ2),・・・のように与えられる。ここで、ｙ1は０または１の２値を取るラベルで、ｘｉはｉ番目のデータの座標を表す。そして「学習」とは、これらのデータに何らかの基準でもっとも合う関数関係ｙ=ｆ(ｘ)を求めることである。回帰問題でもほぼ同様で、違いは ｙｉが離散値の代わりに実数値を取るということである。

このような関数関係が求められれば、未知のデータｘにそれを適用して、予言 ｙ=ｆ(ｘ)を与えることができる。分類問題であればこれを分類器、回帰問題であればこれを回帰曲線などという。

教師あり学習を利用した手法としては、決定リスト、Naive Bayes 法、Support Vector Machine(SVM)などがある。本研究ではNaive Bayes 法を利用した。

# 第３章　Naive Bayes

## ３．１　確率の基本性質と条件付確率

まず、確率に関する基本的な性質について述べる。一般には、ある実験や観測を試行と呼び、試行の結果として起こることがらを事象と呼ぶ。

試行Ｔを行うときに、起こり得るすべての場合がｎ通りあり、どれも同時に起こることがなく、また、どの場合が起こることも同程度に期待できるものとする。このとき、事象Ａの起こるのは、このｎ通りのうちａ通りであるとすると、試行Ｔを行うときの事象Ａの起こる確率は以下のように定義される。

$$P(A)=\frac{a}{n} \qquad (3.1)$$

この試行を行うときに起こる事象について、起こり得るすべての事柄からなる事象を全事象と呼び、Ωで表す。

事象Ａ1，A2，A3・・・がともに起こる事象をＡ1，A2，A3・・・の積事象といい、A1∩A2∩A3∩・・・で表す。

A1，A2，A3・・・のうち少なくても１つは必ず起こる事象をＡ1，A2，A3・・・の和事象といい、A1∪A2∪A3∪・・・で表す。

事象Ａが起こらないことも１つの事象であり、これをＡの余事象といい、$\overline{A}$ で表す。

これらの事象の関係をベン図で表すと図3.1のように表される。

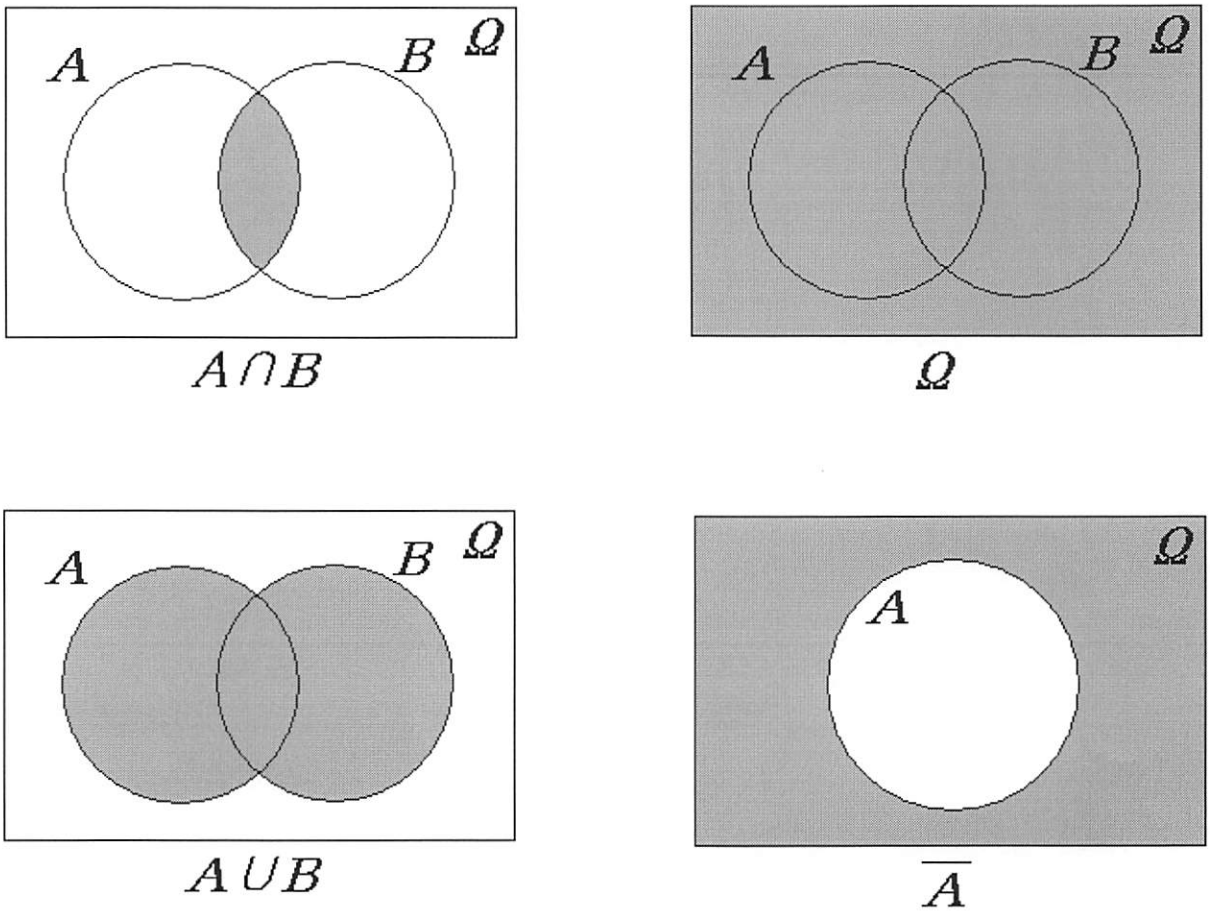


図 3.1:事象の関係図

サイコロを振る試行を例にして考えてみる。サイコロの目は１から６にいずれかの目が出るので、この場合の全事象は次のようになる。

$$\Omega=\{1,2,3,4,5,6\} \quad (3.2)$$

出る目の数が奇数の場合を事象Ａとし、出る目の数が３以上の場合を事象Ｂとすると、Ａ∩Ｂ、Ａ∪Ｂ、は$\overline{A}$は以下のようになる。

$$A\cap B=\{3,5\} \quad (3.3)$$

$$A\cup B=\{1,3,4,5,6\} \quad (3.4)$$

$$\overline{A}=\{2,4,6\} \quad (3.5)$$

出る目の数が偶数の場合を事象Ｃとすると、Ａ∩Ｃが起こることはありえない。このような起こりえない事象を空事象といい、$\phi$で表す。

また、事象Ａ1，Ａ2，Ａ3・・・があった場合、任意のｉ、ｊ（ｉ≠ｊ）についてＡｉ∩Ａｊ＝$\phi$であるとき、これらの事象は互いに背反であるといわれる。サイコロの例でいうと、サイコロの目は偶数の目が出る事象と奇数の目が出る事象が同時に起こることはないので、事象Ａと事象Ｃは互いに背反である。

次に条件付確率について述べる。トランプを例にして条件付確率について説明していく。今よくきったトランプ５２枚から１枚トランプを抜くとき、その札がスペードである事象をＡ、絵札である事象をＢとすると、起こり得るすべての場合は５２通りであり、次のようになる。

$$P(A)=\frac{13}{52} \quad (3.6)$$

$$P(B)=\frac{12}{52} \quad (3.7)$$

$$P(A\cap B)=\frac{3}{52} \quad (3.8)$$

このとき、抜いた札がスペードだとわかっていると、起こり得るすべての場合がスペー

ドの札に制限されるため、この中から抜いた札が絵札となる確率は$\frac{3}{13}$になる。

このように１つの試行を行うときに起こる事象Ａ,Ｂに対して、Ｐ(Ａ)>0のとき、Ｐ(Ａ∩Ｂ)/Ｐ(Ａ)を事象Ａが起こったという条件のもとで事象Ｂの起こる条件付確率といい、Ｐ(Ｂ|Ａ)とあらわす。式で表すと次のようになる。

$$P(B|A)=\frac{P(A\cap B)}{P(A)} \qquad (3.9)$$

先ほどのトランプの例で、抜いた札がスペードであった条件のもとで、それが絵札である確率は、式(3.9)を用いると次のように算出される。

$$P(B|A)=\frac{\frac{3}{52}}{\frac{13}{52}}=\frac{3}{13} \qquad (3.10)$$

また、式(３,10)からＰ(Ａ)>0, Ｐ(Ｂ)>0のとき、次の確率の乗法定理が得られる。

$$P(A\cap B)=P(A)P(B|A)=P(B)P(A|B) \qquad (3.11)$$

## ３．２　ベイズの定理

ベイズの定理は、今日では、いくつかの未観測要素を含むコンピュータによる推論等で利用され、迷惑メールの発見・分類といった作業の自動化(フィルタリング)といった情報工学上の情報ふるい分けに利用されている。この定理は、ある結果が得られた時に、その結果に対する原因の確率を求めるのに使われている。Ｐ(Ａ)を事象Ａが発生する確率、Ｐ(Ａ|Ｂ)を事象Ｂが既に発生している場合に事象Ａが発生するという条件付確率とする。事象Ｂｉが発生した時、事象Ａが発生する確率はＰ(Ａ|Ｂｉ)であるが、これを逆に見て、事象Ａが発生したときにＢｉが発生していた確率Ｐ(Ａ|Ｂｉ)を知りたい場合などに使われる。

事象Ａの条件のもと、以下の式が成り立つ。

$$P(A)=\sum_{i=1}^{n}P(A\cap B_i)=\sum_{i=1}^{n}P(A\mid B_i)P(B_i) \qquad (3.12)$$

式(３,12)から確率Ｐ(Ｂｉ|Ａ)は次のように表すことが出来る。

$$P(B_i\mid A)=\frac{P(A\mid B_i)P(B_i)}{P(A)}=\frac{P(A\mid B_i)P(B_i)}{\sum_{i=1}^{n}P(A\mid B_i)P(B_i)} \qquad (3.13)$$

この式(3.13)をベイズの公式と呼ぶ。Ｐ(Ｂｉ)を事前確率、Ｐ(Ｂｉ|Ａ)を事後確率という。

例えば、千人に１人の割合で感染している病気があるとする。検査機によって、感染していれば0.98の確率で陽性反応が出る。しかし感染していない場合でも、0.01の確率で陽性反応が出てしまう。陽性反応が出た人が感染している確率を求めるには、事象Ａを感染の有無、事象Ｂを陽性反応の有無として、以下の式をベイズの公式に当てはめればよい。

$$P(\text{感染})=0.001 \qquad (3.14)$$

$$P(\text{非感染})=0.999 \qquad (3.15)$$

$$P(\text{陽性}\mid\text{感染})=0.98 \qquad (3.16)$$

$$P(\text{陽性}\mid\text{非感染})=0.01 \qquad (3.17)$$

これらをベイズの公式に当てはめると以下のようになる。

$$P(\text{感染}|\text{陽性}) = \frac{P(\text{感染})P(\text{陽性}|\text{感染})}{P(\text{陽性})}$$
$$= \frac{0.001 \times 0.98}{0.001 \times 0.98 + 0.999 \times 0.01} = 0.089 \qquad (3.18)$$

## ３．３　Naive Bayes

機会による評価として、実際に機械学習手法を実装させる必要がある。本研究では機械学習の手法として、Naive Bayes 法を用いた。

Naive Bayes 法は全ての未知量を確率変数として扱い、未知パラメータも確率分布として推定を行う確率論に基礎をおく推定方法を用いた分類方法であり、スパムメールの自動フィルタリングなどの文書分類においても広く利用されている。その特徴としては、最終的な仮定成立の確率計算に事前知識を利用し、仮定が成立するかどうかを真か偽ではなく、確率で明示的に表現できる点にある。

ある事例ｘを素性のリストとして、ｘ=(ｘ1,ｘ2,・・・,ｘn)とし、ｘの分類先のクラスの集合をC={ｃ1,ｃ2,・・・,ｃm}としたとき、分類問題はＰ(ｃ|ｘ)の分布を推定することで解決できる。ｘのクラスｃｘは以下の式で決まる。

$$c_x = \arg\max_{c \in C} P(c \mid x) \qquad (3.19)$$

また、ベイズの定理より、

$$P(c \mid x) = \frac{P(c)P(x \mid c)}{P(x)} \qquad (3.20)$$

となる。よって、式(3.19)、式(3.20)より、以下の式が成立する。

$$c_x = \arg\max_{c \in C} P(c)P(x \mid c) \qquad (3.21)$$

Ｐ(ｃｉ)は事前確率なので、クラスの割合などから比較的容易に推定できる。しかしＰ(ｘ|ｃｉ)の推定は困難である。ここでNaive Bayes 法では以下の仮定を導入する。

$$P(x \mid c) = \prod_{i=1}^{n} P(x_i \mid c) \qquad (3.22)$$

Ｐ(ｘｉ|ｃ)の推定は比較的容易であるために、結果としてＰ(ｘ|ｃ)が推定できる。Naive Bayes 法がうまくいくためには、式(3.22)の仮定をできるだけ満たすような素性を選択する必要がある。本研究などの自然言語の分野では、多くの場合は素性には単語が使われてい

る。これは単語同士ではクラスに属する確率に関わりがないと仮定しているということである。

このNaive Bayes法を利用して分類を行う分類器をNaive Bayes分類器という。Naive Bayes分類器の概要を表した図を図3.2に示す。

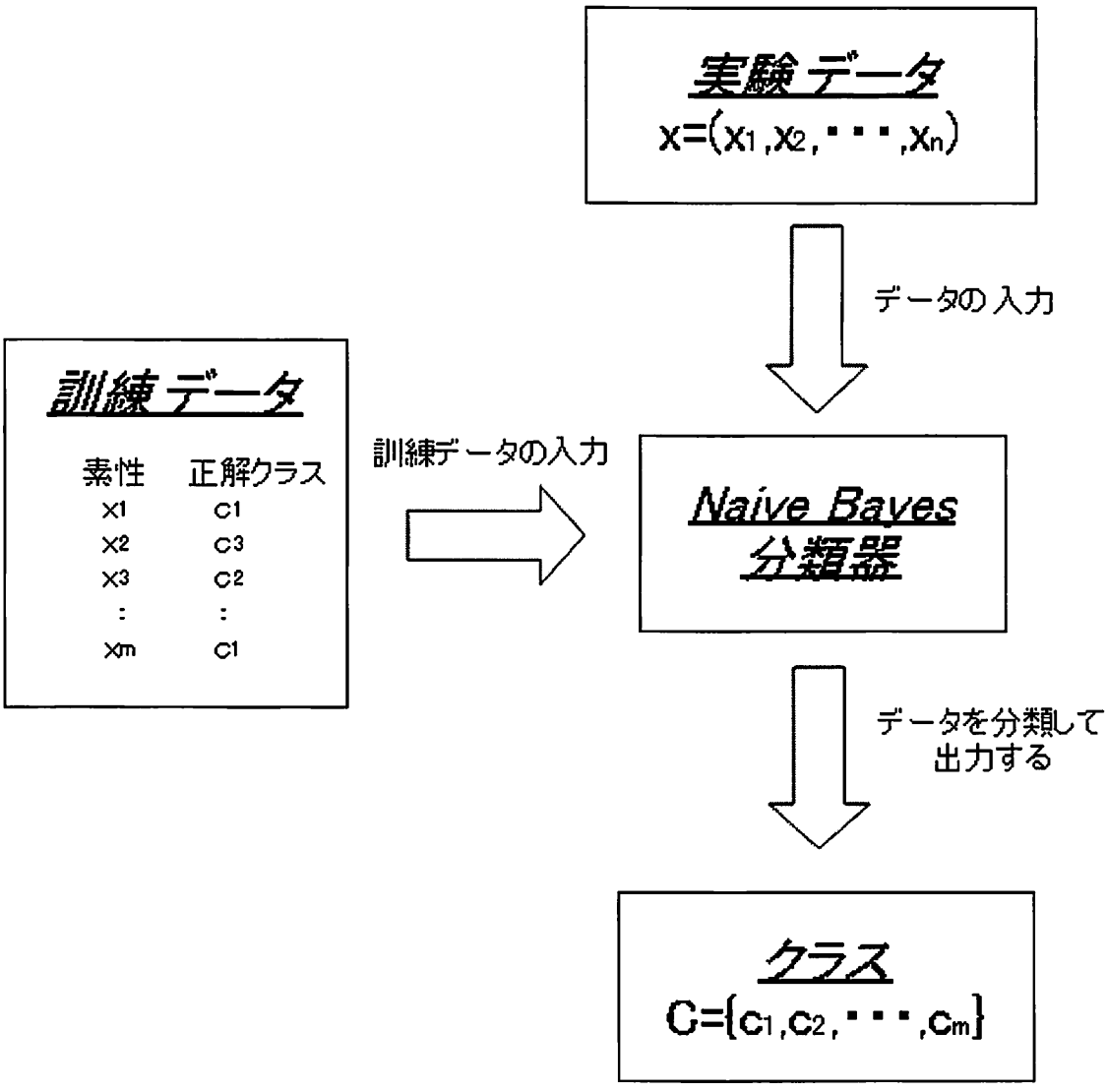


図3.2:Naive Bayes分類器

# 第４章　主要語義の推定

## ４．１　語義コード

自然言語処理では、文脈をベクトル化することが頻繁に行われる。しかし文脈の単語のみを利用するのでは、異なる言い回しの同じ意味の単語や、似たような意味で使われる単語を、全く違う単語としてしかベクトル化できない。例えば、「高校」と「大学」という単語を考えてみると、単語的には全く違うものだが、どちらにも英語でいう「school」の意味がある。

素性に語義を考慮させる方法として、日本語 WordNet に登録されている単語の語義コードを利用する方法がある。日本語 WordNet とは、日本語の概念辞書(意味辞書)である。単語が synset(語義コード)と呼ばれる同義語のグループに分類されている。

例えば、「クリスタル」という名詞の語義コードは以下の３個である。

| 09260466-n、03142834-n、14883206-n |
|---|

「本」という名詞の語義コードは以下の 10 個である。

| 08511777-n、02870092-n、06413666-n、13597794-n、<br>06410904-n、06394865-n、13356002-n、03841417-n<br>07009946-n、09215315-n |
|---|

このように、語義コードの数はその単語によって異なり、１個だけのものもあれば 10 個以上のものもある。

## ４．２　全語彙を利用した手法

ベクトルの素性に全ての語義コードを加えることにより、語義の特徴を見つける精度を上げることができる。

ベクトル化するための素性としては、文脈で使われている多義語の前後の単語を利用し、素性e1,e2,e3,e4までは語義コードを全く使わない手法と同様の素性を利用する。ここでe3とe4の自立語から名詞のみを取り出す。各名詞の日本語WordNetに載っている全ての語義コードを素性e5とする。本研究では名詞の語義コードのみを素性に利用した。語義コードを利用した文脈のベクトル化の概要を表した図は図4.2に示す。

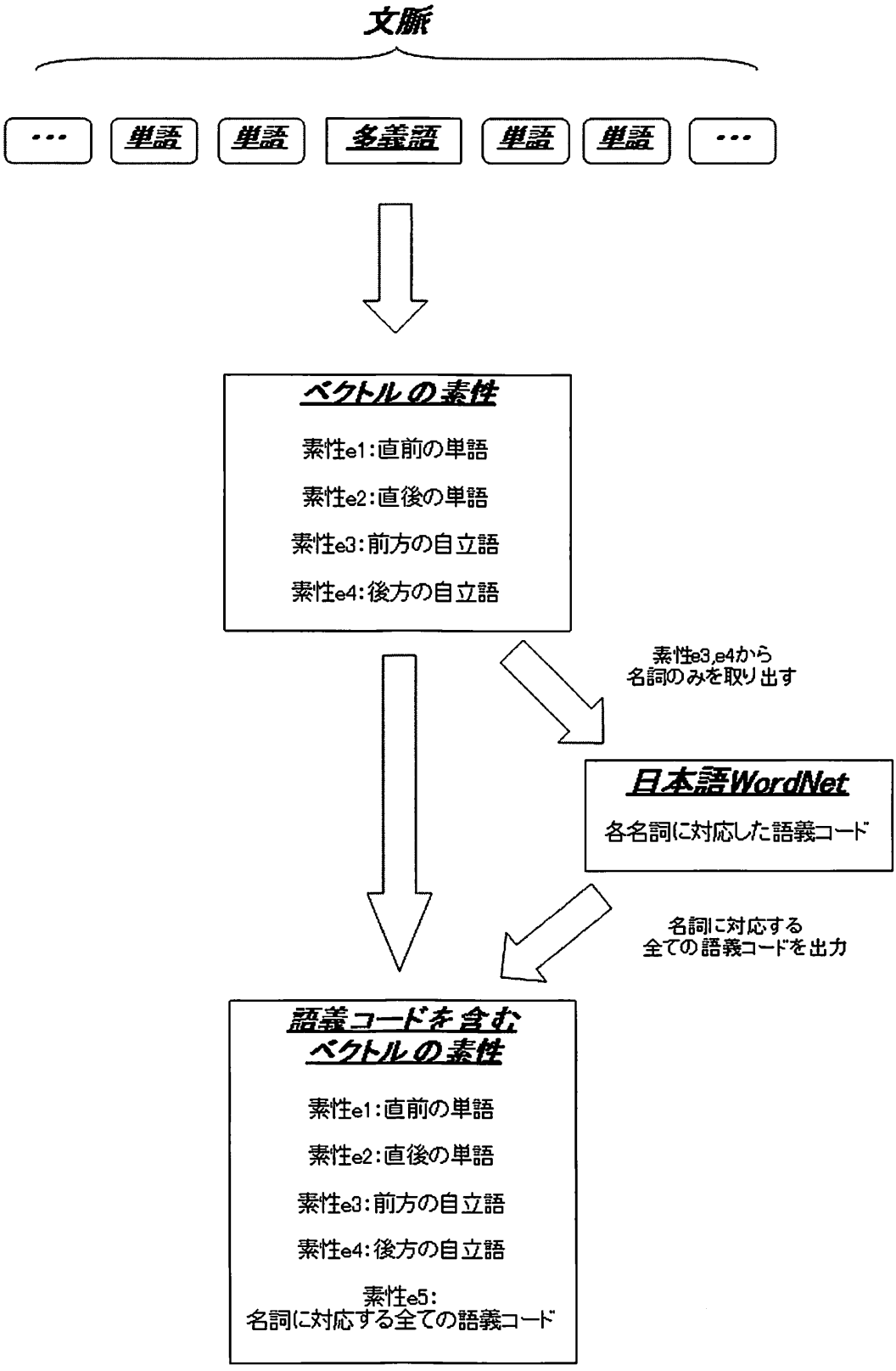


図 4.2:全ての語義を用いたベクトル化

例として、語義識別の対象単語を「守る」として、以下の文を全語義コードを素性に用いてベクトル化してみる。

彼/は/時間/を/守る/性格/だ/。

この場合、「守る」の直前、直後の単語から、e1 が「を」、e2 が「性格」となる。次に「守る」の前方の自立語から近い順に取り、e3 は「彼」と「時間」になる。「守る」の後方の自立語から近い順に取り、e4 は「性格」と「。」となる。
次に e3 と e4 から名詞を取り出す。名詞は「彼」「時間」「性格」の３つであり、各語義コードを日本語 WordNet から調べて、以下の語義コードを得る。

彼：10682953-n、09871364-n

時間 :00028270-n、15135822-n、15122231-n、15134340-n、15227846-n、15270431-n、15246353-n、15129927-n、07288215-n、15269513-n、05131023-n、15228378-n、15113229-n

性格 :04617562-n、05849284-n、04620216-n、04623612-n、04623113-n、04726724-n、04763293-n

結果として以下の 28 個が素性となる。

表 4.1:全ての語義コードを利用した素性

| | |
|---|---|
| e1 | を |
| e2 | 性格 |
| e3 | 彼、時間 |
| e4 | 性格、。 |
| e5 | 10682953-n、09871364-n、00028270-n、15135822-n、15122231-n、15134340-n、15227846-n、15270431-n、15246353-n、15129927-n、07288215-n、15269513-n、05131023-n、15228378-n、15113229-n、04617562-n、05849284-n、04620216-n、04623612-n、04623113-n、04726724-n、04763293-n |

## ４．３　全語彙を利用した問題点

文脈をベクトル化する際に語義コードを含めれば、異なる単語間に対しても意味が似ていれば類似度を与えることができるために、より適切なベクトル化が可能になる。しかし、素性とする単語が多義語である場合、その多義語の利用されていない別の語義が、もう一方の文脈との類似度誤って高める危険性もある。

例として、名詞「記録」の語義識別問題を例にして文脈のベクトル化の問題を述べる。「記録」には以下の２つの語義がある。

**語義１**　のちのちに伝える必要から、事実を書き記すこと。

**語義２**　競技などの成績・結果。

以下の(a)の文の「記録」は(語義１)の意味であり、(b)の文は(語義２)の意味である。

(a)これがこの地方の儀式の記録です。

(b)このタイムが公式記録です。

(a)の「記録」の文脈に「儀式」がある。「儀式」の日本語 WordNet での語義コードは以下の６つである。

| |
|---|
| 07450842-n、01027379-n、07447261-n、01027859-n、<br>01204055-n、04911420-n |

また(b)の「記録」の文脈に「公式」がある。「公式」の日本語 WordNet での語義コードは以下の５つである。

| |
|---|
| 01204055-n、04911420-n、06731802-n、05846932-n、<br>00186491-n |

(a)の「記録」の文脈と(b)の「記録」の文脈は異なるはずなので、共通の素性が現れないのが理想である。しかし全語義コードを使うと、"01204055-n"と"04911420-n"の２つが

一致してしまう。結果として(a)の「記録」に対するベクトルと、(b)の「記録」に対するベクトルに類似度が現れてしまう。このような予期せぬ類似度により、語義識別を誤ってしまう可能性がある。

## ４．４　主要語義の推定

文脈のベクトル化の際に、文脈中に多義語存在した場合、その多義語を識別できることが理想である。しかし現実には、多義語の語義を識別するのは困難である。本来、多義語の語義を識別するために文脈のベクトル化が必要なので、文脈のベクトル化の際に多義語の語義を識別することはできない。

本研究ではこの問題を解決するために名詞の主要語義を推定し、多義語の名詞に対してはその主要語義だけを利用することを提案する。

ここでは名詞の主要語義を推定するためにコーパスを利用する。コーパスとしてはBCCWJコーパスの「白書」を利用した。まずコーパスをMeCabによって解析して名詞のみを全て取り出す。そして取り出した名詞をカウントしてコーパス内の名詞の頻度を求める。

次に、日本語WordNetで割り当てられた名詞に対する語義コードのリストを作り、使われている名詞の頻度を全て加算していく。

最後に値の大きい順にこの語義コードのリストをソートする。これを語義コードの順位表と呼ぶ。語義コードの順位表の作り方の概要を表した図は図4.3に示す。

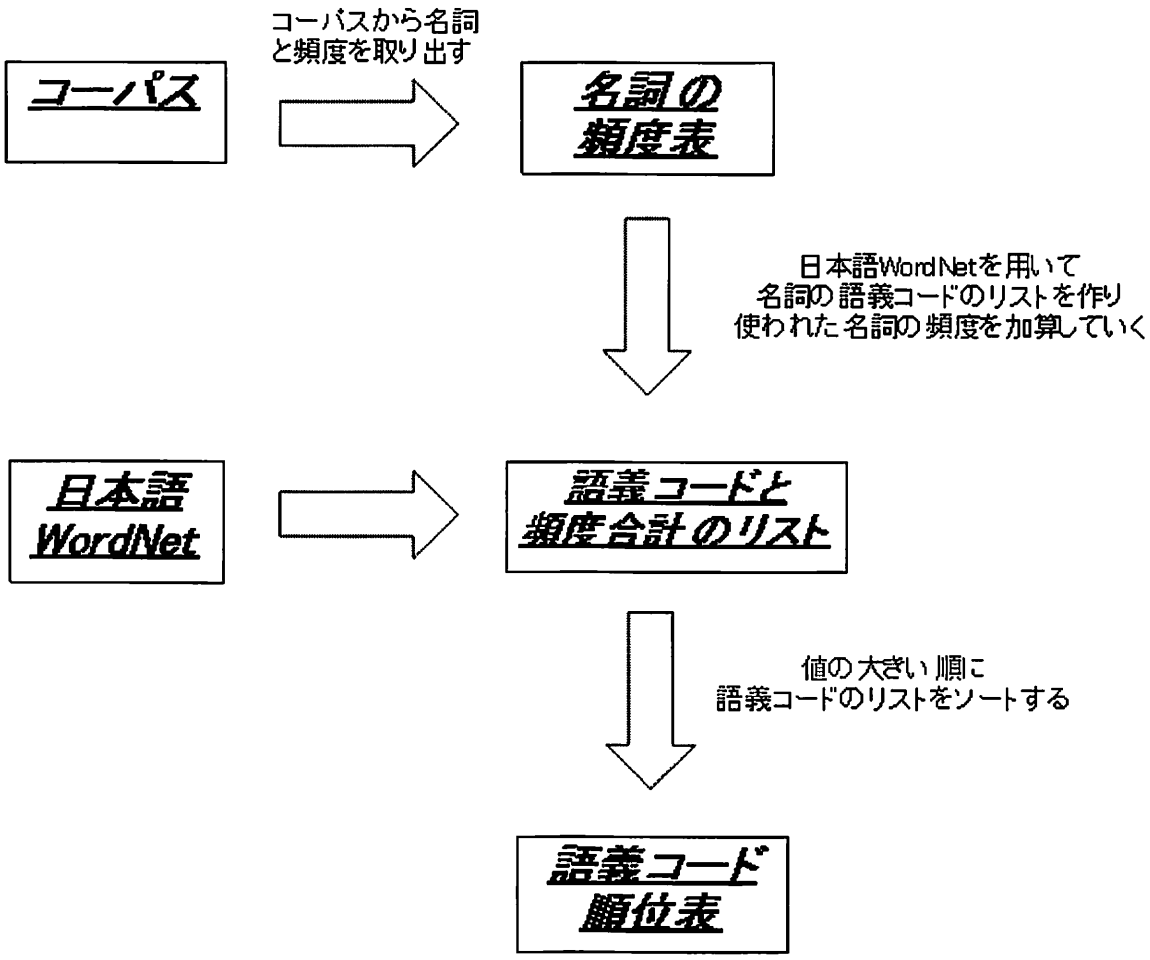


図 4.3:語義コード順位表の作り方

## ４．５　語義識別問題への応用

ベクトルの素性に主要な語義コードだけを加えることにより、語義の特徴を見つける精度が上がることを期待する。

ベクトル化するための素性としては、文脈で使われている多義語の前後の単語を利用し、素性 e1,e2,e3,e4 までは語義コードを全く使わない手法と同様の素性を利用する。素性 e5 として、e3 と e4 の名詞の語義コードから、語義コード順位表より上位３つだけを利用した。

例として、語義識別の対象単語を「描く」として、以下の文を考える。

過酷/な/世界/を/描く/ドラマ/。

この場合、「描く」の直前、直後の単語から、e1 が「を」、e2 が「ドラマ」となる。次に「描く」の前方の自立語から近い順に取り、e3 は「過酷」と「世界」になる。「描く」の後方の自立語から近い順に取り、e4 は「ドラマ」と「。」となる。次に e3 と e4 から名詞を取り出す。名詞は「過酷」「世界」「ドラマ」の３つであり、各語義コードを日本語 WordNet から調べて、以下の語義コードを得る。

過酷：01803583-a、01041481-a、01507402-a

世界：09270894-n、07966140-n、14514805-n、09480809-n、05809878-n、07965973-n

ドラマ：07290278-n、06376154-n、07007945-n

この語義コードを語義コード順位表より上位３つだけを利用する。これを主要語義とする。すると利用する語義コードは以下の９つになる。

過酷：01803583-a、01041481-a、01507402-a

世界：09270894-n、14514805-n、07965973-n

ドラマ：07290278-n、06376154-n、07007945-n

結果として、以下の 15 個がベクトルの素性となる。

表 4.2:主要語義のみを利用した素性

| | |
|---|---|
| e1 | を |
| e2 | ドラマ |
| e3 | 過酷、世界 |
| e4 | ドラマ、。 |
| e5 | 01803583-a、01041481-a、01507402-a、09270894-n、14514805-n、07965973-n、07290278-n、06376154-n、07007945-n |

この例では全語義を用いた場合の素性が 18 個なのに対し、主要語義だけを利用した場合の素性は 15 個と 3 個少なくなった。素性になる名詞が増えたり、名詞の語義コードが多いものだと、この違いはさらに大きくなるだろう。

# 第５章　実験

## ５．１　実験概要

推定した主要語義だけを用いる手法の効果を調べるために、Naive Bayes を用いて、SENSEVAL2 の日本語辞書タスクで課題とされた動詞 50 単語に対する語義識別を行った。このタスクは各動詞に対して 100 問のテストデータがある。対象とした動詞は表 5.1 に示す。これらの単語は語義の頻度分布のエントロピーを考慮して選定されており、語義判別が容易なものから困難なものまでバランスよく選定されている。

比較として、語義コードを全く利用しな手法と、e5 の素性に全ての語義コードを利用した手法でも同様の語義識別を行った。

表 5.1:動詞 50 単語

| | | | | |
|---|---|---|---|---|
| ataeru | iu | ukeru | uttaeru | umareru |
| egaku | omou | kau | kakaru | Kaku_v |
| kawaru | kangaeru | kiku | kimaru | kimeru |
| kuru | kuwaeru | koeru | shiru | susumu |
| susumeru | dasu | chigau | tsukau | tsukuru |
| tsutaeru | dekiru | deru | tou | toru |
| nerau | nokosu | noru | hairu | hakaru |
| hanasu | hiraku | fukumu | matsu | matomeru |
| mamoru | miseru | mitomeru | miru | mukaeru |
| motsu | motomeru | yomu | yoru | wakaru |

## ５．２　実験結果

実験結果は表5.2のようになった。語義コードを利用することで識別の精度は向上するが、全ての語義コードを利用した手法の方が主要語義だけを用いる手法よりも精度が高いことがわかる。また、表5.3で示す11単語に関しては、主要語義だけを用いる手法の方が全ての語義を利用した手法よりも精度が高かった。総合的にみると、全語義を用いた方が精度は高くなると言える。

表 5.2:実験結果

| 見出し | 訓練事例数 | 語義なし | 全語義 | 主要語義 |
|---|---|---|---|---|
| ataeru | 116 | 0.67 | 0.63 | 0.67 |
| iu | 336 | 0.94 | 0.94 | 0.94 |
| ukeru | 357 | 0.50 | 0.61 | 0.60 |
| uttaeru | 70 | 0.85 | 0.83 | 0.82 |
| umareru | 65 | 0.64 | 0.66 | 0.67 |
| egaku | 70 | 0.63 | 0.65 | 0.63 |
| omou | 465 | 0.90 | 0.89 | 0.89 |
| kau | 87 | 0.83 | 0.84 | 0.85 |
| kakaru | 115 | 0.51 | 0.62 | 0.55 |
| Kaku_v | 135 | 0.68 | 0.68 | 0.67 |
| kawaru | 97 | 0.92 | 0.92 | 0.92 |
| kangaeru | 291 | 0.99 | 0.99 | 0.99 |
| kiku | 180 | 0.60 | 0.62 | 0.64 |
| kimaru | 117 | 0.96 | 0.96 | 0.96 |
| kimeru | 228 | 0.93 | 0.92 | 0.93 |
| kuru | 128 | 0.84 | 0.83 | 0.84 |
| kuwaeru | 109 | 0.89 | 0.90 | 0.89 |
| koeru | 119 | 0.77 | 0.79 | 0.78 |
| shiru | 246 | 0.97 | 0.97 | 0.97 |
| susumu | 112 | 0.46 | 0.46 | 0.44 |
| susumeru | 132 | 0.96 | 0.97 | 0.95 |
| dasu | 208 | 0.29 | 0.31 | 0.28 |
| chigau | 105 | 1.00 | 1.00 | 1.00 |
| tsukau | 270 | 0.97 | 0.96 | 0.97 |
| tsukuru | 183 | 0.59 | 0.63 | 0.61 |
| tsutaeru | 97 | 0.75 | 0.77 | 0.78 |
| dekiru | 75 | 0.81 | 0.81 | 0.81 |
| deru | 412 | 0.58 | 0.53 | 0.53 |
| tou | 71 | 0.68 | 0.63 | 0.65 |
| toru | 112 | 0.27 | 0.32 | 0.28 |
| nerau | 67 | 0.99 | 0.98 | 0.99 |
| nokosu | 98 | 0.79 | 0.79 | 0.79 |
| noru | 64 | 0.54 | 0.58 | 0.55 |
| hairu | 278 | 0.37 | 0.39 | 0.37 |
| hakaru | 84 | 0.92 | 0.92 | 0.92 |
| hanasu | 175 | 1.00 | 1.00 | 1.00 |
| hiraku | 224 | 0.80 | 0.90 | 0.81 |
| fukumu | 92 | 0.99 | 0.99 | 0.99 |
| matsu | 83 | 0.60 | 0.60 | 0.57 |
| matomeru | 93 | 0.80 | 0.81 | 0.80 |
| mamoru | 93 | 0.79 | 0.76 | 0.76 |
| miseru | 100 | 0.98 | 0.98 | 0.98 |
| mitomeru | 212 | 0.89 | 0.89 | 0.89 |
| miru | 408 | 0.72 | 0.73 | 0.72 |
| mukaeru | 93 | 0.89 | 0.89 | 0.89 |
| motsu | 267 | 0.50 | 0.55 | 0.51 |
| motomeru | 306 | 0.87 | 0.86 | 0.87 |
| yomu | 106 | 0.88 | 0.88 | 0.88 |
| yoru | 474 | 0.97 | 0.97 | 0.97 |
| wakaru | 178 | 0.90 | 0.90 | 0.90 |
| 平均 | | 0.7714 | 0.7801 | 0.7734 |

表 5.3:精度が上がった単語

| 単語 | 全語義 → 主要語義 |
|---|---|
| ataeru | 0.63 → 0.67 |
| umareru | 0.66 → 0.67 |
| kau | 0.84 → 0.85 |
| kiku | 0.62 → 0.64 |
| kimeru | 0.92 → 0.93 |
| kuru | 0.83 → 0.84 |
| tsukau | 0.96 → 0.97 |
| tsutaeru | 0.77 → 0.78 |
| tou | 0.63 → 0.65 |
| nerau | 0.98 → 0.99 |
| motomeru | 0.86 → 0.87 |

# 第6章　考察

実験では主要語義の個数を3つに設定した。この部分で最適な数を推定すれば、全語義を使ったものよりも良い精度が得られる可能性がある。しかし、数がある程度大きいと、結果は全語義を使ったものとほぼ同等となってしまうし、少なければ結果は語義を使わないものと同等になってしまう。

実験結果でderuの結果を見てみると、語義を利用することで精度が下がっている。deruの100問のテストの中で「語義なし」が正解し、「全語義」が誤ったものは7問あった。逆に「全語義」が正解し「語義なし」が誤ったものは2問だった。総合して「全語義」の方が0.05精度が低い。「全語義」が誤った上記7問を概観するとe5の語義コードの素性が非常に多く、本論文で示した文脈をベクトル化する際の問題が生じたと考えられる。この7問中、「主要語義」が誤っているものは4問であり、この部分で語義を主要語義に限定した効果は出ている。

実験結果で「全語義」よりも「主要語義」の方が劣っていた原因として、主要語義の推定が適切でなかったことが考えられる。本論文では名詞の主要語義を推定するのにコーパス内の語義の頻度を利用したが、これにより各単語の主要語義が推定できる保障はない。例えばある単語Aの頻度が非常に高く、またAの語義mがややマイナーであるような場合を考えてみる。このとき、ある多義語が語義mを持っていれば、それが主要語義とみなされてしまう。本論文では多義語の主要語義を1つに限定せずに複数使用することでこの問題に対処したが、主要語義を推定する手法としては簡易すぎたと思われる。精緻な手法を考案して、主要語義を推定する必要がある。

# 第7章　結論

本研究では、文脈をベクトル化する際に多義語の語義コードを全て素性として利用する場合の問題点を指摘し、多義名詞の主要語義の語義コードだけを利用する手法を提案した。また、実際にコーパスを利用して名詞の主要語義の語義コードを推定した。

実験ではSENSEVAL2の辞書タスクの動詞50単語の語義識別問題を題材に、Naive Bayes分類器を用いて本手法の効果を調べた。実験では"語義コードなし"、"全語義コード"、"主要語義コードのみ"を比較として行った。語義識別問題に語義を使う効果は認められたが、その語義を主要語義に限定することに効果は見られなかった。これは主要語義の推定方法が適切ではなかったと考えられる。

今後の課題は、語義コードからの主要語義の推定部分の改良である。また今回は名詞の語義コードのみを素性として利用したが、動詞や形容詞などへの拡張も必要である。

# 参考文献

[1]Francis Bond Hitoshi Isahara and Sanac Fujita and Kiyotaka Uchimoto and Takayuki Kuribayashi and Kyoko Kanzaki. Enhancing the Japanese WordNet. In *The 7th Workshop on Asian Language Resources, in conjunction with ACL-IJCNLP 2009*,2009.

[2]Kikuo Maekawa. Design of a Balanced Corpus of Contemporary Witten Japanese. In *Symposium on Large-Scale Knowledge Resources(LKR2007)*,pp.55-58,2007.

[3]白井清昭. SENSEVAL-2 日本語辞書タスク. Vol.10,No.3,pp.3-24.2003.

[4]田河生長 他. 確率統計. 大日本図書株式会社.

[5]藤井丈明. 語義識別の誤り原因の調査とオンザフライの類義語判定. 茨城大学工学部システム工学科 平成 15 年 卒業論文. 2003.

[6]正木裕一. 文書分類手法を利用した画像検索結果のフィルタリング. 茨城大学大学院理工学研究科 平成 17 年度 修士学位論文. 2005.

# 付録A　プログラムソースリスト

単語と語義コードから単語だけを取り出す。

```
import re

jpn = re.compile('jp')

f = open("./word-means.txt", "r")
lines = f.readlines()
f.close()

jp_words = []
for line in lines:
        if not jpn.findall(line):
                jp_words.append(line)

f = open("./jp-words.txt", "w")
f.writelines(jp_words)
f.close()
```

単語と語義コードから語義コードだけを取り出す。

```
import re

jpn = re.compile('jp')

f = open("./word-means.txt", "r")
lines = f.readlines()
f.close()
n = len(lines)

meanlist = []
for line in lines:
        if jpn.findall(line):
                meanlist.append(line)
meanlist = set(meanlist)

f = open("./means.txt", "w")
f.writelines(meanlist)
f.close()
```

語義コードと使われている単語をリストにする。

```
import re

jpn = re.compile('jp')

f = open("./word-means.txt", "r")
lines = f.readlines()
f.close()
n = len(lines)

mean_words = {}
mean_words2= {}
i = 0
while i < n:
        w = lines[i]
        i += 1
        while i < n: #this condition is necessary, 1 makes condition i=n to work.
                if jpn.findall(lines[i]):
                        if lines[i] not in mean_words:
                                mean_words[lines[i]] = w
                                mean_words2[lines[i]]= w.rstrip("Wn")
                        else:
                                mean_words[lines[i]] += w
                                mean_words2[lines[i]]+= "Wt" + w.rstrip("Wn")
                        i += 1
                else:
                        break

f = open("./mean-words.txt", "w")
for k,v in mean_words.iteritems():
        f.write(k)
        f.write(v)
f.close()
```

```
f = open("./mean-words2.txt", "w")
for k, v in mean_words2.iteritems():
        f.write(k)
        f.write(v + "Wn")
f.close()
```

白書の単語を取り出す。

```
import os
import re

f = open("./jp-words.txt", "r")
words = f.readlines()
f.close()

wordnet_words = []
for word in words:
        wordnet_words.append(word.rstrip())

os.chdir("./sentence")
files = os.listdir("./")
nouns = "";
for file in files:
        f = open(file, "r")
        lines = f.readlines()
        f.close()

        for line in lines:
                nouns += line.rstrip("¥n")
nouns = nouns.split(" ")

nouns_in_wordnet = []
for word in nouns:
        if word in wordnet_words:
                nouns_in_wordnet.append(word+"¥n")

f = open("../nouns-in-wordnet.txt", "w")
f.writelines(nouns_in_wordnet)
f.close()
```

単語をカウントする。

```
f = open("./nouns-in-wordnet.txt", "r")
lines = f.readlines()
f.close()

words = []
for line in lines:
        words.append(line.rstrip())

wordlist = set(words)

words_count = {}
for word in wordlist:
        words_count[word] = words.count(word)

f = open("./words-count.txt", "w")
for k, v in words_count.iteritems():
        f.write(k + "¥t" + str(v) + "¥n")
f.close()
```

日本語 WordNet から名詞とその語義コードを取り出す。

```
import re

writtenform = re.compile("writtenForm")
synset = re.compile("synset")

f = open("./jpn_wn_lmf.xml", "r");
lines = f.readlines()
f.close()
n = len(lines)

i = 0
wordlist = []
while i < n:
        if writtenform.findall(lines[i]):
                (w1,w2,w3,w4,w5) = lines[i].split('₩'')
                wordlist.append(w2+"₩n")
        elif synset.findall(lines[i]):
                (w1,w2,w3,w4,w5) = lines[i].split('₩'')
                wordlist.append(w4+"₩n")
        i += 1

f = open("./word-means.txt", "w")
f.writelines(wordlist)
f.close()
```

語義コード順位表を作る。

```
import re

jp = re.compile("jp")

f = open("./word-means.txt", "r")
lines = f.readlines()
f.close()
n = len(lines)

word_means = {}
i = 0
while i < n:
        w = lines[i].rstrip("¥n")
        i += 1
        while i < n:
                if jp.findall(lines[i]):
                        if w not in word_means:
                                word_means[w] = lines[i]
                        else:
                                word_means[w] += lines[i]
                        i += 1
                else:
                        break

f = open("./words-count.txt", "r")
lines = f.readlines()
f.close()

word_count = {}
for line in lines:
        k_v = line.split("¥t")
        k, v = k_v[0], int(k_v[1])
        word_count[k] = v
```

```
for word in word_means.items():
        if word[0] not in word_count:
                del word_means[word[0]]

word_mean_count = {}
mean_total_count = {}
for k, v in word_count.iteritems():
        means = word_means[k]
        means = means.rstrip("₩n").split("₩n")
        n = len(means)
        for i in range(n):
                if means[i] not in mean_total_count:
                        mean_total_count[means[i]] = v
                else:
                        mean_total_count[means[i]] += v
                means[i] += "₩t"+str(v)+"₩n"
        word_mean_count[k] = "".join(means)

f = open("./result1.txt", "w")
for k, v in word_mean_count.iteritems():
        f.write(k+"₩n")
        f.write(v)
f.close()

f = open("./result2.txt", "w")
for k, v in mean_total_count.iteritems():
        f.write(k+"₩t"+str(v)+"₩n")
f.close()

f = open("./mean-words2.txt", "r")
lines = f.readlines()
f.close()

mean_words = {}
n = len(lines)
i = 0
```

```
while i < n:
        mean_words[lines[i].rstrip("₩n")] = lines[i+1]
        i += 2

f1 = open("./result3.txt", "w")
f2 = open("./result4.txt", "w")
f3 = open("./result5.txt", "w")
f4 = open("./result6.txt", "w")

for word in word_means:
        means = word_means[word]
        means = means.rstrip("₩n").split("₩n")
        n = len(means)
        for i in range(n):
                max = i
                for j in range(i+1, n):         # "j=i+1, for j in range(n):" is
wrong!
                        if mean_total_count[means[j]] >
mean_total_count[means[max]]:
                                max = j
                if i != max:
                        means[i], means[max] = means[max], means[i]
        f1.write(word + "₩n")
        for i in range(n):
                 f1.write(means[i] + "₩t" + str(mean_total_count[means[i]]) +
"₩n")

        f2.write(word + "₩n")
        f2.write(means[0] + "₩t" + str(mean_total_count[means[0]]) + "₩n")
        f3.write(word + "₩n")
        f3.write(means[0] + "₩n")
        f4.write(word + "₩n")
        f4.write(means[0] + "₩t" + mean_words[means[0]])
        if n > 1 and mean_total_count[means[1]] > mean_total_count[means[0]] *
0.4:
                f2.write(means[1] + "₩t" + str(mean_total_count[means[1]]) +
"₩n")
```

```
                f3.write(means[1] + "₩n")
                f4.write(means[1] + "₩t" + mean_words[means[1]])
        if n > 2:       #if a mean of word is greater than threshold, and it
also be remained.
                i = 2
                threshold = mean_total_count[means[0]] * 0.7
                while i < n and mean_total_count[means[i]] > threshold:
                        f2.write(means[i] + "₩t" +
str(mean_total_count[means[i]]) + "₩n")
                        f3.write(means[i] + "₩n")
                        f4.write(means[i] + "₩t" + mean_words[means[i]])
                        i += 1
        f4.write("₩n")

f1.close()
f2.close()
f3.close()
f4.close()
```