ブートストラップ法による決定リストのメタパラメータの推定

執筆者：脇坂恭志郎

指導教官：新納 浩幸

平成16年3月2日

目 次

第1章 序論 **5**

1.1 研究概要 5

1.2 本論文の構成 5

第2章 メタパラメータの推定 **6**

2.1 ブートストラップ法 6

2.2 交差検定法 7

第3章 決定リスト **9**

3.1 決定リスト 9

3.2 決定リストの多義語問題への適用 11

3.2.1 証拠の設定 11

3.2.2 頻度の獲得 12

3.2.3 予測力の算出 12

3.2.4 決定リストの作成 13

3.3 決定リストの利用 13

3.4 その他の学習手法 14

3.4.1 Naive Bayes 法
（低機能学習手法） 14

3.4.2 サポートベクトルマシン法
（高機能学習手法） 15

第4章 実験 **17**

4.1 実験の設定 17

4.2 実験手順 18

4.2.1 決定リストの作成 18

4.2.2 メタパラメータの推定 22

第5章　結果　23

第6章　考察　28

第7章　まとめ　29

第8章　謝辞　30

付 録A プログラムソースリスト
（決定リスト作成）　31

付 録B プログラムソースリスト
（ブートストラップ法）　38

付 録C プログラムソースリスト
（シェルスクリプト）　49

図目次

4.1 対象単語（名詞） 18
4.2 対象単語（動詞） 18
4.3 多義語「地方」の二つの語義 19
4.4 多義語「地方」のトレーニングデータ 19
4.5 多義語「地方」のテストデータ 21

表目次

3.1 "plant"における決定リストの例 10

4.1 多義語「地方」の決定リスト 21
4.2 二つのメタパラメータの可変値 22

5.1 各単語の正解率移行（名詞） 23
5.2 各単語の正解率移行（動詞） 25
5.3 各単語の正解数移行 27

6.1 各々の正解率 28

第1章　序論

1.1　研究概要

ある問題を帰納学習の手法から解決しようとした場合、その手法を適用する前にあらかじめ決めておかねばならないパラメータが存在する。これをメタパラメータという。最適なメタパラメータを用いると、機能の低い機械学習手法でも精度の高い規則を学習できる可能性がある。[1]

一方、高機能な機械学習手法から得られる規則の精度は高いが、複雑で人間可読でなく、なにより後の改良も行なえない。

そこで、これらの特徴を踏まえ、本研究では「人間可読な決定リスト（低機能な学習手法）とメタパラメータの高精度な推定」が語義選択において高機能学習手法と同等以上の機能を発揮できるように目指していく。

メタパラメータの推定にブートストラップ法を用い、各単語に対して最適な決定リストが得られるようにした。また、メタパラメータを固定した場合の正解率とブートストラップ法によってメタパラメータを推定した場合の正解率を求め、その比較を行なった。

1.2　本論文の構成

本論文は、メタパラメータ推定の際に用いたブートストラップ法と、その他の推定手法として用いられる交差検定法について（第２章)、語義識別に使用した決定リストと、その他の学習手法であるナイブベイズ法とサポートベクトルマシン法について（第３章)、実験（第４章）及び結果（第５章)、その考察（第６章）からまとめ（第７章）へと進む。

また、巻末にはプログラムのソースリストを添付した。

第2章　メタパラメータの推定

ここでは本研究においてメタパラメータを推定する際に用いた「ブートストラップ法」と、同じく有用な手法となりうる「交差検定法」の理論を記す。

2.1　ブートストラップ法

任意の統計量を推定する手法として、ブートストラップ法 (bootstrap method) がある [2]。ブートストラップ法には、推定値分散が小さくなる、すなわちパターン集合 X の変動に対して安定するという特長がある。ブートストラップ法は統計量に応じていろいろな推定の仕方をするが、その基本は X からの復元抽出（取り出してはもとに戻す抽出法）にある。以下では e_λ の推定として、ブートストラップ法がどのように応用されているのかについて説明する。ただし、以下では実用面での使用法を重視した直感的な説明にとどめる。

いま、e_λ の推定をする際、X を学習とテストの両方に使用して、推定値 $\hat{e_\lambda}$ を得たとする。学習パターン集合がテストにも利用されたわけだから、明らかに得られた誤識別率の推定値は、真値よりも小さくなるはずである。このときそのずれを、

$$R = e_\lambda - \hat{e_\lambda}$$

で表すと、もしRの値を何らかの方法で推定することが出来れば、

$$e_\lambda = R + \hat{e_\lambda}$$

と求めることが出来る。ブートストラップ法では、X からn回の復元抽出により疑似パターン集合 $X^* = \chi_1^*, \chi_2^*, \cdots, \chi_n^*$ を生成し、この疑似パターン集合を用いて R の推定値を求めようという方法である。

すなわち、ブートストラップ法ではこの疑似のパターン集合 X^* を学習パターンとみなし、元のパターン集合 X をテストパターン集合とみなすことにより、$R = e_\lambda - \hat{e_\lambda}$ を

$$R^* = e^*_\lambda - \hat{e^*_\lambda}$$

と書き換える。ここでは e^*_λ は X^* を学習に、X をテストにして得られた e_λ の推定値を表し、$\hat{e^*_\lambda}$ は X^* を学習とテストの両方に用いて得られた e_λ の推定値を表す。ただし、R^* を特定のサンプリングの影響をなくすためB個の疑似パターン集合 $\chi^*_1, \chi^*_2, \cdots, \chi^*_n$ を生成し、そのおのおのについて $R^{*1}, R^{*2}, \cdots, R^{*B}$ を求め、それらの平均値を R^* とする。

ブートストラップ法による e_λ の推定法を整理すると以下のようになる。

Step 1. 元のパターン集合 X で、モデル λ の識別機を設計した後、同じ X で誤識別率を算出し、その値を $\hat{e_\lambda}$ とする。

Step 2. $b = 1, \cdots, B$ のおのおのについて以下を行なう。元のパターン集合 X からn回の復元抽出により X^{*b} を生成し、X^{*b} をパターン集合としてモデル λ の識別機を設計した後、X で誤識別率を算出したものを e^{*b}_λ とし、X^{*b} で誤識別率を算出したものを $\hat{e^{*b}_\lambda}$ とする。これらの値を用いて、

$$R^{*b} = e^{*b}_\lambda - \hat{e^{*b}_\lambda}$$

を計算する。

Step 3. Step 2で得られたB個の $R^{*1}, R^{*2}, \cdots, R^{*B}$ の平均値を R^* とすると、求めるべき推定値は $\hat{e_\lambda} + R^*$ として得られる。

2.2 交差検定法

交差検定法 (cross-validation method) による e_λ の推定では、X のすべての要素が学習とテストに使用されるようになっている。具体的には、まず X をm個のグループ $\chi_1, \chi_2, \cdots, \chi_m$ に分割する。このとき、各グループのパターン数は n/m となっている。そして、X_i で語義識別を算出する。この手順を $i = 1, 2, \cdots, m$ の全てについて行ない、得られたm個の語識別率の平均値を求め、それを e_λ の推定値とする。

最も単純かつよく用いられている分割は、要素数が1となる分割である。すなわち、$(\chi - x_i)$で学習し、x_iでテストするという手順を$i = 1, 2, \cdots, n$について行ない、n回のテストによって得られた誤識別率をe_λの推定値とする。この方法は、一つ抜き法(Leace-one-out method)と呼ばれる。

第3章　決定リスト

3.1　決定リスト

決定リストとは従来より語義選択問題に適用されてきたクラス分類手法の一つである。

特定の辞書に基づいて意味タグ（語義）が付与されたコーパスであれば、品詞付けと同様の手法により語義付与が可能かも知れないが、意味タグつきコーパスを作成するコストは非常に大きい。そこでYarowskyは、語義が付与された少数の初期データ（seed collocation）から、教師なし学習により語義選択のための確率モデルを作製する方法を提案した。その確率モデルが決定リストである。

決定リストは図に示すようなif-thenのルールを学習する手法と言える。

```
if(直前の証拠 == "の") return class1;
if(直前の証拠 == "は") return class2;
 ...
return class1
```

図から分かるように、決定リストの規則の表現形式は容易であり、直感的で分かりやすい構造をしている。語義の選択に影響を与えるための証拠（*evidense*）がリスト状に並んでおり、その順番は証拠の強さ（予測力）の順番で降順にソートされたものである。証拠としては、様々な単語共起の形（単語・品詞及びその位置関係）が考慮され、予測力はその証拠$evidense_i$のもとで、語義$sense_a$と語義$sense_b$が選ばれる確率との対数尤度比で表される。

$$\log(\frac{P(sense_a|evidence_i)}{P(sense_b|evidence_i)})$$

テキスト中の単語の語義は、文脈中に存在する様々な証拠から総合的に判断するのではなく、最も予測力の高い一つの証拠に基づいて決定される。例えば表に英

単語"plant"の語義選択に関する決定リストの例を示す。

表 3.1: "plant"における決定リストの例

共起単語（証拠）	予測力	語義
plant growth	10.12	LIVING
car(±k 語以内)	9.68	FACTORY
plant height	9.64	LIVING
union (±k 語以内)	9.61	FACTORY
equipment (±k 語以内)	9.54	FACTORY
assembly plant	9.68	FACTORY
nuclear plant	9.68	FACTORY
...	...	...

英単語"plant"には「植物(LIVING)」と「工場(FACTORY)」という２つの意味がある。この決定リストでは、"growth"や"height"という語が"plant"の直後にあれば「植物」を意味し、"car"や"union"が周辺(±k 語以内)にあれば「工場」を意味するとしている。

3.2 決定リストの多義語問題への適用

ここでの決定リストは語義選択に影響を与える文脈中の証拠を語義の予測力の順に並べたリストである [2]。決定リストの作成は概略以下の手順に従う。

3.2.1 証拠の設定

問題を解消するための証拠（文脈情報）を設定する。本研究では以下の６つの証拠を用いた。

・e1 直前の単語

・e2 直後の単語

・e3 前方の内容語２つまで

・e4 後方の内容語２つまで

・e5 e3 の分類語彙表の番号

・e6 e5 の分類語彙表の番号

例えば、語義判別対象の単語を「出す」として、以下の文を考える（形態素解析され各単語は原型に戻されているとする）。[3]

短い／コメント／を／出す／に／とどまる／た／。

この場合、「出す」の直前、直後の単語は「を」と「に」なので、'e1=を'、'e2=に' となる。次に、「出す」の前方の内容語は「短い」と「コメント」なので、'e3=短い'、'e3=コメント' の２つが作られる。またここでは句読点も内容語に設定しているので、「出す」の後方の内容語は「とどまる」「。」となり、'e4=とどまる'、'e4=。' が作られる。次に「短い」の分類語彙表の番号を調べると、3.19201 である。ここでは分類語彙表の 4 桁目と 5 桁目までの数値を取ることにした。つまり、'e3=短い' に対しては'e5=3192'、'e5=31920' が作られる。「コメント」は分類語彙表には記載されていないので、'e3=コメント' に対しては e5 に関する素性は作られない。次は「とどまる」の分類語彙表を調べるはずだが、ここでは平仮名だけで構成される単語の場合、分類語彙表の番号を調べないことにしている。これは平仮名だ

けで構成される単語は多義性が高く、無意味な素性が増えるので、その問題を避けた為である。もしも分類語彙表上で多義になっていた場合には、それぞれの番号に対して並列に全ての素性を作成する。

結果として、上記の例文に対しては以下の８つの素性が得られる。

e1=を, e2=に, e3=短い, e3=コメント,
e4=とどまる, e4=。, e5=3192, e5=31920,

3.2.2 頻度の獲得

クラス内の語義c_iと証拠evd_jとが共起する頻度$frq(c_i, evd_j)$をトレーニングデータから得る。

具体的な例を示す。例として多義語の名詞（地方）を用いた以下の例文を見る。

例文 「外国人の地方参政権付与に関する問題。」

例文からは「地方」に対する証拠として、

e1=の e2=参政権 e3=人 e5=1196 e5=11960 e5=1202 e5=1202 e3=外国 e5=1253
e4=参政権 e6=1340 e6=13401 e4=付与 e6=1377 e6=13770

が取り出される。以上のように証拠を取り出した後、その中で予測力が最高となる共起頻度$frq(c_i, evd_j)$を求める。

3.2.3 予測力の算出

証拠が生じている場合に語義がc_iである予測力$est(c_i, evd_j)$を以下のように定義する。

$$est(c_i, evd_j) = \log \frac{frq(c_i, evd_j)+\alpha}{sum-frq(c_i, evd_j)+\alpha}$$

最終的な決定リストを求めるには、予測力の強い順に証拠をソートする。なお、ここではsumを$\sum_i frq(c_i, evd_j)$と定義している。

上式に用いられている α はメタパラメータである。また、default という特別な証拠を設定し、$frq(c_i, default)$ をクラス c_i の総頻度とする。default は以下のように求める。

$$default = \log \frac{frq(c_i,default)+\alpha}{sum-frq(c_i,default)+\alpha}$$

3.2.4 決定リストの作成

$est(c_1, evd_j)$,$est(c_2, evd_j)$,$\cdots$,$est(c_n, evd_j)$ の中で最も値の大きな $est(c_k, evd_j)$ を取り出し、この w_k を証拠 evd_j が現れた時の解答とする。また、この時の予測力は $est(w_k, evd_j)$ である。

各 evd_j に対して、evd_j が現れた時の解答 w_{k_j} を求め、予測力 $est(w_{k_j}, evd_j)$ が高い順のリストを作成する。これが決定リストとなる。ただし、あまりに弱い予測力を持つ事項をリストに含めても、実際にそれが正しい選択を示しているかどうかは疑わしい。そこで、そのような無意味な事項をリストから除外する為に、先程設定した default よりも予測力 $est(w_{k_j}, evd_j)$ が低いものはリストから外す。当然ながらこの default はリストの最終におかれる。また総頻度がメタパラメータ h 以下のものもリストから外す。

3.3 決定リストの利用

実際に決定リストを用いて語義選択するためには、まず文中から予め用意してあるリスト中の単語 w を見つけ、「証拠の設定」で設定した w に対する証拠

$$E = (evd_1, evd_2, \cdots, evd_N)$$

を取り出す。

次に作成してある決定リストの最上位の証拠から順に、その証拠が先ほど取り出した証拠の集合 E に属するかどうか調べる。もし evd_j が属していれば、evd_j に対する解答 w_{k_j} が識別結果となる。

3.4 その他の学習手法

3.4.1 Naive Bayes法 （低機能学習手法）

語義識別における機械学習の手法は様々なものが報告されているが、本研究ではNaive Bayes法を用いる．その理由としては，Senseval2 の辞書タスクでは，Naive Bayes法が最も良い成績を収めたと報告されているためである．[?]

機械学習による学習規則を用いて語義識別を行なう場合、主に統計的手法を用いる．まず多義語の前後にある素性に着目する．多義語の前後にある素性を学習させ、文脈上にある素性が現われた場合、その素性のとき最も確率の高い語義を識別結果として返す．

ある事例 x が要素ベクトルとして以下の様に表せるとする。

$$x = (x_1, x_2, \cdots, x_m)$$

ここで x を構成する各要素は、多義語の前後にある素性を示している．

また x の分類先のクラスの集合を以下の様に表せるとする..

$$C = (c_1, c_2, \cdots, c_k)$$

ここで C を構成する各要素は、多義語の語義を示している．

文脈上にある要素ベクトル x が出現した場合、クラス c_i となる確率が最も高いものが識別結果となる。つまり語義識別を行なうには $P(c_i|x)$ を最大にする c_i を求めれば良い。ベイズの定理より、$P(c_i|x)$ は以下の様に表せる。

$$P(c_i|x) = \frac{P(c_i)P(x|c_i)}{P(x)}$$

ここで、$P(x)$ は不変であるので、$P(c_i|x)$ が最大となる確率を求めるには、$P(c_i)P(x|c_i)$ を最大にする c_i を求めれば良い。$P(c_i)$ は事後確率であるのでクラスの割合などから比較的容易に推測できる。しかし $P(x|c_i)$ は、クラス c_i が出現したとき、要素ベクトル x が現われる確率であるので推定が困難である。ここで

$$P(x|c) \approx \prod_{i=1}^{m} P(x_i|c)$$

と仮定する．この仮定により、要素ベクトルの全てを一度に考慮せず、$P(x_i|c)$ の総積を考えることで結果として $P(x|c)$ が推定できる。これを Naive Bayes 法という。

3.4.2 サポートベクトルマシン法 （高機能学習手法）

サポートベクトルマシン法は、空間を超平面で分割することにより2つの分類からなるデータを分類する手法である。このとき、2つの分類が正例と負例からなるものとすると、学習データにおける正例と負例の間隔（マージン）が大きいものほどオープンデータで誤った分類をする可能性が低いと考えられ、このマージンを最大にする超平面を求めそれを用いて分類を行なう。通常、学習データにおいてマージンの内部領域に小数の事例が含まれてもよいとする手法の拡張や、超平面の線形の部分を非線形にする拡張（カーネル関数の導入）がなされたものが用いられる。この拡張された方法は、以下の識別関数を用いて分類することと等価であり、その識別関数の出力値が正か負かによって二つの分類をすることができる。

$$f(\mathbf{x}) = sgn\left(\sum_{i=1}^{l} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

$$b = -\frac{\max_{i;y_{i=-1}} b_i + \min_{i;y_{i=-1}} b_i}{2}$$

$$b_i = \sum_{j=1}^{l} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i)$$

ただし、$\mathbf{x}$ は識別したい事例の文脈（素性の集合）を、$\mathbf{x}_i$ と $y_i (i = 1, \cdots, l, y_i \in \{1, -1\})$ は学習データの文脈と分類先を意味し、関数 sgn は、

$$sgn(x) = \begin{cases} 1 & (x \geq 0) \\ -1 & (otherwise) \end{cases}$$

であり、また、各 α_i は以下の制約のもと、$L(\alpha)$ を最大にする場合ものである。

$$0 \geq \alpha_i \geq C(i = 1, \cdots, l)$$

$$\sum_{i=1}^{l} \alpha_i y_i$$

$$L(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j K(\mathbf{x_i}, \mathbf{x_j})$$

また、関数 K はカーネル関数と呼ばれ、様々なものが用いられるが、論文＼citesen2j では以下の多項式のものが用いられている。

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^d$$

C, d は実験的に設定される定数である。ここで、 $\alpha_i > 0$ となる $\mathbf{x}_i$ は、サポートベクトルと呼ばれ、通常、$f(\mathbf{x}) = sgn\left(\sum_{i=1}^{l} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right)$ の和をとっている部分はこの事例のみを用いて計算される。つまり、実際の解析には学習データのうちサポートベクトルと呼ばれる事例のみしか用いられない。

サポートベクトルマシン法は分類の数が2個のデータを扱うもので、通常これにペアワイズ手法を組み合わせて用いることで、分類の数が3個以上のデータを扱うことになる。

ペアワイズ手法とは、 N 個の分類を持つデータの場合、異なる二つの分類先のあらゆるペア ($N(N-1)/2$ 個) を作り、各ペアことにどちらがよいかを2値分類器で求め、最終的に $N(N-1)/2$ 個の2値分類先の多数決により、分類先を求める方法である。

第4章　実験

本研究で行なった実験について説明する。

4.1　実験の設定

ここでは決定リストを用いて、SENSEVAL2の日本語辞書タスクで課題とされた名詞50単語、動詞50単語の多義語の語義判別を試みた。

SENSEVAL2の日本語辞書タスクは、単純な語義選択問題である。対象単語は50単語、動詞50単語の計100単語である。これら100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良く選定されている。トレーニングデータは1単語平均して名詞は177.4事例、動詞は172.4事例用意されている。またテストデータは各単語に対して100問のテストが用意されている。つまり名詞に対しては計5000問、動詞に対しても計5000問のテストが行なえる。

ただし、ラベルなし訓練事例はSENSEVAL2からは提供されていない。これには通常のテキストが使えそうだが、実際は制限がある。それはラベルつき訓練事例を作成した際に用いた単語辞書や品詞分類を合わせる必要があるからである。そのためここではＲＷＣテキストデータベース第２版に納められた毎日新聞95年度版の一年分の記事を利用して、ラベルなしの訓練事例を収集した。このデータはラベル付き訓練事例のもとになったデータであり、同一の形態素解析システムを用いて形態素解析されている。収集できたラベルなし訓練事例の数は1単語平均して名詞は7571.2事例、動詞は6571.9事例である。

以下、実験に使用する100単語を、図4.1と図4.2に示す。(ローマ字順)

間、頭、場合、地方、近く、中心、代表、電話、同日、現在、技術、午後、花、反対、今、意味、一般、一方、自分、時代、事業、時間、情報、開発、核、関係、気持ち、記録、子供、国内、言葉、交渉、前、目、民間、問題、物、胸、娘、男、精神、社会、市場、市民、少年、姿、対象、手、程度、疑い

図 4.1: 対象単語（名詞）

与える、違う、出す、出来る、出る、描く、含む、入る、計る、話す、開く、言う、かかる、書く、考える、買う、変わる、聞く、決まる、決める、越える、来る、くわえる、守る、まとめる、待つ、見る、見せる、認める、求める、持つ、迎える、狙う、残す、乗る、思う、知る、進める、進む、取る、問う、使う、作る、伝える、受ける、生まれる、訴える、分かる、読む、寄る

図 4.2: 対象単語（動詞）

4.2 実験手順

ここでは実際に本研究で行なった実験の手順を説明し、その結果を示す。

4.2.1 決定リストの作成

STEP1 共起頻度ファイルの作成トレーニングデータから決定リストを作成するのに必要な対数尤度比を計算するために、その前準備として証拠頻度を計算し、その結果を記述したファイルを作成する。また、前述のように、

- e1 直前の単語
- e2 直後の単語
- e3 前方の内容語２つまで
- e4 後方の内容語２つまで
- e5 e3 の分類語彙表の番号
- e6 e5 の分類語彙表の番号

と、表記に”e1,e2,e3,e4,e5,e6”のいずれかの位置情報を付与する。また、最後に語義のクラス番号を付ける。

「地方」という単語に対する二つの語義を図4.3に、そして実際のトレーニングデータを図4.4に示す。

class	意味
32849-0-0-1-0（語義１）	ある一定の地域
32849-0-0-2-0（語義２）	首府以外の地域

図 4.3: 多義語「地方」の二つの語義

e1=シベリア e2=イルクーツク e3=シベリア e3=・ e4=イルクーツク e4=旅客機 e6=1467 32849-0-0-1-0

e1=保守党 e2=議員 e3=保守党 e3=女性 e5=1204 e5=1550 e4=議員 e6=1240 e4=ジュリア 32849-0-0-2-0

e1=を e2=自治体 e3=施設 e5=1385 e5=13850 e3=治療 e5=1383 e5=13831 e4=自治体 e6=1270 e4=建設 e6=1382 e6=13822 32849-0-0-2-0

e1=西南 e2=や e3=西南 e5=1173 e5=11731 e3=多い e5=3195 e4=、 e4=ロシア 32849-0-0-1-0

e1=SS e2=都市 e4=都市 e6=1254 e4=総理 e6=1360 32849-0-0-2-0

e1=とりわけ e2=の e3=とりわけ e4=中小 e4=製造 e6=1386 32849-0-0-2-0

e1=は e2=中小 e3=本書 e5=1316 e5=13160 e3= e4=中小 e4=製造 e6=1386 32849-0-0-2-0

e1=、 e2=自治体 e3=、 e3=地方単独事業 e4=自治体 e6=1270 e4=公社 e6=1264 32849-0-0-2-0

e1=は e2=組織 e3=質疑 e5=1313 e5=13132 e4=組織 e6=1132 e4=代議員 32849-0-0-2-0

e1=、 e2=組織 e3=、 e3=これ e4=組織 e6=1132 e4=中心 e6=1174 e6=11742 32849-0-0-2-0

e1=源流 e2=から e3=源流 e5=1111 e5=11111 e3=アマゾン e4=運ぶ e6=2152 e6=21521 e6=2383 e4=材料 e6=1410 32849-0-0-1-0

e1=バタングランデ e2=の e3=バタングランデ e3=キロ e5=1196 e5=11961 e5=1196 e5=11961 e5=1196 e5=11962 e4=亜熱帯 e6=1528 e4=森林地帯 32849-0-0-1-0

… … … … … … … … … … … … … … … … … … … …

図 4.4: 多義語「地方」のトレーニングデータ

STEP2 決定リストファイルの作成

次に、トレーニングデータから対数尤度比を利用した予測力を算出し、その予測力の順にソートして決定リストを作成する。初めの試行ではメタパラメータを $\alpha = 0.1, h = 0$ に固定した。

例えば、多義語「地方」については以下のようになる。まず、「地方」に対応

するトレーニングデータを開き、二つの語義の総頻度を調べる為にそれぞれの頻度の統計を調べる。ここで、語義1及び語義2に対する共起頻度が97、74であったとすると、頻度の高い語義1を用いてdefaultの設定を行なう。その予測力は、

$$default = \log \frac{frq(\text{語義}1, default)+\alpha}{sum-frq(\text{語義}1, default)+\alpha} = 0.270614$$

となる。ここで予測力が最大になるものをその証拠の予測力とし、その時の語義を証拠に対する解答とする。ここでは証拠defaultに対して予測力1.30902、解答「語義1」が得られたことになる。

総頻度の予測力の計算が終ったら、一般の証拠の予測力の計算に移る。算出方法は総頻度と同様である。例えば証拠「e3=統一」に対し、「語義2」の事例が6だったとすれば、証拠「e3=統一」が出現した時の予測力は、

$$est(\text{語義}2, e3 = \text{統一}) = \log \frac{frq(\text{語義}2, e3=\text{統一})+\alpha}{sum-frq(\text{語義}2, e3=\text{統一})+\alpha} = 4.110874$$

となる。これより結果的に証拠「e3=統一」の予測力は4.110874で、解答は語義2であることが得られる。

このとき、先に導出したdefaultの予測力を下回る証拠については、意味のないリスト要素として決定リストには挿入しない。

全ての証拠について予測力の計算とリストへの挿入が終ったら、今度は予測力について降順にソートする。これが最終的な決定リストとなる。以上より、図4.1のような決定リストが得られる。

STEP3 決定リストファイルの利用

決定リストが作成されたら、テストデータから多義語についての正解判定を行なうことができる。多義語「地方」のテストデータを4.5に示す。テストデータもトレーニングデータと同様の形式で、一つの多義語に6種類の証拠が並べて記述されている。これらの証拠を使い、先ほど作成した決定リストから各証拠に対応する予測力とその解答を調べることで判定を行なった。

証拠	予測力	総頻度	クラス
e2=で	4.615121	10	32572-0-0-1-0
e3=)	4.262680	7	32572-0-0-2-0
e4=議員	4.110874	6	32572-0-0-2-0
e3=統一	4.110874	6	32572-0-0-2-0
…	…	…	…
e2=議会	4.110874	6	32572-0-0-2-0
e2=へ	4.110874	6	32572-0-0-2-0
e1=が	4.110874	6	32572-0-0-1-0
(default)	0.270614	—	32572-0-0-1-0

表 4.1: 多義語「地方」の決定リスト

chihou.000 e1=の e2=選挙 e3=今年 e5=1164 e5=11641 e3=一 e4=選挙 e6=1363 e4=

chihou.001 e1=・ e2=公聴会 e3=・ e3=中央 e5=1174 e5=11742 e4=公聴会 e6=1351 e6=13510 e4=経る e6=2152 e6=21521

chihou.002 e1=、 e2=分 e3=、 e3=もの e4=分 e6=1100 e6=1155 e6=11552 e6=1196 e6=11961 e6=1196 e6=11962 e6=1196 e6=11963 e6=1197 e6=11970 e6=1198 e6=11980 e6=1341 e6=1370 e6=13701 e4=強硬

chihou.003 e1=ベンガル e2=で e3=ベンガル e3=する e4=、 e4=乾季 e6=1162 e6=11620

chihou.004 e1=トスカーナ e2=の e3=トスカーナ e3=・ e4=メシーナ e4=町 e6=1196 e6=11961 e6=1254 e6=1255

chihou.005 e1=は e2=遊説 e3=首相 e5=1241 e5=12411 e4=遊説 e6=1313 e6=13135 e4=会見 e6=1352 e6=13520

chihou.006 e1= e2=組織 e3= e3=メド e4=組織 e6=1132 e4=代表 e6=1104

chihou.007 e1=の e2=組織 e3=ブロック e5=1198 e5=11981 e3=十 e5=1195 e5=11950 e4=組織 e6=1132 e4=代表 e6=1104

chihou.008 e1=、 e2=組織 e3=、 e3=もの e4=組織 e6=1132 e4=代表 e6=1104

chihou.009 e1=の e2=都市 e3=南部 e5=1173 e5=11731 e3=インド e4=都市 e6=1254 e4=バンガロール chihou.010 e1=た e2=議員 e3=参謀 e5=1242 e3=選挙 e5=1363 e4=議員 e6=1240 e4=「

… …

図 4.5: 多義語「地方」のテストデータ

4.2.2 メタパラメータの推定

次に先ほど固定したメタパラメータを可変にして、各々の単語に対する最適なメタパラメータを求めていく。

STEP1 復元抽出メタパラメータを求める手順として、まず正解つきデータ X からn回の復元抽出（取り出しては元に戻す抽出法）により疑似のパターン集合（トレーニングデータ）X^* を生成する。

実験では正解つきデータから8割の問題をランダムに復元抽出した、10個のトレーニングデータを作成した。

STEP2 正解率取得10個のトレーニングデータから学習された分類器（決定リスト）を用いて正解つきデータ X を識別する。その結果、10個の正解率が得られるが、この平均をメタパラメータ α, h の正解率とする。

値を様々に変化させ、最も正解率の高かったメタパラメータ α, h を推定値とする。各々の値の変域を図4.2に示す。

—	可変域
α	0.01, 0.05, 0.1, 0.2, 0.3,…,0.9
h	0, 1, 2, 3, 4, 5

表 4.2: 二つのメタパラメータの可変値

STEP3 単語の語義判別STEP2から得られた推定値を用いて、今度は正解つきデータ X から直接決定リストを作成する。それによってテストデータを語義判別した結果が、多義語の最終的な正解率となる。

第5章　結果

まず、メタパラメータを固定したうえで各単語の正解率を求めた。その後にブートストラップ法で求めたメタパラメータので正解率を求め、その対比を各単語別に以下に示す。

表 5.1: 各単語の正解率移行（名詞）

—	メタパラメータ固定	ブートストラップ法	変化率
間	81	78	-3
頭	54	62	+8
一般	85.6665	88.6665	+3
一方	84	83	-1
今	89	88	-1
意味	54	50	-4
疑い	100	100	0
男	88	92	+4
開発	62	62	0
核	69	71	+2
関係	80	82	+2
気持ち	64	63	-1
記録	70	73	+3
技術	83	96	+13
現在	94	97	+3
交渉	98	100	+2
国内	46	49	+3
言葉	40	41	+1
子供	63	68	+5
午後	87	86	-1

名詞（続き）			
—	メタパラメータ固定	ブートストラップ法	変化率
市場	78	81	+3
市民	54	62	+8
社会	82	81	-1
少年	87	93	+6
時間	53	53	0
事業	61	64	+3
時代	76	68	-8
自分	100	94	-6
情報	76	75	-1
姿	65	61.3333	-3.6667
精神	59	66	+7
対象	95	94	-1
代表	85	86	+1
近く	77	80	+3
地方	75	75	0
中心	94	98	+4
手	49	52	+3
程度	98	99	+1
電話	79	81	+2
同日	85	63	-22
花	93	99	+6
反対	97	97	0
場合	85	79	-6
前	88	88	0
民間	94	100	+6
娘	86	85	-1
胸	49	61.5	12.5
目	17	17	0
物	25	27	+2
問題	96	96	0

表 5.2: 各単語の正解率移行（動詞）

—	メタパラメータ固定	ブートストラップ法	変化率
与える	70	58	-12
言う	94	93	-1
受ける	58	61	+3
訴える	80	77	-3
生まれる	69	68	-1
描く	69	55	+14
思う	89	91	+2
買う	82	86	+4
かかる	57	60	+3
書く	69	67	-2
変わる	90	92	+2
考える	99	99	0
聞く	58	63	+5
決まる	96	96	0
決める	93	96	+3
来る	87	89	+2
くわえる	89	89	0
越える	77	79	+2
知る	96	96	0
進む	44	42	-2
進める	96	95	-1
出す	35	34	-1
違う	97	99	+2
使う	92.5	91.75	-0.75
作る	61	61	0
伝える	73	76	+3
出来る	70	77	+7
出る	52	53	+1
問う	64	63	-1

動詞（続き）			
—	メタパラメータ固定	ブートストラップ法	変化率
取る	29	34	+5
狙う	97	98	+1
残す	76.25	75.25	-1
乗る	59	67	+8
入る	41	41	0
計る	91	92	+1
話す	99	100	+1
開く	83	89	+6
含む	92	98	+6
待つ	52	49	-3
まとめる	77	77	0
守る	75	63.5	-11.5
見せる	95	95	0
認める	88	88	0
見る	75	69	-6
迎える	89	93	+4
持つ	52	49	-3
求める	87	87	0
読む	86	89	+3
寄る	97	97	0
分かる	90	90	0
(全体)			
名詞	74.99	76.11	+1.12
動詞	76.46	76.95	+0.49

表5.1と表5.2より、個々の多義語ではそれぞれの正解率の変化が異なったものの、若干ではあるがブートストラップ法を用いた手法の方が正解率が上回っていることが分かる。各々の単語のブートストラップ法適用前と適用後での正解率移行状況を、表に示した。

表 5.3: 各単語の正解数移行

—	名詞	動詞
正解率上昇	27 個	25 個
正解率不変	8 個	10 個
正解率下降	15 個	15 個

第6章　考察

今回の実験ではブートストラップ法より導出したメタパラメータを用いてテストデータを識別したが、適当な値にメタパラメータを設定した時よりも若干の差ではあるが全体的に正解率が向上した。

しかし、中には固定したメタパラメータよりも正解率が減少したものが全100単語中30個もあった。つまり、本手法はある単語については効果があるが、ある単語については逆効果になっているといえる。これはテストデータとトレーニングデータの傾向が異なったために、最適な決定リストを得ることが出来なかったからであると考えられる。理想的な解決策として、二つのデータの偏りをなくす等の方法が挙げられる。

表 6.1: 各々の正解率

—	名詞	動詞
Naive Bayes 法	78.2	79.8
サポートベクトルマシン法	78.0	78.6
決定リスト	74.9	77.2

表6.1に、ENSEVAL2の辞書タスクで報告されている各学習手法の正解率を示す。この結果と比較すると、本実験での手法は決定リストの効果を若干上げることが出来たが、他の手法と比べて数%下回っていることが分かる。

第7章　まとめ

本研究の目的は、ブートストラップ法を実装することよって決定リストのメタパラメータを高精度に推定し、高機能な学習手法と同等以上の語義識別を行なうことにある。

結果、僅かではあったが正解率が上昇したが、高機能学習手法には今一つ及ばなかった。

ブートストラップ法による正解率の上昇を更に顕著なものとするためには、考察で挙げたような改善を行なう必要がある。また、決定リストを作成する際に必要な膨大な計算時間を減らすことが出来なかったので、何かしらの工夫をして計算量を減らすことも今後の課題となる。

第8章　謝辞

本研究を進めるにあたって、理論の御指導及び論文作成における多大な御助言を賜わりました新納 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。

さらに、様々な御協力を頂きました岩崎 唯史 教官（茨城大学工学部システム工学科)、また、同研究室の紺野 憲一 氏、結城 隆 氏、藤井 丈明 氏、大城 亜里沙 氏、時田 陽一氏に深く感謝致します。

付 録A　プログラムソースリスト
（決定リスト作成）

```
//////////////////////////////////////////
//sotuken12.c
//////////////////////////////////////////


//default を求める。

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define LINESIZE  256

#define TBSIZE    150
#define NAMESIZE  50
#define WORDSIZE  100
void quit(char *);

int main(int argc, char *argv[])
{
  FILE *f1,*f2;
  char buf[LINESIZE], nametable[TBSIZE][LINESIZE],word1[WORDSIZE],
word2[WORDSIZE],word3[WORDSIZE],str[TBSIZE][LINESIZE],*c,*p;
```

```
//nametable[][]は1次元配列の集合

 int  a,b,r,w,i,j,k,l,flag;
 int  counttb[TBSIZE], lastid = 0, counter=0;
 double z;


  if(argc != 2) quit(" 引数の数が違う prog datafile ");
  if((f1 = fopen(argv[1],"r")) == NULL) quit("ファイルが開けない");

  while(fgets(buf,LINESIZE,f1) != NULL) {

     i=j=k=flag=0;
     strcpy(word1,"");
     strcpy(word2,"");
     strcpy(word3,"");

      while(buf[j]!='\0')
       {
       word2[i]=buf[j];
       if(word2[i]==' ')
          {
          word2[i+1]='\0'; //文字列の最後に\0を追加
          i=-1;
          flag=1;

         strcpy(word1,word2);
           }
        if(word2[i]=='\n' && flag==1)
          {
         flag=0;
         counter++;

         strcpy(word3,word1);

          i=-1;
```

```
          }
        i++;
        j++;

      }
 c=word3;

for(k=0;k<lastid;k++)
       {
        if(c==NULL) break;
        if (strcmp(str[k],word3) == 0)
           {
          counttb[k]++;
          break;
            }
        }

  if(k==lastid)
        {
         strcpy(str[k],word3);

         counttb[k]=1;
          lastid++;
        }


  }

for(l=0;l<lastid;l++) {                    //最大頻度 MAX のクラスを記憶
  if(a<counttb[l]) {
    a=counttb[l];
    b=l;
   }
  }
z=log((counttb[b]+0.01)/(counter-counttb[b]+0.01));
printf("%f %s \n",z,str[b]);
```

```
 printf("%d番目  %d個 / 全%d個\n",b+1,counttb[b],counter);

  if((f2 = fopen("def","w")) == NULL) quit("ファイルが開けない");
  fprintf(f2,"%f %s\n",z,str[b]);
  if((w = fclose(f2)) == -1) quit("ファイルが閉じれない");


 if((r = fclose(f1)) == -1) quit("ファイルが閉じれない");
}

void quit(char *s)
{
  puts(s); exit(1);
}
```

```
////////////////////////////////////////////////
//tyukan2.c
////////////////////////////////////////////////

//決定リストを作成する。

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void quit(char *);
int skip_space(char [],int);
int get_string(char [],char [],int);
int get_integer(char [],int *,int);
```

```
int main(int argc, char *argv[])
{
   FILE *f1,*f2;
   char buf[2000],buf2[128],word[100],str[200],str2[200],cls[200];
   int  class,a,r,n,i,sum,h,pos,th;
   double  c,d, alpha;


   //  if(argc != 4) quit("引数の数が違う");
  if(argc != 5) quit("引数の数が違う");

  alpha = atof(argv[3]);

  if((f1 = fopen(argv[1],"r")) == NULL) quit("ファイルが開けない");
  if((f2 = fopen(argv[2],"r")) == NULL) quit("ファイルが開けない");

  //  while(fgets(buf2,128,f2) != NULL) {
      fgets(buf2,128,f2);
      sscanf(buf2,"%lf %s",&d,cls);
//    d = atof(buf2);
      //        }
printf("default %f  %s\n",d,cls);

  while(fgets(buf,2000,f1) != NULL) {
     a=c=pos = 0;
     pos = skip_space(buf,pos);
     pos = get_string(buf,word,pos);          //素材
     pos = skip_space(buf,pos);
     pos = get_integer(buf,&sum,pos);         //総頻度
    strcpy(str2,word);
//printf("%s ",word);     printf("その数 %d\n",sum);
     while(pos != -1) {
        pos = skip_space(buf,pos);
        pos = get_string(buf,word,pos);       //クラス
```

```
        pos = skip_space(buf,pos);
        pos = get_integer(buf,&h,pos);          //頻度
//printf("次の列  %s\n",word);   printf("その数  %d\n",h);

  if(a<h)  {
       a=h;                              //h の最大値を a に代入
       strcpy(str,word);
      }
  }

    c = log((sum+alpha)/(sum-a+alpha));
    th = atoi(argv[3]);
    if((c > d) && (sum > th)) {
                //c が def より小さく sum が th 以下なら表示しない
       printf(" %s ",str2);
       printf(" %f  %s    %d\n",c,str,sum);
      }

 }
  if((r = fclose(f1)) == -1)  quit("ファイルが閉じれない");
  if((r = fclose(f2)) == -1)  quit("ファイルが閉じれない");
}



int skip_space(char buf[],int pos)            //空白
{
   while(buf[pos] == ' ') pos++;
   return pos;
}


int get_string(char buf[],char word[],int pos)    //文字
{
   int i = 0;
   char c;
```

```
    if (pos == -1) return -1;
    if (buf[pos] == '\n') return -1;
    for(     ;((c = buf[pos]) != ' ') && (c != '\n');pos++) {
       word[i] = c;   i++;          //空白でかつ改行がない限り処理続行
    }
    word[i] = '\0';
    return pos;
}


int get_integer(char buf[],int *h,int pos)        //頻度
{
    int i = 0;
    char c,num[10];

    if (pos == -1) return -1;
    if (buf[pos] == '\n') return -1;
    for(      ;((c = buf[pos]) != ' ') && (c != '\n');pos++) {
       num[i] = c;   i++;          //空白でかつ改行でない限り処理続行
    }
    num[i] = '\0';   *h = atoi(num);
    return pos;
}



void quit(char *s)
{
  puts(s); exit(1);
}
```

付 録B　プログラムソースリスト（ブートストラップ法）

```
////////////////////////////////////////////////
//randam3.c
////////////////////////////////////////////////

//ランダムに抽出する。

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>


#define LINESIZE  1000

#define TBSIZE    300
#define NAMESIZE  100
#define WORDSIZE  300
void quit(char *);



int main(int argc, char *argv[])
{
  FILE *f1,*f2;
  char buf[LINESIZE],*p;
  int  r,w;        //r,w(ファイル用)
  int  n,j,k,m;    //j(一文字読み込み), n(行数), k(nの0.8倍), z(乱数用)
```

```
  int  str[TBSIZE];

  char  str2[TBSIZE][LINESIZE],word3[WORDSIZE],buf2[LINESIZE];
  int  flag,x,y,z;

  int  a,b;

  time_t t;

  if(argc != 3) quit(" 引数の数が違う prog datafile ");
  if((f1 = fopen(argv[1],"r")) == NULL) quit("ファイルが開けない");

  n=-1;
  j=k=0;

  t = time(NULL);
  srand((unsigned int)t);

  while(fgets(buf,LINESIZE,f1) != NULL) {
     while(buf[j]!='\n')
       {
         j++;
       }
     j=0;
     n++;
   }
     k=0.8*n;
/*     printf("元データ行数%d, ランダム抽出数%d \n",n,k); */

     str[0]='\0';

     for(m=0; m<k; m++){
        str[m]=rand()%n;
/*        printf("%d ",str[m]); */
       }
```

```
/*    printf("\n"); */

    if((r = fclose(f1)) == -1) quit("ファイルが閉じれない");



  if((f1 = fopen(argv[1],"r")) == NULL) quit("ファイルが開けない");
  while(fgets(buf2,LINESIZE,f1) != NULL) {

   x=y=flag=0;

     while(buf2[x]!='\0')
        {
        word3[y]=buf2[x];

        if(buf2[x] == ' '){
           flag=1;
          }

        if(buf2[x] == '\n' && flag==1){
           word3[y+1]='\0';
           strcpy(str2[z],word3);
//           printf("%s\n",str2[z]);
             y=-1;
             flag=0;
             z++;
           }
          x++;
          y++;
        }
    }

  if((r = fclose(f1)) == -1) quit("ファイルが閉じれない");
```

```
  if((f2 = fopen(argv[2],"w")) == NULL) quit("ファイルが開けない");

     for(a=0; a<k; a++){
       b=str[a];
/*       fprintf(f2,"%s\n",str2[b]); */
         fprintf(f2,"%s",str2[b]);
     }
/*  printf("\n"); */


  if((w = fclose(f2)) == -1) quit("ファイルが閉じれない");


}

void quit(char *s)
{
  puts(s); exit(1);
}
```

```
////////////////////////////////////////////////////
//hantei1-6.c
////////////////////////////////////////////////////

//語義判別を行なう

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#define LINESIZE  1000

#define TBSIZE    1000
#define WORDSIZE  500
void quit(char *);                          //10

int main(int argc, char *argv[])
{
  FILE *f1,*f2;
  char buf1[LINESIZE],buf2[LINESIZE],class[TBSIZE][LINESIZE],
   word1[WORDSIZE],word2[WORDSIZE],word3[WORDSIZE],word4[WORDSIZE],
   word5[WORDSIZE],
   str1[TBSIZE][LINESIZE],str2[TBSIZE][LINESIZE],str3[TBSIZE][LINESIZE],
   *p;

 int  r,i,j,k,m,l,n,flag;
 double  pow[LINESIZE],kazu[LINESIZE],maxp[LINESIZE],a,y;
 int  x;

  if(argc != 3) quit(" 引数の数が違う prog datafile ");

  if((f1 = fopen(argv[1],"r")) == NULL) quit("ファイルが開けない");
  if((f2 = fopen(argv[2],"r")) == NULL) quit("ファイルが開けない");



//決定リストからデータを獲得

   i=x=0;
  word1[0]=word2[0]=word3[0]=word4[0]='\0';

  while(fgets(buf1,LINESIZE,f1) != NULL) {

        sscanf(buf1,"%s %s %s %s",word1,word2,word3,word4);
```

```
        pow[i] = atof(word2);
        strcpy(str1[i],word1);
        strcpy(str2[i],word3);

//      printf("%d %s %lf %s\n",i,str1[i],pow[i],str2[i]);
//                                  単語、予測力、クラス

        i++;                    //決定リストの数
      }

  i--;



//トレイニングデータからデータを取得

  l=m=maxp[0]=0;

  while(fgets(buf2,LINESIZE,f2) != NULL) {

   j=k=0;
   word5[0]='\0';

     while(buf2[j]!='\0')
      {
      word5[k]=buf2[j];

      if(word5[k]==' ')                 //空白がきたら
         {
         word5[k]='\0';
         k=-1;
         flag=1;

         strcpy(str3[l],word5);
```

```
                for(x=0; x<=i; x++){                          //予測力の代入
                    if(strcmp(str1[x],str3[l])==0){
                       strcpy(class[m],str2[x]);
                       kazu[m]=pow[x];  break;
                        }
                    }

                    if(x==i){
                       strcpy(class[m],str2[x]);
                       kazu[m]=pow[x];  break;
//                       printf("!!!!!!!!!");
                    }


                  if(maxp[m]<=kazu[m])
                            maxp[m]=kazu[m];              //最大の予測力を記憶

                  l++;
                 }


              if(word5[k]=='\n' && flag==1)                  //改行がきたら
                {
               word5[k]='\0';
               strcpy(str3[l+1],"\0");
//               printf("%d %lf %s %s ",m,maxp[m],str3[l-1],class[m]);


              if(strcmp(str3[l-1],class[m])==0){             //クラスの判定
//                 printf("...TRUE! \n");
                y++; }
//               else
//                 printf("...FALSE \n");
```

```
            m++;
            flag=x=l=0;
              }

          k++;
          j++;
           }
       }

    a = 100*y/m;                                    //正解率の計算表示
//    printf("\n%d %d %f 正解率%f %\n\n",m,i,y,a);
    printf("%d %d %f %f\n",m,i,y,a);

  if((r = fclose(f1)) == -1) quit("ファイルが閉じれない");
  if((r = fclose(f2)) == -1) quit("ファイルが閉じれない");
}

void quit(char *s)
{
  puts(s); exit(1);
}
```

```
////////////////////////////////////////////////////
//heikin.c
////////////////////////////////////////////////////

//疑似データ群から平均正解率を求める

#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>


#define LINESIZE  1000

#define TBSIZE    300
#define WORDSIZE  50
void quit(char *);                              //10


int main(int argc, char *argv[])
{
  FILE *f1;
  char buf1[LINESIZE],word1[WORDSIZE],word2[WORDSIZE],word3[WORDSIZE],
  word4[WORDSIZE],a[WORDSIZE],h[WORDSIZE],a_max[WORDSIZE],h_max[WORDSIZE],*p;

  int  r,i,j,k,m,flag;
  double  pow[LINESIZE], sum, avg, max;                    //20

  i=m=sum=avg=0;
  pow[0]=word1[0]=word2[0]=word3[0]=word4[0]='\0';

  if(argc != 2) quit(" 引数の数が違う prog datafile ");
  if((f1 = fopen(argv[1],"r")) == NULL) quit("ファイルが開けない");



  while(fgets(buf1,LINESIZE,f1) != NULL) {                //30

        word2[0]=word3[0]='\0';
        sscanf(buf1,"%s %s %s %s",word1,word2,word3,word4);
        pow[i]=atof(word4);

        if(i==0){
//          printf("--h=%s, a=%s--\n",word2,word3);
```

```
            strcpy(h,word2);
            strcpy(a,word3);
           }


         i++;

           if(i==11){
              for(k=1; k<=11; k++){
              sum = sum + pow[k];
                 }
               avg = sum/10;
//               printf("平均=%lf% \n\n",avg);

              i=sum=0;

                if(max<avg){
                      max=avg;
                      strcpy(h_max,h);
                      strcpy(a_max,a);

                     }

             }

       }


//       printf("\n\n h=%s a=%s の時、\n 最大正解率 %lf%\n\n\n",h_max,a_max,max);
      printf("%s %s %.4f\n",h_max,a_max,max);


  if((r = fclose(f1)) == -1) quit("ファイルが閉じれない");
}
```

```
void quit(char *s)
{
  puts(s); exit(1);
}
```

付 録C　プログラムソースリスト（シェルスクリプト）

```
////////////////////////////////////////
//mkdl.sh
////////////////////////////////////////

//デフォルトの取得と決定リストの作成

#!/bin/bash

# all.sh train dl0 1 0.01

./p2 $1 | sort | uniq -c >| k2
./p3 k2 >| k3
./getdef $1 >| def
./tyukan2 k3 def $3 $4 | sort -k 2 -rn >| $2
```

```
////////////////////////////////////////
//all-n.sh
////////////////////////////////////////

//ブートストラップ法に用いる名詞リスト

#!/bin/bash
```

```
export dirs="aida atama ippan ippou ima imi utagai otoko kaihatsu kaku_n\
             kankei kimochi kiroku gijutsu genzai koushou kokunai kotoba\
             kodomo gogo shijo shimin shakai shonen jikan jigyou jidai\
             jibun joho sugata seishin taishou daihyou chikaku chihou\
             chushin te teido denwa doujitsu hana hantai baai mae minkan\
             musume mune me mono mondai"

for d in $dirs; do
   echo === $d START =======
   \cp -f ../data/noun/$d/label .
   all3.sh
   \cp -f KEKKA  ../data/noun/$d
   \cp -f PARA  ../data/noun/$d
done
```

```
////////////////////////////////////////
//all-n.sh
////////////////////////////////////////

//ブートストラップ法に用いる動詞リスト

#!/bin/bash

export dirs="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"

for d in $dirs; do
   echo === $d START =======
```

```
   \cp -f ../data/verb/$d/label .
   all3.sh
   \cp -f KEKKA  ../data/verb/$d
   \cp -f PARA  ../data/verb/$d
done
```

```
////////////////////////////////////////
//all3.sh
////////////////////////////////////////

//ブートストラップ法によるメタパラメータの推定値取得

#!/bin/bash

\cp -f /dev/null KEKKA

declare -i h=0
while [ $h -lt 6 ]; do
  for alpha in "0.01" "0.05" "0.1" "0.2" "0.3" "0.4" "0.5" "0.6" "0.7" "0.8" "0.9"; do
     echo "--- $h $alpha ---";
     echo "--- $h $alpha ---"  >>  KEKKA
     declare -i j=0
     while [ $j -lt 10 ]; do
         all.sh label $h $alpha  >> KEKKA
         let j="$j + 1"
     done
  done
  let h="$h + 1"
done

./getmaxpara KEKKA >| PARA
```

関連図書

[1] 新納浩幸： “複合語からの証拠に重みを付けた決定リストによる同音異義語判別”, vol.39, pp.3200-3206, 1998.

[2] 石井健一郎, 上田修功, 前田英作, 村瀬洋一： “分かりやすいパターン認識”, オーム社, 1998.

[3] 野澤洋一： “決定木による同音異義語の誤り検出とその修正”, 平成 12 年度卒業論文, 1996.