SVDPACKC とその語義判別問題への利用

新納浩幸[†] ，佐々木稔[††]

本論文ではフリーの特異値分解ツール SVDPACKC を紹介する．その利用方法を解説し，利用事例として語義判別問題を扱う．近年，情報検索では潜在的意味インデキシング（ Latent Semantic Indexing，LSI）が活発に研究されている．LSI では高次元の索引語ベクトルを低次元の潜在的な概念のベクトルに射影することで，ベクトル空間モデルの問題点である同義語や多義語の問題に対処する．そして概念のベクトルを構築するために，索引語文書行列に対して特異値分解を行う．SVDPACKC は索引語文書行列のような高次元かつスパースな行列に対して特異値分解を行うツールである．また LSI は，高次元の特徴ベクトルを重要度の高い低次元のベクトルに圧縮する技術であり，情報検索以外にも様々な応用が期待される．ここでは SVDPACKC の利用事例として語義判別問題を取り上げる．SENSEVAL2 の辞書タスクの動詞 50 単語を対象に実験を行った．LSI に交差検定を合わせて用いることで，最近傍法の精度を向上させることができた．また最近傍法をベースとした手法は，一部の単語に対して決定リストや Naive Bayes 以上の正解率が得られることも確認できた．

キーワード： 特異値分解，SVDPACKC，LSI，語義の曖昧性解消

Introduction of SVDPACKC and its application to word sense disambiguation problems

Hiroyuki Shinnou Minoru Sasaki

In this paper, we introduce a free software package SVDPACKC computing the singular value decomposition (SVD) of large sparse matrics. First we explain how to use it, and then solve word sense disambiguation problems by using it. In information retrieval domain, Latent Semantic Indexing (LSI) has actively been researched. LSI maps a high dimensional term vector to the low dimensional concept vectors to overcome synonymy and polysemy problems over information retrieval using vector space model. To build low dimensional concept vectors LSI computes the SVD of term-document matrics. SVDPACKC is a software tool to computes the SVD of large sparse matrics like term-document matrics. LSI compresses a high dimensional future vector to the low dimensional concept vectors, so has many applications besides information retrieval. In this paper, we attack word sense disambiguation problems of 50 verbs in Japanese dictionary task of SENSEVAL2. By using cross validation and LSI, we improved simple Nearest Neighbor method (NN). And we showed that the methods based on NN achieve better precision than the decision list method and Naive Bayes method for some words.

KeyWords: *singular value decomposition, SVDPACKC, LSI, word sense disambiguation*

[†] 茨城大学工学部システム工学科，Department of Systems Engineering, Ibaraki University
[††] 茨城大学工学部情報工学科，Department of Computer and Information sciences, Ibaraki University

1　はじめに

本論文はフリーの特異値分解ツール SVDPACKC (Berry,Do,Krishna and Varadhan 1993) を紹介する．その利用方法を解説し，利用事例として多義語の曖昧性解消問題（以下，語義判別問題と呼ぶ）を扱う．

情報検索ではベクトル空間モデルが主流である．そこでは文書とクエリを索引語ベクトルで表し，それらベクトル間の距離をコサイン尺度などで測ることで，クエリと最も近い文書を検索する．ベクトル空間モデルの問題点として，同義語（synonymy）と多義語（polysemy）の問題が指摘されている．同義語の問題とは，例えば，“car” というクエリから “automobile” を含む文書が検索できないこと．多義語の問題とは，例えば，ネットサーフィンについてのクエリ “surfing” に対して，波乗りに関する文書が検索されることである．これらの問題は文書のベクトルに索引語を当てることから生じている．そこでこれら問題の解決のために文書のベクトルを潜在的（latent）な概念に設定することが提案されており，そのような技術を潜在的意味インデキシング（Latent Semantic Indexing，以下 LSI と略す）と呼んでいる．LSI の中心課題はどのようにして潜在的な概念に対応するベクトルを抽出するかである．その抽出手法に LSI では特異値分解を利用する．具体的には索引語文書行列 A に対して特異値分解を行い，その左特異ベクトル（AA^T の固有ベクトル）を固有値の大きい順に適当な数 k だけ取りだし[1]，それらを潜在的な概念に対応するベクトルとする (北研二, 津田和彦, 獅々堀正幹 2001)．

LSI は魅力的な手法であるが，実際に試してみるには，特異値分解のプログラムが必要になる．低次元の特異値分解のプログラムは比較的簡単に作成できるが，現実の問題においては，高次元かつスパースな行列を扱わなくてはならない．このような場合，特異値分解のプログラムを作成するのはそれほど容易ではない．そこで本論文では，この特異値分解を行うためのツール SVDPACKC を紹介する．このツールによって高次元かつスパースな行列に対する特異値分解が行え，簡単に LSI を試すことができる．

また LSI の情報検索以外の応用として，語義判別問題を取り上げ SVDPACKC の利用例として紹介する．実験では SENSEVAL2 の日本語辞書タスク (黒橋禎夫, 白井清昭 2001) で出題された単語の中の動詞 50 単語を対象とした．LSI に交差検定を合わせて用いることで，最近傍法 (石井健一郎, 上田修功, 前田英作, 村瀬洋 1998) の精度を向上させることができた．また最近傍法をベースとした手法は，一部の単語に対して決定リスト (Yarowsky 1994) や Naive Bayes (Mitchell 1997) 以上の正解率が得られることも確認できた．

2　LSI と 特異値分解

情報検索において，m 個の索引語を予め決めておけば，文書は m 次元の索引語ベクトルとして表現できる（ここでは列ベクトルとして考える）．検索対象の文書群が n 個ある場合，各文

[1] ここでは索引語ベクトルを列ベクトルとしている．また A^T は A の転置行列を表す．

書 d_i に対する m 次元の索引語ベクトルが n 個並ぶので，$m \times n$ の索引語文書行列 A ができる．

$m \times n$ の行列 A の特異値分解とは，行列 A を以下のような行列 U，Σ，V の積に分解することである．

$$A = U\Sigma V^T \tag{1}$$

ここで，U は $m \times m$ の直交行列，V は $n \times n$ の直交行列である．また Σ は $m \times n$ の行列であり，$rank(A) = r$ とすると，対角線上に r 個の要素 $\sigma_1, \sigma_2, \cdots, \sigma_r$（ ただし $\sigma_1 \geq \sigma_2 \geq \cdots, \geq \sigma_r$）が並んだ行列である．それ以外の Σ の要素は 0 である．

行列 A の特異値分解を行ったとき，U は A の列ベクトルが張る空間の正規直交基底となっている．そして U 内の列ベクトルは左側にあるものほど基底としての重要度が高い．そこで，U の最初の k 個の列ベクトルを使って，索引語ベクトルを表すことにする．具体的には索引語ベクトルを U の最初の k 個の列ベクトルに射影させればよい．つまり，U の最初の k 個の列ベクトルで作成される $m \times k$ の行列を U_k とおくと，索引語ベクトル d は $U_k^T d$ によって k 次元のベクトルで表現できることになる．実際の検索では，m 次元の索引語ベクトルで表現されていたクエリ q も $U_k^T q$ によって k 次元のベクトルで表現し，$U_k^T d$ と $U_k^T q$ の距離によって，d と q の距離を測ればよい．例えば，d と q の距離 $dist(d, q)$ は以下のように測ることができる (北研二他 2001)．

$$dist(d, q) = \frac{(U_k^T d, U_k^T q)}{||U_k^T q|| ||U_k^T q||} \tag{2}$$

以上より，情報検索に LSI を利用するためには，索引語文書行列 A の特異値分解から得られる行列 U が求まれば良いことがわかる．

3 特異値分解ツール SVDPACKC

3.1 入手とコンパイル

行列 A の特異値分解を行うツールが SVDPACKC である．SVDPACKC はフリーで配布されており，以下の URL から入手できる．

`http://www.netlib.org/svdpack/svdpackc.tgz`

SVDPACKC には特異値分解を行う C 言語のプログラムが 8 つ入っている．この中で最も計算速度が優れているのは `las2` と名付けられているランチョス法 (北研二他 2001) を使ったプログラムである．単に特異値分解の結果だけを得たいのであれば `las2` を利用すれば良く，他のプログラムをコンパイルする必要はない．ここでは `las2` だけをコンパイルする．

`las2.c` をコンパイルする前に，`las2.c` の中でコメントアウトされているマクロ定数 `UNIX_CREAT` を有効にしておく．

```
/* #define  UNIX_CREAT */   →   #define  UNIX_CREAT
```

これによって las2 による特異値分解の結果がファイルに保存される．次に，las2.h のマクロ定数 LMTNW，NMAX，NZMAX の値を適当に調整する．これらは取り得るメモリの最大サイズ，行列の最大サイズ，行列中の非ゼロ要素の最大個数を定義したものであり，どの程度の大きさの行列を扱えるかを示している．扱う問題や利用する計算機にもよるだろうが，本論文での実験では予め与えられている値の 10 倍の数に変更した．実行時に行列のサイズに関するエラーが出た場合は，これらの値を設定し直してコンパイルする．また las2.c の中では random 関数を自前で用意しているために，stdlib.h で定義されている random 関数と競合する場合がある．ここでは las2.c 中の stdlib.h を include しないことにした．

```
#include <stdlib.h>   →   /* #include <stdlib.h> */
```

コンパイルは makefile のコンパイラの指定（ CC）を利用するコンパイラに合わせて，以下を実行する．ここでは Linux の gcc で問題なくコンパイルできた[2]．

```
make  las2
```

またマニュアルには，CRAY Y-MP，IBM RS/6000-550，DEC 5000-100，HP 9000-750，SPARCstation 2 及び Machintosh II/fx で SVDPACKC が動作することが記載されている．また Windows2000 上の Borland C++ Compiler 5.5 を利用しても las2.c をコンパイルできた[3]．更に Windows2000 上の cygwin + gcc 環境[4]でもコンパイルできた．特別なライブラリは使われていないので，多くの環境でコンパイル可能と思われる．

3.2　利用方法

las2 は内部で2 つのファイルを読む込む．1 つは特異値分解を行いたい対象の行列が記述されたファイル matrix であり，もう1 つはパラメータを記述したファイル lap2 である．この 2 つのファイルを適切に用意することで，las2 を実行することができる．

配布キットでは，サンプルの行列が belladit.Z という名前の圧縮されたファイルとして提供されている．このファイルから，例えば以下のコマンドにより，matrix ファイルを作り，las2 を試してみる．lap2 はこのサンプル用にキット内に用意されている．

```
zcat belladit.Z > matrix
```

実行は以下のように単にコマンド名だけを入力する．

```
./las2
```

[2] gcc，libc 及び libm のバージョンはそれぞれ egcs-2.91.66，2.1.2，2.1.2 のものを用いた．libm 以外のライブラリは使用されない．
[3] ただし構造体 rusage の定義がないので，若干変更する必要があった．
[4] cygwin の dll のバージョンは 1.2.12-2，gcc のバージョンは 2.95.3-5 のものを用いた．

結果は lav2 と lao2 というファイルに保存される．lav2 には特異値分解したときの式 1 における U や V の配列が保存される．ただし，バイナリファイルなので直接見ることはできない．lao2 には特異値分解したときの式 1 の Σ，つまり特異値の列とその他の情報（ 行列の大きさや実行時間等）が保存される．これはテキストファイルなので中身を確認できる．

3.2.1 matrix ファイルの記述方法

特異値分解の対象となる行列 A は，ハーウェル・ボーイング形式（ Harwell-Boeing format）と呼ばれる列方向の圧縮形式を用いてファイル matrix に記述する．これによりスパース行列を少ない記述量で簡単に表現することができる．

最初に注意として，行列 A の大きさを $m \times n$ とした場合，SVDPACKC では $m \geq n$ を仮定している．そのために，実際に特異値分解したい行列 A の列数の方が行数よりも大きい場合は，行列 A の転置行列 A^T に対して特異値分解を行う必要がある．この場合，式 2 の U は V と置き換えなければならないこともある．

ここでは，matrix の記述形式の説明として，以下のような行列 A を考える．

$$A = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 2.1 & 0 & 0.5 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0.8 & 0 & 0 \\ 0 & 0.8 & 1.0 & 0 \\ 1.0 & 0 & 2.2 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

この行列に対して，1 列目から順に非ゼロ要素を取り出し，以下のような表を作る．

データ番号	1	2	3	4	5	6	7	8	9	10
位置	(1,1)	(6,1)	(2,2)	(3,2)	(4,2)	(5,2)	(5,3)	(6,3)	(2,4)	(7,4)
値	1.0	1.0	2.1	1.0	0.8	0.8	1.0	2.2	0.5	1.0

次にこの表から位置の行の部分だけを取り出す．

データ番号	1	2	3	4	5	6	7	8	9	10
位置	(1,1)	(6,1)	(2,2)	(3,2)	(4,2)	(5,2)	(5,3)	(6,3)	(2,4)	(7,4)
値	1.0	1.0	2.1	1.0	0.8	0.8	1.0	2.2	0.5	1.0
行位置	1	6	2	3	4	5	5	6	2	7

次に各列の最初の非ゼロ要素のデータ番号を列ポインタに記述する．例えば，1 列目であれ

ば，最初の非ゼロ要素は (1,1) の 1.0 であり，これに対するデータ番号は 1 である．次に 2 列目であれば，最初の要素は (2,2) の 2.1 であり，これに対するデータ番号は 3 である．これを各列，順に記述したものが列ポインタである．つまり列ポインタの要素数は配列 A の列数となる．

データ番号	1	2	3	4	5	6	7	8	9	10
位置	(1,1)	(6,1)	(2,2)	(3,2)	(4,2)	(5,2)	(5,3)	(6,3)	(2,4)	(7,4)
値	1.0	1.0	2.1	1.0	0.8	0.8	1.0	2.2	0.5	1.0
行位置	1	6	2	3	4	5	5	6	2	7
列ポインタ	1	3	7	9	-	-	-	-	-	-

ハーウェル・ボーイング形式とは，行列に対して，このような表を作り，列ポインタ，行位置，値を記述した形式である．これはスパース行列を圧縮した表現となる．

matrix では 4 行目以降に，列ポインタ，行位置，値が記述されている．列ポインタについては最後の非ゼロ要素のデータ番号に1 を足したものが付け加えられることに注意する．先の例では以下のようになる．

```
      1       3       7       9       11
      1       6       2       3       4       5       5       6       2       7
      1.0       1.0       2.1       1.0       0.8       0.8       1.0       2.2
      0.5       1.0
```

matrix の最初の 4 行は行列に関するその他の情報が記述されている．3 行目以外意味はない．1 行目はデータの名前であり，サンプルファイルの belladit.Z を参考に適当につければよい．2 行目，4 行目も意味はなく，belladit.Z の通り に記述すれば良い．3 行目は以下のように 5 つのデータを空白で区切って記述すればよい．

```
      rra                     7           4          10           0
```

この行の 1 列目(rra)，5 列目(0) はこの通り 記述すれば良い．2 列目(7) は行列 A の行数，3 列目(4) は列数，4 列目(10) は非ゼロ要素の総数を 記述する．結果，先の例において，matrix は以下のようになる．

```
Jikken Data                                                          jikken
Transposed
rra                    7           4          10           0
          (10i8)          (10i8)          (8f10.3)          (8f10.3)
    1       3       7       9       11
    1       6       2       3       4       5       5       6       2       7
    1.0       1.0       2.1       1.0       0.8       0.8       1.0       2.2
    0.5       1.0
```

3.2.2 lap2 ファイルの記述方法

lap2 ファイルは las2 で使われるパラメータが 1 行で記述されている．例えば，配布キットに入っている lap2 ファイルの中身は以下のような 1 行のファイルであり，8 個のデータが空白で区切られて入っている．

```
    'belladit' 44 10 -1.0e-30 1.0e-30 TRUE 1.0e-6 0
```

1 列目（ 'belladit'）は matrix ファイルの 1 行目に記述したデータの名前である．matrix と lap2 とのデータ名の一致は検査していないので，実質意味はない．適当な名前をつければ良い．4 列目（ -1.0e-30），5 列目（ 1.0e-30），6 列目（ 1.0e-6）の数値は，繰り返しの収束条件にあたるものであり，特に変更する必要はない．7 列目（ TRUE）は特異値分解の結果の U や V の行列をファイルに保存するかどうかの指定であり， TRUE にしておけば保存される．8 列目に意味はない．実際は何も書かなくてもよい．

問題は 2 列目（ 44）と 3 列目（ 10）の整数値である．結論から述べれば，どちらも行列 A の列数 n を設定すればよい．今，2 列目の整数値を lanmax ，3 列目の整数値を maxprs とおく． lanmax は las2 のアルゴリズムであるランチョス法の最大の繰り返し回数を意味する．一方， maxprs の意味はやや不明確である．マニュアルには，所望の U や V の次元数と記載されているが，例えば， maxprs = 20 と設定したからといって，必ずしも U や V の次元数が 20 になって出力されるわけではなく，10 であったり，25 であったりする．このような違いは lanmax の数値とも関連しており，依存関係は複雑である．しかし las2 内部では，U や V を最大の次元数に設定して計算しており，最後の出力の部分で指定した次元数を考慮して出力させている．そのため maxprs の値は実行時間等に影響はなく，現実的には得られる最大の次元数を出力させ，その結果から所望の次元数を得た方が取り扱いが簡単である．

3.3 出力結果の利用

las2 による特異値分解の結果は lav2 と lao2 というファイルに保存される．lao2 はテキス

トファイルであり，内容の確認は容易である．重要部分はファイルの下方に記載されている固有値の列である．この部分を適当に切り取って利用すればよい．また，lao2では，固有値は値の小さい順に出力されていることに注意すべきである．固有値は大きい方が重要な意味を持つため，ファイルの下方に書かれた特異値ほど重要である．

lav2はバイナリファイルであり，las2のソースをみて出力形式を確認すれば，特異値分解結果のUやVを得ることが可能である．結局，lav2をテキストファイルの形式に変換する何らかのプログラムを自作する必要がある．ただし，そのようなプログラムを作成するのであれば，las2のソースを直接変更して，テキストファイルの形式で出力させた方が簡単である．

例えば，las2.cの334行目でUが出力されているので，以下のように変更する．

変更前

```
write(fp_out2, (char *)&xv1[ida], size1);
```

変更後

```
long kk;   /* この変数をはじめの方で作っておく */

...

/* write(fp_out2, (char *)&xv1[ida], size1); */
for(kk = 0;kk < nrow;kk++) fprintf(fp_out2,"%lf\n",xv1[ida+kk]);
fprintf(fp_out2,"EOV\n"); /* ベクトルの終りの記号も入れる */
```

これで lav2 にUの中身がテキスト形式で出力される．ファイル名も変更したいときは，148行目の以下の部分を書き換える．

```
out2 = "lav2";
```

またlas2.cの756行目でVが出力されているので，以下のように変更する．

変更前

```
for (i = 0; i < n; i++) xv1[id++] = w1[i];
```

変更後

```
FILE *fp_out3;   /* 他と合わせるために las2.h に
                    書いておく．大域変数となる．  */
...

fp_out3 = fopen("V-matrix", "w");
                /* 行列 V のファイル名は V-matrix とする．
                   main 中で fopen で開いておく */
...

/* for (i = 0; i < n; i++) xv1[id++] = w1[i]; */
for (i = 0; i < n; i++) {
   xv1[id++] = w1[i];
   fprintf(fp_out3,"%lf\n",w1[i]);
}
fprintf(fp_out3,"EOV\n"); /* ベクトルの終りの記号も入れる */
```

以上のようにして，テキストファイルの形式で U や V を得ることができる．これらのファイルは，U や V の列ベクトルが，順に出力されている形になるが，その順序は las2 の固有値の順序に対応している．つまり，固有値の大きな順に k 個の列ベクトルを取り出すときには，下方にあるベクトルから順に k 個取り出さなければならないことに注意する．

4 語義判別問題への利用

ここでは情報検索以外への LSI の応用として語義判別問題を取り上げる．SENSEVAL2 の日本語辞書タスクで課題として出された動詞 50 単語を実験の対象とする．

4.1 最近傍法の利用

単語 w は k 個の語義を持つとし，各語義を c_i $(i = 1\sim k)$ で表す．単語 w の語義判別問題とは，テキストに単語 w が現れたときに，その文脈上での単語 w の語義 c_j を判定する問題である．文脈を m 個の素性のベクトル $(f_1, f_2, \cdots, f_m)$ で表現した場合，この語義判別問題は分類問題となり，帰納学習の手法により解決できる．ここでは最近傍法（ Nearest Neighbor 法，以下 NN 法と略す）(石井健一郎他 1998) を用いる．NN 法は与えられた素性ベクトルと最も距離が近い訓練事例中の素性ベクトルを選び，そのクラスを出力とする手法である[5]．今，単語 w の訓練データの事例数を n とし，各事例を $m \times 1$ の素性ベクトル d_i（ $i = 1\sim n$）で表す．すると

[5] これは用例ベースの手法であり，帰納学習手法とは位置づけない見方もできる．

訓練データ全体の集合は $m \times n$ の行列 A として表せる．実際の語義判別は，単語 w の現れた文脈を素性ベクトル q で表し，以下の式で求められる訓練事例 $\hat{d}$ のクラスを返すことで行える．

$$\hat{d} = arg \min_{d_i} dist(q, d_i)$$

ここでの NN 法は，$dist(q, d_i)$ を単純なコサイン尺度で計算することにする．また行列 A を特異値分解し，式 2 を利用して $dist(q, d_i)$ を定義したものを LSI 法と呼ぶことにする．

4.2　素性の設定

ここでは語義判別の手がかりとなる属性として以下のものを設定した．

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)．

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 (国立国語研究所 1994) の番号を調べると， 3.1920_1 である．ここでは分類語彙表の 4 桁目と 5 桁目までの数値をとることにした．つまり ‘e3=短い’ に対しては，‘e5=3192’ と ‘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，その問題を避けたためである．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すべ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，

上記の例のようにして「出す」に対するすべての訓練事例の素性を集め，各素性に 1 番から順に番号をつける．例えば，本論文の実験では「出す」に対しては 978 種類の素性があり，上記例の素性には表 1 のように番号が振られた．

表 1: 素性と次元番号

素性	次元番号
e1=を	21
e2=に	60
e3=コメント	134
e3=短い	302
e4=。	379
e4=とどまる	406
e5=3192	789
e5=31920	790

以上より，上記例文に対する素性ベクトルは第 21 次元目，第 60 次元目，第 134 次元目，第 302 次元目，第 379 次元目，第 406 次元目，第 789 次元目，第 790 次元目の各要素が 1 であり，その他の要素がすべて 0 の 978 次元のベクトルとなる．

4.3 交差検定の利用

LSI 法を利用した場合，NN 法と比較して，必ずしも精度が向上するわけではなく，逆に精度が悪化する場合もある．そのため単純にすべての単語に対して，LSI 法を用いることはできない．そこで交差検定を行い，次元圧縮の効果が確認できる単語のみ LSI 法を用いることにする．このように NN 法と LSI 法を融合した手法を LSI+NN 法と呼ぶことにする．

ここでの交差検定では訓練データを 4 分割し，3 つを訓練データ，1 つをテストデータとする．組合わせを変えて，合計 4 通りの実験を行う．各実験では，NN 法と LSI 法のテストデータに対する正解率を測る．また特異値分解を使って圧縮する次元数は 75 とした．ただし行列 A のランク数が 75 以下の場合は，行列 A のランク数にした．付録の表 7 に `las2` の結果をまとめている．そこでは SENSEVAL2 の日本語辞書タスクの動詞 50 単語の各単語に対する行列

A の大きさ，非ゼロ要素の密度，圧縮した次元数，次元圧縮に要したメモリと時間が記されている．ただし，これらは 4 通りの実験での平均である．また次元圧縮に要したメモリと時間は `las2` の出力ファイル `lao2` から得ている．

各単語に対して，4 通りの実験の平均をとった結果が表 2 である．

表 2: 交差検定による比較

単語	NN	LSI		↓	↓	↓	
ataeru	0.621	0.517		tsutaeru	0.703	0.629	
iu	0.803	0.836	○	dekiru	0.722	0.708	
ukeru	0.608	0.513		deru	0.432	0.279	
uttaeru	0.828	0.659		tou	0.718	0.703	
umareru	0.722	0.690		toru	0.464	0.232	
egaku	0.628	0.533		nerau	0.894	0.895	○
omou	0.918	0.845		nokosu	0.592	0.530	
kau	0.930	0.896		noru	0.672	0.641	
kakaru	0.644	0.452		hairu	0.443	0.392	
kaku_v	0.741	0.733		hakaru	0.941	0.905	
kawaru	0.907	0.948	○	hanasu	0.983	0.965	
kangaeru	0.925	0.949	○	hiraku	0.853	0.813	
kiku	0.667	0.456		fukumu	0.946	0.946	○
kimaru	0.881	0.915	○	matsu	0.589	0.470	
kimeru	0.899	0.886		matomeru	0.655	0.669	○
kuru	0.766	0.773	○	mamoru	0.774	0.795	○
kuwaeru	0.899	0.899	○	miseru	0.920	0.880	
koeru	0.841	0.716		mitomeru	0.934	0.934	○
shiru	0.866	0.834		miru	0.806	0.777	
susumu	0.339	0.366	○	mukaeru	0.925	0.893	
susumeru	0.886	0.826		motsu	0.566	0.472	
dasu	0.476	0.303		motomeru	0.882	0.807	
chigau	0.905	0.971	○	yomu	0.963	0.967	
tsukau	0.715	0.704		yoru	0.973	0.931	
tsukuru	0.578	0.569		wakaru	0.848	0.916	○
↓	↓	↓		平均	0.764	0.719	

特異値分解を利用することで正解率が向上したものは，表 2 で○印のつけた以下の 14 単語である．これらに対して LSI 法を用いることにする．

iu, kawaru, kangaeru, kimaru, kuru, kuwaeru, susumu,
chigau, nerau, fukumu, matomeru, mamoru, mitomeru, wakaru

4.4 特異値分解を用いた語義判別実験

実際は選出した 14 単語のみに対して LSI 法を行えば良いが，交差検定の効果も示すために，すべての単語に対して LSI 法を試みた．圧縮する次元数は 100 に設定した．ただし行列 A のランク数が 100 以下の場合は，行列 A のランク数にした．付録の表 8 に `las2` の結果をまとめている．そこでは SENSEVAL2 の日本語辞書タスクの動詞 50 単語の各単語に対する行列 A の大きさ，非ゼロ要素の密度，圧縮した次元数，次元圧縮に要したメモリと時間が記されている．また次元圧縮に要したメモリと時間は `las2` の出力ファイル `lao2` から得ている．

次に SENSEVAL2 で配布されたテスト文を用いて正解率を測った結果が表 3 である．スコアの算出は解答結果に部分点を与える mixed-gained scoring という方式 (黒橋禎夫, 白井清昭 2001) を用いている．

わずかではあるが，LSI+NN 法の方が NN 法よりも精度が高かった．

また選択した 14 単語のうち LSI 法を利用することで精度が上がった単語（ 選択が正しかった単語）は 8 単語，下がった単語（ 選択が誤った単語）は 6 単語である．逆に選択しなかった 36 単語のうち NN 法の方が精度が良かった単語（ 選択が正しかった単語）は 31 単語，LSI 法の方が精度が良かった単語（ 選択が誤った単語）は 5 単語であった．つまり，全体の 50 単語のうち選択が正しかった単語は 39 単語（ 78%），選択が誤った単語は 11 単語（ 22%）である．単純にすべて NN 法を選択した場合，選択が正しくなる単語は 37 単語（ 74%），選択が誤る単語は 13 単語（ 26%）であるため，交差検定の効果が確認できる．

また同様の素性を用いて，決定リスト（ DL と略す），Naive Bayes（ NB と略す）を用いた判別も行った．結果を表 4 に示す．

ほとんどの単語で，決定リストや Naive Bayes は NN 法や LSI 法よりも良い結果を出しているが，一部では NN 法や LSI 法の方が良い値を出している．単語によっては NN 法をベースとした方が良い場合もあることを示している．

5 考察

SVDPACKC が扱える行列の大きさについて述べておく． `las2.c` からメモリ割り当ての関数 `malloc` の部分を抜き出してみると， `las2` は $m \times n$ の行列の特異値分解を行うのに，大ざっぱに見積もって，$8mn$ バイト強のメモリを必要としていることがわかる[6]．この点から

[6] `sizeof(double)` の mn 倍である．`sizeof(double) = 8` として $8mn$ を得ている．

表 3: 実験結果

単語	NN	LSI	LSI+NN	↓	↓	↓	↓
ataeru	0.680	0.560	0.680	tsutaeru	0.663	0.780	0.663
iu	0.820	0.880	0.880	dekiru	0.760	0.690	0.760
ukeru	0.550	0.410	0.550	deru	0.440	0.320	0.440
uttaeru	0.810	0.680	0.810	tou	0.670	0.640	0.670
umareru	0.720	0.670	0.720	toru	0.280	0.270	0.280
egaku	0.560	0.570	0.560	nerau	0.980	0.920	0.920
omou	0.930	0.710	0.930	nokosu	0.705	0.595	0.705
kau	0.860	0.850	0.860	noru	0.680	0.600	0.680
kakaru	0.620	0.540	0.620	hairu	0.390	0.250	0.390
kaku_v	0.770	0.560	0.770	hakaru	0.980	0.980	0.980
kawaru	0.920	0.890	0.890	hanasu	1.000	0.990	1.000
kangaeru	0.960	0.950	0.950	hiraku	0.880	0.790	0.880
kiku	0.550	0.470	0.550	fukumu	0.960	0.990	0.990
kimaru	0.910	0.900	0.900	matsu	0.490	0.540	0.490
kimeru	0.920	0.910	0.920	matomeru	0.710	0.740	0.740
kuru	0.740	0.830	0.830	mamoru	0.635	0.735	0.735
kuwaeru	0.870	0.860	0.860	miseru	0.950	0.980	0.950
koeru	0.770	0.710	0.770	mitomeru	0.860	0.880	0.880
shiru	0.940	0.930	0.940	miru	0.740	0.730	0.740
susumu	0.390	0.340	0.340	mukaeru	0.940	0.890	0.940
susumeru	0.920	0.830	0.920	motsu	0.540	0.390	0.540
dasu	0.340	0.210	0.340	motomeru	0.810	0.790	0.810
chigau	0.870	0.970	0.970	yomu	0.880	0.830	0.880
tsukau	0.715	0.895	0.715	yoru	0.960	0.900	0.960
tsukuru	0.660	0.590	0.660	wakaru	0.820	0.890	0.890
↓	↓	↓	↓	平均	0.750	0.717	0.7570

考えると，必要メモリが約 200M バイトとなる 25000 × 1000 位の大きさが現実的な最大サイズだと思われる[7]．確認のために，非ゼロ要素の密度が 1% であり，平均 2 のポアソン分布に従って，非ゼロ要素の整数値（ 1 〜 6）が配置されるような 25000 × 1000 の行列 A を人工的に作成し[8]，その行列に対して `las2` で特異値分解を行ってみた．Pentium-4 1.5GHz メモリ 512M バ

[7] これは個人的な感覚である．
[8] 実際の索引語文書行列に似せるよう考慮している．

表 4: 他手法との比較

単語	NN	LSI	DL	NB	↓	↓	↓	↓	↓
ataeru	0.680	0.560	0.660	0.740	tsutaeru	0.663	0.780	0.750	0.780
iu	0.820	0.880	0.940	0.930	dekiru	0.760	0.690	0.790	0.800
ukeru	0.550	0.410	0.550	0.660	deru	0.440	0.320	0.560	0.570
uttaeru	0.810	0.680	0.820	0.870	tou	0.670	0.640	0.600	0.700
umareru	0.720	0.670	0.680	0.710	toru	0.280	0.270	0.330	0.390
egaku	0.560	0.570	0.560	0.580	nerau	0.980	0.920	0.990	0.990
omou	0.930	0.710	0.890	0.900	nokosu	0.705	0.595	0.770	0.800
kau	0.860	0.850	0.850	0.850	noru	0.680	0.600	0.590	0.660
kakaru	0.620	0.540	0.630	0.660	hairu	0.390	0.250	0.440	0.380
kaku_v	0.770	0.560	0.740	0.730	hakaru	0.980	0.980	0.920	0.920
kawaru	0.920	0.890	0.920	0.920	hanasu	1.000	0.990	1.000	0.990
kangaeru	0.960	0.950	0.990	0.990	hiraku	0.880	0.790	0.830	0.910
kiku	0.550	0.470	0.590	0.640	fukumu	0.960	0.990	0.990	0.990
kimaru	0.910	0.900	0.960	0.960	matsu	0.490	0.540	0.530	0.490
kimeru	0.920	0.910	0.940	0.920	matomeru	0.710	0.740	0.780	0.750
kuru	0.740	0.830	0.890	0.880	mamoru	0.635	0.735	0.800	0.750
kuwaeru	0.870	0.860	0.890	0.880	miseru	0.950	0.980	0.980	0.970
koeru	0.770	0.710	0.780	0.760	mitomeru	0.860	0.880	0.890	0.890
shiru	0.940	0.930	0.960	0.960	miru	0.740	0.730	0.730	0.740
susumu	0.390	0.340	0.430	0.450	mukaeru	0.940	0.890	0.920	0.900
susumeru	0.920	0.830	0.960	0.940	motsu	0.540	0.390	0.520	0.550
dasu	0.340	0.210	0.350	0.340	motomeru	0.810	0.790	0.880	0.870
chigau	0.870	0.970	1.000	1.000	yomu	0.880	0.830	0.880	0.880
tsukau	0.715	0.895	0.935	0.963	yoru	0.960	0.900	0.970	0.970
tsukuru	0.660	0.590	0.590	0.710	wakaru	0.820	0.890	0.900	0.900
↓	↓	↓	↓	↓	平均	0.750	0.717	0.777	0.790

イトの Linux 環境での実行時間は 227 秒，要したメモリは 228M バイトであった．この程度の大きさの行列であれば，実行時間は大きな問題にはならないと思われる．

ただし，行列の大きさを変更して同じ条件で試してみると表 5 の結果が得られた．メモリは行列のサイズにほぼ比例するが実行時間は指数関数的に増加しているので，実行時間の面からも，この程度の大きさの行列が SVDPACKC で扱える限度だと思われる．ちなみに 25000×2000 の行列ではメモリ不足で実行できなかった．ただし実験で用いたマシンにスワップは設定され

ていないことを注記しておく．スワップを利用すれば，更に大きな行列も扱えるが，その場合は実行時間の方で問題が生じるであろう．

表 5: 行列の大きさと速度・メモリの関係

	25000×125	25000×250	25000×500	25000×1000
使用メモリ (MB)	26.2	52.6	108	228
実行時間（秒）	7.49	16.8	53.5	227

実際に情報検索で用いられる索引語ベクトルの次元数は少なくとも数十万単位になり，検索対象の文書も 100 万文書以上となるであろう．その場合の索引語文書行列 A の大きさは巨大なスパース行列である．このような巨大な行列になると，SVDPACKC によって一気に特異値分解を行うのは不可能である．この問題に対しては最初に小さな行列で特異値分解を行い，その後に文書や索引語の追加に従って特異値分解の更新を行う folding-in とよばれる手法や大規模な文書集合から文書をランダムサンプルし，そこから特異値分解を行う手法などが提案されている (北研二他 2001)．あるいは概念ベクトルの選択に特異値分解以外の手法を使うアプローチもある（(佐々木稔, 北研二 2001) など）．最近では言語横断検索にも LSI が利用されているが (Susan Dumais and Todd Letsche and Michael Littman and Thomas Landauer 1997)，そこでも大規模な行列の特異値分解をどう行うかが問題点として上がっている (森辰則, 國分智晴, 田中崇 2002)．結局，現実の情報検索で現れるような大規模な行列に対しては，SVDPACKC を直接利用することはできない．しかしアイデアを試すための中規模の実験であれば，十分にその役割を果たせる．

実験では圧縮する次元の数を交差検定では 75 に，実際の評価では 100 に固定している．この値は適当である．最適な次元数については様々な議論があるが，ここでは SVDPACKC の利用例として紹介した実験であるため，最適な次元数を推定する処理は行わなかった．ちなみに実際の評価における次元数を 100 から増減させた場合の，実験結果を表 6 に示す．次元数を 100 に圧縮するといっても行列のランク数がそれ以下であれば，100 よりも小さい数になるので，100 のときに圧縮された次元数を基準に -20, -10, +10, +20 と次元数を変更させて実験を行った．また表中に MAX とあるのは，行列のランク数で圧縮した場合を示す．これが圧縮できる次元の最大値である．大まかな傾向としては次元数が多い方が精度は高いようである．ただし最高精度を記録する次元数は個々の単語によって異っており，最適な次元数は問題に依存すると言える．

LSI のアイデア自体は情報検索以外にも適用できる．ここでは語義判別問題への利用を試みた．他にも文書分類への応用が報告されている (Zelikovitz and Hirsh 2001)．このような教師付き学習のタイプでは，訓練事例数が大規模なものになることはないため，SVDPACKC が利用できる．また素性ベクトルの次元圧縮という手法は，統計学では主成分分析，パターン認識では Karhunen-Loève 展開や線形判別法 (石井健一郎他 1998) として知られている手法であ

表 6: 圧縮次元数と精度の関係

	-20	-10	100	+10	+20	MAX
判別精度	0.7030	0.7168	0.7202	0.7212	0.7202	0.7203

る．またデータマイニングの分野ではデータ数が非常に大きいために，現実的には機械学習手法を直接適用できないという問題がある．そのために類似する事例集合を抽象表現として表される事例に変換し，変換後の事例に対して機械学習手法を適用する Data Squashing という手法が使われる (鈴木英之進 2002)．これは索引語ベクトルではなく文書ベクトルに対する次元圧縮の手法に対応する．このように次元圧縮の手法は様々な分野で重要であり，新しい手法が次々と提案されている（ 例えば (末永高志, 佐藤新, 坂野鋭 2002) など）．次元圧縮の手法として，特異値分解は古典的と言えるが，ベースとなる手法として容易に試すことのできる意味でも SVDPACKC は有用であろう．

最後に LSI を分類問題に利用する場合の注意を述べておく．次元圧縮を行う手法は種々あるが，それらは 2 つに大別できる．1 つは「 表現のための次元圧縮」であり，もう 1 つは「 判別のための次元圧縮」である (石井健一郎他 1998)．「 表現のための次元圧縮」は素性ベクトルの分布全体のもつ情報をできるだけ反映できるように次元を圧縮する．一方，「 判別のための次元圧縮」はクラスをできるだけ明確に分離できるように次元を圧縮する．主成分分析や Karhunen-Loève 展開は前者であり，線形判別法は後者である．そして特異値分解も「 表現のための次元圧縮」に属する手法である．このため，特異値分解を行ったからといって必ずしも判別精度が高まることは保証されない．「 表現のための次元圧縮」が判別精度向上に寄与できる問題は，非常に高次元のベクトルを扱う問題（ 例えば情報検索や音声・ 画像認識）だと思われる．このような場合，「 表現のための次元圧縮」は “次元の呪い” に対抗できる可能性がある．あるいは次元数が多くなったときに素性間に共起性（ 依存関係）が生じる傾向があり，それが精度向上に悪影響を及ぼすが，そのような依存関係を解消できる可能性ももつ．

特異値分解による次元圧縮が判別精度の向上に寄与できるかどうかは未知である．本研究では交差検定を行うことで，精度向上に寄与できそうな問題を選別しておくというアプローチをとった．しかし，LSI 法は平均的には他の学習手法よりも精度が低かった．LSI 法を語義判別問題に利用するためには，また別の工夫が必要になるだろう．1 つの利用可能性としては bagging 手法 (Breiman 1996) の 1 つの学習器として使うことが考えられる．実際に，LSI 法は数個の単語に関しては他の学習手法よりも精度が高かった．また NN 法まで含めるとその数は更に増える．SENSEVAL2 の辞書タスクでは様々な学習手法を融合して用いる手法が最も良い成績を納めた (村田真樹, 内山将夫, 内元清貴, 馬青, 井佐原均 2001)．そこでは，決定リスト，Naive Bayes，SVM の学習手法を用意し，交差検定の結果から単語毎に利用する学習手法を設定している．ここで用いた NN 法 や LSI 法も 1 つの学習手法としてエントリーさせておけばよい．

今回の実験では，多くの単語に対して LSI 法は NN 法よりも正解率が低かった．特に，語義

判別の場合，素性ベクトルの次元数 m に比べ，訓練事例数 n が小さい．特異値分解で圧縮する次元数の最大値は n なので，この点でかなり制約があった．交差検定を用いることで NN 法の精度を高めることができたが，他の学習手法と比べると精度の面ではまだ十分ではない．今後は NN 法や LSI 法が他の学習手法よりも正解率が高かった単語について，その原因を調査する．これによって LSI を語義判別問題のような分類問題に利用する方法を探ってゆく．また LSI が利用可能な他の問題を調べゆく．

6　おわりに

本論文ではフリーの特異値分解ツール SVDPACKC を紹介した．その利用方法を解説し，利用事例として語義判別問題を扱った．SENSEVAL2 の辞書タスクの動詞 50 単語を対象に実験を行ったところ，交差検定を合わせて用いることで，NN 法を改良できた．また NN 法や LSI 法は，一部の単語に対して決定リストや Naive Bayes 以上の正解率が得られることも確認できた．特異値分解は，情報検索の LSI だけではなく，高次元の特徴ベクトルを重要な低次元のベクトルに射影する手法で必要とされる．このために様々な応用が期待される．今後はここでの実験の結果を詳しく調査し，LSI が利用可能な問題を調べてゆきたい．

参考文献

Berry, M.,Do, T.,Krishna, G. O. V.,and Varadhan, S. (1993). “SVDPACKC (Version 1.0) User’s Guide.” In *”www.netlib.org/svdpack”*.

Breiman, L. (1996). “Bagging Predictors.” *Machine Learning*, **24** (2), 123–140.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill Companies.

Susan Dumais and Todd Letsche and Michael Littman and Thomas Landauer (1997). “Automatic cross-language retrieval using latent semantic indexing.” In *AAAI Symposium on Cross-Language Text and Speech Retrieval. American Association for Artificial Intelligence*.

Yarowsky, D. (1994). “Decision Lists for Lexical Ambiguity Resolution: Application to Accent Restoration in Spanish and French.” In *32th Annual Meeting of the Association for Computational Linguistics (ACL-94)*, pp. 88–95.

Zelikovitz, S. and Hirsh, H. (2001). “Using LSI for Text Classification in the Presence of Background Text.” In *10th ACM International Conference on Information and Knowledge Management (CIKM-01)*, pp. 113–118.

国立国語研究所 (1994). 分類語彙表. 秀英出版.

黒橋禎夫, 白井清昭 (2001). “SENSEVAL-2 日本語タスク.” 電子情報通信学会言語とコミュニケーション研究会, NLC-36〜48, pp. 1–8.

佐々木稔, 北研二 (2001). “ランダムプロジェクションによるベクトル空間情報検索モデルの次元圧縮.” 自然言語処理, **8** (1), 5–19.

森辰則, 國分智晴, 田中崇 (2002). “空間分割型 CL-LSI による大規模言語横断情報検索.” 情報処理学会データベース, **43** (SIG-2(TOD 13)), 27–35.

石井健一郎, 上田修功, 前田英作, 村瀬洋 (1998). わかりやすいパターン認識. オーム社出版局.

村田真樹, 内山将夫, 内元清貴, 馬青, 井佐原均 (2001). “SENSEVAL2J 辞書タスクでの CRL の取り組み.” 電子情報通信学会言語とコミュニケーション研究会, NLC-36〜48, pp. 31–38.

北研二, 津田和彦, 獅々堀正幹 (2001). 情報検索アルゴリズム. 共立出版.

末永高志, 佐藤新, 坂野鋭 (2002). “クラスタ構造に着目した特徴空間の可視化—クラスタ判別法—.” 信学論 D-II, **J85-D** (5), 785–795.

鈴木英之進 (2002). “データマイニングにおけるデータ変換.” 人工知能学会第 17 回 AI シンポジウム, SIG-J-A103, pp. 23–27.

付録

表 7: las2 による特異値分解結果 (1)

単語	行列サイズ	非ゼロ密度	圧縮次元数	メモリ (MB)	実行時間 (秒)
ataeru	431 × 87	0.0232	75.00	0.51	0.1225
iu	937 × 274	0.0087	75.00	3.93	1.8625
ukeru	1138 × 267	0.0092	75.00	4.21	2.0050
uttaeru	292 × 52	0.0315	26.50	0.20	0.0225
umareru	309 × 48	0.0338	32.75	0.19	0.0325
egaku	339 × 52	0.0283	36.25	0.22	0.0350
omou	981 × 348	0.0079	75.00	5.67	3.0700
kau	375 × 65	0.0241	65.25	0.32	0.0700
kakaru	482 × 86	0.0233	66.75	0.54	0.1150
kaku_v	498 × 101	0.0197	65.00	0.68	0.1400
kawaru	424 × 72	0.0224	72.75	0.40	0.0950
kangaeru	953 × 218	0.0103	75.00	2.87	1.2775
kiku	569 × 135	0.0154	75.00	1.09	0.2925
kimaru	407 × 87	0.0239	75.00	0.47	0.1050
kimeru	656 × 171	0.0137	75.00	1.52	0.3975
kuru	493 × 96	0.0194	75.00	0.63	0.2925
kuwaeru	418 × 81	0.0215	66.00	0.46	0.1000
koeru	460 × 89	0.0227	67.00	0.54	0.1125
shiru	797 × 184	0.0111	75.00	2.04	0.6225
susumu	473 × 84	0.0215	75.00	0.51	0.1175
susumeru	529 × 99	0.0193	75.00	0.68	0.1675
dasu	791 × 156	0.0130	75.00	1.61	0.3900
chigau	522 × 78	0.0190	55.75	0.50	0.0950
tsukau	1056 × 202	0.0093	75.00	2.75	1.1400
tsukuru	648 × 137	0.0146	75.00	1.09	0.3200
tsutaeru	377 × 72	0.0252	72.25	0.36	0.0875
dekiru	364 × 56	0.0266	56.25	0.26	0.0600
deru	1153 × 309	0.0083	75.00	5.17	2.8725
tou	265 × 53	0.0405	52.25	0.19	0.0475
toru	457 × 84	0.0221	57.25	0.50	0.0925
nerau	318 × 50	0.0290	50.25	0.20	0.0425
nokosu	436 × 73	0.0216	73.50	0.41	0.1000
noru	258 × 48	0.0386	33.75	0.17	0.0300
hairu	974 × 208	0.0106	75.00	2.71	0.9075
hakaru	365 × 63	0.0285	63.00	0.30	0.0700
hanasu	382 × 131	0.0195	75.00	0.82	0.1575
hiraku	657 × 168	0.0161	75.00	1.60	0.4325
fukumu	520 × 69	0.0223	49.25	0.42	0.0725
matsu	380 × 62	0.0274	45.00	0.30	0.0525
matomeru	381 × 69	0.0293	69.75	0.35	0.0800
mamoru	392 × 69	0.0236	69.75	0.36	0.0800
miseru	357 × 75	0.0248	74.50	0.37	0.0800
mitomeru	729 × 159	0.0132	75.00	1.58	0.4475
miru	1229 × 306	0.0076	75.00	5.29	2.8975
mukaeru	348 × 69	0.0318	69.00	0.33	0.0750
motsu	1020 × 200	0.0101	75.00	2.64	0.8900
motomeru	975 × 229	0.0104	75.00	3.09	1.1875
yomu	406 × 79	0.0238	75.00	0.43	0.1025
yoru	1001 × 355	0.0092	75.00	5.97	3.0350
wakaru	425 × 133	0.0196	65.50	0.90	0.1725

表 8: las2 による特異値分解結果 (2)

単語	行列サイズ	非ゼロ密度	圧縮次元数	メモリ (MB)	実行時間 (秒)
ataeru	529 × 116	0.0189	100	0.85	0.22
iu	1158 × 366	0.0070	100	6.68	4.16
ukeru	1382 × 357	0.0076	100	7.05	3.97
uttaeru	365 × 70	0.0252	27	0.34	0.03
umareru	385 × 65	0.0272	64	0.32	0.07
egaku	429 × 70	0.0223	32	0.38	0.05
omou	1201 × 465	0.0064	100	9.77	6.47
kau	472 × 87	0.0192	87	0.54	0.14
kakaru	598 × 115	0.0187	100	0.89	0.23
kaku_v	606 × 135	0.0162	100	1.12	0.30
kawaru	532 × 97	0.0179	97	0.67	0.16
kangaeru	1156 × 291	0.0085	100	4.81	2.55
kiku	711 × 180	0.0123	100	1.85	0.62
kimaru	498 × 117	0.0196	100	0.77	0.20
kimeru	807 × 228	0.0111	100	2.78	0.79
kuru	612 × 128	0.0156	100	1.06	0.30
kuwaeru	517 × 109	0.0174	53	0.77	0.11
koeru	580 × 119	0.0180	100	0.92	0.23
shiru	986 × 246	0.0090	100	3.46	1.09
susumu	581 × 112	0.0175	100	0.86	0.21
susumeru	652 × 132	0.0157	100	1.15	0.30
dasu	978 × 208	0.0105	100	2.69	0.97
chigau	653 × 105	0.0152	61	0.84	0.15
tsukau	1309 × 270	0.0075	100	4.65	1.97
tsukuru	811 × 183	0.0117	100	1.80	0.51
tsutaeru	464 × 97	0.0205	58	0.61	0.11
dekiru	461 × 75	0.0211	75	0.43	0.11
deru	1387 × 412	0.0069	100	8.63	5.81
tou	328 × 71	0.0326	69	0.32	0.07
toru	560 × 112	0.0180	54	0.84	0.13
nerau	404 × 67	0.0229	67	0.34	0.07
nokosu	544 × 98	0.0174	98	0.69	0.16
noru	321 × 64	0.0311	28	0.28	0.04
hairu	1199 × 278	0.0086	100	4.52	2.04
hakaru	448 × 84	0.0232	84	0.50	0.12
hanasu	480 × 175	0.0155	100	1.45	0.28
hiraku	795 × 224	0.0133	100	2.69	0.85
fukumu	649 × 92	0.0179	91	0.70	0.18
matsu	473 × 83	0.0220	42	0.51	0.07
matomeru	467 × 93	0.0239	93	0.58	0.13
mamoru	492 × 93	0.0188	93	0.60	0.16
miseru	448 × 100	0.0198	99	0.62	0.16
mitomeru	890 × 212	0.0108	100	2.65	0.75
miru	1502 × 408	0.0062	100	8.90	5.87
mukaeru	423 × 93	0.0262	92	0.55	0.13
motsu	1246 × 267	0.0083	100	4.40	2.13
motomeru	1171 × 306	0.0086	100	5.08	2.50
yomu	513 × 106	0.0188	100	0.73	0.17
yoru	1218 × 474	0.0076	100	10.10	5.30
wakaru	528 × 178	0.0158	100	1.44	0.45

略歴

新納 浩幸: 1985 年東京工業大学理学部情報科学科卒業. 1987 年同大学大学院理工学研究科情報科学専攻修士課程修了. 同年富士ゼロックス，翌年松下電器を 経て, 1993 年茨城大学工学部システム工学科助手. 1997 年同学科講師，2001 年同学科助教授. 情報処理学会，人工知能学会，言語処理学会，ACL

各会員．博士 (工学)．

佐々木 稔： 1996 年徳島大学工学部知能情報工学科卒業．2001 年徳島大学大学院博士後期課程修了．博士 (工学)．現在，茨城大学工学部情報工学科助手．機械学習や統計的手法による情報検索，自然言語処理等に関する研究に従事．情報処理学会，言語処理学会各会員．

(2002 年 8 月 12 日 受付)
(2002 年 10 月 23 日 再受付)
(2003 年 1 月 10 日 採録)