SVMを用いたCBCによる文書クラスタリング

執筆者：吉田　雄介

指導教官：新納 浩幸

平成２０年２月２８日

目次

図目次

表目次

第１章　序論

1.1　目的と背景

情報検索の分野では、図書や雑誌論文などの文書（documents）の集合を内容的に均質ないくつかの群に分けるため、文書クラスタリング（document　clustering）の研究が長年にわたって試みられてきた[1]。

ＣＢＣ（clustering by committee）とは、文書クラスタリング手法の１つである[2]。これは、文書全体を分類する前に、クラスタの核となるcommitteeと呼ばれる小さなクラスタを用いるクラスタリングである。はじめにcommitteeを作成し、それらをクラスタの中心として各文書を振り分ける。これにより、クラスタに部分的に属するような文書に、クラスタの中心が影響されないようになる。この手法によって算出されるクラスタの数は不定である。

本研究では、このＣＢＣの各committeeに文書を振り分ける作業を分類問題と捉え、ＳＶＭ（Support Vector Machine）を用いることで、より高精度のクラスタを求めることを目指す。

1.2　本論文の構成

第２章において、基本的な文書クラスタリングの手法について述べる。ここでは文書と単語のベクトル空間モデルとして用いる索引語文書行列の算出方法、単語の出現頻度をそのまま文書の特徴に用いることができない理由、および出現頻度をもとに、文書の特徴付けの方法を説明する。また、既存のクラスタリング手法と文書クラスタリングの問題点である次元の呪いについても説明する。

第３章では本研究のメインである、CBCについての概要、およびアルゴリズムについて説明する。

第４章ではCBCの問題点を述べ、新しい手法の提案とそのアルゴリズムの説明を行う。

第５章において、CBCの問題点を解決するために導入するSVMの概要、特徴を述べる。さらに、このSVMをどのようにしてCBCの問題解決に当てるかも説明する。

第２章　文書クラスタリング

2.1　クラスタリング

クラスタリングとはデータ解析手法の１つであり、教師なしデータ分類手法、つまり与えられたデータを外的基準なしに自動的に分類する手法のことである[3]。データの集合を部分集合（クラスタ）に切り分けて、それぞれの部分集合に含まれるデータが（理想的には）ある共通の特徴を持つようにする。この特徴は多くの場合、類似性や、ある定められた距離尺度に基づく近さで示される。さまざまな手法が提案されているが、大きく分けるとデータの分類が階層的になされる階層型手法と、特定のクラスタ数に分類する非階層的手法とがある。

また、１つのデータが唯一のクラスタに属するようにするクラスタリングと、１つのデータが複数のクラスタに属することを許すクラスタリングがある。前者は“排他的”、後者を“非排他的”なクラスタリングである。

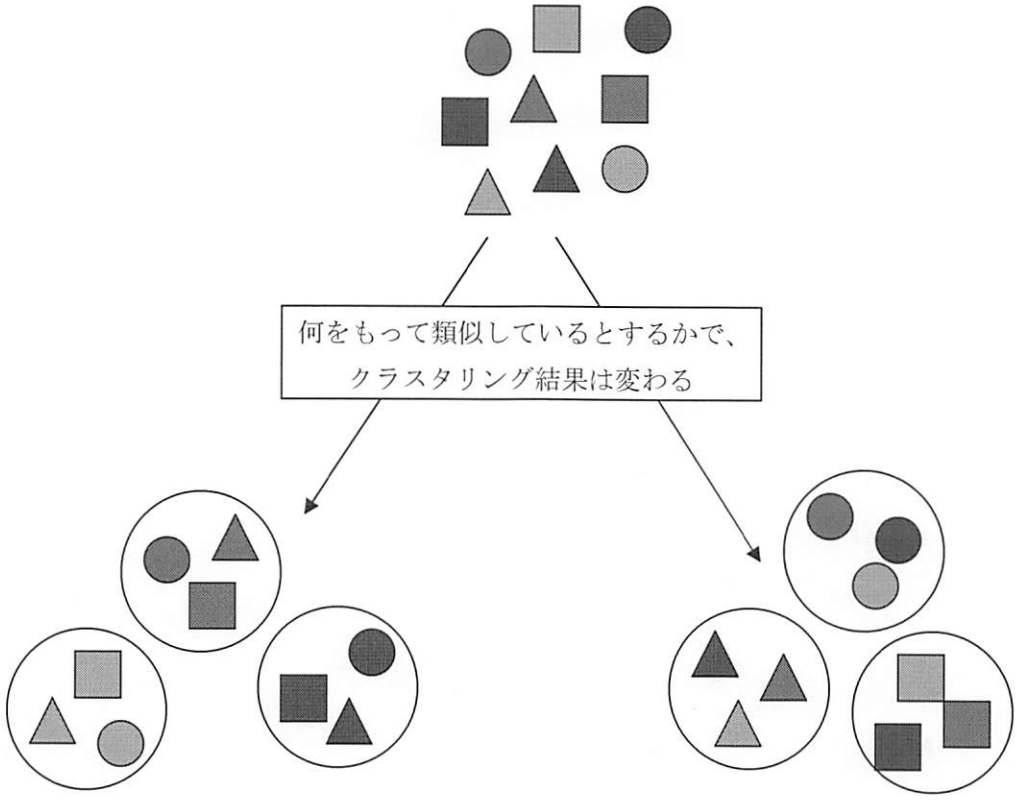

図２－１．クラスタリング

2.2 文書クラスタリング手法

文書クラスタリングとは文字通り、文書をクラスタリングすることである。文書は文がいくつも連なってできたものであり、クラスタリングをするためには文書をベクトルで表現する必要がある。

文書をベクトル化にあたって、文書に使われる各単語の出現頻度をベクトルの要素とすると、文書 d_i は

$$d_i = (c_{i1}, c_{i2}, ..., c_M) \qquad (2.1)$$

となる。ここで c_j は単語 t_j の出現頻度を表す。

N 件の文書 $d_i\ (i = 1,2,\cdots,N)$ を要素とする集合 D とすると

$$D = \begin{bmatrix} d_1 \\ \vdots \\ d_i \\ \vdots \\ d_N \end{bmatrix} \qquad (2.2)$$

となり、式（2．1）を代入すると

$$D = \begin{bmatrix} d_1 \\ \vdots \\ d_i \\ \vdots \\ d_N \end{bmatrix} = \begin{bmatrix} c_{11} & \cdots & c_{1j} & \cdots & c_{1M} \\ \vdots & \ddots & & & \vdots \\ c_{i1} & & c_{ij} & & c_{iM} \\ \vdots & & & \ddots & \vdots \\ c_{N1} & \cdots & c_{Nj} & \cdots & c_{NM} \end{bmatrix} \qquad (2.3)$$

となり、この行列を索引語文書行列という。ここで、Mは文書集合Dに含まれる単語の種類数である。

文書クラスタリングを実行するにあたって、文書が長ければ長いほど単語の出現頻度も高くなる。つまり出現頻度をそのまま特定の文書における単語の重さ、つまりその文書を表す特徴として利用することはできない。よって、なんらかの方法で各文書における単語の重要度（重み付け）を行う必要がある。

2.3 *tf-idf*

*tf-idf*とは、単語t_jが文書d_iをどの程度特徴つけているかを示す方法の１つである[4]。*tf-idf*は単語の出現頻度を示す*tf*（term frequency）と、単語の逆出現頻度を示す*idf*（inverse document frequency）の２つを掛け合わせたものである。

*tf*はある単語が、文書中に出現する出現頻度のことであり、これが大きければその単語が文書に何度も出現していること示し、その文書の特徴を強く表している単語であるという考えである。ある文書d_iにおける語t_jの*tf*は以下のように定義される。

$$tf(d,t) = \text{文書 } d \text{における単語 } t \text{の出現頻度} \qquad (2.4)$$

*idf*とは*df*の逆数である。*df*とは、ある単語が出現する文書の数を示す。これの逆数をとることにより、出現数が少ない単語ほど、文書の特徴を捉えているという考えである。ある語t_jにおける*idf*は以下のように定義される。

$$idf(t) = \log \frac{N}{df(t)} + 1 \qquad (2.5)$$

文書が長ければそれだけ単語の出現頻度が増す。そのため*tf*だけでは単語が文書の特徴を強く表しているとは言えず、*idf*と掛け合わせることにより、文書の特徴を表す重みとして使用される。よって*tf-idf*は

$$tf - idf(d,t) = tf(d,t) \times idf(t) \qquad (2.6)$$

と、定義される。

2.4 類似度

文書をベクトルで表現することによって、文書間の類似度を求めることができる[1]。2件の文書d_i, d_hの文書間の類似度は、それらのベクトルの成す角度の余弦として定義される。すなわち、

$$s(d_i, d_h) = \frac{\sum_{j=1}^{M} w_{ij} w_{hj}}{\sqrt{\sum_{j=1}^{M} w_{ij}^2} \sqrt{\sum_{j=1}^{M} w_{ij}^2}} \qquad (2.7)$$

であり、この類似度$s(d_i, d_h)$に基づいてクラスタリングを実行できる。ここで、w_{ij}は文書d_iにおける単語t_jの重みを表す。

ベクトル間の類似度がユークリッド距離ではなく、余弦係数（またはそれに類した尺度）に基づいて計算されることは文書クラスタリングの特徴の一つである。ただし、文書ベクトルが標準化され、その長さが1、すなわち、$\tilde{d}_i = d_i / \|d_i\|$であるならば順位付けという点では平方ユークリッド距離と余弦係数とは同一の結果を与える。そのため、この距離を非類似度として用いる研究もある。

文書ベクトル$\tilde{d}_i$と$\tilde{d}_h$との間のユークリッド距離$S(\tilde{d}_i, \tilde{d}_h)$は

$$S(\tilde{d}_i, \tilde{d}_h) = \sqrt{\sum (w_{ij} - w_{hj})^2} \qquad (2.8)$$

と、定義される。

2.5 クラスタリング手法

クラスタリング手法には大きく分けて階層的手法（hierarchical）と、分割最適化手法（partitioning-optimization）に分けられる。ここではこれらの基本的手法を説明する[5]。

・階層的手法

この手法にはさらに分枝型（divisive）と凝集型（agglomerative）に分けることができるが、ここでは後者のみについて説明する。

凝集型について。はじめにN件のデータが与えられたとき、初期状態としてデータを1つだけ含むクラスタがN個ある状態を作る。次に、対象x1とx2との距離$S(x_1,x_2)$（非類似度）をもとにクラスタ間の距離$S(C_1,C_2)$を計算し、その距離が最も近いクラスタ同士を合併していく。これを全てのクラスタが合併し、1つになるまで繰り返すことで、階層構造を持ったクラスタを得る。クラスタ間の距離$S(C_1,C_2)$を求めるには、以下のような手法がある。

・最短距離法

最短距離法では、2つのクラスタ間の距離は、それぞれのクラスタに含まれるデータの対の中で、最も近いデータの距離として定義されている。よって、距離$S(C_1,C_2)$は以下に示す式によって表される。

$$S(C_1,C_2)=\min_{x_1\in C_1,x_2\in C_2} S(x_1,x_2) \qquad (2.9)$$

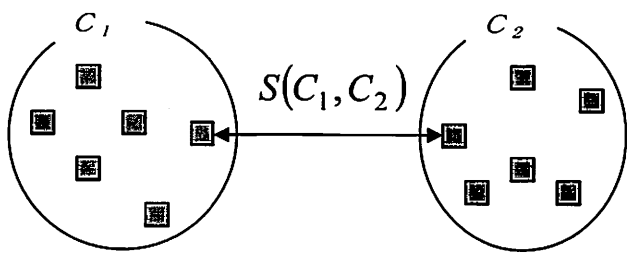

図2－2．最短距離法

・最長距離法

最長距離法では、最短距離法と逆に、それぞれのクラスタに含まれるデータの対の中で、最も遠いデータの距離をクラスタ間の距離として定義している。よって、距離$S(C_1,C_2)$は以下に示す式によって表される。

$$S(C_1,C_2)=\max_{x_1\in C_1,x_2\in C_2} S(x_1,x_2) \qquad (2.10)$$

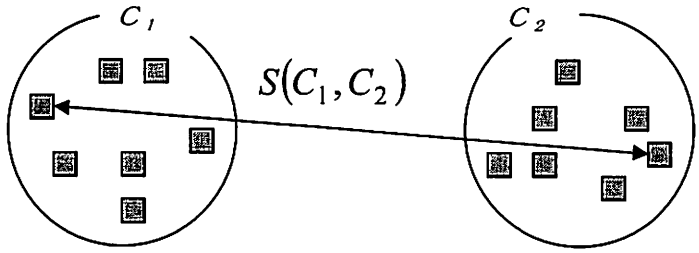

図２－３．最長距離法

・群平均法

群平均法では２つのクラスタ間の距離は、２つのクラスタ内のすべてのデータのペアの距離の平均として定義される[6]。クラスタC_1、C_2のデータの数をそれぞれn_1, n_2とすると、合併後に出来るクラスタC_tと任意のクラスタC_rとの距離$S(C_t,C_r)$は以下に示す式によって表される。

$$S(C_t,C_r)=\frac{(n_1S(C_1,C_r)+n_2S(C_2,C_r))}{(n_1+n_2)} \qquad (2.11)$$

・ウォード法

ウォード法では、クラスタ間の距離を分散分析的アプローチで評価しているため，他の方法とはかなり異なる。簡単にいえば、この方法は、各ステップで形成される任意の２つのクラスタの平方和を最小にするよう試みる。一般にこの方法は、非常に有効だが、小さいクラスタを作りやすい傾向にある。クラスタC_1、C_2のデータの数をそれぞれn_1, n_2とすると、合併後に出来るクラスタC_tと任意のクラスタC_rとの距離$S(C_t,C_r)$は以下に示す式によって表される。

$$S(C_t,C_r)=\frac{n_1+n_r}{n_t+n_r}S(C_1,C_r)+\frac{n_2+n_r}{n_t+n_r}S(C_2,C_r)-\frac{n_r}{n_t+n_r}S(C_1,C_2)$$

（２．１２）

・分割最適化手法

分割最適化手法は分割のよさの評価関数を定め、その評価関数を最適にする分割を探索する手法である[5]。可能な分割の総数はNに対して指数的なので、実際は準最適解を求めることになる。k-means法は代表的な手法である。

・k-means法

k-means法では、セントロイド（クラスタの重心点）x_iをクラスタの代表点とし、

$$\sum_{i=1}^{k}\sum_{d\in C_i}(S(d,x_i))^2$$

（２．１３）

の評価関数を最小化するようにk個のクラスタを分割する[5]。最適解の探索は図２－５に示すように、対象のクラスタへの割り当てと代表点の再計算を交互に繰り返して行う。図２－４はこの手法のイメージ図である。この手法は山登り法であり、局所最適解しか求められないため、ランダムに初期値を変更して、評価関数を最小にする結果を選択するのが一般的である。この手法の分類対象のクラスタへの割り当てと、代表点の更新とが反復的に繰り返され、この反復回数をrとすれば計算量は

$O(NMLr)$となる。ここでNは文書数、Mは単語数、Lはクラスタの数である。階層的クラスタ分析の計算量がN^2に比例してしまうのに対し、k-means法は計算量が少ないという利点があるが、クラスタの個数とその代表点を先験的に与える必要があり、教師なしの分類である文書クラスタリングにおいては望ましい条件ではない。

k-means法は重心の再計算と文書の割り当てを繰り返して最終的なクラスタを求めるわけだが、図２－６に示すような、クラスタとクラスタの境界付近にあるような文書が加わると、重心の再計算の際に重心が大きく動いてしまうことがある。

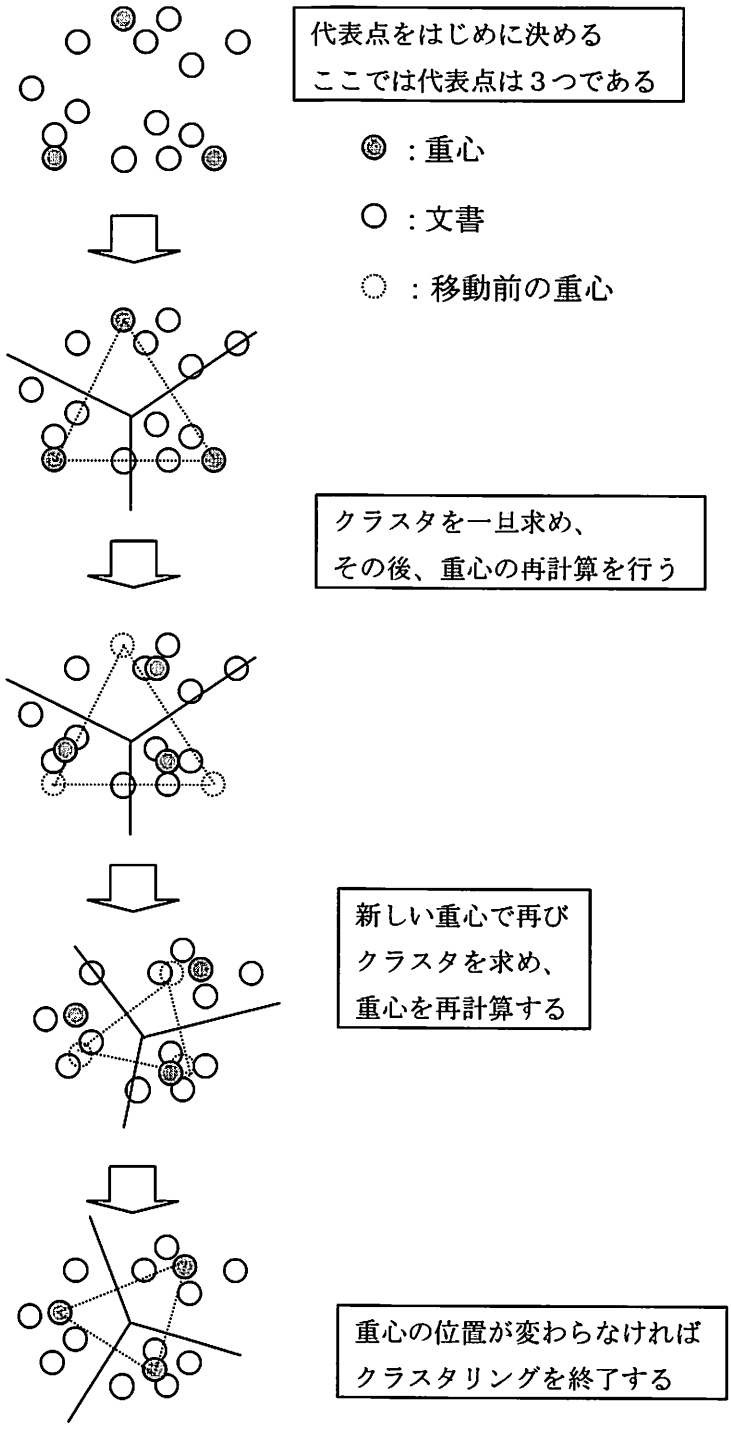

図２－４．k-means 法のイメージ

1．k個の代表点$x_1,...,x_k$をランダムに選択する

2．Dの全ての対象dを$c^* = \arg\min_{x_i} S(d, x_i)$なる代表点に割り当てる

3．もし代表点への割り当てが変化しないならば終了し、
そうでなければ各クラスタのセントロイドを代表点にしてステップ２へ

図２－５．k-means法のアルゴリズム

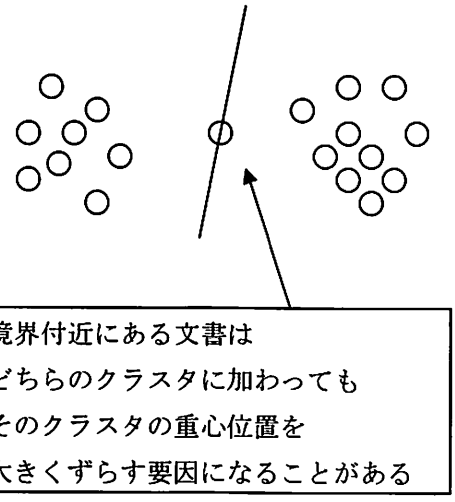

図２－６．k-means 法の問題点

第3章　CBC

3.1　CBC

一般的な文書クラスタリング手法の1つk-means法では、クラスタの重心の再計算の際、そのクラスタに部分的にしか属さないような文書が不当な影響を与えてしまうことがある。その結果、各クラスタ内の結合がゆるくなり、1つのクラスタの内容が多様になってしまう可能性がある。この問題を解決するために Pantel と Lin は CBC（clustering by committee）というアルゴリズムを考案した[2]。

CBCとは、クラスタの核としてcommitteeと呼ばれる小さいクラスタを用いた文書クラスタリングである。このcommitteeとは非常に信頼性の高いクラスタリング結果の一部であり、この核が中心になるように文書を割り当てていきクラスタを求める。これによって、クラスタに部分的にしか属さない文書がクラスタの重心の計算に不当な影響を与えないようにしている。

3.2 相互情報ベクトル

第2章の 2.3 節では類似度は各文書における単語の重みから計算することを述べたが、Pantel と Lin は類似度の計算に相互情報ベクトル $MI(d)=(mi_{e1}, mi_{e2}, ..., mi_{eM})$ を用いた[2]。相互情報ベクトルとは文書 d_c と、各単語との相互情報量を表すものである。その定義式は

$$mi_{dt} = \log \frac{\frac{c_{dt}}{N}}{\frac{\sum_i c_{it}}{N} \times \frac{\sum_j c_{dj}}{N}} \qquad (3.1)$$

となる。ここで N は

$$N = \sum_i \sum_j c_{ij} \qquad (3.2)$$

である。

類似度は相互情報ベクトルの成す角度の余弦と定義されるので、文書 d_i と d_j の類似度 $s(d_i, d_j)$ は

$$s(d_i, d_j) = \frac{\sum_t mi_{d_i t} \times mi_{d_j t}}{\sqrt{\sum_t mi_{d_i t}^2 \times \sum_t mi_{d_j t}^2}} \qquad (3.3)$$

と表される。

3.3 アルゴリズム

CBC は３つのフェーズからなる[1]。

・フェーズ１

文書ごとに類似度の高い k 個の文書を求める。

・フェーズ２

フェーズ１で求めた類似度の高い文書群をもとに committee の候補を作成する。作成した committee の候補は、既存の committee の重心との類似度を計算し、類似度が低いものを committee に追加する。これによって互いの類似性が低い committee を算出できる。フェーズ２の具体的なアルゴリズムは図３－１の通りである。

・フェーズ３

各文書を算出した committee の重心との類似度をもとに振り分け、クラスタを求める。

この手順から明らかなように、CBC アルゴリズムではフェーズ１、フェーズ２にて各クラスタの核となる committee を作成し、フェーズ３で最終的なクラスタを求めている。これによってクラスタに部分的にしか属さない文書がクラスタの重心の計算に不当な影響を及ぼすことを防いでいる。

入力．２つの閾値θ_1とθ_2を設定し、リストEにすべての文書を入れる。

ステップ１．committee の候補を入れるリストL_cを空にし、リストEに含まれる文書ごとに以下の処理を行う。

- その文書と類似度の高いk個の文書に対して群平均クラスタリングを行う。
- 群平均クラスタリングで生成されたクラスタ C ごとにスコア $|C| \times avgsim(C)$ の計算を行う。ここで$|C|$はクラスタCに含まれる文書の数、$avgsim(C)$はクラスタC中の文書の組ごとの類似度の平均である。
- スコアの高いクラスタをリストL_cに追加する。

ステップ２．リストL_cをスコアの降順に並び替える。

ステップ３．リストL_c中の上位のクラスタCから順に以下の処理を行う。

- クラスタCの重心と、その時点で作成された committee の重心との類似度を求める。
- すべての committee との類似度が閾値θ_1を下回っているならばそのクラスタCを committee のリストL_kに追加する。

ステップ４．リストL_kへの追加がなければフェーズ２を終了する。

ステップ５．リストEに含まれる文書ごとに以下の処理を行う。

- その文書とすべての committee の重心との類似度を計算し、すべて閾値θ_2を下回るならその文書をリストL_Rに追加する。

ステップ６．リストL_Rに追加された文書がなければフェーズ２を終了する。文書が追加されていたらリストRをリストEとしてステップ１を繰り返す。

図３－１．フェーズ２

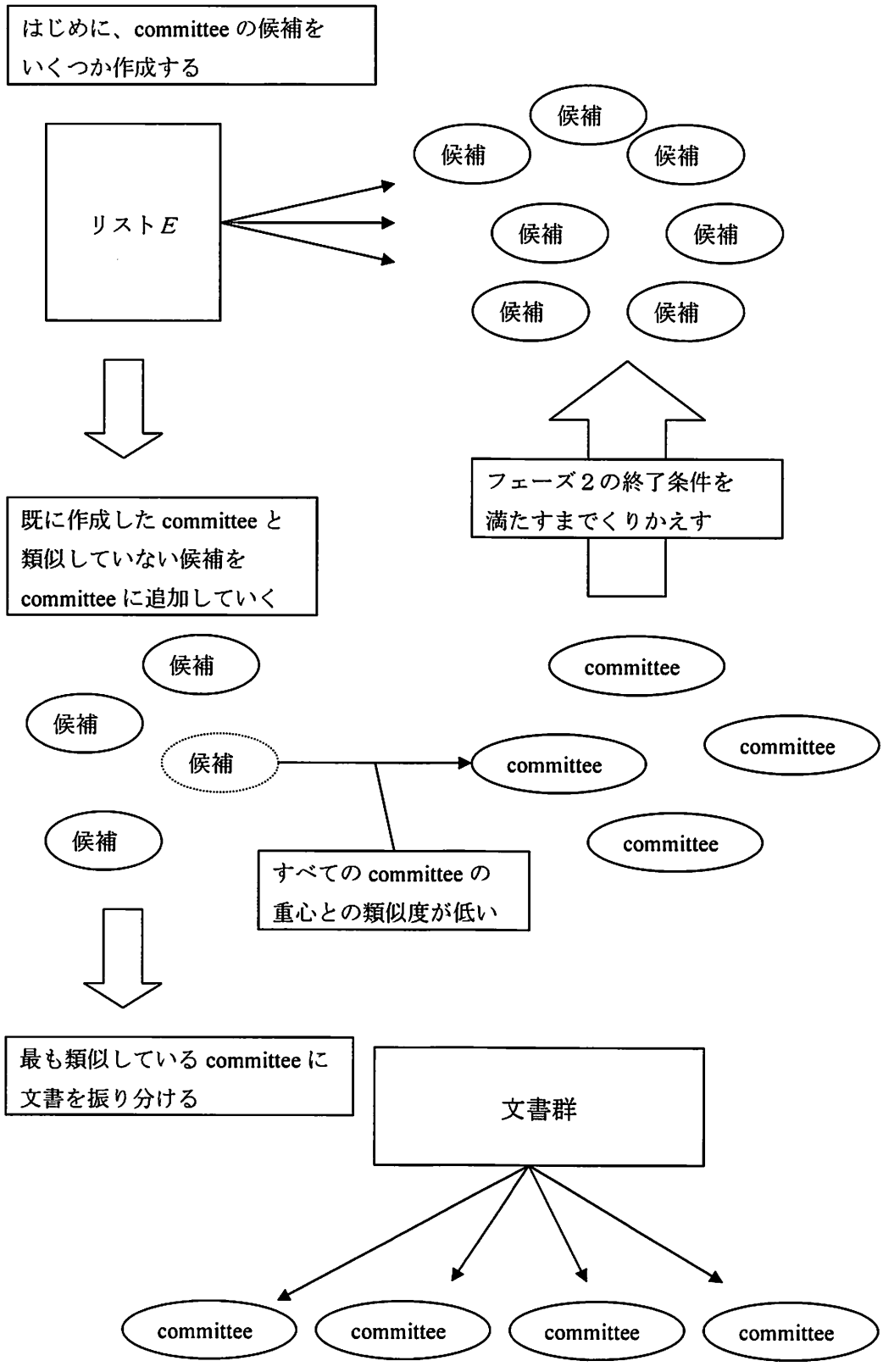

図３－２．CBC の大まかな流れ

第４章　CBC の改良

4.1　改良点

本研究の目的は CBC のクラスタリング結果の精度の向上である。今回、精度の向上を目指し、以下の２つの改良を行う。

１．算出されるクラスタの数を指定できるようにする

CBC は、はじめに与えたパラメータによって算出されるクラスタの数が異なってくる。これは実質、算出するクラスタの数をはじめに決めていると考えられる。

クラスタの算出される数を直接指定できるようにすることに対して疑問視する人もいるかもしれないが、クラスタリングとは結果を見て考察するためのツールであり、正解というものがない。よって、クラスタの数を指定できるようにすること自体に大きな問題はないと言える。

２．各 committee へ文書を振り分ける作業に SVM を適用する

CBC の、文書を各 committee に振り分ける方法は committee と文書との類似度に基づいて行われるが、この方法はシンプルで分かりやすい反面、正確な振り分けが期待できない。つまり、より信頼できる振り分け方法を取り入れることにより、クラスタリング結果の向上を期待できる。

本研究ではこの文書の振り分け作業を分類問題と捉え、分類問題に強い SVM (Support Vector Machine) を適用する。SVM の詳細や具体的な適用方法については次章にて行うが、簡潔に述べると、これはトレーニングデータとしていくつかの正例と負例を与えることにより識別関数というものを学習する。その識別関数にデータを与えるとそのデータが正に属する信頼度、もしくは負に属する信頼度が算出される。この SVM の結果をもとに文書を各 committee に振り分け、クラスタを求めることで、クラスタリング結果の向上を目指す。

本研究では以上の２点の改良を行い、CBC のクラスタリング結果の向上を目指す。

4.2 作成したプログラムの説明

本研究において、２つのプログラムを作成した。１つは指定された数、committee を算出するプログラム「MC.java」。もう１つは SVM の結果から各文書を committee に振り分けるプログラム「Distributer.java」である。

- MC.java

  このプログラムは CBC のフェーズ１とフェーズ２のアルゴリズムをもとに作成したものである。与えるパラメータは閾値 θ_1 と θ_2、文書ごとに求める類似した文書の数 k、算出する committee の数の４つである。また、このプログラムには引数としてデータセットの他に予め計算された *tf-idf* の結果を与えている。

  アルゴリズム

  はじめにデータセットや *tf-idf* のデータを変数に格納し、次にユークリッド距離と式（３．１）、（３．２）（３．３）を用いて各文書間の類似度を求めている。その後、図４－１のアルゴリズムに従い committee を算出する。committee の算出後は、算出した committee の表示と、SVM で使えるようにフォーマットしたトレーニングデータを作成する。

  データセットのフォーマット

  １行目に行数、列数、非ゼロ要素の数を指定し、２行目以降が文書のデータとなっている。行列の行が文書に対応し、行列の列が単語に相当する。非ゼロ要素の数はプログラム中に使用していないため、適当な整数を充てておくだけでも問題ない。文書のデータを示す部分は　列 頻度 列 頻度...列 頻度　となっており、頻度がゼロの部分は記載されていない。図４－２はデータセットのフォーマットの例である。

  tf-idf のフォーマット

  すべての行で　行 列 *tf-idf*　と並んでいる。データセットと同様に、行が文書に対応し、列が単語に相当する。図４－３は *tf-idf* のフォーマットの例である。

ステップ１．すべての文書をリストEに入れる。

ステップ２．リストEに含まれる文書ごとに、以下の処理を行う。

- その文書と類似度の高いk個の文書で、群平均クラスタリングを行う。
- 群平均クラスタリングで生成されたクラスタ C ごとにスコア $|C| \times avgsim(C)$ の計算を行う。ここで $|C|$ はクラスタ C に含まれる文書の数、$avgsim\ (C)$ はクラスタ C 中の文書の組ごとの類似度の平均である。
- スコアが最も高かったクラスタをリストL_cに入れる。

ステップ３．リストL_cを降順にする。

ステップ４．リストL_c中の上位のクラスタCから順に以下の処理を行う。

- クラスタ C の重心と、その時点で作成された committee の重心との類似度を求める。
- すべての committee との類似度が閾値θ_1を下回っているならばそのクラスタ C を committee のリストL_kに追加する。
- もし、リストL_kに追加された committee が指定した数に達した場合は committee の算出処理を終了する。

ステップ５．リストL_kへの追加がなければ閾値θ_1を⊿だけ増やしてステップ４を繰り返す。ここでは⊿=0.01 とした。

ステップ６．リストEに含まれる文書ごとに以下の処理を行う。

- その文書とすべての committee の重心との類似度を計算し、すべて閾値θ_2を下回るならその文書をリストL_Rに追加する。

ステップ６．リストL_Rに追加された文書がｎ個未満なら閾値θ_2を⊿だけ増加し、リストRの中を空にしてステップ６を繰り返す。追加された文書がｎ以上ならリストRをリストEとしてステップ２を繰り返す。ここでは ｎ＝k×残りの作成する committee の数とした。

図４－１．改良 CBC のアルゴリズム

```
414 6429 116613
  29 1.00   31 9.00   34 3.00  259 1.00  330 1.00  347 2.00  352 2.00 ...
  31 7.00   37 1.00  257 1.00  270 2.00  348 3.00  351 1.00  352 2.00 ...
  28 2.00   29 1.00   31 9.00   36 1.00  352 2.00  354 1.00  356 1.00 ...
  28 2.00   29 1.00   31 9.00   37 3.00  352 2.00  354 1.00  356 1.00 ...
  29 1.00   31 9.00   36 1.00  352 2.00  354 2.00  356 1.00  359 1.00 ...
  31 7.00  112 1.00  154 1.00  270 2.00  352 2.00  357 2.00  360 1.00 ...
   :
   :
```

図４－２．データセットのフォーマットの例

```
1 29 0.005188
1 31 0.000233
1 34 0.063401
1 259 0.036399
1 330 0.034964
1 347 0.038537
1 352 0.000052
1 353 0.028578
:
:
```

図４－３．*tf-idf* のフォーマットの例

- Distributer.java

このプログラムは SVM で学習させた識別関数にデータセットを与えたことで得られた分類結果と、文書数、committee の数を与えることにより、各文書を最も適切な committee に振り分けるものである。

アルゴリズム

文書数、クラスタ数、SVM の結果が書かれたテキストファイルを受けとり、それらをそれぞれの変数に格納する。次に各文書に対し、最も正である信頼度が高い committee を調べ、そこに振り分けてクラスタを求める。最後にクラスタ結果を表示して終了。

データのフォーマット

1行目に文書数、committee の数を指定する。2行目以降から文書数まで committee 1 に属する信頼度、もしくは属さない信頼度が入る。その次の行から文書数分まで committee 2 に属する信頼度、もしくは属さない信頼度が入る。同様のフォーマットですべての committee の分、その committee に属する信頼度、もしくは属さない信頼度が入る。図4－4がデータの例である。

```
文書数 クラスタの数
文書1が committee 1 に属する信頼度 or 属さない信頼度
文書2が committee 1 に属する信頼度 or 属さない信頼度
  :
  :
  :
  :
文書 d_i が committee X に属する信頼度 or 属さない信頼度
  :
  :
  :
```

図4－4．Distributer.java で使うデータのフォーマット

第５章　SVM

5.1　SVM

SVM (Support Vector Machine) とは統計学習理論にもとづく学習システムの１つである[7]。トレーニングデータとしていくつかの正例と負例を与えることで識別関数を学習する。その識別関数にデータを与えると、そのデータが正である信頼度、もしくは負である信頼度が算出される。つまり、トレーニングデータを与えることで、与えられたデータが正であるか負であるかを識別することができることができるわけである。

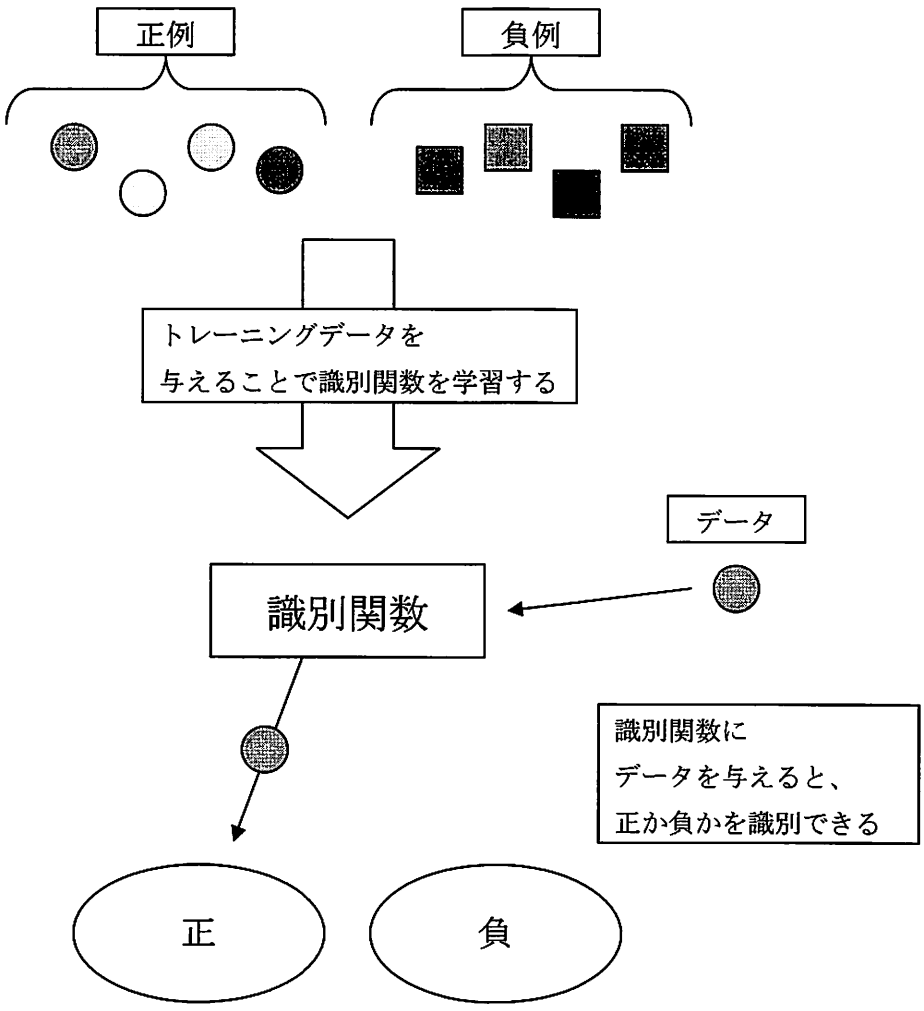

図５－１．SVM の概要

5.2 マージン最大化

SVM はパターン認識手法の１つである[8]。SVM 以外にもパターン認識手法はいくつも提唱されているが、SVM にはマージン最大化という特徴を持つ。マージン最大化とは、データを２つの種類に分離するための分離平面の決定を各データ点との距離が最大になるようにすることである。

具体的な例として、図５－２のような２次元の特徴空間に図のような２つのクラスＡとＢに属するデータがあったとする。

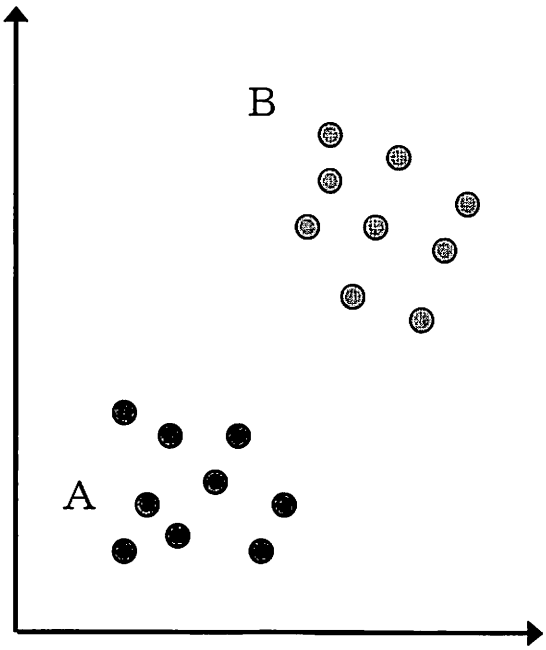

図５－２．２次元の特徴空間

この図に線を引いて分離させるとき、図５－３（ａ）のような分け方だけでなく、図５－３（ｂ）、図５－３（ｃ）のような分け方もできてしまう。ここで出てくるのがマージン最大化である。

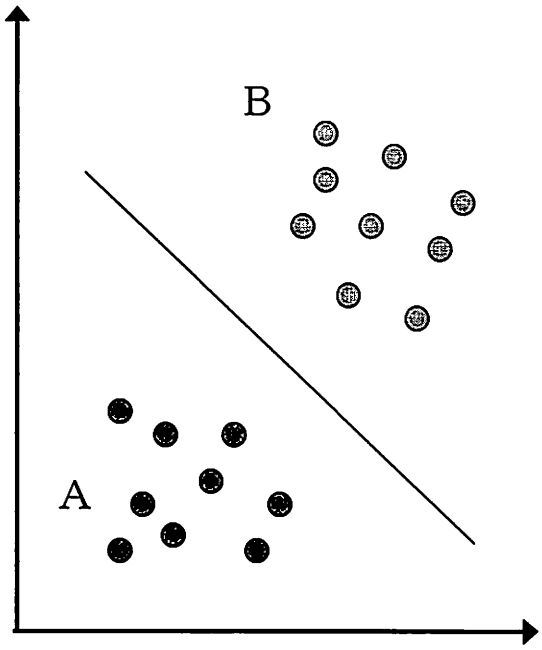

図５－３（ａ）．境界線の例１

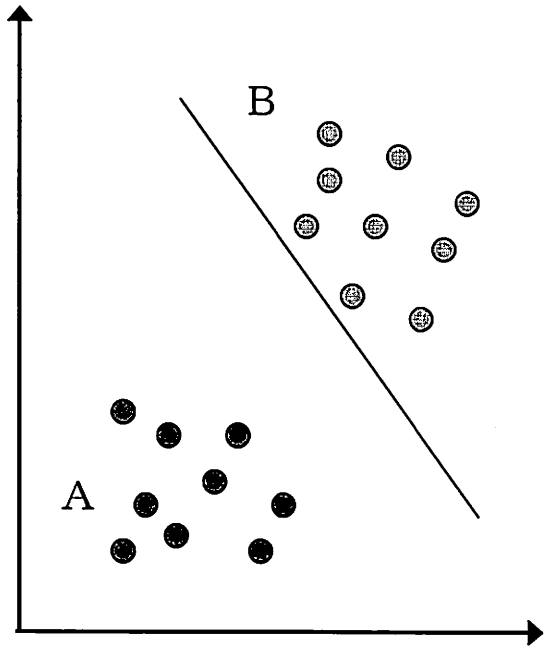

図５－３（ｂ）．境界線の例２

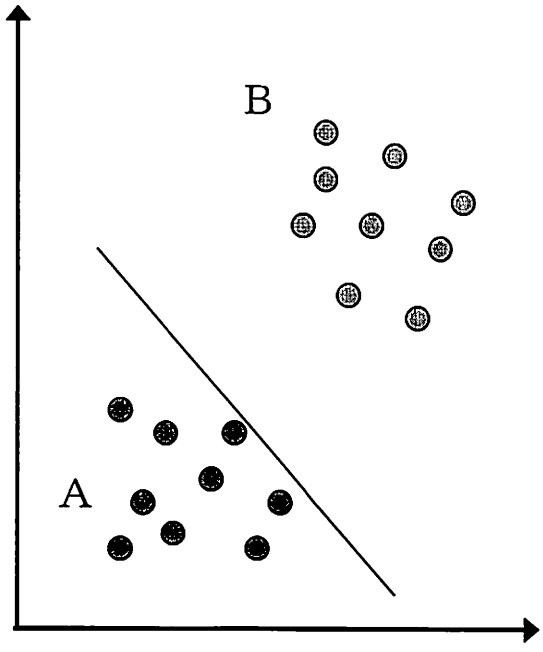

図５－３（ｃ）．境界線の例３

最も適した識別境界はどこか、という考え方は大きく分けて以下の２通り。

・パラメトリックな方法
　統計的に学習データの分布を考えて識別境界を引く

・ノンパラメトリックな方法
　与えられた学習データを全て正しく認識できるように識別境界を引く

SVM は後者のノンパラメトリックな方法である。SVM では、学習データの中の最も他クラスと近い位置にあるもの（これをサポートベクトルと呼ぶ）を基準として、そのユークリッド距離が最も大きくなるよう内地に識別境界を設置する。つまり、クラスの最端から他クラスまでのマージンを最大にするようにする。これがマージン最大化である。先程の例であれば図５－４のようになる。

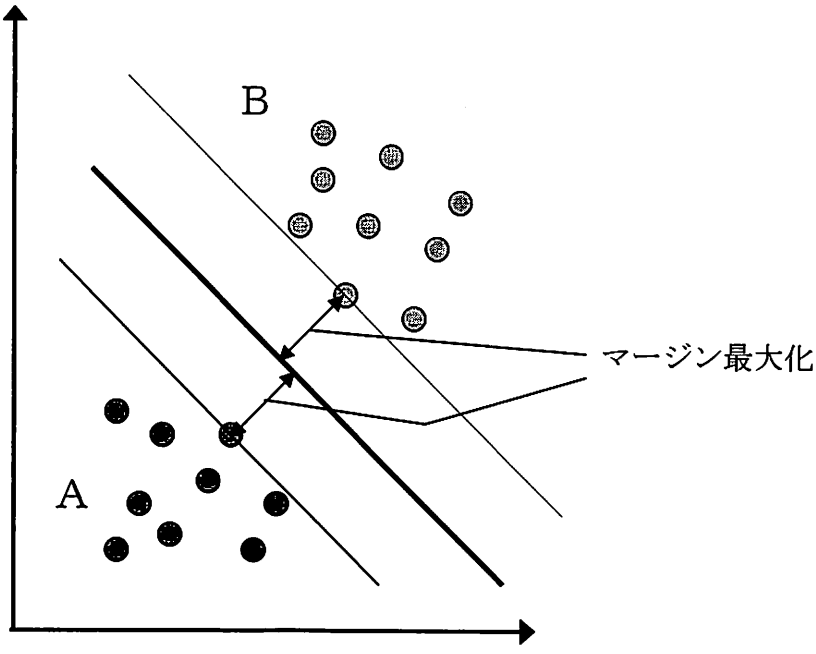

図５－４．マージン最大化

SVM 以外にもニューラルネットワークを使ったパターン識別手法として、多層パーセプトロンをバックプロパゲーションで学習させる方法などがあるが、このバックプロパゲーション学習は、学習アルゴリズムの本質として与えられたトレーニングデータに関してのみ学習結果を保証するものである。つまり、特徴空間上の学習データが与えられていない領域に関しては、学習結果は初期値に依存することになる。そのため、図５－３（ｂ）や

図５－３（ｃ）のような境界線を引いてしまうこともある。一方のクラスに属するデータの周りの領域が、もう一方のクラスに属するデータの周りの領域より大きくなると、領域が狭いほうのクラスは汎化能力（学習データとは異なったパターンのデータに対して正しく応答できる能力）が低くなってしまう。また、バックプロパゲーション学習もSVMと同様にノンパラメトリックな手法で識別境界を決めているが、マージンを最大にするという、識別境界の位置を決定する明確な基準を持っていることもSVMの特徴である。

5.3 SVM の適用方法

本研究では、算出した committee をトレーニングデータとし、識別関数を学習させ、分類していくことで、クラスタを求めることが目的である。SVM は２値分類であり、同時に３つ以上のものを識別することはできない。つまり、算出した committee が３個以上あると、SVM をそのまま適用することができない。そこで、ある特定の committee のみを正例とし、それ以外の committee を負例として識別関数を学習させ、データセットを分類させる。これにより、特定の committee に文書が加わる信頼度を知ることができる。つまり、この作業をすべての committee に対し行うことで、文書ごとに各 committee に振り分けられる信頼度を知ることができる。

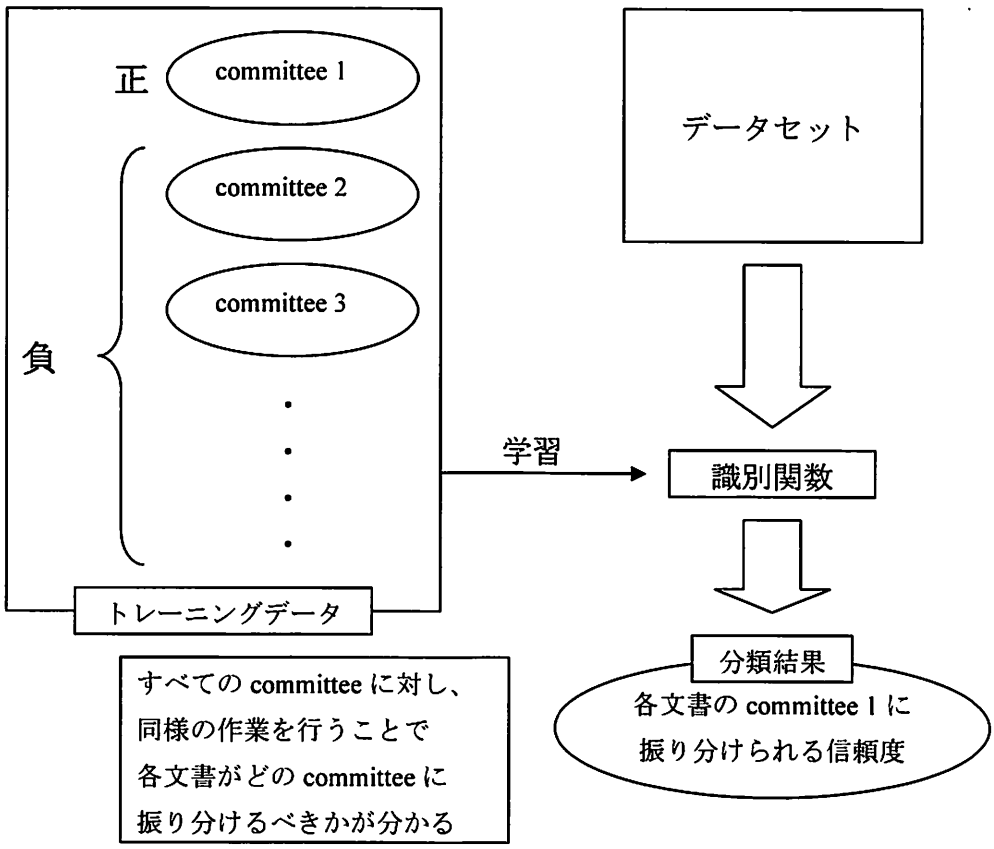

図５－５．SVM の適用方法

第６章　実験

6.1　実験概要

本研究では、改良したCBCの精度を調べるために指定した数、committeeを算出させた後に以下の２つの方法で５つのデータセットtr11、tr12、tr31、tr41、tr45のクラスタリングを行った。使用したデータセットの詳細は表６－１に示す。

- 従来のcommitteeの重心との類似度にもとづき各文書を振り分け、クラスタを求める。
- 第５章の5.3節にある方法で文書を振り分け、クラスタを求める。

表６－１．データセット

データセット	文書数	単語数	正解クラスタの数
tr11	414	6429	9
tr12	313	5804	8
tr31	927	10128	7
tr41	878	7454	10
tr45	690	8261	10

また、実験で与えたパラメータについては表６－２に示す。

表６－２．パラメータ

データセット	k	θ_1	θ_2	算出するcommitteeの数
tr11	19	0.25	0.10	9
tr12	28	0.23	0.10	8
tr31	35	0.05	0.10	7
tr41	24	0.19	0.10	10
tr45	29	0.03	0.10	10

この２つのクラスタリング結果の精度を比較し、SVM を用いた文書振り分けによって精度が向上したかを調べる。精度の比較には Purity と Entropy を用いる。

Purity

クラスタの純度を表す。正解クラスタとの一致の度合いであり、高い方が精度が良いことを示す。

Entropy

クラスタの情報の曖昧さを表す。１つのクラスタに含まれる正解クラスタと一致しない文書のばらつきが多いと高くなる。Entropy は低い方が精度が良いことを示す。

6.2 実験結果

実験結果は表６－３のようになった。精度の向上が見られたのは tr41 だけであり、他のデータセットでは精度にあまり差がでなかったものや、従来の手法の方が精度が高かったものがあり、SVM を用いることで committee の精度が向上したとは言えない。

表６－３．実験結果

データセット	振り分け方法	Purity	Entropy
tr11	従来	0.804	0.275
	SVM	0.746	0.331
tr12	従来	0.700	0.401
	SVM	0.712	0.383
tr31	従来	0.684	0.454
	SVM	0.688	0.490
tr41	従来	0.673	0.387
	SVM	0.822	0.276
tr45	従来	0.812	0.313
	SVM	0.678	0.436

第７章　考察

実験結果から、本研究で行った改良では CBC の精度の向上はできなかった。この原因を探るため、実験で得られた各データセットの committee の精度を調べたら表７－１のようになった。

表７－１．committee の精度

データセット	平均メンバー数	Purity	Entropy
tr11	13.667	0.849	0.206
tr12	17.250	0.876	0.173
tr31	18.857	0.932	0.149
tr41	23.800	0.983	0.027
tr45	20.700	0.841	0.221

実験にて、精度が従来の手法よりも明らかに悪くなった tr11、tr45 の committee の精度は、５つのデータセットの中でも特に精度が低いことが分かる。また、表の中で２番目、３番目に良い精度の committee を得られた tr12、tr31 は従来の手法と精度にあまり差がみられなかったが、SVM を用いることで精度が向上した tr41 の committee の精度は他のどの committee よりも明らかに精度が高い。

SVM はトレーニングデータを用いる教師あり学習である。間違いのあるトレーニングデータ、つまり正しく分類できていないトレーニングデータを与えてしまうと SVM の分類結果も悪くなってしまうのは言うまでもない。今回の実験では精度の改善がみられたデータセットは tr41 だけである。

tr41 の committee は非常に精度が高く、平均メンバー数も今回の実験で得られた committee の中では最も多く、正解クラスタ数も tr45 と並んで 10 個と、実験に使用したデータセットの中では多い部類に入る。つまり正しく分類できている信頼の高い committee をトレーニングデータとして他のどの committee よりも多く SVM に与えることができた。このことが tr41 だけ精度が向上した理由と考えられる。

次に精度が従来の手法より劣る結果になった tr11 と tr45 の committee を比較する。この２つの committee はトレーニングデータとしては精度が悪かったために従来の手法に劣る分類結果になったと考えられるが、tr11 の committee は tr45 の committee に比べ、精度がわずかに良いものの平均メンバー数、正解クラスタ数が少ない、つまりトレーニングデータとし

て与えられたデータが明らかに少ない。しかしtr11のSVMを用いたクラスタリング結果の方がtr45のそれよりも精度が良い。従来の手法で振り分けた場合はtr45の方が精度が良く、精度の悪化の度合いはtr11よりtr45の方が大きいと言える。このことより、SVMに与えるトレーニングデータは量よりも質、数が少なくても、少しでも良い精度のcommitteeを与えることが重要だと考えられる。

第8章　結論

本研究ではCBCの高精度化を目指し、算出されるクラスタの数を指定し、SVMによる文書の振り分けを用いる改良を行った。

5つのデータセットで実験を行ったところ、算出できたcommitteeの精度はSVMのトレーニングデータとしては不十分であった。また、SVMにトレーニングデータとして与えるデータは、数多く与えるよりも、少しでも精度が良いものを与えると良い結果になることが分かった。

SVMを使って高精度化を目指すために、より精度の高いcommitteeの算出アルゴリズムを考案することが、今後の課題となる。

参考文献

[1] 岸田和明 :"文書クラスタリングの技法：文献レビュー", Library and Information Science, No.49, pp.33-75 (2003).

[2] Patrick Pantel, Dekang Lin : "Document Clustering with Committees". Proceedings. Of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp.199-206 (2002).

[3] "データクラスタリング –Wikipedia–",

<http://ja.wikipedia.org/wiki/%E3%83%87%E3%83%BC%E3%82%BF%E3%83%BB%E3%82%AF%E3%83%A9%E3%82%B9%E3%82%BF%E3%83%AA%E3%83%B3%E3%82%B0>, (2008/02/27アクセス)

[4] "tfidf", <http://nlp.nagaokaut.ac.jp/~sekiguti/doc/pdf/tfidf/tfidf/tfidf.html>, (2008/02/28アクセス)

[5] 神嶌 敏弘, "データマイニング分野のクラスタリング手法(1) ー クラスタリングを使ってみよう！ ー", 人工知能学会誌, vol.18, no.1, pp.59-65 (2003).

[6] 阿部圭司, "クラスター分析",
<http://www1.tcue.ac.jp/home1/abek/htdocs/stat/Excel/cluster/cluster.pdf>, (2008/02/27 アクセス)

[7] Nello Cristianini, John Shawe-Taylor 著, 大北剛 訳: "サポートベクターマシン入門", 共立出版 (2005).

[8] "SVM を使うと，なにが嬉しいの？",
< http://www.neuro.sfc.keio.ac.jp/~masato/study/SVM/SVM_1.htm >, (2008/02/25アクセス)

付録 A

(MC.java)

```
import java.text.*;
import java.io.* ;

class MC {
  public static int hyouka ;

  public static int data[] = new int[3] ;//行数、列数、非ゼロ数を格納
  public static int sim_k ;//フPhase２の群平均する類似文書数
  public static int committee_num ;//算出するcommittee数
  public static float C_dt[][] ;//頻度数ベクトル
  public static double theta1, theta2 ;//閾値θ1, θ2
  public static double delta = 0.01 ;//閾値の増加量
  public static double Euclid[][] ;//ユークリッド距離
  public static double tf_idf[][] ;//tf-idf
  public static double sim[][] ;//文書間類似度
  public static Cluster committees[] ;//committee

  public static void main( String args[] ) {
    int h, i, j, k, l, m, n ;//汎用変数
    DecimalFormat exFormat1 = new DecimalFormat("0.000") ;
    try {

      //文字を読み取って、表示する
      double a, b, c ;//汎用変数
      double t1, t2 ;//閾値θ1, θ2
      String str ;//データの読み書きに使用

      //最初にデータを取得
      {
        int df[] ;//tf-idfで使う

        theta1 = Double.parseDouble( args[1] ) ;
        theta2 = Double.parseDouble( args[2] ) ;
        t1 = theta1 ;
        t2 = theta2 ;
        sim_k = Integer.valueOf( args[3] ).intValue() ;
```

```
committee_num = Integer.valueOf( args[4] ).intValue() ;
committees = new Cluster[committee_num] ;

FileReader fr = new FileReader( args[0] ) ;
BufferedReader br = new BufferedReader( fr ) ;

str = br.readLine() ;
for( i = 0 ; i < 3 ; i++ ) {
  str = str.trim() ;
  n = str.indexOf( ' ' ) ;
  if( n != -1 ) {
    data[i] = Integer.valueOf(str.substring(0,n)).intValue() ;
    str = str.substring( n+1 ) ;
  }
  else
    data[i] = Integer.valueOf(str).intValue() ;
}

//行列のデータを取得(頻度数ベクトルにデータを入れる)
C_dt = new float[data[0]][data[1]] ;
df = new int[data[1]] ;
double c_sum[] = new double[data[1]] ;

for( i = 0 ; i < data[0] ; i++ ) {
  str = br.readLine() ;
  while(true) {
    str = str.trim() ;
    n = str.indexOf( ' ' ) ;
    j = Integer.valueOf(str.substring(0,n)).intValue()-1 ;
    str = str.substring( n+1 ) ;
    str = str.trim() ;
    n = str.indexOf( ' ' ) ;
    df[j]++ ;
    if( n != -1 ) {
      C_dt[i][j] = Float.valueOf(str.substring(0,n)).floatValue() ;
```

```
        c_sum[j] += C_dt[i][j] ;
        str = str.substring( n+1 ) ;
      }
      else {
        C_dt[i][j] = Float.valueOf(str).floatValue() ;
        c_sum[j] += C_dt[i][j] ;
        break ;
      }
    }
  }
  br.close() ;
  fr.close() ;

  //tf-idfを得る
  tf_idf = new double[data[0]][data[1]] ;
  FileReader frmtx = new FileReader( args[5] ) ;
  BufferedReader brmtx = new BufferedReader( frmtx ) ;
  while( (str = brmtx.readLine()) != null ) {
    while(true) {
      str = str.trim() ;
      n = str.indexOf( ' ' ) ;
      i = Integer.valueOf(str.substring(0,n)).intValue()-1 ;
      str = str.substring( n+1 ) ;
      str = str.trim() ;
      n = str.indexOf( ' ' ) ;
      j = Integer.valueOf(str.substring(0,n)).intValue()-1 ;
      str = str.substring( n+1 ) ;
      str = str.trim() ;
      n = str.indexOf( ' ' ) ;
      if( n != -1 ) {
        tf_idf[i][j] = Double.valueOf(str.substring(0,n)).doubleValue() ;
        str = str.substring( n+1 ) ;
      }
      else {
        tf_idf[i][j] = Double.valueOf(str).doubleValue() ;
        break ;
```

```
      }
    }
  }
}
System.out.println( "データ読み込み完了" ) ;

sim = new double[data[0]][data[0]] ;//類似度
{
  double sigma_c_it[] = new double[data[1]] ;
  double sigma_c_dj[] = new double[data[0]] ;
  double N = 0.0 ;

  //ΣCif, ΣCejを求める
  for( i = 0 ; i < data[0] ; i++ ) {
    for( j = 0 ; j < data[1] ; j++ ) {
      sigma_c_it[j] += C_dt[i][j] ;
      sigma_c_dj[i] += C_dt[i][j] ;
      N += C_dt[i][j] ;
    }
  }

  //相互情報ベクトルを求める
  {
    double MI_dt[][] = new double[data[0]][data[1]] ;//相互情報ベクトル

    for( i = 0 ; i < data[0] ; i++ ) {
      c = sigma_c_dj[i] / N ;
      for( j = 0 ; j < data[1] ; j++ )
      {
        if( C_dt[i][j] == 0 )
          continue ;
        a = C_dt[i][j] / N ;
        b = sigma_c_it[j] / N ;
        MI_dt[i][j] = Math.log( a /(b*c) ) ;
      }
    }
```

```
    //類似度を調べる
    k = data[0] -1 ;
    for( i = 0 ; i < k ; i++ ) {
      b = 0.0 ;
      for( n = 0 ; n < data[1] ; n++ )
        b += MI_dt[i][n]*MI_dt[i][n] ;
      for( j = i+1 ; j < data[0] ; j++ ) {
        a = 0.0 ;
        c = 0.0 ;
        for( n = 0 ; n < data[1] ; n++ ) {
          a += MI_dt[i][n]*MI_dt[j][n] ;
          c += MI_dt[j][n]*MI_dt[j][n] ;
        }
        sim[i][j] = a/Math.sqrt(b * c)  ;
        sim[j][i] = sim[i][j] ;
      }
      sim[i][i] = 1.0 ;
    }
    sim[i][i] = 1.0 ;
  }
}
System.out.println( "類似度計算完了" ) ;

int List_R[] = new int[data[0]] ;
int R_num ;
Cluster List_Lk[] = new Cluster[data[0]/2] ;
int all_Lk_num = 0 ;

boolean end1 = true, end2 = true ;
double avgsim ;//スコア計算用
double sim_min ;//類似値の最小値
double max, bf ;
int List_E[] = new int[data[0]] ;

for( i = 0 ; i < data[0] ; i++ )
```

```
        List_E[i] = i ;

      //非類似度(ユークリッド距離)
      n = data[0]-1 ;
      Euclid = new double[data[0]][data[0]] ;
      for( i = 0 ; i < n ; i++ ) {
        for( j = i+1 ; j < data[0] ; j++ ) {
          a = 0.0 ;
          for( k = 0 ; k < data[1] ; k++ ) {
            b = tf_idf[i][k]-tf_idf[j][k] ;
            a += b*b ;
          }
          Euclid[i][j] = Math.sqrt( a ) ;
          Euclid[j][i] = Euclid[i][j] ;
        }
      }

      all_Lk_num = MC_phase2.MC_p2( List_E, data[0], committees, 0 ) ;
      if( all_Lk_num != committee_num ) {
        System.out.println( "エラーにより、指定数のcommitteeを作成できませんでした。" ) ;
        return ;
      }

      System.out.println( "committee算出完了¥n" ) ;

      //committeeとそれに属する文書の表示
      System.out.println( "データセット名：" + args[0] ) ;
      System.out.println( "文書数：" + data[0] + "¥t単語数：" + data[1] ) ;
      System.out.println( "パラメータ：¥n  θ1 = " + theta1 + "¥tθ2 = " + theta2
+ "¥tk = " + sim_k ) ;
      if( (t1!=theta1) || (t2!=theta2) )
        System.out.println( "元のパラメータ：¥n  θ1 = " + t1 + "¥tθ2 = " + t2 ) ;
      n = 0 ;
      a = 0.0 ;
      for( i = 0 ; i < committee_num ; i++ ) {
```

```
        n += committees[i].n ;
        a += committees[i].score ;
      }
      System.out.println( "平均メンバー数 = " +
exFormat1.format((double)n/committee_num) ) ;
      System.out.println( "committee数:" + all_Lk_num ) ;

      for( i = 0 ; i < committee_num ; i++ ) {
        System.out.print( "¥ncommittee" + (i+1) ) ;
        for( j = 0 ; j < committees[i].n ; j++ )
          System.out.print((committees[i].elements[j]+1) + " " ) ;
      }
      System.out.println( "¥n" ) ;

      FileWriter fw[] = new FileWriter[committee_num] ;
      BufferedWriter bw[] = new BufferedWriter[committee_num] ;

      //committeeの数だけトレーニングデータを作成
      for( i = 0 ; i < committee_num ; i++ ) {
        j = args[0].indexOf( '.' ) ;
        str = args[0].substring( 0, j ) ;
        str = str.concat( "_training" ) ;
        str = str.concat( String.valueOf(i+1) ) ;
        str = str.concat( ".svmdata" ) ;
        fw[i] = new FileWriter( str ) ;
        bw[i] = new BufferedWriter( fw[i] ) ;

        for( j = 0 ; j < committee_num ; j++ ) {
          for( k = 0 ; k < committees[j].n ; k++ ) {
            if( i == j )
              bw[i].write( "+1" ) ;
            else
              bw[i].write( "-1" ) ;
            for( m = 0 ; m < data[1] ; m++ )
              if( tf_idf[committees[j].elements[k]][m] != 0.0 )
                bw[i].write( " " + (m+1) + ":" +
```

```
tf_idf[committees[j].elements[k]][m] ) ;
            bw[i].write( "¥n" ) ;
          }
        }
        bw[i].flush() ;
        bw[i].close() ;
        fw[i].close() ;
      }

      //SVM用にデータセットをフォーマット
      j = args[0].indexOf( '.' ) ;
      str = args[0].substring( 0, j ) ;
      str = str.concat( "_test.svmdata" ) ;
      FileWriter fww = new FileWriter( str ) ;
      BufferedWriter bww = new BufferedWriter( fww ) ;

      for( j = 0 ; j < data[0] ; j++ ) {
        bww.write( "+1" ) ;
        for( k = 0 ; k < data[1] ; k++ )
          if( tf_idf[j][k] != 0.0 )
            bww.write( " " + (k+1) + ":" + tf_idf[j][k] ) ;
        bww.write( "¥n" ) ;
      }
      bww.flush() ;
      bww.close() ;
      fww.close() ;

      System.out.println( "データの作成完了" ) ;
    }
    catch( Exception e ) {
      System.out.println( "Exception: " + e ) ;
    }
  }
}

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

```
class Cluster {
  int elements[] ;
  int n = 0 ;//文書数
  double score, Vector[] ;//スコア、重心ベクトル

  int i, j, k ;//汎用

  Cluster() {
     elements = new int[MC.sim_k+1] ;
  }
  Cluster( int d ) {
     elements = new int[d] ;
  }

  //文書dを要素に加える
  void elementIn( int d ) {
    elements[n] = d ;
    n++ ;
  }
  //重心の計算
  void center() {
    Vector = new double[MC.data[1]] ;
    for( i = 0 ; i < n ; i++ )
      for( j = 0 ; j < MC.data[1] ; j++ )
        Vector[j] += MC.tf_idf[elements[i]][j] ;
    for( i = 0 ; i < MC.data[1] ; i++ )
      Vector[i] /= n ;
  }

  //スコアの計算
  void Cal_score() {
    if( n == 1 ) {
      score = 1.0 ;
      return ;
    }
```

```
    double avgsim = 0.0 ;
    k = n-1 ;
    for( i = 0 ; i < k ; i++ )
      for( j = i+1 ; j < n ; j++ )
        avgsim += MC.sim[elements[i]][elements[j]] ;
    score = 2*avgsim/k ;
  }
}

//*************************************************************************//

class MC_phase2 {
  static int MC_p2( int List_E[], int E_num, Cluster List_Lk[], int Lk_num ) {
    //汎用変数
    int i, j, k, l, m, n ;
    double a, b, c ;
    double t1, t2 ;
    i = j = k = l = m = n = 0 ;
    int sim_num1 = -1, sim_num2 = -1 ;//List_Eの数、特定の文書の位置1,2

    int List_R[] = new int[E_num] ;
    int R_num = 0 ;

    try {
      t1 = MC.theta1 ;
      t2 = MC.theta2 ;
      boolean end1 = true, end2 = true ;
      double avgsim ;//スコア計算用
      double sim_min ;//類似値の最小値
      Cluster List_Lc[] = new Cluster[E_num] ;
      int Lc_num = 0 ;//List_Lcの数

      //類似度の高いk個の文書を見つけ、群平均クラスタリング
      for( i = 0 ; i < E_num ; i++ ) {
        int sim_dcmnt[] = new int[MC.sim_k+1] ;//類似したsim_k個の文書を格納
```

```
int sim_d ;

//類似しているk個の文書を群平均クラスタリング
//類似しているk個の文書を探す
n = MC.sim_k ;//sim_k個の文書をリストEからsim_dcmntに入れる
l = n-1 ;
sim_d = 0 ;
sim_num1 = 0 ;
if( i == 0 )
  sim_min = MC.sim[List_E[0]][List_E[1]] ;
else
  sim_min = MC.sim[List_E[i]][List_E[0]] ;
//sim_dcmntに群平均するk個の文書をList_Eの順に格納
for( j = 0 ; sim_d < n ; j++ ) {
  if( i == j )
    continue ;
  sim_dcmnt[sim_d] = List_E[j] ;
  //最小の類似値とその文書を記憶
  if( MC.sim[List_E[i]][List_E[j]] < sim_min ) {
    sim_min = MC.sim[List_E[i]][List_E[j]] ;
    sim_num1 = sim_d ;//類似度の最も低い文書の場所
  }
  sim_d++ ;
}
//sim_dcmntに類似値の高い文書を格納
while( j < E_num ) {
  if( i == j ) {
    j++ ;
    continue ;
  }
  if( MC.sim[List_E[i]][List_E[j]] > sim_min ); {
    sim_dcmnt[sim_num1] = List_E[j] ;//一番類似度の低いのに上書き
    sim_min = MC.sim[List_E[i]][sim_dcmnt[0]] ;
    sim_num1 = 0 ;
    for( k = 1 ; k < n ; k++ ) {
      if( sim_min > MC.sim[List_E[i]][sim_dcmnt[k]] ) {
```

```
            sim_min = MC.sim[List_E[i]][sim_dcmnt[k]] ;
            sim_numl = k ;
          }
        }
      }
      j++ ;
    }
    sim_dcmnt[n] = List_E[i] ;

    //Average Link
    n++ ;
    l = n-1 ;
    {
      int nn_1 = 2*n-1 ;//群平均で出来るクラスタの数
      Cluster AL[] = new Cluster[nn_1] ;
      double euc_al[][] = new double[nn_1][nn_1] ;//クラスタ間の非類似度
      int ef[] = new int[nn_1] ;//クラスタ間非類似度の有効無効

      for( j = 0 ; j < l ; j++ ) {
        AL[j] = new Cluster() ;
        AL[j].elementIn( sim_dcmnt[j] ) ;
        for( k = j+1 ; k < n ; k++ ) {
          euc_al[j][k] = MC.Euclid[sim_dcmnt[j]][sim_dcmnt[k]] ;
          euc_al[k][j] = euc_al[j][k] ;
        }
      }
      AL[j] = new Cluster() ;
      AL[j].elementIn( sim_dcmnt[j] ) ;
      j++ ;
      for( ; j < nn_1 ; j++ )
        AL[j] = new Cluster() ;

      while( true ) {
        //距離が最小のものを見つける
        sim_min = -1.0 ;
        for( j = 0 ; j < l ; j++ ) {
```

```
        if( ef[j] == -1 )
          continue ;
        for( k = j+1 ; k < n ; k++ ) {
          if( ef[k] == -1 )
            continue ;
          if( (euc_al[j][k]<sim_min)||(sim_min<0.0) ) {
            sim_num1 = j ;
            sim_num2 = k ;
            sim_min = euc_al[j][k] ;
          }
        }
      }
      ef[sim_num1] = -1 ;
      ef[sim_num2] = -1 ;
      l++ ;
      n++ ;

      //クラスタの合併
      for( j = 0 ; j < AL[sim_num1].n ; j++ )
        AL[l].elementIn( AL[sim_num1].elements[j] ) ;
      for( j = 0 ; j < AL[sim_num2].n ; j++ )
        AL[l].elementIn( AL[sim_num2].elements[j] ) ;

      //終了判定
      if( n == nn_1 )
        break ;

      //非類似度更新
      for( j = 0 ; j < n ; j++ ) {
        if( ef[j] == -1 )
          continue ;
        euc_al[l][j] =

(AL[sim_num1].n*euc_al[sim_num1][j]+AL[sim_num2].n*euc_al[sim_num2][j])
          /(AL[sim_num1].n+AL[sim_num2].n) ;
        euc_al[j][l] = euc_al[l][j] ;
```

```
      }
    }
    endl = true ;

    a = 0.0 ;
    //スコア計算
    for( j = 0 ; j < nn_1 ; j++ ) {
      AL[j].Cal_score() ;
      if( AL[j].score > a ) {
        a = AL[j].score ;
        sim_num1 = j ;
      }
    }

    //スコアの最も高いクラスタをリストに追加
    List_Lc[Lc_num] = new Cluster() ;
    List_Lc[Lc_num] = AL[sim_num1] ;
    Lc_num++ ;
  }//Average Link終了
}

{
  Cluster clstr = new Cluster() ;

  m = Lc_num -1 ;
  for( j = 0 ; j < m ; j++ ) {
    for( k = m ; k > j ; k-- ) {
      if( List_Lc[k].score > List_Lc[k-1].score ) {
        clstr = List_Lc[k] ;
        List_Lc[k] = List_Lc[k-1] ;
        List_Lc[k-1] = clstr ;
      }
    }
  }
}
```

```
for( i = 0 ; i < Lc_num ; i++ )
  List_Lc[i].center() ;//List_Lc[i]の重心を求める

end1 = true ;
end2 = true ;
while( end2 ) {

  m = Lk_num-1 ;
  for( i = 0 ; i < Lc_num ; i++ ) {
    if( Lk_num == 0 ) {
      List_Lk[0] = new Cluster() ;
      List_Lk[0] = List_Lc[i] ;
      Lk_num++ ;
      m = 0 ;
      end1 = false ;
      continue ;
    }
    b = 0.0 ;
    for( k = 0 ; k < MC.data[1] ; k++ )
      b += List_Lc[i].Vector[k]*List_Lc[i].Vector[k] ;
    for( j = 0 ; j < Lk_num ; j++ ) {
      a = 0.0 ;
      c = 0.0 ;
      for( k = 0 ; k < MC.data[1] ; k++ ) {
        a += List_Lc[i].Vector[k]*List_Lk[j].Vector[k] ;
        c += List_Lk[j].Vector[k]*List_Lk[j].Vector[k] ;
      }
      //θ1と類似度の比較
      if( MC.theta1 <= a/Math.sqrt(b * c) )//類似度がθ1以上
        break ;
      else//θ1を下回る
        //全ての比較対象のcommittee(List_Lk)との類似度がθ1未満
        if( j == m ) {
          List_Lk[Lk_num] = new Cluster() ;
          List_Lk[Lk_num] = List_Lc[i] ;
          Lk_num++ ;
```

```
            m++ ;
            end1 = false ;
            if( MC.committee_num <= Lk_num )
              i += Lc_num ;
            break ;
          }
      }
    }
    if( end1 )
      MC.theta1 += MC.delta ;
    else
      end2 = false ;
  }

  if( MC.committee_num <= Lk_num )
    return Lk_num ;

  m = Lk_num-1 ;
  end2 = true ;
  while( end2 ) {
    for( i = 0 ; i < E_num ; i++ ) {
      b = 0.0 ;
      for( k = 0 ; k < MC.data[1] ; k++ )
        b += MC.tf_idf[List_E[i]][k]*MC.tf_idf[List_E[i]][k] ;
      for( j = 0 ; j < Lk_num ; j++ ) {
        a = 0.0 ;
        c = 0.0 ;
        for( k = 0 ; k < MC.data[1] ; k++ ) {
          a += MC.tf_idf[List_E[i]][k]*List_Lk[j].Vector[k] ;
          c += List_Lk[j].Vector[k]*List_Lk[j].Vector[k] ;
        }
        if( MC.theta2 <= a/Math.sqrt(b*c) )
          break ;
        else
          if( j == m ) {
            List_R[R_num] = List_E[i] ;
```

```
                R_num++ ;
              }
            }
          }
          if( R_num < MC.sim_k *(MC.committee_num-Lk_num) ) {
            MC.theta2 += MC.delta ;
            R_num = 0;
          }
          else
            end2 = false ;
        }
      }
      catch( Exception e ) {
        System.out.println( "Exception: " + e ) ;
        return -1 ;
      }

      if( E_num == MC.data[0] )
        MC.hyouka = Lk_num ;
      return MC_phase2.MC_p2( List_R, R_num, List_Lk, Lk_num ) ;
    }
  }
```

付録 B

(Distributer.java)

```
import java.io.* ;

class Distributer {
  public static void main( String args[] ) {
    int i, j, k, n ;//汎用変数

    try {
      int data[] = new int[2] ;//文書数、committeeの数
      int committee[][] ;
      int com_num[] ;
      int max_i = -1 ;
      double score[][] ;
      double max ;
      String str ;

      //データ読み込み
      FileReader fr = new FileReader( args[0] ) ;
      BufferedReader br = new BufferedReader( fr ) ;

      str = br.readLine() ;
      str = str.trim() ;
      n = str.indexOf( ' ' ) ;
      data[0] = Integer.valueOf(str.substring(0,n)).intValue() ;
      str = str.substring( n+1 ) ;
      str = str.trim() ;
      n = str.indexOf( ' ' ) ;
      data[1] = Integer.valueOf(str).intValue() ;

      committee = new int[data[1]][data[0]] ;//committee番号、文書番号
      com_num = new int[data[1]] ;//committee番号
      score = new double[data[0]][data[1]] ;//文書番号、committee番号

      for( i = 0 ; i < data[1] ; i++ ) {
        for( j = 0 ; j < data[0] ; j++ ) {
          str = br.readLine() ;
          k = str.length() ;
```

```
    if( 0 < k && k < 13 )
      score[j][i] = Double.valueOf(str).doubleValue() ;
    else
      j— ;
  }
}
fr.close() ;

//振り分け
for( i = 0 ; i < data[0] ; i++ ) {
  max = -1024.0 ;
  for( j = 0 ; j < data[1] ; j++ ) {
    if( score[i][j] > max ) {
      max = score[i][j] ;
      max_i = j ;
    }
  }
  committee[max_i][com_num[max_i]] = i+1 ;//1から始まる文書番号
  com_num[max_i]++ ;
}

//結果
i = args[0].indexOf( '.' ) ;
str = args[0].substring( 0, i ) ;
str = str.concat( "_result.txt" ) ;
FileWriter fw = new FileWriter( str ) ;
BufferedWriter bw = new BufferedWriter( fw ) ;

bw.write( data[0] + " " + data[1] + "¥n" ) ;
for( i = 0 ; i < data[1] ; i++ ) {
  System.out.println( "cluster " + (i+1) + "¥tメンバー数:" + com_num[i] ) ;
  for( j = 0 ; j < com_num[i] ; j++ ) {
    bw.write( committee[i][j] + " " ) ;
    System.out.print( " " + committee[i][j] ) ;
  }
  bw.write( "¥n" ) ;
```

```
            System.out.println( "" ) ;
          }
          bw.close() ;

        }
        catch( Exception e ) {
          System.out.println( "Exception: " + e ) ;
          return ;
        }
      }
    }
```