平成 20 年度茨城大学工学部情報工学科卒業研究論文

文書クラスタリングを対象とした Weighted Kernel K-means の 初期値設定法

平成 21 年 2 月 10 日

茨城大学工学部情報工学科

著者：茂木 哲矢（05T4078Y）

指導教員：新納 浩幸 准教授

平成 20 年度茨城大学工学部情報工学科卒業研究

文書クラスタリングを対象とした Weighted Kernel K-means の初期値設定法

著者： 茂木 哲矢（05T4078Y）
指導教員： 新納 浩幸 准教授

論文要旨

文書クラスタリングは，文書の集合に対して，知的な処理を行う基本的な処理であり，その重要性は明らかである．例えばテキストマイニングの分野では，クラスタリングは基本的な構成要素である，情報検索の分野では，検索された文書の集合を俯瞰的に見るためにクラスタリングするシステムが盛んに研究されている．

クラスタリングの標準的手法として K-means が存在する．K-means はクラスタの中心と各データの距離を基準としてクラスタリングを行う手法である．K-means は山登り法で解を求めるため，局所最適解しか求めることができない．これは初期値によって求まる解が異なることを意味する．安定した解を得るためには，ランダムな初期値を用いて，複数回 K-means を実行する必要がある．また，K-means は非線形な分離境界をもつデータに対して最適な分類を行えないという問題もある．後者の問題の対策として，データ空間を非線形関数によって高次元空間に写像し，表現力を高めてから線形分離する Kenrel K-means が提案されている．

Kernel K-means 以外にも，非線形の分離境界をもつデータに対するクラスタリング手法にスペクトラルクラスタリングがある．スペクトラルクラスタリングはデータから得られる隣接行列に関する固有値を求めることでクラスタリング結果を得る手法であり，精度の高いクラスタリング手法として知られている．

一方，Kernel K-means を一般化した Weighted Kernel K-means（以下 WKK と省略する）の評価関数はスペクトラルクラスタリングの評価関数と等価であることが知られている．スペクトラルクラスタリングは固有値を求める処理の負荷が大きいが，WKK は固有値を求めずに，繰り返しの処理によって，その評価関数の最適解を見つけようとする．そのため WKK により効率的かつ高精度のクラスタリングが期待できる．ただし WKK は K-means と同様，初期値依存の問題があり，安定した解を得るにはランダムな初期値を用いて，複数回 WKK を実行しなければならない．本論文では K-means の初期値依存の問題に効果的な KKZ を WKK に適用することを試みる．

実験では 5 個のデータセットに対して，KKZ で初期値を選択してから，WKK の手法でクラスタリングを行いエントロピーと純度により評価を行った．その結果，初期値を KKZ で設定した方がランダムに設定した場合よりも 4 個のデータセットで良い結果を得ることができた．また，3 個のデータセットでは初期値を KKZ で設定した方がランダムに設定した場合より良い結果を得ることができた．

目次

図目次

表目次

第1章
はじめに

1.1 研究の背景

近年の計算機の高性能化やネットワークの整備によるインターネットの普及，大容量外部記憶装置の低廉化に伴い，データ化された情報は蓄積され続けている．

文書データは発信・蓄積が容易になった一方で，その量の増大に伴って，整理・管理していくことが煩雑になる．蓄積した文書データから必要な文書データを取り出す手段として，文書クラスタリングの研究が行われてきた．

クラスタリング手法には，階層的クラスタリングと非階層的クラスタリングの2種類がある．

1. 階層的クラスタリング
   各データを1つのクラスタとし，類似度に従ってクラスタを併合する．最終的に1つのクラスタになるまで再帰的に併合を繰り返す手法である．図1.1のような樹木図で表すことができる．

2. 非階層的クラスタリング
   クラスタ数をあらかじめ指定し，分類の良さを表す評価関数の最適解を求める．

階層的クラスタリングでは，文書 N 件すべての組み合わせの類似度計算が必要となる．その計算量が $O(N^2)$ となるため，一般的に文書クラスタリングに用いることは難しい．そこで本論文では非階層的クラスタリング手法を中心的に扱う．

非階層的クラスタリングの代表的手法に K-means がある．K-means はクラスタの中心と各データの距離を基準にして分類を行う手法である．

K-means は山登り法で解を求めるため，局所最適解しか求めることができないことが知られている．これは初期値によって求まる解が異なることを意味する．安定した解を得るためには，ランダムな初期値を用いて，複数回 K-means を実行する必要がある．また，K-means は分離境界が非線形なデータに対して最適な分類を行えないという問題もある．分離境界が非線形となるデータに対しては，カーネル法を用いて表現力を高めてから，K-means で線形分離する Kernel K-means が提案されている．

Kernel K-means 以外の非線形な分離境界をもつデータに対するクラスタリグ手法にはスペクトラルクラスタリングがある．クラスタリングをグラフの分割する問題に置き換えて考え，評価関数によりカットするエッジを決定する．評価関数の最適解を求めることとデータから得られる隣接行列の固有値問題の解を求めることが対応する関係を利用してクラスタリングを行う手法がスペクトラルクラスタリングであり，精度の高いクラスタリング手法として知られている．

このように非線形な分離境界をもつデータに対するクラスタリング手法は大きく異なるが，Kernel K-means を一般化した Weighted Kernel K-means（以下 WKK ）の評価関数とスペクトラルクラスタリングの Normalized cut の評価関数は等価であることが示されている [1]．スペクトラルクラスタリングは固有値を求める処理の負荷が大きいが，WKK は固有値を求めずに，繰り返しの処理によって，その最適解を見つける．そのため WKK より効率的かつ高精度のクラスタリングが期待できる．

しかし WKK も山登り法で解を求める手法であるため，K-means と同様に局所最適解しか得られない問題は解決していない．

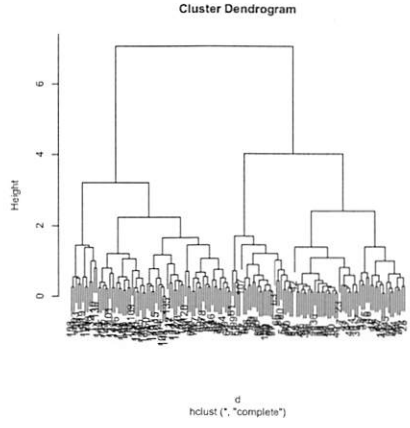


図 1.1: 樹木図（データセット：iris，手法：最長距離法）

1.2 研究の目的

1.1 で述べたように，非階層的クラスタリングの代表的手法である K-means は初期値に依存するという問題を抱えている．カーネル法と組み合わせ，一般化した WKK でもこの問題は解決していない．

本論文では，WKK のクラスタリグの性能が従来より良くなるような初期値設定法を提案する．これによりランダムな初期値を用いて，複数回クラスタリングすることを避けることができ，クラスタリング全体の処理が効率化することが期待できる．

第2章
文書クラスタリング

2.1 クラスタリング

データの集合をデータ間の類似度（あるいは非類似度）に従って，いくつかのグループに分けることをクラスタリングという．

図2.1の9個の図形データをいくつかのグループに分けることを考える．

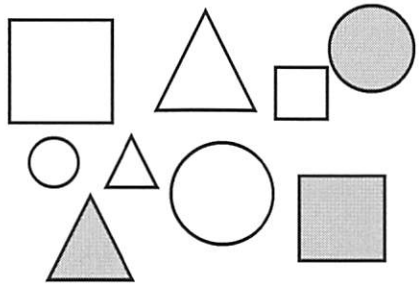


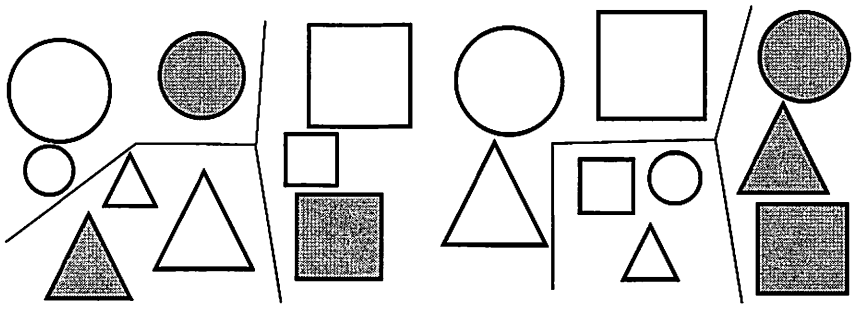
図 2.1: クラスタリング前

与えられた図形は

- 円
- 三角形
- 四角形

なので，図形の形状に注目して図2.2(a)のように3つのグループに分けられる．また，図形の大きさに注目すると図2.2(b)のようにわけることもできる．

このように，どのような観点で類似度を設定するかによってクラスタリングの結果は異なり，クラスタリングには厳密な正解がない．そのためク



(a) 形状でクラスタリング　(b) 大きさでクラスタリング

図 2.2: 異なる観点によるクラスタリング

ラスタリングはデータから何らかの結論を導くための道具ではなく，人間がデータを分析するための道具といえる．

2.2　ベクトル空間モデル

一般にデータ解析を行うためには，データを n 次元のベクトルで表現する．文書クラスタリングでは文書がデータに対応し，文書を n 次元のベクトルで表現するには，一般にベクトル空間モデルを使う．

ベクトル空間モデルでは，文書は文書集合中に現れた単語とその重みで構成されるベクトル

$$\boldsymbol{d}_i = (t_{i1}, t_{i2}, \cdots, t_{iM}) \tag{2.1}$$

として表現される．ここで t_{ij} は i 番目の文書 d_i の j 番目に現れる単語 w_j の重みである（$j = 1, 2, \cdots, N$）．

通常，文書は複数存在するので式（2.2）のような表現になる．

$$\boldsymbol{d} = \begin{pmatrix} \boldsymbol{d}_1 \\ \boldsymbol{d}_2 \\ \vdots \\ \boldsymbol{d}_M \end{pmatrix} = \begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1M} \\ t_{21} & t_{22} & \cdots & t_{2M} \\ \vdots & \vdots & \vdots & \vdots \\ t_{N1} & t_{N2} & \cdots & t_{NM} \end{pmatrix} \tag{2.2}$$

ベクトル空間モデルでは，文書中のすべての単語が同じ重みで評価される．単語には文書の分野を推定できる単語やどんな文書にも出現するような一般的な単語が存在する．

文書の分野を推定できる単語の重みを重く，一般的な単語の重みを軽くすることによって文書をよく表したベクトルが生成されたり，ベクトル間の位置関係が実際のデータ間の位置関係により近くなると考えられる．

このように重みを付ける処理は，ベクトル空間モデルでは通常行われ，重みの付け方としては TF*IDF という方法が標準的である．

2.3 TF*IDF

TF term frequency
　文書 d_i 中の単語 w_j の出現頻度

IDF inverse document frequency
　全文書数 N を w_j を含む文書数 n_j で割った値に対数をとったもの

と定義する．

TF を f_{ij}，IDF を $\log(N/n_j)$ と書くと，TF*IDF とは文書 d_i 中の単語 w_j の重みを

$$f_{ij} * \log(N/n_j) \tag{2.3}$$

にすること．

式 2.3 の定義では，単語 w_j がすべての d_i に出現するとき $n_j = N$ になるため，IDF が $\log(N/N) = 0$ となる場合がある．IDF が 0 になると不都合が生じる手法があるため，IDF を $\log((N+1)/n_j))$ と補正し，

$$f_{ij} * \log((N+1)/n_j) \tag{2.4}$$

とすることもある．

2.4 正規化

TF*IDF で重み付けするとサイズの大きな文書は，大きなベクトルになるため，ベクトルの長さを 1 に正規化する処理もよく行われる．

2.5 類似度

類似度とは 2 つの文書の表現がどのくらい類似しているかを表し，値が大きいほど 2 つの文書が類似していると定義する．

類似度の例を式（2.5），(2.6）に示す．

- ユークリッド距離

$$||\boldsymbol{x} - \boldsymbol{y}||^2 \tag{2.5}$$

- 余弦尺度

$$\cos\theta = \frac{\boldsymbol{x}\cdot\boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||} \tag{2.6}$$

ここで $\boldsymbol{x}, \boldsymbol{y}$ は文書ベクトル，θ はベクトル $\boldsymbol{x}, \boldsymbol{y}$ がなす角度.

2.6　基本的なクラスタリング手法

クラスタリング手法は大きく階層的クラスタリングと非階層的クラスタリングに分けられる．これらの基本的手法について述べる.

2.7　階層的クラスタリング

階層的クラスタリングは，分枝型と凝集型に分けられ，ここでは凝集型について述べる.

2.7.1　凝集型クラスタリングのアルゴリズム

凝集型クラスタリングは各データを 1 つのクラスタと考え，再帰的にクラスタを併合して階層構造を獲得するクラスタリング手法である.

凝集型クラスタリングのアルゴリズムを図 2.3 に示す．また，クラスタを併合する様子を図 2.4 に示す.

[step 1] 各データ 1 個のみからなる N 個のクラスタを作成する.

[step 2] クラスタ間の距離関数に基づき，最も近いクラスタ対を見つける.

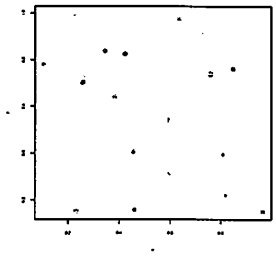
[step 3] そのクラスタ対を併合して、新しいクラスタを作成する.

[step 4] (a) クラスタ数が 1 個になれば終了.

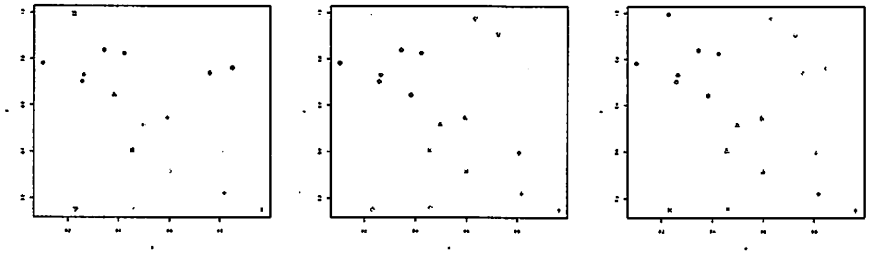
(b) クラスタ数が 1 個でなければ step 2 から繰り返すことにより階層構造を獲得する.

図 2.3: 凝集型クラスタリングのアルゴリズム

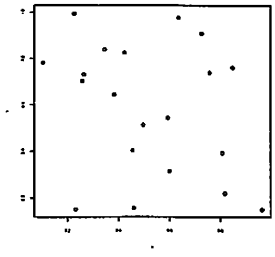
クラスタ C_1, C_2 の距離関数 $D(C_1, C_2)$ の違いにより次のような手法がある．凝集型クラスタリングの手法の中には文書数 N に対して，$O(N^2)$



(a) step 1



(b) step 2, step 3



(c) step 4

図 2.4: クラスタ併合の様子

の計算量を必要とするアルゴリズムが存在し，一般に文書クラスタリングには向いていない．

最短距離法

クラスタ間の最小の距離を与えるデータ対を選び，そのデータ間の距離をクラスタ間の距離とする手法である．

$$D(C_1, C_2) = \min_{x_1 \in C_1, x_2 \in C_2} d(x_1, x_2) \tag{2.7}$$

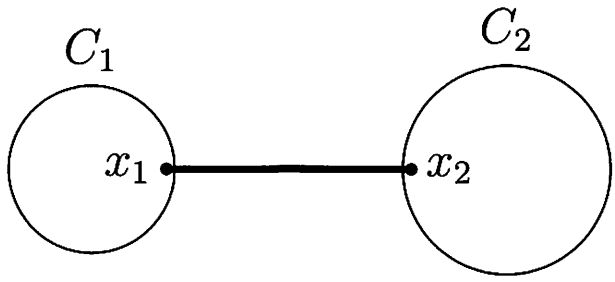


図 2.5: 最短距離法のクラスタ間の距離

最長距離法

クラスタ間の最大の距離を与えるデータ対を選び，そのデータ間の距離をクラスタ間の距離とする手法である．

$$D(C_1, C_2) = \min_{x_1 \in C_1, x_2 \in C_2} d(x_1, x_2) \tag{2.8}$$

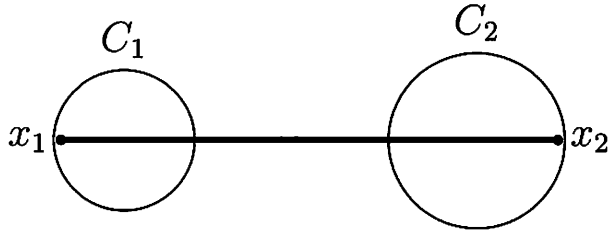


図 2.6: 最長距離法のクラスタ間の距離

群平均法

クラスタ間のすべての要素間の距離を求め，その平均をクラスタ間の距離とする手法である．

$$D(C_1, C_2) = \frac{1}{n_1 n_2} \sum_{x_1 \in C_1} \sum_{x_2 \in C_2} D(x_1, x_2) \tag{2.9}$$

ウォード法

クラスタを併合した場合にクラスタ内の平方和の増加分をクラスタ間の距離とする手法である．

$$\begin{aligned} D(C_1, C_2) &= E(C_1 \cup C_2) - E(C_1) - E(C_2) & (2.10) \\ E(C_i) &= \sum_{x \in C_i} (D(x, c_i))^2 & (2.11) \end{aligned}$$

重心法

クラスタの重心の間の距離の 2 乗をクラスタ間の距離とする手法である．

$$D(C_1, C_2) = \left((\sum_{x_i \in C_1} x_i) - (\sum_{x_j \in C_2} x_j) \right)^2 \tag{2.12}$$

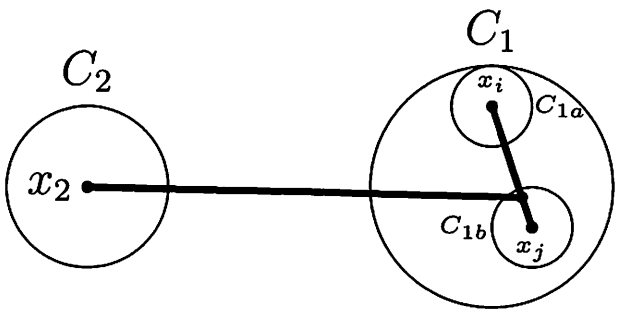


図 2.7: 重心法のクラスタ間の距離

メディアン法

クラスタ C_{1a} と C_{1b} を併合して C_1 としたとき，C_{1a} の重心と C_{1b} の重心の中点を求め，この中点とクラスタ C_2 の重心の間の距離の 2 乗をクラスタ間の距離とする手法である．

$$D(C_1, C_2) = \left(\frac{1}{2} \left(\left(\sum_{x_i \in C_{1a}} x_i \right) + \left(\sum_{x_j \in C_{1b}} x_j \right) \right) - \sum_{x_k \in C_2} x_k \right)^2 \tag{2.13}$$

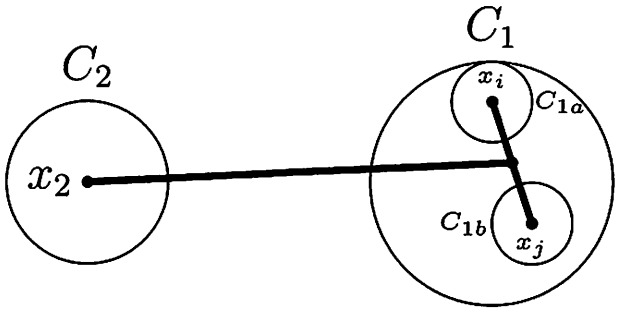


図 2.8: メディアン法のクラスタ間の距離

2.8　非階層的クラスタリング

階層的クラスタリングは文書数 N に対して $O(N^2)$ の計算量を必要とするため，文書クラスタリングには向いていないことが知られている．

非階層的クラスタリングは，クラスタリングの性能に関する評価関数を定め，その評価関数を最小にする分類を探索する手法である．ここでは，非階層クラスタリングの代表的手法である K-means 及び精度の高いクラスタリングであるスペクトラルクラスタリングについて述べる．

2.8.1　K-means

クラスタリングの重心をそのクラスタの代表点 $c_1, c_2, \cdots, c_K$ に設定して，式 (2.14) の評価関数を最小化するようなデータのクラスタへの割り当てを求める．

$$\sum_{i=1}^{K} \sum_{x \in C_i} ||x - c_i||^2 \tag{2.14}$$

K-means のアルゴリズム

K-means を使って文書ベクトル $\boldsymbol{x}$ を K 個のクラスタに分類する場合，図 2.9 の反復アルゴリズムを用いる．

[step 1] クラスタの個数 K および $C_1, C_2, \cdots, C_K$ の代表点 $c_1, c_2, \cdots, c_K$ を設定する．

[step 2] データ $\boldsymbol{x}_i$ とクラスタ C_j の代表点 $\boldsymbol{c}_j i$ との距離 $||\boldsymbol{x}_i - \boldsymbol{c}_j||$ を計算する．

$$d_{ij} = ||\boldsymbol{x}_i - \boldsymbol{c}_j||^2 \tag{2.15}$$

[step 3] 以下の式で $\boldsymbol{x}_i$ のクラスタを設定する．

$$\alpha = \underset{j}{\operatorname{argmin}}\, d_{ij} \tag{2.16}$$

$$\boldsymbol{x}_i \in C_\alpha \tag{2.17}$$

[step 4] (a) step 2 により各データの $\boldsymbol{x}_i$ のクラスタが変化しなければ終了．

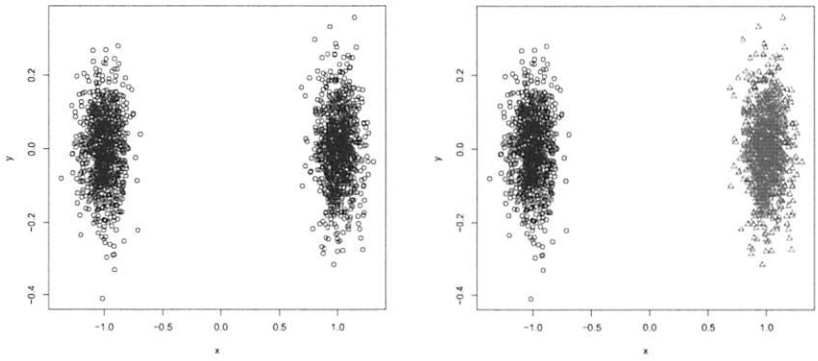
(b) 変化があったときには，式 (2.18) で各クラスタの重心を代表点 $\boldsymbol{c}_j$ に設定し，step 2 に戻る．

$$\boldsymbol{c}_j = \frac{1}{|C_j|} \sum_{\boldsymbol{x}_i \in C_i} x \tag{2.18}$$

図 2.9: K-means のアルゴリズム

アルゴリズムによって式 (2.14) は単調に減少するが，それが式 (2.14) の本当の最小値（大域解）であるとは限らなく，図 2.9 のアルゴリズムによって求まる解は局所解である．

K-means によるクラスタリング例を図 2.10 に示す．



(a) K-means 実行前　　(b) K-means 実行後

図 2.10: K-means によるクラスタリング例

2.8.2　スペクトラルクラスタリング

クラスタリングをグラフ分割の問題として解く手法がある．グラフ分割の問題では，文書データをグラフのノードとして表現し，ノード間のエッジの重みに両端の文書データの類似度を与える．類似度が 0 の場合はエッジを張らないこととすると，文書データの集合をグラフとして表現した場合，クラスタリングとはエッジをカットして，複数のサブグラフに分割することに対応する．このとき，サブグラフ内のエッジは密になり，サブグラフ同士は疎になるように分割する．

適切なカットを見つけるために評価関数を設定する．この評価関数の最適化がある固有値問題の解に対応することを利用して，クラスタリングを行う手法がスペクトラルクラスタリングである．

式（2.19），(2.20）で定義される評価関数がある．

- Min-max cut [2]

$$Mcut = \frac{cut(A,B)}{W(A)} + \frac{cut(A,B)}{W(B)} \tag{2.19}$$

- Normalized cut [3]

$$Ncut = \frac{cut(A,B)}{d_A} + \frac{cut(A,B)}{d_B} \tag{2.20}$$

ここで $cut(A,B)$ はサブグラフ A，B の類似度であり，

$$cut(A,B) = W(A,B) \tag{2.21}$$

と定義する．$W(A,B)$ はサブグラフ A，B の間にあるエッジの重みの総和である．また，

$$W(A) = W(A,A) \tag{2.22}$$

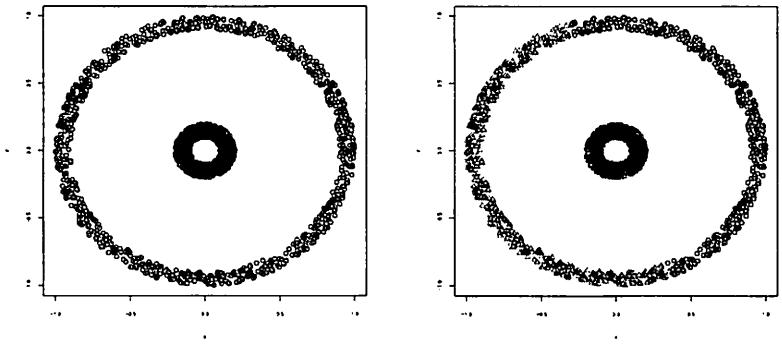
と定義する．d_i は i 番目のデータとその他のデータとの類似度の和と定義する．

第3章

Weighted Kernel K-means

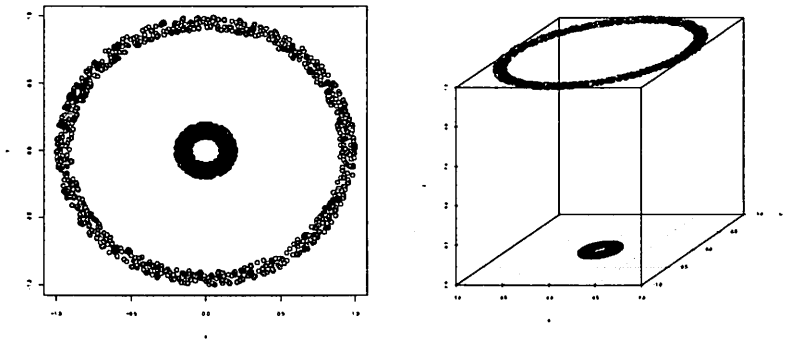
3.1 K-means の問題

K-means では，評価関数にデータ点とクラスタの重心との距離を用いるため，分離境界が各クラスタの中心を結ぶ垂直 2 等分線を形成する．そのため，非線形な分離境界をもつデータに対しては最適な分類を行うことが出来ない（図 3.1）．分離境界が非線形となるようなデータに対するクラスタリング法として，Kernel K-means が提案されている．Kernel K-means は，非線形関数を用いてデータ点を高次元空間へ写像して表現力を高めてから，K-means を行い線型分離する方法である（図 3.2）．



(a) K-means 実行前　　(b) K-means 実行後

図 3.1: K-means によるクラスタリングがうまくいかない例（中心の円に 1000 個，輪に 1000 個のデータ）



(a) 変換前　　(b) 変換後

図 3.2: Kernel K-means のイメージ

3.2　カーネル法

もとのデータ空間を Ω，変換後の高次元特徴空間を H，非線形変換を Φ とすると，もとのデータ空間から高次元特徴空間への変換は

$$\Phi : \Omega \rightarrow H \tag{3.1}$$

のように表すことができる．

この高次元特徴空間 H 上において，

$$K(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}) \tag{3.2}$$

で定義されるカーネル関数を用いることで，非線形変換したベクトル $\phi(\boldsymbol{x})$ を計算することなく，もとのデータ空間における計算だけで，高次元特徴空間 H 上における内積を定義することができる．カーネル関数を用いて高次元特徴空間における複雑な計算をせずに，低次元空間の内積のみでクラスタリング技法を高次元空間に拡張するアプローチをカーネルトリックという．

主なカーネル関数として式 (3.3) ～ (3.5) がある．

- 多項式カーネル

$$K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x} \cdot \boldsymbol{y} + c)^d \tag{3.3}$$

- ガウスカーネル

$$K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(\frac{-||\boldsymbol{x} - \boldsymbol{y}||^2}{2\sigma^2}\right) \tag{3.4}$$

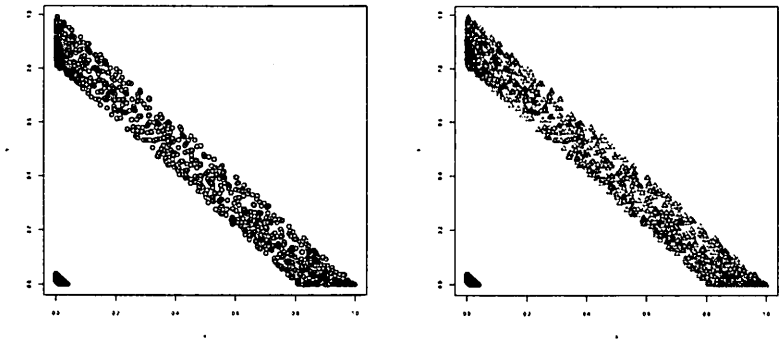
- シグモイドカーネル

$$K(\boldsymbol{x}, \boldsymbol{y}) = \tanh(c(\boldsymbol{x} \cdot \boldsymbol{y}) + \theta) \tag{3.5}$$

図 3.1 の座標を

$$(x', y') = (x^2, y^2) \tag{3.6}$$

と変換したデータに対する K-means のクラスタリング結果を図 3.3 に示す.



(a) K-means 実行前　　(b) K-means 実行後

図 3.3: 図 3.1 の平方に対する K-means のクラスタリング結果

3.3　カーネル K-means の評価関数

3.2 で述べたカーネル法を K-means に組み合わせた手法がカーネル K-means である.

データ $\boldsymbol{x}$ を高次元特徴空間に移すことを $\phi(\boldsymbol{x})$ と書くとると, 式 (2.14) で与えられる K-means の評価関数は

$$\begin{aligned} &\sum_{j=1}^{k} \sum_{a \in \pi_j} ||\phi(\boldsymbol{a}) - \boldsymbol{m}_j||^2 \\ &\boldsymbol{m}_j = \frac{\sum_{b \in \phi_j} \phi(\boldsymbol{b})}{|\phi_j|} \end{aligned} \tag{3.7}$$

と書くことができる.

$$||\phi(\boldsymbol{a}) - \boldsymbol{m}_j||^2$$

を展開すると，

$$\begin{aligned} ||\phi(a) - m_j||^2 = \phi(a) \cdot \phi(a) \\ - \frac{2\sum_{b \in \pi_j} \phi(a) \cdot \phi(b)}{|\pi_j|} \\ + \frac{\sum_{b,c \in \pi_j} \phi(b) \cdot \phi(c)}{|\pi_j|^2} \end{aligned} \tag{3.8}$$

を得る．

式（3.8）の内積を

$$K(x, y)$$

で置き換えると

$$K(a,a) - \frac{2\sum_{b \in \pi_j} K(a,b)}{|\pi_j|} + \frac{\sum_{b,c \in \pi_j} K(b,c)}{|\pi_j|^2} \tag{3.9}$$

を得る．

3.4　Weighted Kernel K-means の評価関数

本論文で扱うクラスタリング手法の WKK は K-means にカーネル法を組み合わせた カーネル K-means において，各データの重みを考えることによってさらに一般化した手法である．

WKK の評価関数はカーネル K-means の評価関数と同様の手順で求めることができる．データ x を高次元特徴空間に移すことを $\phi(x)$ と書く．また，各データには重み $w(a)$ があるとすると，式（2.14）で与えられる K-means の評価関数は

$$\begin{aligned} \sum_{j=1}^{k} \sum_{a \in \pi_j} w(a)||\phi(a) - m_j||^2 \\ m_j = \frac{\sum_{b \in \pi_j} w(b)\phi(b)}{\sum_{b \in \pi_j} w(b)} \end{aligned} \tag{3.10}$$

と書き直すことができる．

$$||\phi(a) - m_j||^2$$

を展開すると式（3.11）を得る．

$$
\begin{aligned}
||\phi(\boldsymbol{a}) - \boldsymbol{m}_j||^2 = \phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{a}) \\
- \frac{2\sum_{\boldsymbol{b}\in\pi_j} w(\boldsymbol{b})\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})}{\sum_{\boldsymbol{b}\in\pi_j} w(\boldsymbol{b})} \\
+ \frac{\sum_{\boldsymbol{b},\boldsymbol{c}\in\pi_j} w(\boldsymbol{b})w(\boldsymbol{c})\phi(\boldsymbol{b}) \cdot \phi(\boldsymbol{c})}{(\sum_{\boldsymbol{b}\in\pi^2} w(\boldsymbol{b}))^2}
\end{aligned}
\tag{3.11}
$$

式（3.11）の内積を

$$K(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$$

で置き換えると

$$K(\boldsymbol{a}, \boldsymbol{a}) - \frac{2\sum_{\boldsymbol{b}\in\pi_j} w(\boldsymbol{b})K(\boldsymbol{a}, \boldsymbol{b})}{\sum_{\boldsymbol{b}\in\pi_j} w(\boldsymbol{b})} + \frac{\sum_{\boldsymbol{b},\boldsymbol{c}\in\pi_j} w(\boldsymbol{b})w(\boldsymbol{c})K(\boldsymbol{b}, \boldsymbol{c})}{(\sum_{\boldsymbol{b}\in\pi^2} w(\boldsymbol{b}))^2} \tag{3.12}$$

を得る．

WKK の評価関数である式（3.12）は Normalized cut の評価関数である式（2.20）と等価であることが知られている．スペクトラルクラスタリングは評価関数の最適解を求めることとある固有値問題の解に対応することを利用して，クラスタリングを行うが，固有値を求める処理の負荷は大きい．WKK は固有値を求めずに，繰り返し処理によって，その評価関数の最適解を見つける．そのため WKK により効率的かつ高精度のクラスタリングが期待できる．

3.5 Weighted Kernel K-means のアルゴリズム

WKK を使って文書ベクトル $\boldsymbol{x}$ を K 個のクラスタにクラスタリングする場合，図 3.4 の反復アルゴリズムを用いる．

このアルゴリズムは図 2.9 と評価関数は違うものの処理の流れは共通である．

[step 1] クラスタの個数 K およびクラスタ $C_1, C_2, \cdots, C_K$ を設定する.

[step 2] データ $\boldsymbol{a}$ とクラスタ C_i との距離 $||\boldsymbol{a} - \boldsymbol{m}_i||$ を計算し，以下の式で $\boldsymbol{a}$ のクラスタを設定する.

$$j^*(\boldsymbol{a}) = \underset{j}{\operatorname{argmin}} ||\phi(\boldsymbol{a}) - \boldsymbol{m}_j||^2 \tag{3.13}$$

$$C_j = \{\boldsymbol{a} : j^*(\boldsymbol{a}) = j\} \tag{3.14}$$

[step 3] (a) step 2 により各データの $\boldsymbol{a}$ のクラスタが変化しなければ終了.

(b) 変化があったときには step 2 に戻る.

図 3.4: WKK のアルゴリズム

第4章
KKZ

4.1 KKZ による Weighted Kernel K-means の初期値設定

非線形な分離境界となるデータに対して最適なクラスタリングができないほかにも，K-means には問題がある．それは K-means は山登り法で解を求めるため，局所最適解しか求めることができないことである．これは初期値によって求まる解が異なることを意味する．安定した解を得るためには，ランダムな初期値を用いて，複数回 K-means を実行する必要がある．K-means を応用し，一般化した WKK でも山登り法で解を求めるため，この問題は存在する．

この問題に対して，本論文で提案する手法は K-means の初期値依存の問題に効果的な初期値設定法である KKZ[4] を WKK に使用する方法である．これによりランダムな初期値を用いて，複数回クラスタリングを行うことを避けることができ，クラスタリングの全体の処理が効率化されることが期待される．

4.2 KKZ のアルゴリズム

KKZ はクラスタとデータ間の距離を基準にして初期値を決定する手法である．新しく決定されるクラスタの代表点は既存の代表点から遠くなるように選択される．KKZ による初期値設定のアルゴリズムを図 4.1 に示す．また，代表点が決定する様子を図 4.2 に示す．

[step 1] 各データのノルムを計算し，最大となるデータを 1 番目のクラスタの代表点 c_1 とする．

$$c_1 = \underset{x_i}{\text{argmax}}\{||x_i||\} \tag{4.1}$$

[step 2] 各データと既に決まっている代表点 $c_1, c_2, \cdots, c_k$ との距離 d_{ij} を計算する．

$$d_{ij} = ||x_i - c_j|| \tag{4.2}$$

[step 3] 各データの最小の d_{ij} を選択する．

$$d_i = \min\{d_{ij}\} \tag{4.3}$$

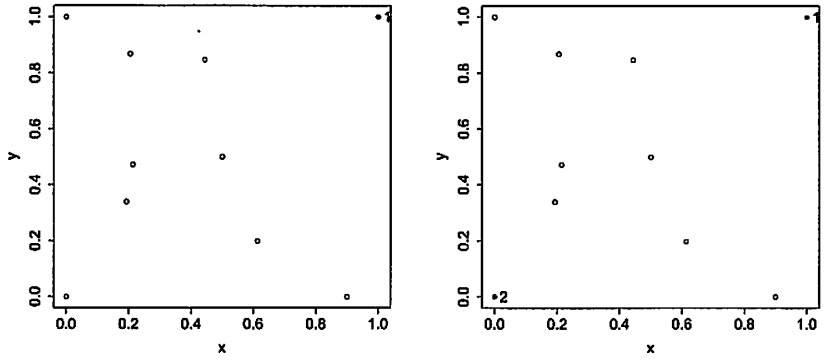
[step 4] d_i の中で最大のものを k 番目のクラスタの代表点 c_k に設定する．

$$c_k = \underset{x_i}{\text{argmax}}\{d_i\} \tag{4.4}$$

[step 5] (a) 設定したクラスタ数だけ初期値が作成されたならば終了する．

(b) 設定したクラスタ数だけ初期値が作成されていなければ step 2 に戻る．

図 4.1: KKZ による初期値設定手順



(a) 代表点 c_1 決定　(b) 代表点 c_2 決定

(c) 代表点 c_3 決定　(d) 代表点 c_4 決定

図 4.2: KKZ によって代表点が決定する様子

第5章

実験

5.1 使用するデータ

実験に使用する文書データはクラスタリングツール CLUTO [1] に付属するデータセットの中から表5.1に示すデータを用いる.

表 5.1: データセット

データセット名	文書数	単語数	出現単語数	クラスタ数
fbis	2463	2000	393386	17
tr11	414	6429	116613	9
tr12	313	5804	85640	8
tr41	878	7454	171509	10
wap	1560	8460	220482	20

5.2 クラスタリングの評価

クラスタリング結果の評価方法として，エントロピーと純度を用いる.

また，クラスタリングが評価関数を小さくしていることを確認するため K-means の評価関数を用いる.

5.2.1 エントロピー

クラスタ C_i に対するエントロピー E_i を求めて，クラスタのデータ数による重み付平均をとることで全体のエントロピーを定義する.

[1] http://glaros.dtc.umn.edu/gkhome/views/cluto/

式（5.1）～（5.3）にエントロピーを定義する.

$$\sum_{i=1}^{K} \frac{|C_i|}{N} E_i = \sum_{i=1}^{K} \frac{\sum_{j=1}^{K} x_{ij}}{N} E_i \tag{5.1}$$

$$E_i = -\sum_{h=1}^{K} P(A_h|C_i) \log P(A_h|C_i) \tag{5.2}$$

$$\frac{|A_h \hat{C}_i|}{|C_i|} = \frac{x_{ih}}{\sum_{j=1}^{K} x_{ij}} \tag{5.3}$$

5.2.2 純度

クラスタ C_i に対する純度 P_i とは，正解クラスタのデータをどの程度含んでいるかを表す．全体の純度は各クラスタのデータ数による重み付平均をとることで定義する.

式（5.4），(5.5）に純度を定義する.

$$\sum_{i=1}^{K} \frac{|C_i|}{N} P_i = \frac{1}{N} \sum_{i=1}^{K} \max_h |C_i \hat{A}_h| \tag{5.4}$$

$$P_i = \frac{1}{|C_i|} \max_h |C_i \hat{A}_h| \tag{5.5}$$

5.3 従来の初期値設定法によるクラスタリング

実験方法を図 5.1 に示す.

[step 1] TF*IDF で重み付けし，正規化する.

[step 2] 距離行列を求める.

[step 3] 初期値をランダムに取り出す.

[step 4] WKK でクラスタリングする.

図 5.1: 従来の手法によるクラスタリングの流れ

クラスタリングには WKK を利用してマルチレベルクラスタリングを行うツールである graclus [2] を WKK のみを行うように変更したものを

[2] http://www.cs.utexas.edu/users/dml/Software/graclus.html

使用する.

5.4 KKZ を使用したクラスタリング

実験方法を図 5.2 に示す.

[step 1] TF*IDF で重み付けし，正規化する.

[step 2] 距離行列を求める.

[step 3] KKZ で初期値を設定する.

[step 4] WKK でクラスタリングする.

図 5.2: 初期値設定法に KKZ を用いたクラスタリングの流れ

また，提案手法（wkkm+kkz）と比較のための K-means（km）および WKK（wkkm）の平均を算出した.

5.5 実験結果

5.5.1 データセット fbis

従来の初期値設定法によるクラスタリングの性能を表5.2，図5.3に示す.

初期値設定法に KKZ を用いたクラスタリングの性能と評価関数値を表 5.3，図 5.4 に示す.

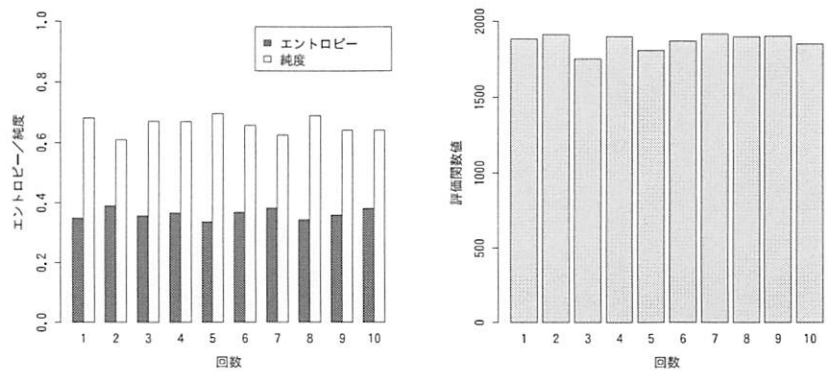
表 5.2 より，10 回のうちクラスタリングの性能が最良となる場合は 5 回目のエントロピー 0.337，純度 0.695，評価関数値 1808 と分かる．評価関数値が最小となる場合は 3 回目のエントロピー 0.357，純度 0.671，評価関数値 1754 と分かる.

表 5.2: 従来の初期値設定法によるクラスタリングの性能（データセット fbis）

回数	エントロピー	純度	評価関数値
1	0.349	0.682	1881
2	0.390	0.609	1913
3	0.357	0.671	1754
4	0.367	0.670	1900
5	0.337	0.695	1808
6	0.369	0.657	1870
7	0.382	0.625	1917
8	0.344	0.689	1898
9	0.360	0.641	1901
10	0.381	0.641	1852

表 5.3: クラスタリングの性能の比較（データセット fbis）

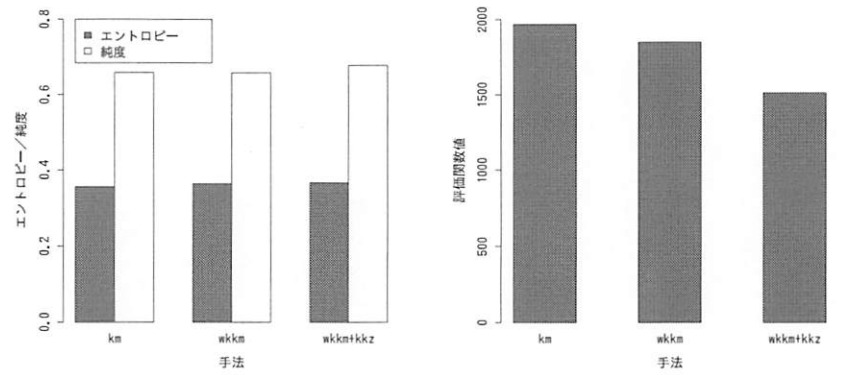
	km	wkkm	wkkm+kkz
エントロピー	0.356	0.364	0.366
純度	0.659	0.658	0.678
評価関数値	1968	1851	1515



(a) 従来の初期値設定法によるクラスタリングの性能

(b) 従来の初期値設定法によるクラスタリングの評価関数値

図 5.3: 従来の初期値設定法によるクラスタリングの結果（データセット fbis）



(a) クラスタリングの性能比較　　(b) クラスタリングの評価関数値比較

図 5.4: クラスタリング結果の比較 （データセット fbis）

5.5.2　データセット tr11

従来の初期値設定法によるクラスタリングの性能を表5.4，図5.5に示す．

初期値設定法に KKZ を用いたクラスタリングの性能と評価関数値を表5.5，図5.6に示す．

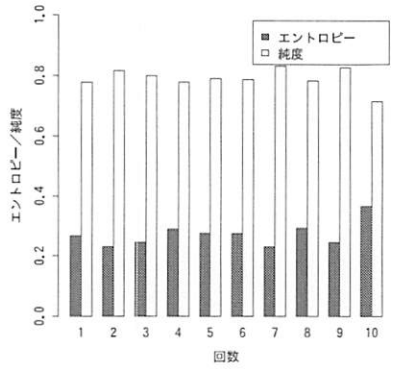
表5.4より，10回のうちクラスタリングの性能が最良となる場合および評価関数値が最小となる場合は7回目のエントロピー 0.231，純度 0.831，評価関数値 245 と分かる．

表 5.4: 従来の初期値設定法によるクラスタリングの性能（データセット tr11）

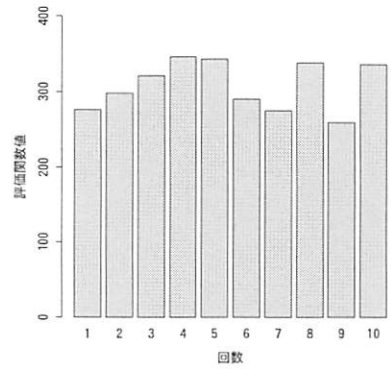
回数	エントロピー	純度	評価関数値
1	0.267	0.778	276
2	0.231	0.816	298
3	0.246	0.800	321
4	0.290	0.778	346
5	0.276	0.790	343
6	0.275	0.787	290
7	0.231	0.831	245
8	0.294	0.783	338
9	0.246	0.826	259
10	0.367	0.715	336

表 5.5: クラスタリングの性能の比較（データセット tr11）

	km	wkkm	wkkm+kkz
エントロピー	0.307	0.272	0.215
純度	0.750	0.790	0.838
評価関数値	360	317	309

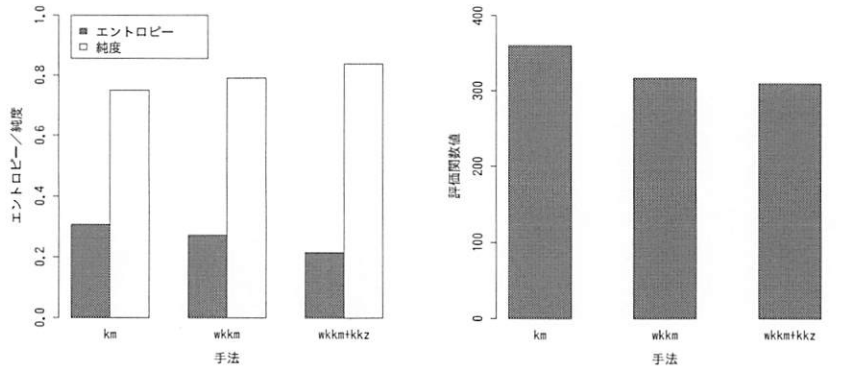


(a) 従来の初期値設定法によるクラスタリングの性能



(b) 従来の初期値設定法によるクラスタリングの評価関数値

図 5.5: 従来の初期値設定法によるクラスタリングの結果（データセット tr11）



(a) クラスタリングの性能比較　　(b) クラスタリングの評価関数値比較

図 5.6: クラスタリング結果の比較（データセット tr11）

5.5.3　データセット tr12

従来の初期値設定法によるクラスタリングの性能を表 5.6，図 5.7 に示す．

初期値設定法に KKZ を用いたクラスタリングの性能と評価関数値を表 5.7，図 5.8 に示す．

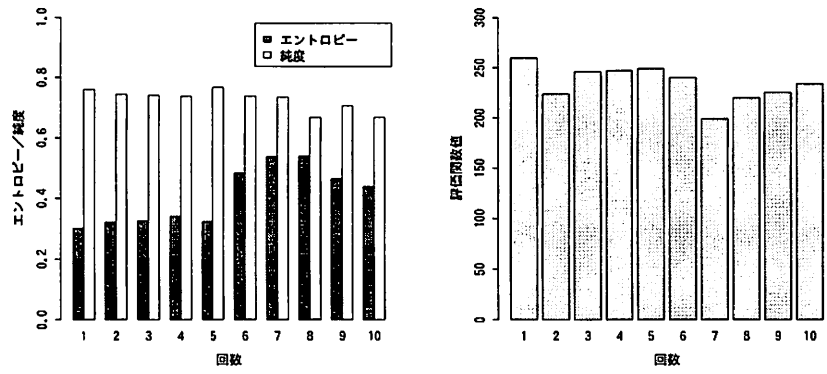
表 5.6 より，10 回のうちクラスタリングの性能が最良となる場合は 5 回目のエントロピー 0.323，純度 0.767，評価関数値 249 と分かる．評価関数値が最小となる場合は 7 回目のエントロピー 0.337，純度 0.735，評価関数値 199 と分かる．

表 5.6: 従来の初期値設定法によるクラスタリングの性能（データセット tr12）

回数	エントロピー	純度	評価関数値
1	0.301	0.760	260
2	0.321	0.744	224
3	0.325	0.741	246
4	0.340	0.738	247
5	0.323	0.767	249
6	0.349	0.738	240
7	0.337	0.735	199
8	0.391	0.668	220
9	0.392	0.706	225
10	0.406	0.668	236

表 5.7: クラスタリングの性能の比較（データセット tr12）

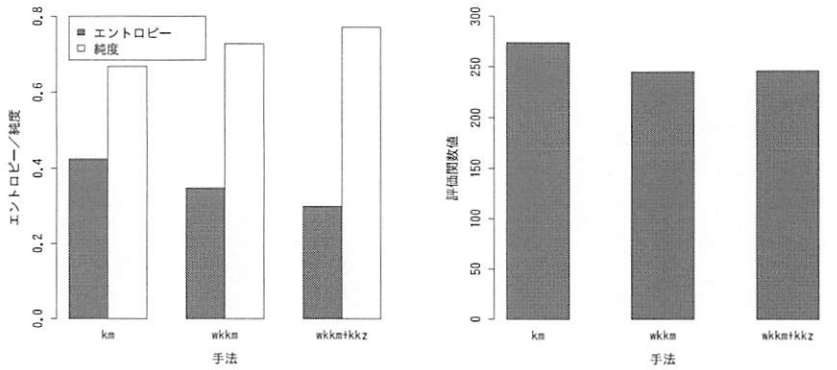
	km	wkkm	wkkm+kkz
エントロピー	0.425	0.348	0.299
純度	0.668	0.727	0.770
評価関数値	274	245	246



(a) 従来の初期値設定法によるクラスタリングの性能

(b) 従来の初期値設定法によるクラスタリングの評価関数値

図 5.7: 従来の初期値設定法によるクラスタリングの結果（データセット tr12）



(a) クラスタリングの性能比較　　(b) クラスタリングの評価関数値比較

図 5.8: クラスタリング結果の比較（データセット tr12）

5.5.4　データセット tr41

従来の初期値設定法によるクラスタリングの性能を表5.8，図5.9に示す．

初期値設定法に KKZ を用いたクラスタリングの性能と評価関数値を表5.9，図5.10に示す．

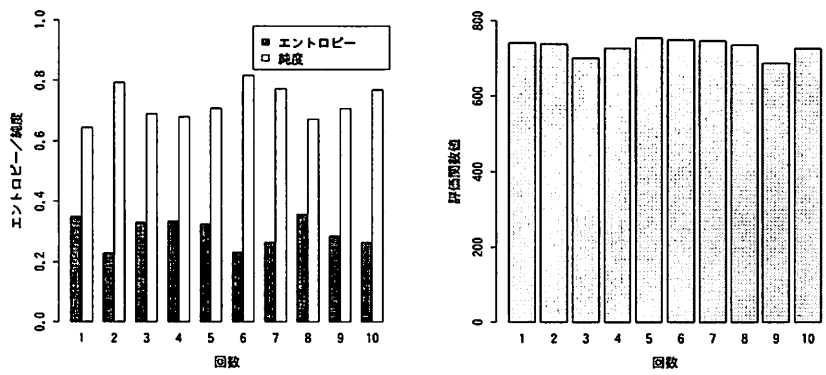
表5.8より，10回のうちクラスタリングの性能が最良となる場合は6回目のエントロピー 0.232，純度 0.817，評価関数値 750 と分かる．評価関数値が最小となる場合は9回目のエントロピー 0.286，純度 0.708，評価関数値 688 と分かる．

表 5.8: 従来の初期値設定法によるクラスタリングの性能（データセット tr41）

回数	エントロピー	純度	評価関数値
1	0.350	0.644	742
2	0.229	0.793	739
3	0.330	0.690	701
4	0.336	0.679	728
5	0.325	0.708	755
6	0.232	0.817	750
7	0.265	0.773	747
8	0.357	0.672	736
9	0.286	0.708	688
10	0.265	0.770	727

表 5.9: クラスタリングの性能の比較（データセット tr41）

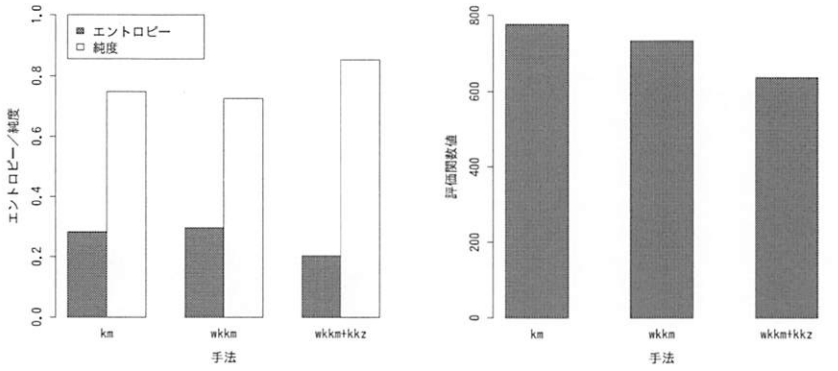
	km	wkkm	wkkm+kkz
エントロピー	0.283	0.297	0.203
純度	0.749	0.725	0.852
評価関数値	776	733	637



(a) 従来の初期値設定法によるクラスタリングの性能

(b) 従来の初期値設定法によるクラスタリングの評価関数値

図 5.9: 従来の初期値設定法によるクラスタリングの結果（データセット tr41）



(a) クラスタリングの性能比較　　(b) クラスタリングの評価関数値比較

図 5.10: クラスタリング結果の比較（データセット tr41）

5.5.5　データセット wap

従来の初期値設定法によるクラスタリングの性能を表 5.10，図 5.11 に示す.

初期値設定法に KKZ を用いたクラスタリングの性能と評価関数値を表 5.11，図 5.12 に示す.

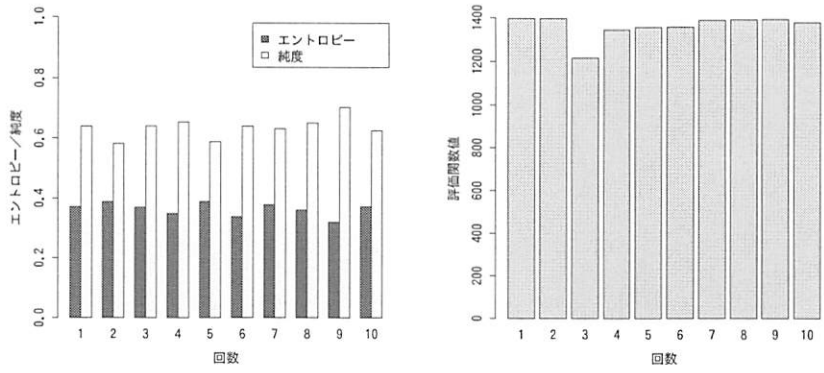
表 5.10 より，10 回のうちクラスタリングの性能が最良となる場合は 9 回目のエントロピー 0.320，純度 0.704，評価関数値 1359 と分かる．評価関数値が最小となる場合は 3 回目のエントロピー 0.369，純度 0.641，評価関数値 1217 と分かる.

表 5.10: 従来の初期値設定法によるクラスタリングの性能（データセット wap）

回数	エントロピー	純度	評価関数値
1	0.371	0.640	1398
2	0.388	0.582	1398
3	0.369	0.641	1217
4	0.349	0.654	1345
5	0.389	0.589	1358
6	0.338	0.641	1358
7	0.379	0.632	1390
8	0.361	0.651	1393
9	0.320	0.704	1359
10	0.372	0.626	1379

表 5.11: クラスタリングの性能の比較（データセット wap）

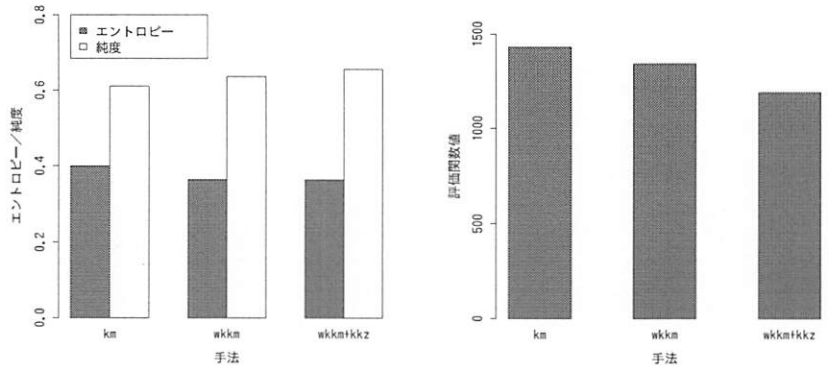
	km	wkkm	wkkm+kkz
エントロピー	0.399	0.364	0.363
純度	0.611	0.636	0.656
評価関数値	1432	1343	1191



(a) 従来の初期値設定法によるクラスタリングの性能

(b) 従来の初期値設定法によるクラスタリングの評価関数値

図 5.11: 従来の初期値設定法によるクラスタリングの結果（データセット wap）



(a) クラスタリングの性能比較　　(b) クラスタリングの評価関数値比較

図 5.12: クラスタリング結果の比較（データセット wap）

第6章
考察

表5.2，5.4，5.6，5.8，5.10より，従来の初期値設定法ではクラスタリングの性能および評価関数値が安定しないことが確認できる．

手法 wkkm の結果の平均と wkkm+kkz の結果を比較する．表5.3より，データセット fbis では，エントロピーは 0.002 増加し，純度は 0.020 減少し，評価関数値では 336 減少して，性能が向上してる．表5.5より，データセット tr11 では，エントロピーは 0.057 減少し，純度は 0.048 増加し，評価関数値は 8 減少して，性能が向上している．表5.7より，データセット tr12 では，エントロピーは 0.049 減少し，純度は 0.043 増加し，評価関数値は 1 増加し，性能が向上している．表5.9より，データセット tr41 では，エントロピーは 0.094 減少し，純度は 0.133 増加し，評価関数値は 96 減少して，性能は向上している．表5.11より，データセット wap では，エントロピーは 0.001 減少し，純度は 0.020 増加し，評価関数値は 152 減少して，性能が向上している．これらの結果から，提案手法の方が全体的に良いクラスタリング結果を得ることができた．

また，クラスタリングの性能が最良となる場合と比較する．データセット fbis では，エントロピーは 0.029 増加し，純度は 0.017 減少し，評価関数値は 293 減少して，性能が悪化している．データセット tr11 では，エントロピーは 0.016 減少し，純度は 0.007 増加し，評価関数値は 64 増加して，性能が向上している．データセット tr12 では，エントロピーは 0.024 減少し，純度は 0.003 増加し，評価関数値は 3 減少して，性能が向上している．データセット tr41 では，エントロピーは 0.029 減少し，純度は 0.035 増加し，評価関数値は 113 減少して，性能が向上している．データセット wap では，エントロピーは 0.043 増加し，純度は 0.048 減少し，評価関数値は 189 減少して，性能が悪化している．このように，提案手法の方が半数のデータセットで従来のデータの最良のクラスタリング結果より良いクラスタリング結果を得ることができた．

データセット fbis において，従来の初期値設定法と 提案手法 を比較するとエントロピーが悪化しているが，クラスタリング手法を K-means から WKK へと変更した場合に性能が悪化していることが確認できる．このことから WKK で使用しているカーネルがデータセット fbis に適していないことが考えられる．

第7章
おわりに

本論文では WKK の初期値設定法に KKZ を使用することを提案した．従来の初期値設定法を用いた文書クラスタリング結果と初期値設定法に KKZ を使用した結果を比較すると，複数のデータセットで性能が同等，もしくは向上する結果が得られ，提案手法が有効であることが確認できた．

一方，一部のデータセットでは性能が低下することを確認した．これは WKK で使用するカーネルがデータセットに適していないことが原因であると考えられる．カーネルを適切に設定することによって，クラスタリングの性能はさらに向上すると考えられる．今後これらの点を改良した文書クラスタリングを作成する．

謝辞

最後に本研究の遂行及び論文の作成にあたり，終始適切な助言を賜り，また丁寧に指導して下さった茨城大学工学部情報工学科の新納浩幸准教授，佐々木稔講師に感謝の意を表します．

また，日常の議論を通じて多くの知識や示唆を頂いた自然言語処理研究室の皆様に感謝します．

参考文献

[1] I.S.Dhillon and Y.Guan and B. Kulis. Kernel k-means, Spectral Clustering and Normalized Cuts. In *KDD*, pp. 551–556. ACM Press, 2004.

[2] Chris Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst Simon. Spectral min-max cut for graph partitioning and data clustering. In *Proc. 1st IEEE International Conference on Data Mining*, pp. 107–114, 2001.

[3] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, No. 8, pp. 888–905, 2000.

[4] Ji He and Man Lan and Chew-lim Tan and Sam-yuan Sung and Hwee-boon Low. Initialization of cluster refinement algorithms: A review and comparative study. In *Proceedings of International Joint Conference on Neural Networks (IJCNN)*, pp. 297–302, 2004.

付録A
ソースリスト

A.1 Weighted Kernel K-means のソース

図 A.1 に本論文で作成した WKK のクラスタリング結果を出力するプログラムソースを示す．これは C で記述した．

図 A.1: graclus.cpp

```
1  #include <metis.h>
2
3  int boundary_points;
4  int cutType; //cut type, default is normalized cut
5  int memory_saving; // forbid using local search or empty
       cluster removing
6  //char mlwkkm_fname[256]; //used to store coarsest file
7
8  /*********************
9    Weighted Kernel K-means function
10 *********************/
11
12 main(int argc, char *argv[])
13 {
14   int nparts;
15   GraphType graph;
16   char *filename, *seedfile;
17   char result[255];
18   int wgtflag, seedflag;
19   int *w;
20   int i;
21   //CtrlType ctrl;
22   FILE *fp;
23
24   boundary_points = 0;
25   filename = argv[1];
26   nparts = atoi(argv[2]);
27   sprintf(result, "%s.part.%d", filename, nparts);
```

```
28
29    // 初期値が与えられているかどうか
30    if (argc < 3) {
31      seedflag = 0;
32    } else {
33      seedfile = argv[3];
34      seedflag = 1;
35    }
36
37    ReadGraph(&graph, filename, &wgtflag); // グラフの読み
          込み
38    printf("vertexs is %d\n", graph.nvtxs);
39    printf("edges is %d\n", graph.nedges);
40
41    w = idxsmalloc(graph.nvtxs, 0, "weight"); // 重みの配
          列
42    cutType = NCUT; //
43
44    Compute_Weights(0, &graph, w);
45
46    // 初期値を設定する
47    graph.where = imalloc(graph.nvtxs, "where");
48    if (seedflag) {
49      // 初期値ファイルあり
50      int d;
51      fp = fopen(seedfile, "r");
52      for (i = 0; fscanf(fp, "%d", &d) != EOF; i++) {
53        graph.where[i] = d;
54      }
55      fclose(fp);
56    } else {
57      // 初期値ファイルなし
58      srand((unsigned)time(NULL));
59      for (i = 0; i < graph.nvtxs; i++) {
60        graph.where[i] = rand() % nparts;
61      }
62    }
63
64    // WKK 実行
65    Weighted_kernel_k_means(0, &graph, nparts, w, 0, 0);
66
67    // 結果の書き出し
68    fp = fopen(result, "w");
69    for (i = 0; i < graph.nvtxs; i++)
70      fprintf(fp, "%d\n", graph.where[i]);
71    fclose(fp);
72  }
```

図A.2に本論文で作成したデータセットを距離行列に変換するプログラムソースを示す．これはRubyで作成した．内部でRのソースを生成し，Rのバッチ

処理でデータセットを距離行列に変換し，保存している．

図 A.2: dst_r_batch.rb

```
#! ruby -Ku
class R_Batch
  def initialize(filename)
    @infile = filename
    @batch = "rbatch.r"
    @outdir = "dst"
    @outfile = filename.sub("mtx", "dst")
  end

  def make_r_script
    rscript = <<-EOS
    library(Matrix)
    source("def.r")
    x <- readMM("#{@infile}")
    w <- x %*% t(x)
    w <- as.matrix(w)
    diag(w) <- 0
    write(w, file="#{@outdir}/#{@outfile}", ncol=ncol(w
        ))
    EOS

    open(@batch, "w") { |file|
      file.puts rscript
    }
  end

  def run
    unless File.directory?(@outdir)
      Dir::mkdir(@outdir)
      print "mkdir ", @outdir, "\n"
    end
    system("R --max-mem-size=2000M --vanilla -q < rbatch
        .r")
  end
end

ARGV.each { |e|
  a = R_Batch.new(e)
  a.make_r_script
  a.run
}
```

図 A.3 に本論文で作成した距離行列を graclus の入力に変形するプログラムソースを示す．

図 A.3: dst2graph.rb

```
1  #! ruby -Ku
2  require 'tempfile'
3
4  class MakeGraph
5    def initialize(filename)
6      @inname = filename
7      @outname = filename.sub("dst", "graph")
8      @node = 0
9      @edge = 0
10     read_write_weight_percent(@inname, @outname)
11     #check_weight(@outname)
12   end
13
14   # ノード間の重みなし
15   def read_write(inname, outname)
16     print inname, " => ", outname, "\n"
17     tmp  = Tempfile.open("temp")
18     open(inname) { |input|
19       input.each { |l|
20         count = 1
21         outdata = ""
22         l.strip.split(" ").each { |e|
23           unless e.to_f == 0
24             outdata = outdata + count.to_s + " "
25             @edge = @edge + 1
26           end
27           count = count + 1
28         }
29         tmp.puts outdata.strip
30         @node = @node + 1
31       }
32     }
33     tmp.close
34     @edge = @edge/2
35
36 ____# ヘッダを書く
37 ____tmp.open
38 ____open(outname, "w") { |file|
39 ______file.print @node, " ", @edge, "\n"
40 ______tmp.each { |l|
41 ________file.puts l.strip
42 ______}
43 ____}
44 ____tmp.close
45 __end
46
47 __# ノード間の重みあり，値を mult 倍して重みにする
48 __def read_write_weight(inname, outname, mult)
49 ____print inname, " => ", outname, "\n"
50 ____tmp  = Tempfile.open("temp")
```

```
51  ____open(inname)_{_|input|
52  ______input.each_{_|l|
53  ________count_=_1
54  ________outdata_=_""
55  ________l.strip.split("_").each_{_|e|
56  __________w_=_(e.to_f*mult).round
57  __________unless_w_==_0
58  ____________outdata_=_outdata_+_count.to_s_+_"_"_+_w.
        to_s_+_"_"
59  ____________@edge_=_@edge_+_1
60  __________end
61  __________count_=_count_+_1
62  ________}
63  ________tmp.puts_outdata.strip
64  ________@node_=_@node_+_1
65  ______}
66  ____}
67  ____tmp.close
68  ____@edge_=_@edge/2
69
70      tmp.open
71      open(outname, "w") { |file|
72        file.print @node, "_", @edge, "_", 1, "\n"
73        tmp.each { |l|
74          file.puts l.strip
75        }
76      }
77      tmp.close
78    end
79
80    # ノード間の重みあり，重みを%と考え100倍して使う
81    def read_write_weight_percent(inname, outname)
82       read_write_weight(inname, outname, 100)
83    end
84
85    def check(filename)
86      node = 0; edge = 0; count_n = 0; count_e = 0
87      open(filename) { |file|
88        e = file.gets.strip.split("_")
89        node = e[0].to_i
90        edge = e[1].to_i
91        file.each { |l|
92          count_n = count_n + 1
93          count_e = count_e + l.strip.split("_").length
94        }
95      }
96      count_e = count_e/2
97  ____print_"first_line_:_",_node,_"_",_edge,_"\n"
98  ____print_"count_node_:_",_count_n,_"\n"
99  ____print_"count_edge_:_",_count_e,_"\n"
```

```
100 ␣␣end
101
102 ␣␣def␣check_weight(filename)
103 ␣␣␣␣node␣=␣0;␣edge␣=␣0;␣format␣=␣0;␣count_n␣=␣0;␣count_e
    ␣=␣0
104 ␣␣␣␣open(filename)␣{␣|file|
105 ␣␣␣␣␣␣e␣=␣file.gets.strip.split("␣")
106 ␣␣␣␣␣␣node␣=␣e[0].to_i
107 ␣␣␣␣␣␣edge␣=␣e[1].to_i
108 ␣␣␣␣␣␣format␣=␣e[2].to_i
109 ␣␣␣␣␣␣file.each␣{␣|l|
110 ␣␣␣␣␣␣␣␣count_n␣=␣count_n␣+␣1
111 ␣␣␣␣␣␣␣␣count_e␣=␣count_e␣+␣l.strip.split("␣").length
112 ␣␣␣␣␣␣}
113 ␣␣␣␣}
114 ␣␣␣␣count_e␣=␣count_e/4
115     print "first␣line␣:␣", node, "␣", edge, "␣", format,
          "\n"
116     print "count␣node␣:␣", count_n, "\n"
117     print "count␣edge␣:␣", count_e, "\n"
118   end
119 end
120
121 if ARGV.length == 0
122   print "usage:␣", $0, "<input␣file>\n"
123 end
124
125 ARGV.each { |e|
126   a = MakeGraph.new(e)
127 }
```

A.2　KKZ のソース

図 A.4 に本論文で作成した KKZ により代表点を決定し出力するプログラムソースを示す．これは R で記述した．

図 A.4: kkz.r

```
1  library(Matrix)
2
3  idx_max <- function(dist) {
4    n <- length(dist)
5    max.dist <- max(dist) # dist中のmaxを保存
6    idxs <- c(1:n)[dist == max.dist]
7    if(length(idxs) > 1) {
8      idx <- idxs[1]
9    }else{
10     idx <- idxs
```

```
11    }
12    return(idx)
13  }
14
15  idx_min <- function(dist) {
16    n <- length(dist)
17    min.dist <- min(dist) # dist中の最小値を保存
18    idxs <- c(1:n)[dist == min.dist]
19    if(length(idxs) > 1) {
20      idx <- idxs[1]
21    }else{
22      idx <- idxs
23    }
24    return(idx)
25  }
26
27  kkz <- function(x, n) {
28    x1 <- as.matrix(x)
29    idx <- numeric(n)
30    # c_1 を決める
31    d <- apply(x1, 1, function(a) { sum(a^2) })
32    idx[1] <- idx_max(d)
33
34    bign <- 10000 # 重複を防ぐ
35    # c_i(i=2..K) を決める
36    for (i in 2:n) {
37      c <- x1[idx,] # 行列の一部を取り出すのはスパースじゃ
          ダメっぽい
38      c <- matrix(c, nc=ncol(x))
39      c <- as(c, "CsparseMatrix")
40      d <- x %*% t(c)
41      d <- apply(d, 1, max)
42      # d_j = \min{||x_j - c_k||} なので，第2項目が最大の
          ものを選ぶ
43      d[idx] <- bign
44      idx[i] <- idx_min(d)
45      # c_i = argmax{d_j} なので，第2項目が最小のものを選
          ぶ
46    }
47    return(idx)
48  }
```