単語の出現頻度の z 値を用いた論文分類手法

明治大学理工学部数学科

鈴木彩加

2022 年 2 月 16 日

概要

本研究では引用情報を使わず，単語の出現頻度により論文の関係性を測る方法を提案する．これにより引用されていないが関連性の高い主題を扱っているかどうかを見ることができる．単語の出現頻度の z 値を用いて k 平均法により，論文のクラスタリングを行った．また，階層的クラスタリングによって各論文同士の関連性の高さを測った．その結果，機械学習の一つであるクラスタリングにおいて，統計学の仮説検定を応用することにより各論文の特徴を取り出すことができることを示した．

1　はじめに

昨今の情報技術の発達により，論文の出版数は急激に増加している．その中ですべての論文に目を通すことは難しく，また目を通した論文を管理することの難易度も高くなっている．そのため，自動的に論文を管理する技術が求まれる．

論文は引用の仕組みがあるため，従来は共引用と書誌的結合の概念により論文の類似性を見ている．例えば，論文間の連鎖を可視化するツールとして Connected Papers [2] が実装されている．共引用と書籍的結合では，引用と参考文献が非常に重複している 2 つの論文は，関連する主題を扱う可能性が高いとみなしている．

本研究では引用情報を使わず，単語の出現頻度により関連性を測る方法を提案する．この方法では従来の方法と違い，引用されてはいないが関連性の高い主題を扱っているかどうかを見ることができる．具体的には，単語の出現頻度の z 値を用いてクラスタリングを行った．実行環境は，Google Colaboratory 上で，言語は Python を用いた．その結果，30 本の論文を可換環・微分幾何・偏微分方程式の分野によって，クラスタリングすることができた．また，それぞれのクラスタで各分野の内容を示す単語を取り出すことができた．

本論文の構成は以下である．第 2 章では，本研究の主題であるクラスタリングの知識についてまとめる．クラスタリングの具体的な手法として，k 平均法，階層的クラスタリングについて説明する．第 3 章では，検定の知識についてまとめる．本研究では実験を 2 つ行った．特徴量の実験，クラスタリングに用いるデータの次元削除の方法についての実験を行った．クラスタリングを行う特徴量として z 値を使う．z 値は統計的検定における概念であり，その関連知識と概要について説明する．第 4 章では，特徴量 z 値の実験について説明する．具体的な提案アルゴリズム，各手法でのクラスタの内訳とその特徴的な単語についてをまとめた．第 5 章では，クラスタリングに用いるデータの次元削除の方法について説明する．具体的な提案アルゴリズム，各手法でのクラスタの内訳とその特徴的な単語についてをまとめた．第 6 章では，まとめとして考察と今後の課題を議論する．付録 A には本研究のソースコードを添付した．

2　クラスタリング

クラスタリングとは，データをいくつかのグループ (**クラスタ**) に分けることである．各クラスタ内のデータが互いに似たものになるように分けることで，データが何種類に分けられるかを調べることができる．クラスタリングのアルゴリズムは，データセットにはあらかじめ正解がわからない，すなわち，ラベルのないデータを扱う場合に使われる．そのため学習後のモデルが最適かどうか判断するのは困難であり，各グループが何を示しているかは解釈が必要である．

本研究では，arXiv の論文の分野情報や，各クラスタに特徴的な用語を抽出することで，クラスタリングの妥当性や，各クラスタの意味づけについても議論できるようにした．クラスタリングのアルゴリズムでは，類似度を表すデータ間の距離を使うため，クラスタリングの結果は距離の取り方に大きく依存する．本研究で使った距離については，第 3 章で説明する．

本章では今回用いた k 平均法 (k-means) と階層的クラスタリングについて説明する．

2.1　k 平均法

ここでは，k 平均法 (k-means) について説明する．詳しくは [1, 第 9 章] などを参照されたい．

k 平均法とは，クラスタリングの手法の一つである．クラスタの平均を用い，あらかじめ与えられたクラスタ数 k 個に分類するアルゴリズムである．単純なアルゴリズムであり，広く用いられている．k 平均法は，必ずしも正しい答えが導けるとは限らないが，ある程度の精度で正解に近い解を得られる．正しい答えを導く保証がない代わりに結果が出るまでの時間が短い．

最初にランダムに割り振るクラスタによって結果が大きく変わってしまうため 1 回の結果で最良のものが得られるとは限らない．また，クラスタ数をあらかじめ与えなければならないので，最適なクラスタ数を選ぶための考察をしなければならない．

先に述べたように k 平均法は結果が出るまでの時間が短いので，大量のデータを扱うのに適しており，今回の実装で採用することにした．また，今回のデータの数値は絶対値よりも相対値を扱いたい．そのためデータの平均によってクラスタを決定していく方法を用いた．

具体的な k 平均法のアルゴリズムは以下である．

(1) クラスタの数 k を決める．
(2) 初期のクラスタの中心点として，k 個のデータを無作為に取る．
(3) 各データをそこから最も近い中心点のクラスタに割り当てる．
(4) クラスタごとの平均点をそのクラスタの新しい中心点とする．

(5) ステップ (3) と (4) を割り当てに変更がなくなるまで繰り返す.

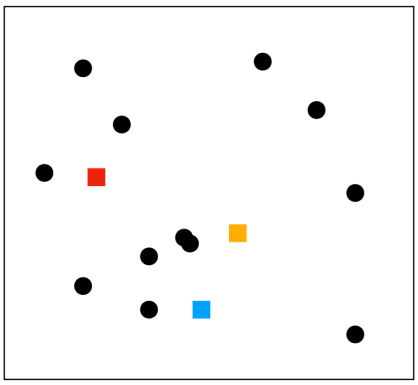

図 1　k 平均法の例:(1), (2)

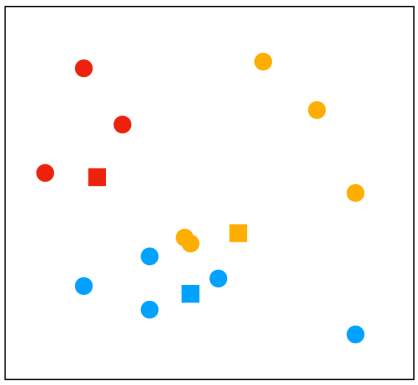

図 2　k 平均法の例: (3)

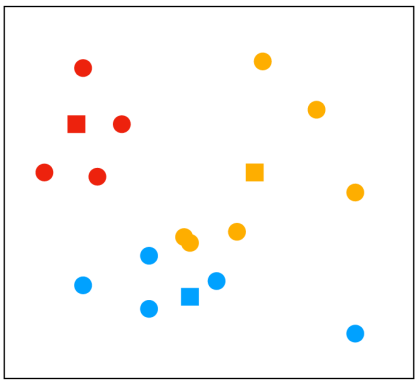

図 3　k 平均法の例: (4)

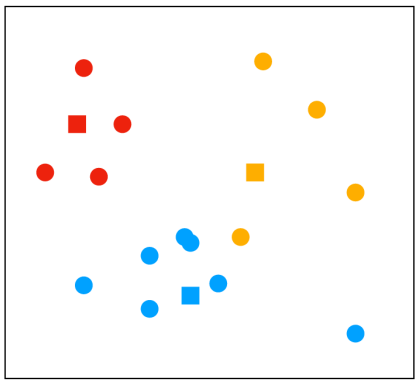

図 4　k 平均法の例: 2 回目の (3)

各ステップの詳細を見ていく.

(1) クラスタの数 k を決める.

k は 2 以上の自然数. k としては, 最適だと思われるクラスタ数を推定する. 最適なクラスタ数を決める方法としてはエルボー法などが知られている.

(2) 初期のクラスタの中心点として, k 個データを無作為に取る.

図 1 では $k = 3$ とした 12 個 のデータのクラスタリングを表しており, ● がデータを示している ■ はクラスタの中心として無作為にとった点である.

(3) クラスタごとの平均点をそのクラスタの新しい中心点とする.

各データと各中心点の間の距離を求める．距離はユークリッド距離やマンハッタン距離など事前に決めた距離の取り方を使う．各データに関して一番距離の小さい中心点のクラスタに割り当て，ラベルを更新する．

(4) クラスタごとの平均点を新しい中心点とする．

各中心点について，それをラベルとするデータの特徴量ベクトルの平均を出す．これが新しい中心点となる．

(5) ステップ (3) と (4) を割り当てに変更がなくなるまで繰り返す．

新しい中心点に対して，各データ x との距離を計算し，ラベルを更新する．各ラベルの中心点を計算する．以上の操作を割り当てに変更がなくなるまで繰り返す．

変化のなくなったラベル割り当てが最終結果である．

2.2 階層的クラスタリング

クラスタリングのもう 1 つの手法として，階層的クラスタリング (hierarchical clustering) を説明する．詳しくは [4, 第 9 章] などを参照されたい．

階層的クラスタリングは，最も似ている組み合わせから順番にクラスタにしていく方法である．階層的クラスタリングは，あらかじめクラスタの数を決めずにクラスタリングを行うことができる．結果は樹形図として視覚的に見ることができる．これにより，データの関係性が把握しやすい．階層的クラスタリングは，すべてのデータの組み合わせを計算しなければならない．そのため，計算量が多くなりたくさんのデータを扱いにくくなる．また，視覚化の方法を工夫しなければ読みにくくなる．

階層的クラスタリングの結果は，**デンドロイドグラム** (樹形図) で表現される (図 5).

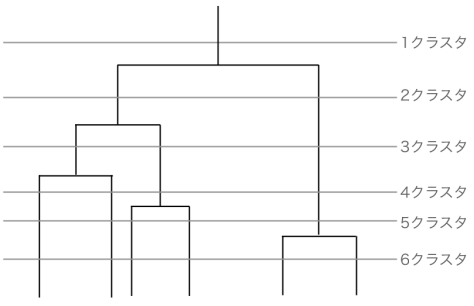

図 5　デンドロイドグラム (樹形図) の例

デンドロイドグラムは結合した 2 つのクラスタの分岐点の高さをクラスタ間の距離にして木を構成している．上に行くほど枝の数が (1 個ずつ) 減っていく．例の図 5 のように枝の数が k の高さで水平に切ると，

クラスタリングが得られる．階層的クラスタリングは事前にクラスタ数 k を決めずにすべての $1 \leq k \leq N$ でのクラスタリングを同時に見ることができる．

具体的なアルゴリズムは以下である．N 個のデータをクラスタリングすることを考える．最初に 1 つのデータのみを含むクラスタをデータの数 N 個作る．ある基準で近いとされるクラスタどうしを結合していき，クラスタ数が 2 まで減ったところで終了する．クラスタ同士の近さはサンプル間の距離 $d(\cdot, \cdot)$ とクラスタ間の距離の取り方によって決まる．サンプル間の距離は，L_1 ノルム，L_2 ノルム，コサイン類似度等が用いられる．クラスタ間の距離は Complete リンケージ，Single リンケージ，Centroid リンケージ，Average リンケージのそれぞれである．A, B をクラスタとして，各データは $x = (x_1, x_2, \cdots, x_p)$ などで表したとき，それぞれの定義は以下である．$|A|$ はクラスタ A に含まれるデータ数である

- Complete リンケージ
$$\max_{i \in A,\, j \in B} d(x_i, y_j)$$
- Single リンケージ
$$\min_{i \in A,\, j \in B} d(x_i, y_j)$$
- Centroid リンケージ
$$d\left(\frac{1}{|A|} \sum_{i \in A} x_i, \frac{1}{|B|} \sum_{j \in B} y_j \right)$$
- Average リンケージ
$$\frac{1}{|A| \cdot |B|} \sum_{i \in A} \sum_{j \in B} d(x_i, y_j)$$

3 検定

ここでは統計学における仮説検定についてまとめる．詳しくは [3, 第 11 章] などを参照されたい．

3.1 仮説検定

統計学において**仮説検定**とは，母集団分布の母数に関する仮定を標本から検証する統計的方法の一つである．仮説が正しいと仮定した上で，それに従う母集団から，実際に観察された標本が出てくる確率を求める．その値が十分に小さければ，その仮説を棄却する．

統計的仮説検定は次のような手順で実施する．

(1) 仮説の設定

(2) 統計量の算出

(3) 統計量の確率分布

(4) 危険域の設定

(5) 判定

である．各手順について詳細を記す．

(1) 仮説の設定

ある変数とある変数が関係していると主張したい．その主張が正しいかどうか検証するためにそれと対立する**仮説** (hypothesis) を立てる．これを**帰無仮説** (null hypothesis) という．すなわち 2 つの変数には関係がないという仮説が帰無仮説である．帰無仮説に対立している主張したい仮説を**対立仮説** (Alternative hypothesis) という．対立仮説は帰無仮説が棄却された際に採択される．

(2) 統計量の算出

標本データから仮説に関係する情報である**検定統計量**を計算する．

(3) 統計量の確率分布

帰無仮説に基づき，検定統計量の確率分布を明らかにする．

(4) 危険域の設定

ある仮説を棄却するかしないかを決める基準の範囲を**危険域** (critical region) と呼ぶ．帰無仮説が正しいのに，危険域に入る (帰無仮説を棄却する) 確率を，**有意水準** (significance level) と呼ぶ．通常は $\alpha = 0.05(5\%)$ か $\alpha = 0.01(1\%)$ を用いる．検定によっては両側検定または片側検定のみということもある．

(5) 判定

データから導いた検定統計量が危険域内にあるかどうか判定する．統計量が仮定した分布の中で，算出した検定統計量と同じかそれよりも極端な値となる確率 p(p 値) を有意水準 α と比較し，$p < \alpha$ ならば危険域の内部にあると判断して，棄却する．

3.2 z 値

標本から作られる検定の核となる統計量を，**検定統計量** (test statistic) という．ここでは，母集団との差や比，z 値等を考えた．

z **値**とは，標本に対して，その平均が 0，標準偏差が 1 になるようにアファイン変換した値である．つまり標準化した値である．母集団の平均が μ，標準偏差が σ が既知であるとする．変数 x の z 値は次のよう

に定義される．

$$z = \frac{x - \mu}{\sigma}$$

データが (ほぼ) 正規分布する場合，z 値を用いて p 値を計算することになる．母集団の平均と標準偏差が未知の場合は，推定する必要がある．

4　実験 1

4.1　実験手法

4.1.1　実験概要

今回の実験では，別の分野の研究をしている数人の著者による論文をそれぞれ 5 つずつと標本の集合から 5 つの論文を用いて実験をする．それぞれの著者の論文が同じクラスタリングに入り，別の分野の研究をしている著者の論文同士が同じクラスタリングに入らなければ良い結果が得られたこととする．

まず代数・幾何・解析の 3 つの分野から論文を選ぶ．すなわち，クラスタリングの対象となる論文は代数の研究を主に行っている 1 人の論文を 5 本，幾何の研究を主に行っている 1 人の論文を 5 本，解析の研究を主に行っている 1 人の論文を 5 本，ランダムに 5 本の論文の合計 20 本の論文である．そのため，代数・幾何・解析以外の論文が含まれている可能性がある．以上から，k 平均法におけるクラスタの数を $k = 4$ と設定した．

4.1.2　アルゴリズム

アルゴリズムの概要は以下である．入力として英語で書かれた複数の pdf ファイルをとる．以下の手順でクラスタごとの特徴的な単語を得る．

Step 1. pdf ファイルを txt ファイルへ変換する．
Step 2. txt ファイルを読み込んで，単語ごとに分ける．
Step 3. 各ファイルごとに出現する単語の数を数える．
Step 4. 特徴量を計算する．
Step 5. クラスタリング
Step 6. クラスタごとに特徴を出す．

各ステップの詳細を見ていく．

Step 1. pdf ファイルを txt ファイルへ変換する．

pdf ファイルに埋め込まれているテキストを抽出し，txt ファイルとして書き出す．今回は単語の出現頻度のみに注目する．文字列に対して処理を行う．

Step 2. txt ファイルを読み込んで，単語ごとに分ける．

txt ファイルを読み込んで文を単語ごとに分ける．今回は，文字列を空白・改行・記号によって分割することで，単語の判定を行った．また，その際に以下の処理を施した．

- 単語をすべて小文字にする．
- !や?などの記号を除く．
- 文字が 2 文字以下の単語は除く．

今回はどのような単語が含まれているかについて見ていくので大文字・小文字の差によって区別をつけないものとした．また，2 文字以下の単語には意味を持っているものが少ないため，今回の処理には加えないこととした．

Step 3. 各ファイルごとに出現する単語の数を数える．

辞書型を使う．単語を key，それに対応する value をその単語の数とした．各ファイルの辞書を作り，最後に一つのデータフレームにする．この時，出現しなかった単語を key に持つ value は 0 とする．

Step 4. 特徴量を計算する．

クラスタリングを行うために単語の数から特徴量を計算する．母集団との差や比，z 値などの検定統計量が候補になる．

Step 5. クラスタリング

k 平均法と階層的クラスタリングによってクラスタリングをすることにした．距離関数としては，ユークリッド距離を取っている．

Step 6. クラスタごとに特徴を出す．

各クラスタに含まれるデータの z 値の平均値を出し，値の大きい単語をそのクラスタの特徴とした．

4.1.3 いくつかの特徴量

クラスタリングに使う特徴量としていくつかを比較する．母集団の単語の出現頻度の推定値を p，あるファイル i での単語の出現頻度を p_i とする. すなわち，あるファイル i での総単語数を n_i，単語の出現回数

を k_i とすると，

$$p = \frac{\sum k_i}{\sum n_i},$$
$$p_i = \frac{k_i}{n_i}$$

である．今回の場合，母集団の単語の出現頻度の推定値 p は，クラスタリングに用いた論文を含め 115 本の論文に含まれる単語の出現頻度とした．

p_i が p と比べて大きい単語に着目してクラスタリングを行いたい．そのため以下のような候補を使ってクラスタリングを行った．

(1) $p_i - p$

(2) p_i/p

(3) z 値

(1)$p_i - p$，(2)p_i/p については，素朴な手法である．(1)，(2) では不十分であった．詳しくは実験結果 4.2 で述べる．

(3) の z 値について説明する．母集団の単語の出現頻度の推定値を p，あるファイル i での総単語数を n_i，単語の出現回数を k_i とする．ここで，z 値の近似値として，

$$z_i = \frac{k_i - n_i p}{\sqrt{n_i p(1-p)}}$$

を考える．

この z_i は以下の意味でほぼ標準正規分布に従う．すなわち，各単語 w_i は独立にある適当な確率 p で現れるとすれば，単語数 n_i のファイルにおける w_i の出現回数は，期待値 $n_i p$，標準偏差 $\sqrt{n_i p(1-p)}$ の二項分布に従う．ラプラス=ド・モアブルの定理 (もしくは中心極限定理) から，p が十分良い精度で近似されていて，n_i が十分大きければ，z_i は標準正規分布 $N(0,1)$ にほぼ従う確率変数とみなせる．

z_i が 0 に近い場合は，そのファイルでのその単語の出現割合が，他のファイルと同程度であることを意味する．この z_i が正の大きい値になる場合は，そのファイルでのその単語の出現割合が，他のファイルと比べて大きいことを意味する．どのくらいの値であれば「不自然」と言ってよいだろうか．仮説検定の言葉を使うと以下のようになる．ある単語について，あるファイルに含まれる割合が他のファイルより大きくないという仮説 H を立てる．計算の単純さのため，有意水準 2.5% とする．確率変数 Z が標準正規分布に従うとき，

$$P(Z > z) = 0.025$$

となる z は約 1.96 であることが統計学においてよく知られている．例えば標準正規分布表を見るか，R 言

語などで確認することができる．そこで，

$$z_i > 1.96 \Rightarrow \text{帰無仮説 } H \text{ を棄却する．}$$

というルールを定める．この帰無仮説 H を棄却した単語を，そのファイルの特徴的な単語として取りだし，クラスタリングに用いる．

(1)(2)(3) の手法どれに対しても以下の操作を行う．特徴量が大きいほど単語は特徴的であるといえる．そこで各ファイルで特徴量の高い上位 100 単語ずつを取り出す．同じ単語が複数回入らないように処理を行う．20 本の論文でクラスタリングするので最大で 2000 単語取り出すことになる．クラスタリングする前に上記の単語以外は削除する．削除することによって特徴のない単語が除外できる．

4.1.4　クラスタリングと特徴量の抽出

クラスタリングに用いた論文は以下である．代数トポロジーの研究を主に行っている A さんの論文を 5 本 (A(1)，A(2)，A(3)，A(4)，A(5))，力学系の研究を主に行っている B さんの論文を 5 本 (B(1)，B(2)，B(3)，B(4)，B(5))，シンプレクティック幾何の研究を主に行っている 1 人の論文を 5 本 (C(1)，C(2)，C(3)，C(4)，C(5))，ランダムに 5 本 (その他 (1)，その他 (2)，その他 (3)，その他 (4)，その他 (5)) の論文の合計 20 本の論文である．ここで，A，B，C の研究者の研究分野は，arXiv への投稿の分類から判断した．

各クラスタリングで特徴的な単語を取り出す．クラスタ内で特徴量の平均をとり，その値の大きい単語 10 単語とした．

階層的クラスタリングでは各ファイルの関係性を見るために用いた．距離の取り方はユークリッド距離，クラスタ間の距離は平均値で取った．k 平均法とできるだけ距離の測り方を同じにしたものを使用している．

4.2　実験結果

4.2.1　特徴量 (1)$p_i - p$ でのクラスタリング結果

特徴量 (1)$p_i - p$ を用いた k 平均法によるクラスタリングの結果は以下である．

クラスタ 1 C さん (3)

クラスタ 2 A さん (1)，A さん (2)，A さん (3)，A さん (4)，A さん (5) B さん (1)，B さん (2)，B さん (4)，B さん (5)，C さん (1)，C さん (2)，C さん (4)，C さん (5)，その他 (1)，その他 (2)，その他 (4)，その他 (5)

クラスタ 3 その他 (3)

クラスタ 4 B さん (3)

クラスタごとの特徴的な単語

クラスタ 1

spc，category，support，tensor，balmer，theorem，supp，spectrum，noetherian，supph

クラスタ 2

the，spin，orbit，periodic，homology，equations，that，equation，data，problem

クラスタ 3

the，and，floer，hamiltonian，pseudo，periodic，product，this，non，see

クラスタ 4

learning，training，task，tasks，state，latent，meta，agent，control，reward

特徴量 (1)$p_i - p$ を用いた階層的クラスタリングの結果は，図 6 である．

4.2.2 特徴量 (2)p_i/p でのクラスタリング結果

特徴量 (2)p_i/p を用いた k 平均法によるクラスタリングの結果は以下である．

クラスタ 1 A さん (2)，A さん (3)，A さん (4)，A さん (5)

クラスタ 2 A さん (1)，B さん (1)，B さん (2)，B さん (3)，B さん (4)，B さん (5)，C さん (5)，その他 (2) B さん (1)，B さん (2)，B さん (4)，B さん (5)，C さん (1)，C さん (2)，C さん (4)，C さん (5)，その他 (1)，その他 (2)，その他 (4)，その他 (5)

クラスタ 3 C さん (1)，C さん (2)，C さん (3)，C さん (4)

クラスタ 4 その他 (1)，その他 (3)，その他 (4)，その他 (5)

クラスタごとの特徴的な単語

クラスタ 1

ψ eq, ceq, athens, hqs, hutchings, outlining, turkish, voisinage, yoccoz, gysin

クラスタ 2

esis, modk, impli, indistinguishability, nerves, kac, komolgorov, moody, mhs, trian

クラスタ 3

deursen, tended, taxi, taiji, synnaeve, swarms, suzuki, subfigure, stoustrup, starcraft

クラスタ 4

pepx, γ iq, complexification, mp θ qw, complexify, complexification, clhb, ck ε, ciocci, cides

特徴量 (1)$p_i - p$，特徴量 (2)p_i/p を用いた階層的クラスタリングの結果は以下である．

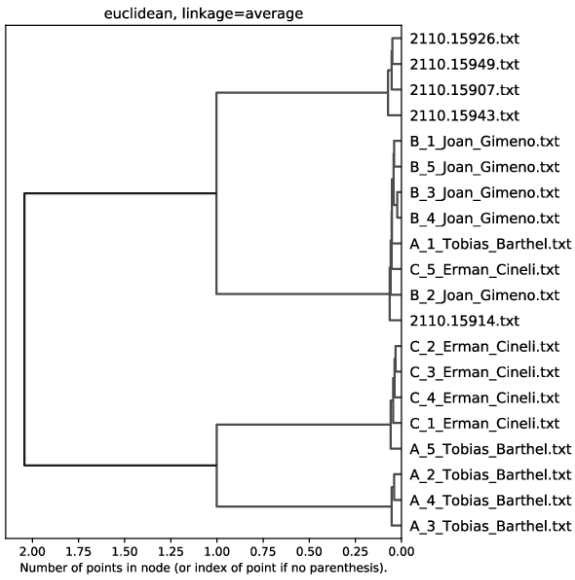

図 6　$p_i - p$ の結果

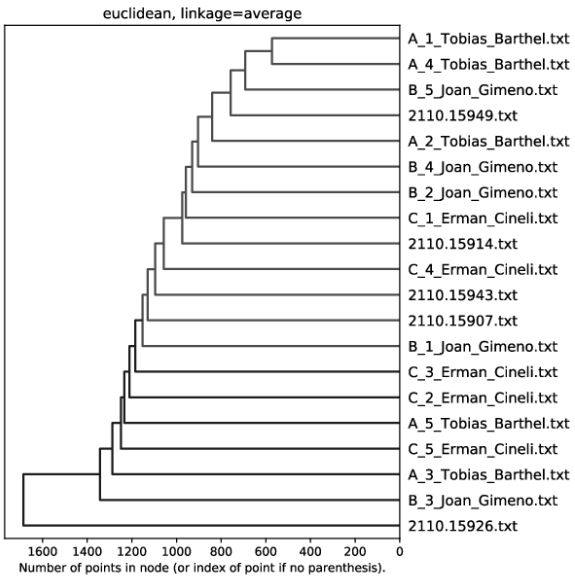

図 7　p_i/p の結果

4.2.3　特徴量 (3)z 値でのクラスタリング結果

特徴量 (3)z 値を用いた k 平均法によるクラスタリングの結果は以下である．

クラスタ 1 A さん (2)，A さん (3)，A さん (4)，A さん (5)

クラスタ 2 C さん (1)，C さん (2)，C さん (3)，C さん (4)，C さん (5)

クラスタ 3 A さん (1)

クラスタ 4 B さん (1)，B さん (2)，B さん (3)，B さん (4)，B さん (5)，
その他 (1)，その他 (2)，その他 (3)，その他 (4)，その他 (5)

クラスタごとの特徴的な単語

クラスタ 1

spc, balmer, tensor,　supp, category, supph, support, noetherian, spectrum, loc

クラスタ 2

floer,hamiltonian, homology, pseudo, degenerate, rotations, equivariant, ginzburg, quantum, symplectic

クラスタ 3

adams, spectral,comodules , morava, milnor, sequence, ext, cts, lim, filtration

クラスタ 4

spin, orbit, periodic, kam, gatelrd, meta, ϕ p θ, llave, gimeno, celletti

特徴量 (3)$_z$ 値を用いた階層的クラスタリングの結果は以下である．

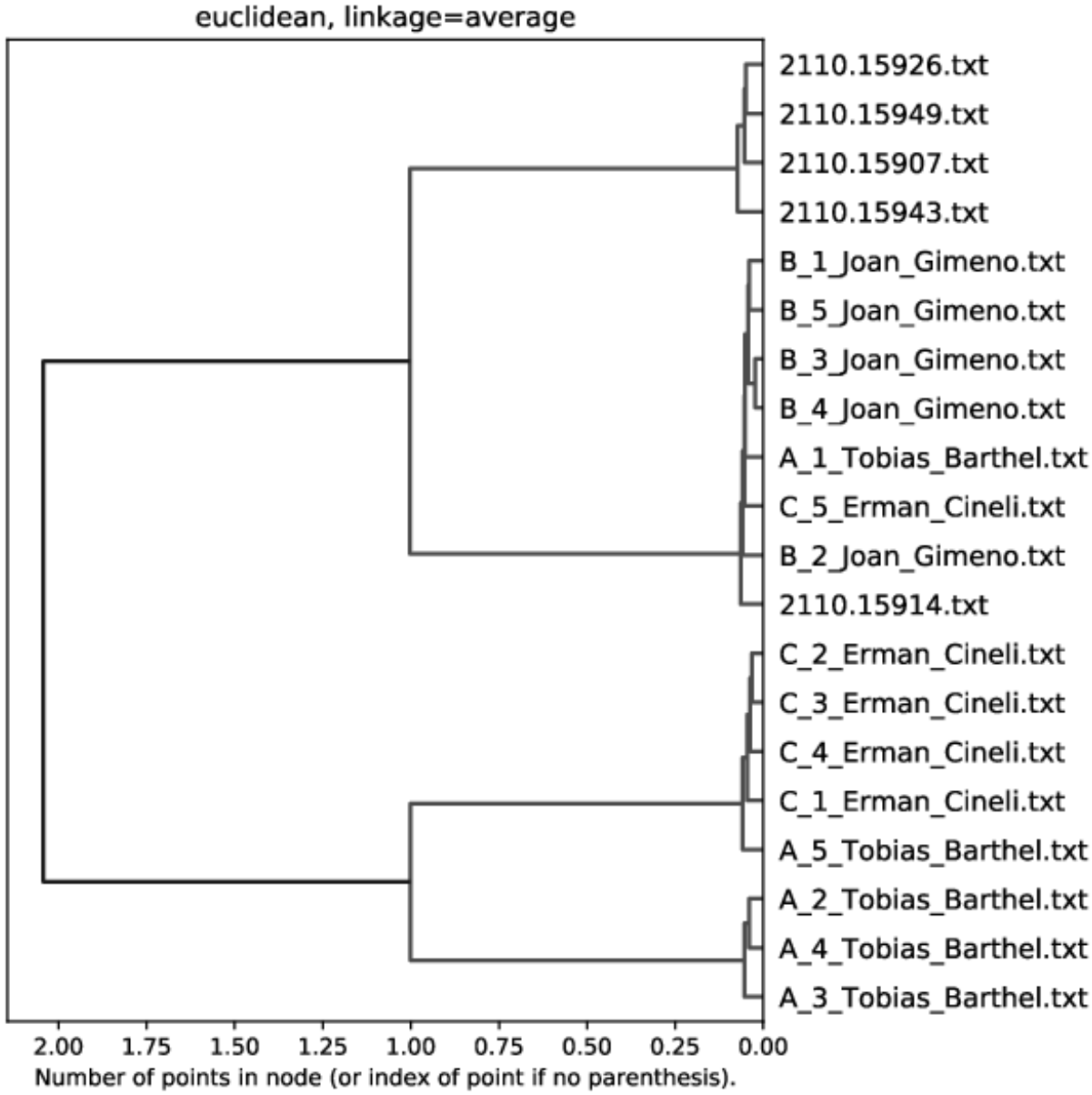
euclidean, linkage=average
2110.15926.txt
2110.15949.txt
2110.15907.txt
2110.15943.txt
B_1_Joan_Gimeno.txt
B_5_Joan_Gimeno.txt
B_3_Joan_Gimeno.txt
B_4_Joan_Gimeno.txt
A_1_Tobias_Barthel.txt
C_5_Erman_Cineli.txt
B_2_Joan_Gimeno.txt
2110.15914.txt
C_2_Erman_Cineli.txt
C_3_Erman_Cineli.txt
C_4_Erman_Cineli.txt
C_1_Erman_Cineli.txt
A_5_Tobias_Barthel.txt
A_2_Tobias_Barthel.txt
A_4_Tobias_Barthel.txt
A_3_Tobias_Barthel.txt
2.00 1.75 1.50 1.25 1.00 0.75 0.50 0.25 0.00
Number of points in node (or index of point if no parenthesis).

図 8　z 値の結果

4.2.4 考察

(1)$p_i - p$，(2)p_i/p のどちらの特徴量も全く別の分野である分野の論文が同じクラスタに集中してしまっている．そのためこれらの特徴量でクラスタリングをしてもよい結果ではなかった．それに対し，(3)z 値は，それぞれの著者の論文が同じクラスタに入っている．また，別の分野の研究をしている著者の論文同士が同じクラスタに入っていない．よって，z 値ならば，より良いクラスタリングできている．

z 値で特徴的な単語を取り出した．特徴量もどのような分野を集めたクラスタなのかわかるように抽出できた．しかし、いくつか特徴のない単語が含まれている．

5 実験 2

クラスタリングをする際にデータフレームを作る．インデックスが単語，値が z 値のデータフレームそのままではインデックスの次元が大きい．そこで，次元を減らしたいのだが，その次元の減らし方，インデックスである単語の選び方を実験していく．

5.1 実験手法

5.1.1 実験概要

実験 2 では，母集団の推定に 1 万本の論文を用いた．クラスタリングを行う論文は，3 つの分野からそれぞれ 10 本ずつの論文を選んだ．分野は固定するが，著者が同じとは限らない．それぞれの分野の論文が同じクラスタリングに入り，別の分野の研究をしている著者の論文同士が同じクラスタリングに入らなければ良い結果が得られたこととする．

今回，可換環・微分幾何・微分方程式の 3 つの分野で行う．実験 1 と比べ分野の近い者同士でクラスタリングをする．そのため，k 平均法におけるクラスタの数を $k = 5$ と設定した．

5.1.2 アルゴリズム

アルゴリズムの概要は以下である．実験 1 の処理では，各データの必要メモリ量が多く処理ができないので変更を加えた．基本は実験 1 と同様である．変更点について説明する．

実験 1 の手順の中で Step4. では，特徴量を計算するのみだった．実験 2 では Step 4.1. で，母集団の推定，Step 4.2. で特徴量 z の計算とクラスタリングに用いる単語の選択をする．以下の手順でクラスタごとの特徴的な単語を得る．

変更のあったステップの詳細を見ていく．

Step 2. txt ファイルを読み込んで，単語ごとに分ける．

実験 1 では文字が 2 文字以下の単語を除いていた．この操作を Step 4.1. 母集団の推定で行った．

Step 4.1. 母集団の推定をする．すべてのファイルの出現回数を出す．

まず，すべてのファイルで出てくる単語のリスト作る．ここで，3 文字以下の単語には意味を持っているものが少ないため，実験 2 の処理には加えないこととした．

次に，インデックスが上記の単語のリストであるデータフレームを作る．一つ一つのファイルデータを読み込んで足していく形で全ファイルでの単語の出現回数を数える．全単語の出現回数を出して処理を終了する．

Step 4.2. 特徴量 z の計算とクラスタリングに用いる単語の選択をする．

クラスタリングに用いるデータフレームのインデックスである単語を選ぶ．クラスタリングを行うために単語の数から特徴量 z を計算する．一つの論文のみで出現回数が多い単語は他の論文との関連性が測れない．しかし，その一つの論文のその単語での z 値はとても大きくなる．そのため，クラスタリングで距離を測ると大きな影響がでる．外れ値の対策として最後に単語ごとに特徴量を正規化する．正規化ならば単語ごとの論文同士の特徴の差がでる．一方単語ごとの差によるクラスタリングへの影響は抑えることができる．

母集団の最大値，最小値がそれぞれ $x_{\max}$，$x_{\min}$ である．変数 X の正規化した値 Y は次のように定義される．

$$Y = \frac{X - x_{\min}}{x_{\max} - x_{\min}}$$

5.1.3 クラスタリング対象の単語

クラスタリングをするデータを作る．インデックスが単語，値が z 値のデータフレームを作るがそのままではインデックスの次元が大きいので減らす．

全論文での出現回数がとても少ない単語を除くために，全ファイルでの出現回数の 100 以下の単語は除外する．the など全論文での出現が多い単語を除くために，全ファイルでの出現回数の上位 100 単語を除く．

さらに以下の候補を使って単語を選ぶ．

(1) クラスタリングの対象それぞれの z 値の上位 10 単語で選ぶ．
(2) クラスタリングの対象全体の z 値で選ぶ．
(3) 事前に学習する．

各候補の操作は以下である．

(1) クラスタリングの対象それぞれの z 値の上位 10 単語で選ぶ.

クラスタリングしたいデータの z 値の上位 10 単語ずつを取り出す．実験 1 では，母集団の推定をするために用いたデータすべての上位 10 単語ずつを取り出していた.

(2) クラスタリングの対象全体の z 値で選ぶ.

クラスタリングしたいデータ全体での単語の z 値を出し，その上位 100 単語を選ぶ.

(1)，(2) では十分な結果が得られなかった．詳しくは，実験結果で述べる.

(3) 事前に学習する.

教師として各分野 10 本の論文からクラスタリングに用いる単語を学習する．この 10 本の論文はクラスタリングの対象となる論文とは別である．学習するデータの z 値を出す．各単語でこの 10 個のデータの中央値を出す．中央値ならば外れ値の影響を受けずにデータの特徴を見ることができる．この中央値の値が 1.96 以上の単語を選ぶ．これを各分野で行い，一つにまとめた単語のリストをクラスタリングする際に用いる単語とする.

5.2 実験結果

5.2.1 クラスタリング結果

5.2.2 (1) クラスタリングの対象それぞれの z 値の上位 10 単語で選んだクラスタリング結果

(1) クラスタリングの対象それぞれの z 値の上位 10 単語で選んだクラスタリングの結果は以下である.

クラスタ 1 DG，DG，DG，DG，
クラスタ 2 DG，DG
クラスタ 3 RA，RA，RA，RA，RA，DG，DG，DG，DG，AP，AP，AP，AP，AP，AP
クラスタ 4 AP，AP
クラスタ 5 RA，RA，RA，RA，RA，AP，AP

クラスタごとの特徴的な単語

クラスタ 1

ahler, curvature, conformally, ricci, robin, metric, graham, manifolds, properness, manifold

クラスタ 2

discrete, algebroid, groupoids, mart, groupoid, iglesias, lagrangian, diego, algebroids, unconstrained

クラスタ 3

epimorphisms, subalgebra, dxdt, homological, algebras, laminations, algebra, subalgebras, extremal, nilpotent

クラスタ 4

korteweg, vera, vries, craig, regularity, kenig, dispersive, posedness, analyticity, kato

クラスタ 5

permitting, indecomposable, nondegenerate, kukl, azumaya, algebra, division, quaternion, jordan, composition

(1) クラスタリングの対象それぞれの z 値の上位 10 単語で選んだ階層的クラスタリングの結果は図 9 である.

5.2.3 (2) クラスタリングの対象の z 値で選んだクラスタリング結果

(2) クラスタリングの対象の z 値で選んだクラスタリングの結果は以下である.

クラスタ 1 RA, RA, RA, RA, RA, DG, DG, DG, DG, DG, DG, DG, DG, AP, AP, AP, AP, AP, AP, AP, AP, AP, AP

クラスタ 2 RA, RA

クラスタ 3 DG

クラスタ 4 RA, RA, RA

クラスタ 5 DG

クラスタごとの特徴的な単語

クラスタ 1

novikov, rhomr, kukl, koszul, korteweg, vera, finsler, solvable, vries, cahn

クラスタ 2

permitting, multiplicative, nondegenerate, indecomposable, composition, jordan, strongly, forms, absolutely, cubic

クラスタ 3

lamination, laminations, spacetimes, earthquake, earthquakes, francesco, geodesic, hyperbolic, spacelike, wick

クラスタ 4

division, crossed, superalgebra, severi, azumaya, etale, galois, brauer, quaternion, indecomposable

クラスタ 5

discrete, algebroid, mart, iglesias, groupoid, lagrangian, diego, groupoids, vector, nondegenerate

(1) クラスタリングの対象それぞれの z 値の上位 10 単語，(2) クラスタリングの対象 z 値で選んだ階層的クラスタリングの結果は以下である．

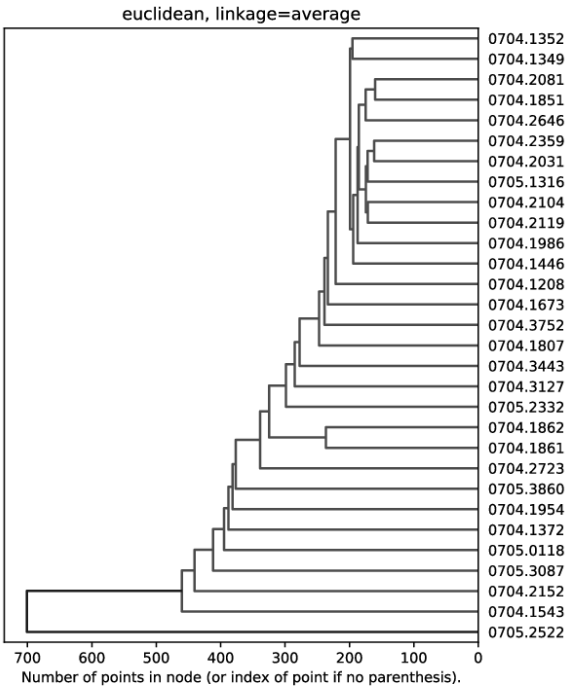

図 9　(1) クラスタリングの対象それぞれの z 値の上位 10 単語の結果

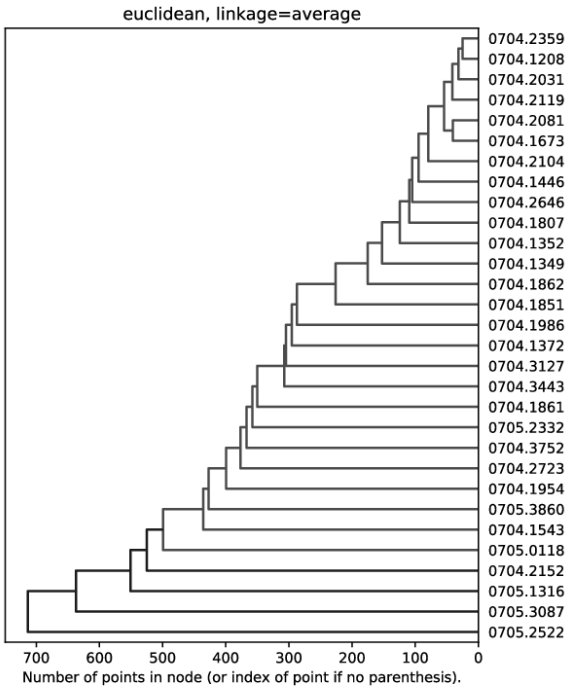

図 10　(2) クラスタリングの対象の z 値で選んだの結果

5.2.4　(3) 事前に学習するのクラスタリング結果

(3) 事前に学習するの k 平均法によるクラスタリングの結果は以下である.

クラスタ 1 DG，DG，DG，DG，DG

クラスタ 2 RA，RA，RA，RA，RA，RA，RA，RA，RA，RA，AP

クラスタ 3 AP，AP，AP，AP，AP，AP，AP，AP

クラスタ 4 DG，DG，DG，DG，AP

クラスタ 5 DG

クラスタ 1 には微分幾何，2 には可換環，3 には微分方程式の論文が集まっている.

クラスタごとの特徴的な単語

クラスタ 1

curvature, ricci, metric, riemannian, manifold, manifolds, metrics, compact, math, proved

クラスタ 2

algebra, algebras, associative, ideal, generated, exists, finitely, commutative, every, rings

クラスタ 3

estimates, regularity, estimate, satisfies, solutions, obtain, solution, inequality, anal, cauchy

クラスタ 4

vector, proposition, manifold, riemannian, curvature, compact, exists, moreover, spaces, problem

クラスタ 5

every, cauchy, compact, finite, metric, proposition, gradient, denote, curvature, remark

クラスタ 1 の単語の中で curvature：曲率，riemannian：リーマン計量，リーマン多様体などとして用いられる，は微分幾何の分野でよく見られる．クラスタ 2 の単語の中で，finitely：有限の，generated：生成される，algebra, algebras：環 $+\alpha$ という意味合いの代数構造，ring, rings：環，ideal, ideals：イデアルは代数学の分野でよく見られる．クラスタ 3 の estimate，estimates：解を近似的に見積もる，solution，solutios：微分方程式の解は微分方程式の分野でよく見られる．

(3) 事前に学習するの階層的クラスタリングの結果は以下である．

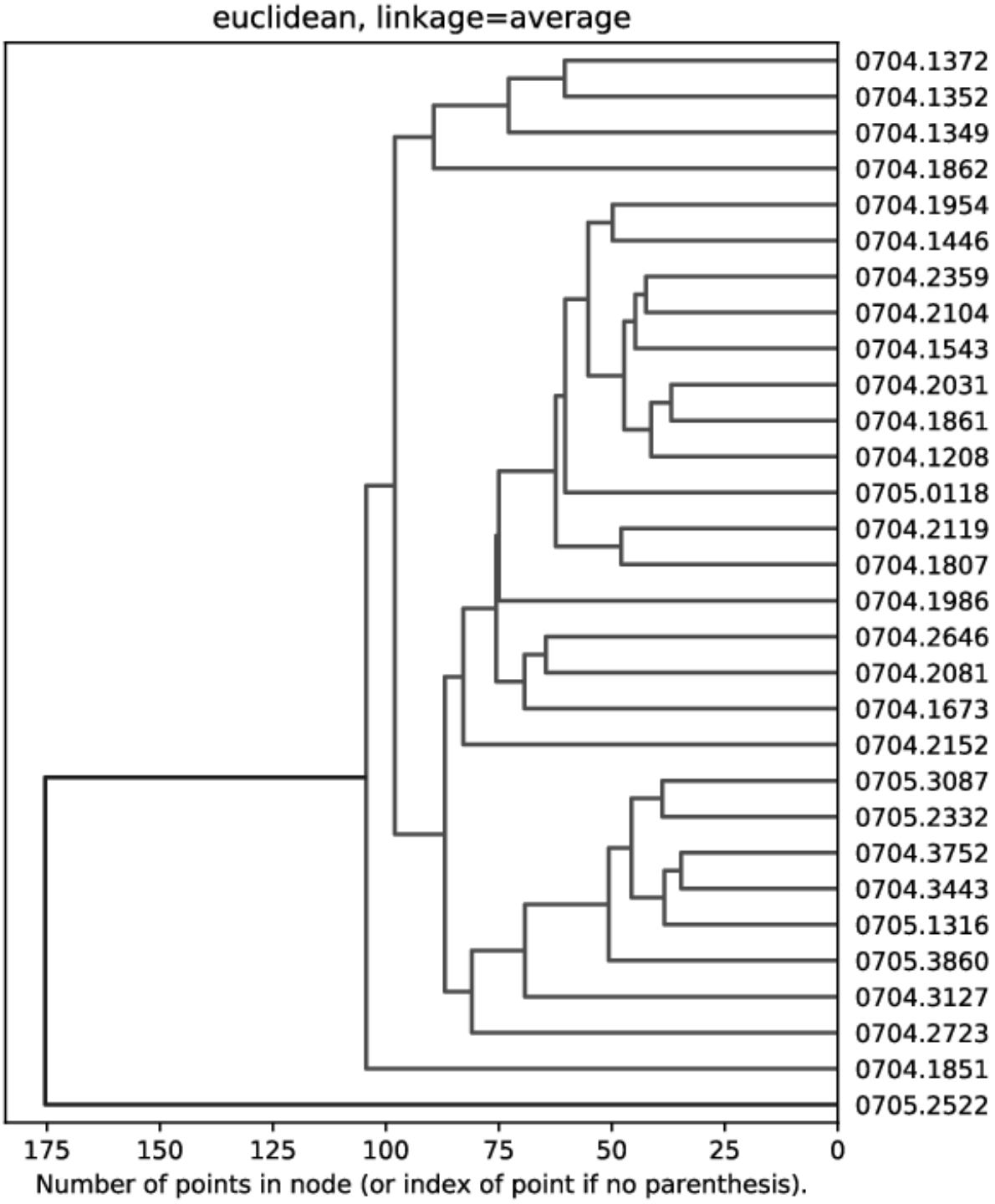

図 11　(3) 事前に学習するの結果果

5.2.5 考察

この実験では，(1) クラスタリングの対象それぞれの z 値の上位 10 単語で選ぶ，(2) クラスタリングの対象の z 値で選んぶでクラスタリングすると一つのクラスタに分野の論文が集中してしまっている．そのためこれらの単語の選び方でのクラスタリングはよい結果ではなかった．クラスタリングに用いたデータを眺めると各単語で一つの論文のみ z 値が高い．そのため，(1)，(2) ではよい結果がでない．(3) 事前に学習するでは，それぞれの分野の論文が同じクラスタに入っている．また，別の分野の研究の論文が最大 1 つのみ入っている．(3) 事前に学習するならば，クラスタリングできている．

各クラスタリングで特徴的な単語を取り出した．特徴的な単語もどのような分野を集めたクラスタなのかわかるように抽出できた．実験 2 では，特徴のない単語を取り除くことができた．(3) 事前に学習するでのクラスタリングされた論文の分野と特徴的な単語からわかる分野が一致した．

単語の選び方によって結果が大きく変化した．クラスタリングをする際に用いる単語の選び方が重要である．

実験 1 でクラスタリングの結果がよかった理由は，” 著者’’ の癖が出ていたのではないか．

6 まとめ

6.1 考察

引用されてはいないが関連性の高い主題を扱っているかどうかを見ることができる．単語で検索をした際，z 値の高いものから表示することで関連性の高いものをより見つけることができる．今まであまり交わりのなかった分野のつながりを見出すことができる．機械学習における特徴量抽出においても，統計学の仮説検定で使われる z 値が有用であることが確認できた．

6.2 課題

今後の課題として以下があげられる．

(1) 分析に用いるファイルの数を増やす．
(2) あらかじめ分野のわかっている論文からクラスタリングに用いる単語を学習した．新しい分野に対応できないなどの汎用性が下がる．
(3) クラスタリングに用いる単語を適切に取り出し方を工夫する．
(4) z 値の基準には何が適切か測る．

(5) 英語で書かれた文にのみ対応しているので，多言語に対応できるようにする．

(6) さらに細かくクラスタリングをした際に精度を保てるようにする．

(7) 単語の単数形・複数形をまとめられないか．

(8) クラスタごとに特徴的な単語を取り出す．

(9) 大量のデータを扱うことによるメモリ不足の問題をどう解決するか．

(1) 分析に用いるファイルの数を増やす．

今回，1 万本の論文で母集団を推定して 30 本の論文をクラスタリングした．実用できるようにクラスタリングする論文数を増やしたい．

(2) あらかじめ分野のわかっている論文からクラスタリングに用いる単語を学習した．新しい分野に対応できないなどの汎用性が下がる．

今回の後半の実験では，あらかじめ分野が判明している論文から特徴的な単語を学習し，クラスタリングしている．新しい分野ができた場合，新しい単語ができたとしてもその単語は特徴として捉えない．通常ではつながりが見えない分野同士の発見ができなくなる．その対策として交差検証の手法を応用することが考えられる．

(3) クラスタリングに用いる単語を適切に取り出し方を工夫する．

クラスタリングに用いる単語の選び方によって結果に大きな影響を与えることが分かった．どのように単語を選ぶかが一番重要である．

(4) z 値の基準には何が適切か測る．

クラスタリングに用いるデータを選ぶ際，クラスタリングに用いる各ファイルの z 値の上位 100 個ずつを取り出した．これは適当に選んだものであるので，何個ずつが妥当なのかもしくは z 値の値がある値以上であるか調べる．

(5) 英語で書かれた文にのみ対応しているので，多言語に対応できるようにする．

何かしらの方法で英語に翻訳して統一することを考えている．

(6) さらに細かくクラスタリングをした際に精度を保てるようにする．

今回は，代数・幾何・解析の分野でクラスタリングを行った．さらに細かく分けていきたい．

(7) 単語の単数形・複数形をまとめられないか．

似たような出現頻度になることが予想されるため単語の単数形・複数形をまとめたい．実際，クラスタリングの特徴的な単語の結果に単数形・複数形どちらも出ている．そのために，形態解析を行うとよいのではと考えている．

(8) クラスタごとに特徴的な単語を取り出す．

さらにより意味の分かる，この分野だとわかる単語を取り出せるようにしたい．意味が伝わらない，余分な単語をとりのぞくようにする．

(9) 大量のデータを扱うことによるメモリ不足の問題をどう解決するか．

今回は最大 1 万本の論文のデータを扱ったがメモリ不足の問題が発生した．arXiv に挙げられている論文数は約 200 万である．本来はこの約 200 万本の論文でクラスタリングしたいのでメモリ不足の問題をどのように解決するか，クラスタリングの仕方・表示の仕方を工夫する．

謝辞

最後に，貴重な指導と助言を賜った指導教官の宮部賢志准教授に感謝いたします．5.2.4 節のクラスタごとの特徴的な単語に，各専門分野からのコメントを野原雄一准教授，松岡直之准教授からいただきました．また，研究を通じて議論にお付き合い頂いた宮部ゼミの皆様に感謝いたします．

参考文献

[1] Andriy Burkov. 機械学習 100+ ページエッセンス. インプレス, 2019. 清水美樹訳.

[2] Alex Tarnavsky Eitan, Eddie Smolyansky, Itay Knaan Harpaz, Sahar Perets. Connected ppers, 2020. https://www.connectedpapers.com/.

[3] 藤澤洋徳. 現代基礎数学 13 確率と統計. 共立出版, 2006.

[4] 鈴木譲. 機械学習の数理 100 問シリーズ 1 統計的機械学習の数理 100 問 with R. 共立出版, 2020.

付録 A　付録

A.1　ソースコード

本実験でプログラミングしたソースコードをリスト 2 に記す．使用言語は Python，Google Colaboratory にて作成．

リスト 1　ファイル操作に関するモジュール

```
import os

#read_file_names_list (str,str)-> str_lst
#("入力の場所(フォルダまでのパス)")->"入力の場所"にあるファイル名(拡張子は除く)を入れた
     string型のlistを返す
#
#"入力の場所"はあるフォルダへのパス.
#フォルダの中にあるファイルの名前を読み込む.
#ファイルの名前をリストに入れていく.
#ファイルの名前を入れたリストを出力として返す.
#
#例
#("/folder" , '.pdf')->["file_1","file_2","file_3"]
#
# read_file_names_list(input_path)
#
#param string input_path 読み込みファイルが存在するフォルダパス
#param string file_target 読み込みたいファイルの拡張子(1つのみ)
#
#return string[] 分割されたファイル名のリスト

def read_file_names_list(input_path, ext, f):
  return([f(fn) for fn in os.listdir(input_path) if (os.path.isfile(os.path.join(input_path, fn)) & (os.
       path.splitext(fn)[1] == ext))])

#txtファイルを読み込んでlistで出力する.
#read_file_to_list(str)-> list
#("ファイルAのパス")->"ファイルA"を読み込んで'\n'ごとで区切り要素としたstring listを出力する.
#
#例
#list.txtの中身
#'apple'
#'banana'
#
#('./list/txt')->['apple', 'banana']
#
# read_file_to_list(input_file_path)
#
#param string input_file_path 読み込みたいファイルのpath
#
#return list listが出力される
#            改行の'\n'は除いている.


def read_file_to_list(file_path):
  with open(file_path) as f:
    list_strip = [s.strip() for s in f.readlines()]
  return(list_strip)


def write_file_from_list(file_path, list):
  with open(file_path, mode='w') as f:
    for s in list:
      f.write(s + "\n")
```

リスト 2　pdf ファイルを txt ファイルへ変換する

```
#マウント
from google.colab import drive
```

```
drive.mount('/content/drive')

##作業するディレクトリを指定する.
# %cd '/content/drive/MyDrive/Colab_Notebooks/arXiv_data'

#関係するツールの導入

#pdfをtxtファイルに変換する際に使用
from bs4 import BeautifulSoup
import urllib
import urllib.request as req

import time
import requests
import os
import numpy as np
import re

#input_folder_path: 入力となるファイルの保存場所
input_folder_path ='/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_1'
#output_folder_path: 出力となるpdfファイルの保存場所
output_folder_path = '/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_2'

#PDFをダウンロードするためにpdf2txtをインストールする.
!pip install PyPDF2

!pip install pdfminer.six

#pdf2txt.pyのダウンロード
py_url ='https://github.com/pdfminer/pdfminer.six/blob/master/tools/pdf2txt.py'
py_fn='pdf2txt.py'
os.system("wget -O " + str(py_fn) + " " + str(py_url))

module_path = '/content/drive/My Drive/Colab_Notebooks/arXiv_data/module/'
os.chdir(module_path)
!ls

#input_file_pathに存在するpdfファイルの名前一覧を作る.
import file_processing

file_names_list = file_processing.read_file_names_list( input_folder_path, '.pdf' , f = lambda x : os.
    path.splitext(x)[0] )

file_names_list

len(file_names_list)

#get_txt_from_pdf (str,str,str)->
#(入力の場所,出力の場所,ファイルの名前)->txtファイルを出力の場所に作る.
#
#
#('./folder_1', './folder_2', file_name )-> ['I'm a doctor.', 'My ... .']
#
# get_txt_from_pdf (import_folder_path,export_folder_path,file_name)
#
#param string input_file_path 読み込みたいファイルの入っているフォルダへのpath
#param string export_file_path 書き込んだファイルを保存するファイルへのpath
#param string file_name 実行対象となるファイルの名前
#
#return なし
#       export_folder_path に file_name + .txt で保存する.
```

```
#       import_pdfpath  のファイルを消去する.

def get_txt_from_pdf (import_folder_path,export_folder_path,file_name):
  import_pdfpath = import_folder_path + "/" + file_name + ".pdf"
  export_pdfpath = export_folder_path + "/"
  #pdf2txt.pyを用いてpdfファイルをテキスト変換
  lines = !pdf2txt.py { import_pdfpath }
  txt = '\n'.join(lines)
  rename = export_pdfpath + file_name + ".txt"
  er = file_processing.write_file_from_list( rename , txt)
  os.remove( import_pdfpath )
  return(er)

#get_txt_from_pdfをlistに対して繰り返し実行
list( map ( lambda x: get_txt_from_pdf(input_folder_path, output_folder_path, x), file_names_list))
```

リスト 3　txt ファイルに変換した文を単語ごとに分けて txt ファイルで保存する.

```
#マウント
from google.colab import drive
drive.mount('/content/drive')

##作業するディレクトリを指定する.
# %cd '/content/drive/MyDrive/Colab_Notebooks/arXiv_data'

#関係するツールの導入
import os
import numpy as np
import pandas as pd
import collections
import re
from string import digits

#input_folder_path: 入力となるファイルの保存場所
input_folder_path ='/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_2'
#output_folder_path: 出力となるpdfファイルの保存場所
output_folder_path = '/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_3'

#モジュールの導入
module_path = '/content/drive/My Drive/Colab_Notebooks/arXiv_data/module/'
os.chdir(module_path)
import file_processing

#input_file_pathに存在するtxtファイルの名前一覧を作る.
file_names_list = file_processing.read_file_names_list( input_folder_path, '.txt' , f = lambda x : os.
    path.splitext(x)[0] )

file_names_list

len(file_names_list)

#take_out_word (str, str)-> str_lst
#(入力の場所,ファイルの名前)->txtファイルを読み込んで単語に分ける.小文字にして,ピリオドなどは除く.
#doxygen文法
#
#指定テキストファイル内の英文を単語ごとの配列に整形します。
#主に半角スペース、コンマ、ピリオド、ハイフンで単語を分割します。
#また、数字の除去と大文字の小文字化も行っています。
#出力は文字列のリスト型で、順番は登場順になります。
#
#例
```

```
#"I'm a apple-pen."->["i","m","a","apple","pen"]
#
#param string path 読み込みファイルが存在するフォルダパス
#param string file_name 読み込むファイルの名前(拡張子込み)
#
#return string[] 分割された単語のリスト
#
#split_txt_to_words_list(read_folder_path,read_file_name)
def take_out_word (path,file_name):
  # split()でスペースと改行で分割したリストから単語を取り出す
  def word_list_of(string_file):
    #小文字にしてから英単語
    table = str.maketrans('', '', digits)
    string_file = string_file.translate(table)
    string_file =string_file.lower()
    string_beta = re.findall("\w+",string_file)
    return(string_beta)
  file_path = path + "/" + file_name + ".txt"
  with open(file_path) as f:
    r = f.read()
    s = word_list_of(r)
  return s

#get_lst_of_word (str,str,str)->
#(入力の場所,出力の場所,ファイルの名前)->take_out_wordを実行して出力の場所にtxtファイルで保存する.
#
#~~~
#内部でtext_out_word()を用いています
#
#
def get_lst_of_word (input_folder_path, output_folder_path, file_name):
  ##pdfpath_1 = path_1 + "/" + file_name
  lst = take_out_word(input_folder_path, file_name)
  output_file_path = output_folder_path + "/" + file_name + ".txt"
  file_processing.write_file_from_list(output_file_path , lst)
  os.remove(input_folder_path + "/" + file_name + ".txt" )

#get_lst_of_wordをlstに対して繰り返し実行する.
txt_data =list(map( lambda x : get_lst_of_word(input_folder_path, output_folder_path, x), file_names_list
    ))
```

リスト 4　ファイルごとに単語が何回出てくるのか数えて csv ファイルに保存する.

```
#マウント
from google.colab import drive
drive.mount('/content/drive')
# %cd "drive/MyDrive"

##作業するディレクトリを指定する.
# %cd '/content/drive/MyDrive/Colab_Notebooks/arXiv_data'

#関係するツールの導入
import os
import numpy as np
import pandas as pd
import collections
import re
from string import digits

#input_folder_path: 入力となるファイルの保存場所
input_folder_path ='/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_3'
```

```
#output_folder_path: 出力となるpdfファイルの保存場所
output_folder_path = '/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_4'

#モジュールの導入
module_path = '/content/drive/My Drive/Colab_Notebooks/arXiv_data/module/'
os.chdir(module_path)
import file_processing

#input_file_pathに存在するtxtファイルの名前一覧を作る.
file_names_list = file_processing.read_file_names_list( input_folder_path, '.txt' , f = lambda x : os.
    path.splitext(x)[0] )

len(file_names_list)

from pandas.core.frame import DataFrame
#read_lst (str,str)->dct
#(入力の場所,ファイルの名前)->単語を数えてdct型にする(key,value)->(str,int)
def read_lst (input_folder_path, output_folder_path, file_name):
  file_path = input_folder_path + "/" + file_name + ".txt"
  list_row = file_processing.read_file_to_list(file_path)
  word_lst = collections.Counter(list_row)
  df = pd.DataFrame( word_lst.values(), index = word_lst.keys()).T
  df.to_csv(output_folder_path + '/' + file_name + '.csv', sep=",")
  os.remove(file_path)

#read_lstをlstに対して繰り返し実行する.
txt_data =list(map( lambda x : read_lst(input_folder_path, output_folder_path, x), file_names_list))
```

リスト 5　母集団の推定をする．すべてのファイルの単語の出現回数をだし，csv ファイルに保存する．

```
#マウント
from google.colab import drive
drive.mount('/content/drive')

##作業するディレクトリを指定する.
%cd '/content/drive/MyDrive/Colab_Notebooks/arXiv_data'

#関係するツールの導入
import os
import numpy as np
import pandas as pd
import collections
import re
from string import digits

import sys
import time

#input_folder_path: 入力となるファイルの保存場所
input_folder_path ='/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_4'
#output_folder_path: 出力となるpdfファイルの保存場所
output_folder_path = '/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_5'

#モジュール
module_path = '/content/drive/My Drive/Colab_Notebooks/arXiv_data/module/'
os.chdir(module_path)
import file_processing

#input_file_pathに存在するcsvファイルの名前一覧を作る.
file_names_list = file_processing.read_file_names_list( input_folder_path, '.csv' , f = lambda x : os.
    path.splitext(x)[0] )
```

```
file_names_list.sort(reverse=False)

#保存しておく.
file_processing.write_file_from_list(output_folder_path + "/" + '/file_names_list_10000.txt',
     file_names_list)
#file_names_list = file_processing.read_file_to_list(output_folder_path + "/" + 'file_names_list_10000.
     txt')

len(file_names_list)

#すべてのファイルで出現する単語のリストを作る.
start = time.perf_counter()
df_dict = []
for i in range(len(file_names_list)):
  input_file_path = input_folder_path + '/' + file_names_list[i] + '.csv'
  d = pd.read_csv(input_file_path)
  d = d.drop('Unnamed: 0', axis=1)
  d = list(d.columns.values)
  df_dict = df_dict+ d
df_dict = list(set(df_dict))
print(time.perf_counter() - start)
print(sys.getsizeof(df_dict)/(1024**2))

print(len(df_dict))

#file_processing.write_file_from_list(output_folder_path + "/" + 'df_dict.txt', df_dict)
#df_dict = file_processing.read_file_to_list(output_folder_path + "/" + 'df_dict.txt')

#出現する単語のリストで3文字以下の単語のみを減らす.
df_dict_3 = [ l for l in df_dict if  len( l ) > 3]

#file_processing.write_file_from_list(output_folder_path + "/" + 'df_dict_3.txt', df_dict_3 )
#df_dict_3 = file_processing.read_file_to_list(output_folder_path + "/" + 'df_dict_3.txt')

#すべてのファイルで出現する単語数を数える.
start = time.perf_counter()

file_names_list_number = len( file_names_list)
ar = np.zeros((2, len(df_dict_3) ))
df = pd.DataFrame(ar, columns = df_dict_3 )
df.insert(0, '全単語数',df.sum(axis=1))

for i in range( file_names_list_number ):
  input_file_path = input_folder_path + '/' + file_names_list[i] + '.csv'
  d = pd.read_csv(input_file_path)
  d = d.drop('Unnamed: 0', axis=1)
  d.insert(0, '全単語数',d.sum(axis=1))
  df.iloc[0, : ] = d.iloc[0,:]
  df = df.fillna(0)
  sum_df = df.loc[0] + df.loc[1]
  df.iloc[1, : ] = sum_df

print(time.perf_counter() - start)
print(sys.getsizeof(df)/(1024**2))

(df.loc[[1]]).to_csv(output_folder_path + '/' +"df_dict_data_10000.csv", sep=",")
```

リスト 6　クラスタリングするファイルと単語を選びその特徴量の値を出す．そして，csv ファイルで保存する

```
#マウント
from google.colab import drive
```

```
drive.mount('/content/drive')

##作業するディレクトリを指定する.
%cd '/content/drive/MyDrive/Colab_Notebooks/arXiv_data'

#関係するツールの導入
import os
import numpy as np
import pandas as pd
import collections
import re
from string import digits

import sys
import time

#input_folder_path: 入力となるファイルの保存場所
input_folder_path ='/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_5'
#output_folder_path: 出力となるpdfファイルの保存場所
output_folder_path = '/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_6'

#モジュール
module_path = '/content/drive/My Drive/Colab_Notebooks/arXiv_data/module/'
os.chdir(module_path)
import file_processing

file_names_list = file_processing.read_file_to_list(input_folder_path + "/" + 'file_names_list_10000.txt'
    )

df = pd.read_csv(input_folder_path + "/df_dict_data_10000.csv")
df = df.drop('Unnamed: 0', axis=1)

df.head()

"""##クラスタリングする際に使用する単語を選ぶ"""

#make_dict_select (dataframe, int, int) -> string_list
#( 1*n のデータフレーム , int, int) -> '全体の単語数' + 条件を満たした単語のみを集めたリスト
#例
#( pd.DataFrame(
#    data={'全体の単語数': [10000],
#          'the': [50],
#          'apple': [10]
#           ... },
#), 30 ,5 ) -> [ '全体の単語数', all, ...., pull]
#
#
#param dataframe df 1*nのデータフレーム
#param int s 条件に用いる数字:
#            データフレームで値を大きい順に並べ替える.
#            上から数えてs個のindexは出力対象から外す.
#param int n 条件に用いる数字:
#            値がn以下のindexは出力対象から外す.
#
#return string[] '全単語数' と 選べれた単語のリスト
#

def make_dict_select (df, s, t):
  df_dic = df.drop('全単語数', axis=1)
  #出現する回数が1回以上の単語
  #出現する回数の多い順に並べてn位以降を選ぶ
```

```
  df_dic_select = pd.DataFrame(df_dic.loc[ 0, df_dic.iloc[0,:] >t].sort_values( ascending = False)[s:]).T
  df_dic_select.insert(0, '全単語数', 0)#df_dic_select.sum(axis=1))
  #上の処理を施したデータフレームからindex(単語)のみを取り出す.
  dict_index_select = list(df_dic_select.iloc[0].index)
  return(dict_index_select)

#dfの中で valueの上位 t より低いかつ valueが t = 100以上のindexを取り出す.
dict_index_select = make_dict_select(df ,100 , 100)

#file_processing.write_file_from_list(output_folder_path + "/" + 'dict_index_select_100_100.txt',
    dict_index_select)
#dict_index_select = file_processing.read_file_to_list(output_folder_path + "/" + '
    dict_index_select_100_100.txt')


"""##make_dict_selectによる制限だけでクラスタリングする場合"""
"""#クラスタリングに使うデータフレームを作る"""
#k_means_listにfile_names_listでの番号を入れる.
#直接 k_means_id_listにファイルの名前を入れてもよい.
k_means_list = []

k_means_id_list = []
for i in k_means_list :
  id_number = file_names_list[i]
  k_means_id_list.append( id_number )

#クラスタリングの対象となるファイルの名前の一覧を保存する.
#これ以降では順序の変更を加えない.
file_processing.write_file_from_list(output_folder_path + "/" + 'k_means_id_list_paper.txt',
    k_means_id_list)


#クラスタリングに用いるファイルの名前のリストから単語とその出現回数を保存したcsvファイルを読み込む.
#全体の単語数のデータフレームに付け加える形で一つのデータフレームとして作る.
for i in k_means_id_list:
   input_file_path = '/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_4' + '/' + i + '.csv'
   d = pd.read_csv(input_file_path)
   d = d.drop('Unnamed: 0', axis=1)
   d.insert(0, '全単語数',d.sum(axis=1))
   df = df.append(d,ignore_index=True)
df = df.fillna(0)
df = df.astype(int)

#クラスタリングに用いるデータフレームを作成する.
#make_dataframe_clustering (dataframe, string_list) -> dataframe
#
#param dataframe df n*mのデータフレーム
#param string_list dict_index_select indexにする要素の一覧
#
#return dataframe : dfに必要な情報を付け足したもの, indexをdict_index_selectのみに制限.
#
def make_dataframe_clustering (df, dict_index_select):
  #クラスタリングに用いるデータすべてで出現する単語の回数を求める.
  df.loc['k-means対象'] = df.loc[1:len(df)].sum(axis=0)
  #母集団の単語の出現頻度の推定値を出す.
  #全体での出現回数を全単語数でわる.
  df.loc['全体の頻度'] = df.loc[0]/df.loc[0, '全単語数']
  #dict_index_selectのみの単語でのデータフレームを取り出す.
  df_select = df.loc[:,dict_index_select]
  return(df_select)

```

```
#必要なデータ情報を作り，クラスタリングに用いる単語を制限する．
df_select = make_dataframe_clustering(df,dict_index_select)
#df_select.to_csv(output_folder_path + '/' +"df_10_select_100_100.csv", sep=",")

"""#z値を求める"""

#新しくデータフレームを作り，そこにz値を求めて入れる．
#make_z_value_dataframe (df, l) -> dataframe
#
#param dataframe df z値を求めたいデータフレーム
#param int l  z値を求めたいデータフレームの数
#
#return dataframe 求めたz値を入れたデータフレーム

def make_z_value_dataframe (df, len_data):
  df_z_value = pd.DataFrame({})
  #全体の頻度
  word_freqs = df.loc['全体の頻度']
  for i in range( len_data + 1):
    #単語の個数
    word_number_i = df.loc[i]
    #word_number_i =word_number_i.drop('全単語数')<-これをすると単語がランダム,不明になってしまう
    #ファイルの単語数
    numbers_file_word = df.loc[i,'全単語数']
    #z値
    z = (word_number_i - numbers_file_word * word_freqs)/np.sqrt(numbers_file_word*word_freqs*(1-
        word_freqs))
    z.name = i
    z = pd.DataFrame([z])
    df_z_value = pd.concat([df_z_value, z], axis=0)
  #単語の個数
  word_number_i = df.loc['k-means対象']
  #word_number_i =word_number_i.drop('全単語数')<-これをすると単語がランダム,不明になってしまう
  #ファイルの単語数
  numbers_file_word = df.loc[i,'全単語数']
  #z値
  z = (word_number_i - numbers_file_word * word_freqs)/np.sqrt(numbers_file_word*word_freqs*(1-word_freqs
      ))
  z.name = 'k-means対象'
  z = pd.DataFrame([z])
  df_z_value = pd.concat([df_z_value, z], axis=0)
  df_z_value_selected = df_z_value.drop('全単語数',axis = 1)
  return ( df_z_value_selected )

l = len(k_means_id_list)
df_z_value_selected = make_z_value_dataframe( df_select, l )

df_z_value_selected.to_csv(output_folder_path + '/' +"df_10_z_value_selected_100_100.csv", sep=",")

"""##各ファイルでz値の上位10単語ずつを取り出して，それでデータフレームを作る．"""

def make_df_top_n_index(df,n,file_names_list):
  top_n_index = df.loc[1].sort_values(ascending = False)[1:n+1].index
  for i in range(2,len(file_names_list)+1):
    top_n_index = top_n_index.append(df.loc[i].sort_values(ascending = False)[1:n+1].index)
  df_z_top_n = df.loc[:,top_n_index]
  df_z_top_n = df_z_top_n.loc[:,~df_z_top_n.columns.duplicated()]
  return( df_z_top_n )

df_z_top_n_100 = make_df_top_n_index(df_z_value_selected,10,k_means_id_list)

```

```
df_z_top_n_100[1: (len(k_means_id_list) +1 )].to_csv(output_folder_path + '/' +"rate_data_100.csv", sep="
    ,")

"""##さらに制限を加えてクラスタリングの対象のファイル全体でのz値上位n単語のみのデータフレームを作る"""

#nで上位何個の単語を用いるか指定する.
#クラスタリング対象すべてでの単語数のz値の上位100単語でデータフレームを作る.
n = 100
top_n_index = df_z_value_selected.loc['k-means対象'].sort_values(ascending = False)[1:n+1].index
df_z_top_n = df_z_value_selected.loc[:,top_n_index]
df_z_top_n = df_z_top_n.iloc[1:len(k_means_id_list)+1,~df_z_top_n.columns.duplicated()]

df_z_top_n.to_csv(output_folder_path + '/' +"rate_data_k_meansfiles__100_paper_10_100_100.csv", sep=",")


"""##先に各分野でクラスタリングに用いる単語を学習する. """

"""#各分野の先頭10個のファイルでz値の中央値を求めて大きい順に並べる. """
#準備
df = pd.read_csv(input_folder_path + "/df_dict_data_10000.csv")
df = df.drop('Unnamed: 0', axis=1)

#make_filed_dataframe (df, string_list, int):
#
#param df filed_df : 加工するデータフレーム
#param string_list dict_index_select : 事前に決めたクラスタリングに使用する単語のリスト
#param inr len_data : データの数, len(dict_index_select )
#
#return df : filed_df のデータからz値の中央値を出してvalueの値の大きい順に並び変えたデータ.

def make_filed_dataframe (filed_df,dict_index_select, len_data ):
  filed_df .loc['全体の頻度'] = filed_df .loc[0]/filed_df .loc[0, '全単語数']
  #dict_index_selectのみの単語でのデータフレームを取り出す.
  filed_df_select = filed_df.loc[:,dict_index_select]

  filed_df_z_value = pd.DataFrame({})
  #全体の頻度
  word_freqs = filed_df_select.loc['全体の頻度']
  for i in range( len_data + 1):
    #単語の個数
    word_number_i = filed_df_select.loc[i]
    #word_number_i =word_number_i.drop('全単語数')<-これをすると単語がランダム,不明になってしまう
    #ファイルの単語数
    numbers_file_word = filed_df_select.loc[i,'全単語数']
    #z値
    z = (word_number_i - numbers_file_word * word_freqs)/np.sqrt(numbers_file_word*word_freqs*(1-
        word_freqs))
    z.name = i
    z = pd.DataFrame([z])
    filed_df_z_value = pd.concat([filed_df_z_value, z], axis=0)

  filed_z_value_med = pd.DataFrame(filed_df_z_value.loc[1:len_data +1 ].apply(lambda x :np.percentile(x,
      50)) ).T
  filed_z_value_med = filed_z_value_med.drop('全単語数', axis=1)
  filed_z_value_med_sorted = filed_z_value_med.iloc[0].sort_values(ascending = False)
  filed_z_value_med_sorted = pd.DataFrame(filed_z_value_med_sorted).T
  return (filed_z_value_med_sorted )

#select_index_from_df (df, float) -> string_list
#
#param df
```

```
#param float t : 基準点
#
#return string_list : dfの0番目のvalueで tより大きいindexを返す.
def select_index_from_df (df, t):
  df_selected = pd.DataFrame(df.loc[ 0, df.iloc[0,:] >t]).T
  df_selected_index = list(df_selected.iloc[0].index)
  return(df_selected_index)

"""各分野から l=10本ずつ t=1.96以上のz値を取り出す"""

l = 10
t = 1.96

#各分野の特徴となる単語を取り出す.
"""#RA"""
#RAの学習用のデータのfile_names_list の要素番号を入れる.
RA_number =[]

RA_id_list = []
for i in RA_number:
  id_number = file_names_list[i]
  AP_id_list.append( id_number )

RA_df = df
#クラスタリングに用いるファイルの名前のリストから単語とその出現回数を保存したcsvファイルを読み込む.
#全体の単語数のデータフレームに付け加える形で一つのデータフレームとして作る.
for i in RA_id_list[0:10]:
   input_file_path = '/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_4' + '/' + i + '.csv'
   d = pd.read_csv(input_file_path)
   d = d.drop('Unnamed: 0', axis=1)
   d.insert(0, '全単語数',d.sum(axis=1))
   RA_df = RA_df.append(d,ignore_index=True)
RA_df = RA_df.fillna(0)
RA_df = RA_df.astype(int)

RA_z_value_selected = make_filed_dataframe(RA_df, dict_index_select, 10)
RA_selected_index = select_index_from_df(RA_z_value_selected, t)

RA_selected_index

#これをRA, DG, APで行う.

"""クラスタリングのためのindexを作る. """

df_index_from_filed = RA_selected_index + DG_selected_index + AP_selected_index

len(df_index_from_filed)
#file_processing.write_file_from_list(output_folder_path + "/" + 'df_index_from_filed.txt',
     df_index_from_filed)
#df_index_from_filed = file_processing.read_file_to_list(output_folder_path + "/" + 'df_index_from_filed.
     txt')

"""クラスタリングするためのデータフレームを作る. """
#k_means_listにはクラスタリングをするデータのfile_names_lsitでの要素番号を入れる.
#k_means_id_listに直接ファイル名を入れてもよい.
k_means_list = []
len(k_means_list)

k_means_id_list = []
for i in k_means_list :
  id_number = file_names_list[i]
```

```
  k_means_id_list.append( id_number )

file_processing.write_file_from_list(output_folder_path + "/" + 'k_means_id_list_10_20.txt',
     k_means_id_list)

df = pd.read_csv(input_folder_path + "/df_dict_data_10000.csv")
df = df.drop('Unnamed: 0', axis=1)
#クラスタリングに用いるファイルの名前のリストから単語とその出現回数を保存したcsvファイルを読み込む.
#全体の単語数のデータフレームに付け加える形で一つのデータフレームとして作る.
for i in k_means_id_list:
   input_file_path = '/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_4' + '/' + i + '.csv'
   d = pd.read_csv(input_file_path)
   d = d.drop('Unnamed: 0', axis=1)
   d.insert(0, '全単語数',d.sum(axis=1))
   df = df.append(d,ignore_index=True)
df = df.fillna(0)
df = df.astype(int)

df_index_from_filed_2 = ['全単語数']+ df_index_from_filed
df_index_from_filed_2 = set(df_index_from_filed_2)
df_select = make_dataframe_clustering(df,  df_index_from_filed_2 )
df_z_value_selected = make_z_value_dataframe( df_select, len_df )
df_z_value_selected_clustering = df_z_value_selected[1: len_df +1]
df_z_value_selected_clustering.to_csv(output_folder_path + '/' +"df_10_z_value_from_field.csv", sep=",")
```

リスト7　クラスタリングをして，各クラスタの特徴的な単語を取り出す.

```
#マウント
from google.colab import drive
drive.mount('/content/drive')

##作業するディレクトリを指定する.
%cd '/content/drive/MyDrive/Colab_Notebooks/arXiv_data'

#関係するツールの導入
import os
import numpy as np
import pandas as pd
import collections
import re
from string import digits

import sys
import time

#input_folder_path: 入力となるファイルの保存場所
input_folder_path ='/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_6'
#output_folder_path: 出力となるpdfファイルの保存場所
output_folder_path = '/content/drive/MyDrive/Colab_Notebooks/arXiv_data/Step_7'

#モジュール
module_path = '/content/drive/My Drive/Colab_Notebooks/arXiv_data/module/'
os.chdir(module_path)
import file_processing

#クラスタリングに用いるデータのファイル名のリストを読み込む.
file_names_list = file_processing.read_file_to_list(input_folder_path + "/" + 'k_means_id_list_10_20_sub.
     txt')

#file_names_list
len(file_names_list)
```

```
34
35  #クラスタリングに用いるデータフレームを読み込んでくる．(csvファイル)
36  #file_names_listとdfの番号が対応していることに注意する．
37  df = pd.read_csv(input_folder_path + "/df_10_z_value_from_field_sub.csv")
38  df = df.drop('Unnamed: 0', axis=1)
39
40  """##クラスタリングの前準備"""
41
42  #正規化
43  def minmax_norm(list_input):
44      return (list_input - list_input.min()) / ( list_input.max() - list_input.min())
45
46  """K-平均法"""
47
48  #k平均法に必要なパッケージ
49  from matplotlib import pyplot as plt
50  from sklearn import datasets, preprocessing
51  from sklearn.cluster import KMeans
52  import numpy as np
53
54  #クラスタリングの結果からfile名をクラスタごとに分ける
55  # show_result (str_list,int_array,int) -> str_list_list
56  def show_result (file, result_k,n):
57    result_file = []
58    for i in range(n):
59      result_file.append([])
60    for j in range(len(file)):
61      l = result_k[j]
62      result_file[l].append(file[j])
63    #return print(result_file)
64    return result_file
65
66  #傾向となる単語を取りだす
67  #df: データ k_means_result:k-means法でクラスタリングした結果 , n:クラスタリングの個数
68  def trend (df,k_means_result,n,m,dictionary):
69    trend = []
70    df['cluster_id']=k_means_result
71    for i in range(n):
72      trend_n = df[df['cluster_id']==i].mean()
73      trend_number = trend_n.drop('cluster_id', axis=0)
74      trend_word = trend_number.sort_values(ascending = False)[0:m].index
75      trend.append((i,trend_word))
76    #return print(trend)
77    return trend
78
79  """階層的クラスタリング"""
80
81  #階層的クラスタリングに必要なパッケージ
82  import numpy as np
83  from matplotlib import pyplot as plt
84  from scipy.cluster.hierarchy import dendrogram
85  from sklearn.datasets import load_iris
86  from sklearn.cluster import AgglomerativeClustering
87
88  def plot_dendrogram(model, **kwargs):
89     # Create linkage matrix and then plot the dendrogram
90
91     # create the counts of samples under each node
92      #要素の値が全てゼロの配列を作成する
93     counts = np.zeros(model.children_.shape[0])
94     n_samples = len(model.labels_)
```

```
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1 # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
counts]).astype(float)

    # Plot the corresponding dendrogram
    #dendrogram(linkage_matrix, **kwargs,labels = file_names_list,leaf_rotation=90 )#, orientation='left')
    dendrogram(linkage_matrix, **kwargs,labels = file_names_list, orientation='left')

"""##データの加工"""

#各単語に対して正規化をする.
df_minmax_norm = df.apply(minmax_norm)

"""##K-means法"""

number_cluster = 5

#k-means
result = KMeans(n_clusters= number_cluster ).fit_predict(df_minmax_norm)

show_result(file_names_list,result,number_cluster)

#それぞれのクラスターでよく用いられる単語10個
trend(df,result,number_cluster,10,df_minmax_norm.columns)

"""##階層的クラスタリング"""

parm_affinity = 'euclidean'
param_linkage = 'average'
  # setting distance_threshold=0 ensures we compute the full tree.
  # affinity: 距離の出す際の計算方法
  #linkage : 観測セット間で使用する距離 ward マージされるクラスターの分散を最小限に抑える
  #distance_threshold クラスターはマージされないリンケージ距離のしきい値
  #n_clusters 検索するクラスターの数
model = AgglomerativeClustering(affinity= parm_affinity,
linkage= param_linkage,
distance_threshold=0,
n_clusters=None)
model = model.fit(df)
fig = plt.figure( figsize=(6,8))
plt.title(parm_affinity + ', linkage=' + param_linkage )
  # plot the top three levels of the dendrogram
plot_dendrogram(model, truncate_mode='level')#, p=3)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
#plt.figure(figsize=(6, 8), dpi=300)
  #fig.savefig("dendrogram.pdf")
fig.savefig( output_folder_path + "/dendrogram_z_euclidean_mean_sub.pdf", bbox_inches='tight')
plt.show()
```