学士学位論文

Web ディレクトリを利用した名詞のジャンルベクトルの作成

茨城大学工学部

情報工学科

執筆者：林　華

指導教官：新納　浩幸

平成 22 年 2 月 8 日

平成21年度 茨城大学工学部情報工学科卒業研究

Web ディレクトリを利用した名詞のジャンルベクトルの作成

著者：林華(06t4071f)
指導教官：新納　浩幸

論文要旨

自然言語処理の多くのタスクにおいて，名詞間の類似度の測定が重要である．本研究では名詞間の類似度を測ることと文書クラスタリングを目的に，名詞に対するジャンルベクトルを構築する手法を提案する．

自然言語処理にとってシソーラスが有用であることは明らかである．用例ベースの翻訳などに見られるように，シソーラスの主な用途は名詞間の類似度の測定である．しかしその測定方法はシソーラスを表す木構造の位置関係から類似度を測るという単純なものであるため，荒い類似度しか得ることができない．本論文では名詞をジャンルベクトルと呼ばれるベクトルで表現する．これによって各名詞間に細かい類似度を与えることができる．

名詞に通常の形態情報の他にジャンルの情報を与えることで，より深い意味解析を行うことができるが，本研究のアイデアは名詞に一つのジャンルを与えるのではなく，設定した各ジャンルとその名詞との関連度を与えるというものである．つまり，ジャンルベクトルとはベクトルの各次元をジャンルに設定し，名詞とそのジャンルとの関連度をその次元の値としたものである．名詞をジャンルベクトルで表現する場合，どのようなジャンルを設定するか，名詞とそのジャンルとの関連度をどのように求めるかという二つの問題がある．

本論文ではWebディレクトリを利用し，上記の問題を解決し，名詞のジャンルベクトルを作成した．作成した名詞のジャンルベクトルを用い，名詞のクラスタリング実験と文書のクラスタリング実験を行った．その結果，作成したジャンルベクトルが既存のシソーラスよりも有効であることがわかった．

目次

第1章　はじめに

1.1　研究概要

自然言語処理の多くのタスクにおいて，名詞間の類似度の測定が重要である．本研究では名詞間の類似度を測ることを目的に，名詞に対するジャンルベクトルを構築する手法を提案する．

自然言語処理にとってシソーラスが有用であることは明らかである．用例ベースの翻訳などに見られるように，シソーラスの主な用途は名詞間の類似度の測定である．しかしその測定方法はシソーラスを表す木構造の位置関係から類似度を測るという単純なものであるため，荒い類似度しか得ることができない．本論文では名詞をジャンルベクトルと呼ばれるベクトルで表現する．これによって各名詞間に細かい類似度を与えることができる．

名詞に通常の形態情報の他にジャンルの情報を与えることで，より深い意味解析を行うことができるが，本研究のアイデアは名詞に一つのジャンルを与えるのではなく，設定した各ジャンルとその名詞との関連度を与えるというものである．つまり，ジャンルベクトルとはベクトルの各次元をジャンルに設定し，名詞とそのジャンルとの関連度をその次元の値としたものである．名詞をジャンルベクトルで表現する場合，どのようなジャンルを設定するか，名詞とそのジャンルとの関連度をどのように求めるかという二つの問題がある．本論文ではWeb ディレクトリを利用して上記二つの問題を解決する．

1.2　本論文の構成

第2章では本研究の中心である名詞のジャンルベクトル作成についてを述べる．

第3章では作成した名詞のジャンルベクトルを評価するための文書クラスタリングについてを説明する．

第4章では従来型の文書クラスタリングと本研究で作成した名詞のジャンルベクトルを用いた文書クラスタリングとの比較実験についてを述べる．

第5章では本研究の結果より，考察を行う．

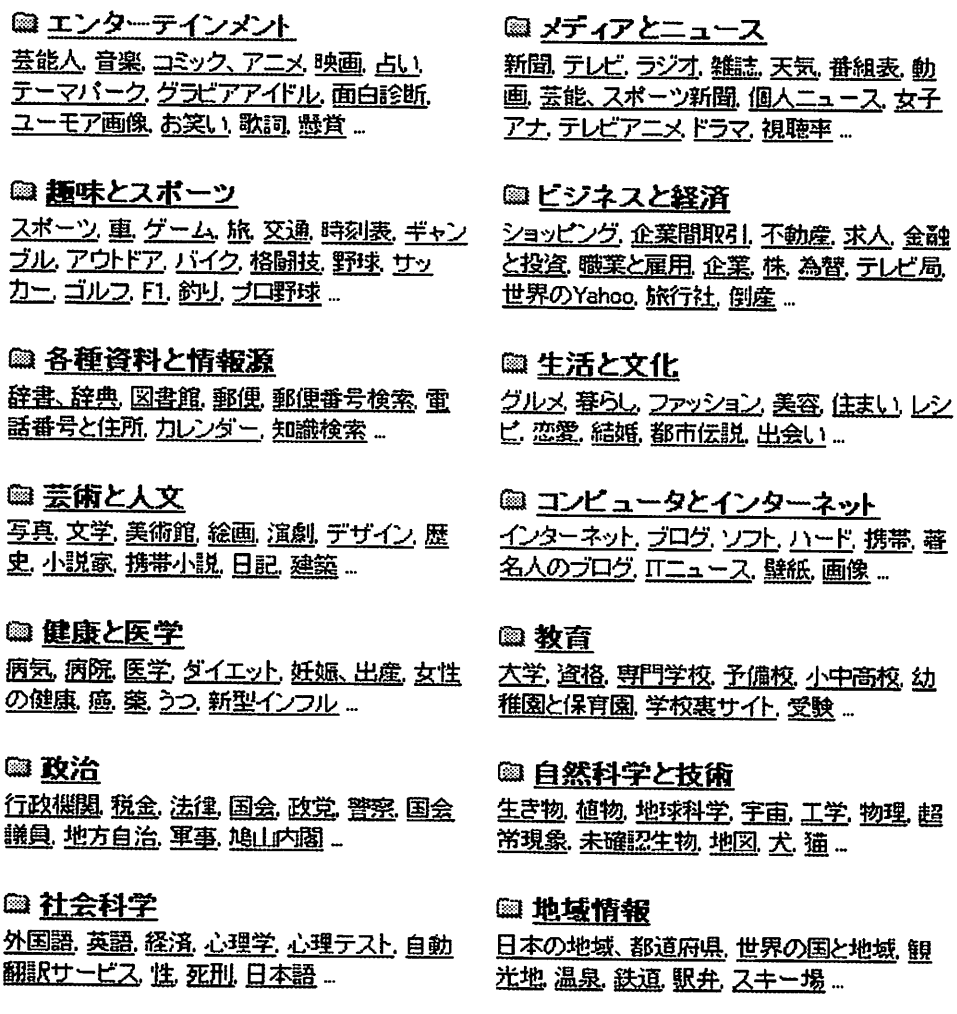
第2章　名詞ジャンルベクトルの作成

2.1　Webディレクトリ

Web ディレクトリとはWeb 上のホームページを階層的に分類したものである. 基本的には木構造になっており, 各ノードが分類のタイトル, リーフノードが各ホームページになっている.

本研究で名詞ジャンルベクトルの作成に, ジャンルの構成に利用するWebディレクトリをYahoo!JAPANのカテゴリ(http://dir.yahoo.co.jp/)を利用した. Yahoo!JAPANは日本最大級のポータルサイトで, 検索, オークション, ニュース, メール, コミュニティ, ショッピング, など80以上のサービスを展開しており, 利用者数が多く, その中のWebディレクトリが各分野を幅広くカバーし, 正確に分類されると考えられるため, 本研究のジャンルとして採用した. 図2.1がそのWeb ディレクトリのトップページである.

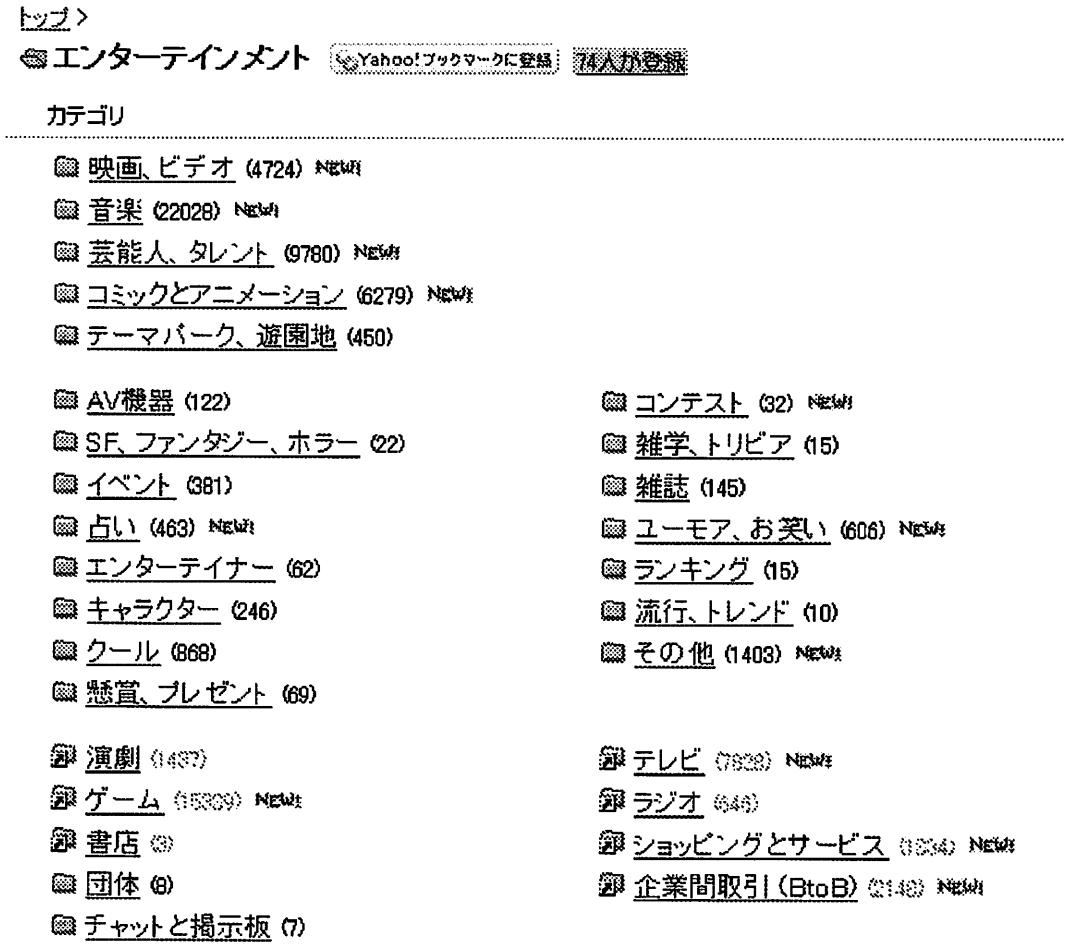
図2.1：Yahoo!Webディレクトリ



上の図2. 1はルートノードの下の第1階層のノードのタイトルが並び, その下に第2階層のノードのタイトルが複数個示されている.

ここで最初のノードの「エンターテインメント」をクリックすると, 「エンターテインメント」の一つ下位のノードが表示される (図2. 2). これらのノードが第2階層である. 以下同様に各階層のノードが表示され, 最終的には指定された分類に属するホームページのタイトルが表示される. このタイトルはそのホームページにリンクされている.

図2. 2:「エンターテインメント」の第2階層



本論文では第2階層に属するノードを各ジャンルに設定する. ジャンルの種類数は362である. このため本論文のジャンルベクトルの次元は362 となる. 次の表2. 1は本研究で利用したジャンルの部分を示した.

表2.1：本研究で利用したジャンルの部分

エンターテインメント - 映画，ビデオ エンターテインメント - 音楽 エンターテインメント - 芸能人，タレント ・・・
メディアとニュース - 新聞 メディアとニュース - インターネット放送 メディアとニュース - テレビ ・・・
・ ・ ・
地域情報 - 世界の国と地域 地域情報 - 観光ガイド 地域情報 - ご当地キャラ ・・・

2.2 種単語

本研究ではYahoo! のWebディレクトリの第２階層に属するノードを各ジャンルに設定している．基本的にはジャンルのノードのサブジャンルの名称を利用する．Yahoo! のWebディレクトリの第１階層の「エンターテイメント」のサブジャンルである「映画，ビデオ」の下には以下の表2.2のようなサブジャンルが並んでいる．これらはWeb ディレクトリの第３階層に属するノードとなっている．

表2.2：「エンターテイメント」の第３階層

「ジャンル」「日本映画」「外国映画」「作品」「映画館」「映画制作」「脚本」「監督」「俳優，女優」「映画祭」「特集上映」「試写会」「予告編」「映画音楽」「映画史」「テレビ番組」「ホームシアター」「イベント」「グッズ」「クラブ」「雑誌」「賞，コンテスト」「大学映画研究会」「団体」「チャットと掲示板」「データベース」「評論，レビュー」「リンク集」「配給会社」「書店」「ビデオカメラ，ビデオデッキ」「映画関連企業」「ビデオ関連企業」

本研究に利用した最初のディレクトリ構造はYahoo!カテゴリの四階層までである．上の表2.2のサブジャンル「日本映画」，「外国映画」のサブジャンル，つまり第4階層の名称を次の表2.3に示した．

表2.3：［日本映画」，［外国映画」のサブジャンル

日本映画	外国映画
「SF」「ファンタジー」「アクション」「アドベンチャー」「インディペンデント」「コメディ」「サスペンス」「ミステリー」「時代劇」「スポーツ」「青春」「伝記」「ドキュメンタリー」「ドラマ」「任侠」「ホラー」「歴史」「恋愛」「ロードムービー」	「SF」「ファンタジー」「アクション」「アドベンチャー」「アニメ」「インディペンデント」「ウエスタン」「コメディ」「サスペンス」「ミステリー」「スポーツ」「青春」「戦争」「伝記」「ドキュメンタリー」「ドラマ」「ファミリー」「ホラー」「ミュージカル」「歴史」「恋愛」「ロードムービー」

上の表2.3のように三層以降重複な情報が多数に存在している．ジャンル2階層以降の分類は細かく過ぎて，かつ重複な情報が多数存在したので，ジャンルを二階層までに自動的に修正を行った．具体例を次の表2.4のように示した．

表2.4：元のディレクトリファイルの修正例

修正前	修正後
エンターテインメント 　映画，ビデオ 　　日本映画 　　　SF 　　　ファンタジー 　　　アクション 　　　コメディ 　　　・ 　　　・ 　　外国映画 　　　SF 　　　ファンタジー 　　　アクション 　　　アドベンチャー 　　　コメディ 　　　戦争 　　　・ 　　　・ 　　・ 　　・ 　　・ 　　ビデオ関連企業 　　　レンタル 　　　販売 　　　・ 　　　・ 　音楽 　　アーティスト 　　　アニメソング 　　　演歌 　　　・ 　　歌詞 　　　アニメソング 　　　作詞 　　　・ ・ ・ ・	エンターテインメント - 映画，ビデオ 　　SF 　　ファンタジー 　　アクション 　　コメディ 　　アドベンチャー 　　戦争 　　レンタル 　　販売 　　・ 　　・ 　　・ エンターテインメント - 音楽 　　アニメソング 　　演歌 　　作詞 　　・ 　　・ 　　・ ・ ・ ・

種単語

このように，実際には，第2層以下におけるすべて異なる第3層の中身(つまり第4層)を一つの層として考え，しかも重複でないようにディレクトリファイルを修正した．これで，第1層と第2層の連結より，一つのジャンルが出来上がった．このような修正の結果より，ジャンル名だけを見れば，表2.1の通りとなる．

本研究では修正後の各ジャンルに存在している名詞を‘種単語’と呼ぶ．

修正後の各ジャンルに種単語の数は均一ではなかった．しかもジャンルと関係薄いものも多数に存在したため，各ジャンルに属する種単語の数を15個にした．詳しい作成方法についてを次の節2.4に述べる．

2.3 ベクトル化に利用した名詞

ベクトル化しようとする名詞はなるべく普段よく使われているものが良い.本研究では言語処理学会から発行した新聞・白書のコーパスBCCWJコーパスを利用した.その一例を付録にご覧ください.

コーパスから名詞を取り出しの作業にはMecab(http://mecab.sourceforge.net/)という形態素解析エンジンを利用した.先ず,Mecabをダウンロードし,インストールしておいた.

ファイルの処理には高水準言語Pythonを用いた.Python2.6をhttp://www.python.orgからダウンロードし,インストールした.ファイル処理には新しいディレクトリを作成したり,処理結果に特定な名前を付け,保存したりする作業が存在したため,その上ファイルの数も膨大で,Mecabだけを利用するにはできないがある.そこで,Mecabを用いた解析処理もPython内で行うようにした.

形態素解析ツールMecabとプログラム言語Pythonの導入を完成し,ファイルから名詞を取り出す処理に着手した.まず,最初に行う作業は用例から,Mecabより解析を行い,次に,解析の結果より,名詞だけを取り出した.詳しい名詞の取り出し方については次の節2.4で述べる.

本研究ではベクトル化に利用された名詞はこの解析の結果より取り出した名詞である.

2.4 名詞間の関連度と種単語の修正

2.3では，ファイルから名詞を取り出す作業について触れた．名詞のベクトル化には，名詞間の関連度を計算することが必須で中心となる．本研究では名詞間の関連度を表す係数についてはダイス係数を利用した．ダイス係数を次のように定義する．

$$dice(v,u) = 2 \times \frac{count(v \ and \ u)}{count(v \ or \ u)}$$

ここで，v,uはそれぞれ異なる名詞を意味し，$v \ and \ u$はある句に名詞vと名詞u両方含まれるのを意味し，$v \ or \ u$は句には名詞v，または名詞uどちらか片方しか含まれないことを意味する．また，$count$はこれらの条件を満たす句の数である．

次に，ファイルから名詞の取り出し方について詳しく説明しておこう．次の表2.5の用例は元のコーパスにあるデータの一部である．

表2.5：コーパスの一用例

母親の希望にしたがってアムンセンは大学の医学部にすすんだが，目的は探検家だったので，そのための鍛練や読書に熱中し，成績は普通以下だった．二一歳のとき，そんな息子の野心に気づかないまま母親は亡くなった．するとアムンセンは，もう母親の希望にしたがう必要もなくなったため，全力を探検のためにささげるべく大学を去った． ノルウェーには兵役の義務があり，健康な青年は一定期間入営して軍事教練を受ける．アムンセンは将来の仕事に役立てるために，すすんで軍事教練を利用しようと思った．しかし近眼だったので，そのために合格できないおそれがある．アムンセンが身体検査ではだかになったとき，老軍医はその鍛練された頑健なからだに感心して，「なんて見事な体格だ!」と叫んだ．一五歳のときからきたえたおかげである．老軍医は，となりの部屋にいた将校たちを呼んでアムンセンのからだを見せているうちに，眼の検査をすっかり忘れてしまった．

まず，上の表2.5の用例をMecabで解析し，次に，解析の結果により，句を単位として名詞を取り出す作業を行った．表2.5解析の結果を次の表2.6に示した．

表2.6：用例を句を単位として名詞を取り出した結果

母親 希望 アムンセン 大学 医学部 目的 探検 家 ため 鍛練 読書 熱中 成績 普通 以下 二 一 歳 とき 息子 野心 まま 母親 アムンセン 母親 希望 必要 ため 全力 探検 ため 大学 ノルウェー 兵役 義務 健康 青年 一定 期間 入営 軍事 教練 アムンセン 将来 仕事 ため 軍事 教練 利用

近眼 ため 合格 それ
アムンセン 身体 検査 とき 軍医 鍛練 頑健 からだ 感心 見事 体格
一 五 歳 とき おかげ
軍医 となり 部屋 将校 たち アムンセン からだ うち 眼 検査

まず，上の例を使って説明しよう．この用例には名詞“検査”と“軍医”と“将校”とがある．それぞれ2個の句，2個の句，1個の句に現れた．また，同じ句に現れたのは“検査”－“軍医”は2回で，“検査”－“将校”は1回である．これより，dice(″検査″，″軍医″) = 2 * 2/(2+2) = 1.0に対し，dice(″検査″，″将校″) = 2 * 1/(2+1) = 0.67となる．1.0は0.67より大きいため，名詞“検査”と“軍医”との関連度は“検査”と“将校”との関連度より高いと定義された．

ダイス係数を利用したのは，関連度の高い二個の名詞は同じ句に現れる確率が関連度低いものより高いという考えの基である．たとえば，名詞“道路”と“渋滞”の場合を考えよう．誰にしても普通に考えれば，“道路”と“渋滞”との関連度は“道路”と“魚”との関連度より高いだろう．“道路”と“魚”とが同じ句に現れるのが数少ない特定な文書に限られるともいえるだろう．勿論，dice(″道路″，″渋滞″)とdice(″道路″，″魚″)の結果はコーパスの文書量と内容によって結果が異なるが，文書の量が多くければ多くほど前者の値が後者の値より大きいに間違いないだろう．また，本研究に利用したコーパスの文書数は6169個である．

2.2節の「種単語」にはジャンルと関係が薄く，多数の種単語が存在したと述べたが，ここで，上に紹介した名詞間の関連度を測るDice係数を用いて，各ジャンルにおける種単語の修正作業を行った．

修正の目的は各ジャンルに属する種単語がなるべくそのジャンルと強く関連するようになることである．ここで具体的な例を挙げる．次の表2.7は修正前と後のジャンル「社会科学 － 経済学」に属する種単語の例である．

表2.7：ジャンル「社会科学 － 経済学」の種単語修正例

修正前	修正後
「学術」「誌」「学会」「誌」「環境」「会計」「学」「資格」「試験」「金融」「工学」「金融」「と」「投資」「技術」「経営」「MOT」「経営」「情報」「学」「国際」「経営」「学」「コーポレート・ガバナンス」「産業」「組織」「心理」「学」「データ」「マイニング」「マーケティング」「産業」「考古学」「PFI」「金融」「ビッグバン」「税」「交通」「不動産」「貿易」「労働」「ゲーム」「理論」「経済」「政策」「と」「規制」「財政」「学」「通貨」「医療」「経済」「学」「銀行」「市場」「造幣局」「地域」「通貨」「システム」「電子」「マネー」「環境」「経済」「学」「経営」「学」「高等」「教育」「政治」「経済」「学」「金融」「比較」「経済」「学」「地域」「経済」「学」「シンクタンク」「会計」「学」「と」「会	「地域」「国際」「産業」「政策」「環境」「市場」「技術」「貿易」「投資」「政治」「財政」「学」「労働」「情報」

計」「監査」「学」「経済」「史」「経済」「理論」「経済」「思想」「公共」「経済」「学」「農業」「経済」「学」	「システム」

ジャンルの種単語の修正は主に第2階層にある名詞と種単語との関連度から決める．この例の場合には，「経済学」と各種単語との関連度を計算し，関連度の高いもの上位15個の種単語をジャンル「社会科学 - 経済学」の種単語とする．

また，仮に第二階層の名詞が2個以上の場合，例えば，「経済学」がMecabより「経済」と「学」と分けられた場合には，「経済」と「学」とがそれぞれ各種単語との関連度を計算し，各種単語とジャンルとの関連度をその種単語とジャンルの第二階層内にあるすべての名詞との関連度の和となる．上の例にすれば，「学術」とジャンル「社会科学 - 経済学」との関連度は「学術」と「経済」との関連度，プラス「学術」と「学」との関連度の和となる．もちろん，これはMecabの解析の結果により，複合詞の「経済学」になるか，単純な「経済」と「学」に分けられるかに結果が異なる．

以上の修正作業より，各ジャンルの種単語がよりシンプルなものになり，かつ関連度が高め，名詞のジャンルベクトル作成に正確度と実行時間効率アップに繋がる．

2.5 ジャンルと名詞の関連度

2.4節には名詞間の関連度についてを述べた．同様な手法を利用してジャンルと名詞間の関連度についてを説明する．

各ジャンルに複数の名詞，いわば種単語が存在することを前節に述べた．また，任意の二個の名詞の間の関連度については2.4節に説明した．ここで，ジャンル j_k に h 個の種単語があるとしよう．すると，このジャンルの種単語は次の式と表せる．

$$\{n_1^{(k)}, n_2^{(k)}, ..., n_h^{(k)}\}$$

また，ベクトル化しようとする名詞を *noun* とする．本研究ではジャンルと名詞との関連度 $v(noun, j_k)$ をジャンル内にあるすべての種単語と名詞との関連度の平均値とした．式で表すと次のようになる．

$$v(noun, j_k) = \left(\sum_{i=1}^{h} dice(noun,\ n_i^{t}) \right) / h$$

2.6　名詞のジャンルベクトルの作成

特定なジャンルと名詞との関連度評価方法を決めたので，最終的に名詞のジャンルベクトル化を行うことができる．

ジャンルの種類の部分を 2.1 節の表 2.1 に示した．本研究ではジャンルの数は 392 である．

名詞のジャンルベクトル化は，名詞に対し，各ジャンルとの関連度を求め，ジャンル順に一列並べたものである．2.5 節の名詞とジャンルとの関連度 $\nu(noun, j_k)$ を ν_k とし，名詞 *noun* に対し，ジャンルベクトルは次のよう表せる．

$$V(noun) = (\nu_1, \nu_2 \ \ldots \ \nu_{392})$$

これで，名詞のジャンルベクトル化作業を完成した．

第3章　文書クラスタリング

3.1　文書クラスタリングとは

文書クラスタリングとは分類対象の集合を，内的結合(internal cohesion)と外的分離(external isolation)が達成されるような部分集合に分割するタスクである．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，テキストマイニングの重要な構成要素となっている．具体的な応用としては，検索文書を絞り込むために，検索された文書の集合をクラスタリングする適合性フィードバックが盛んに研究されている．

文書クラスタリングでは，まずデータとなる文書をベクトルで表現する．データがベクトルで表現できれば，ward法やk-meansなどの古典的クラスタリング手法を用いることができる．文書をベクトルで表現するためには，通常，bag of wordsのモデルを用い，次にTF-IDFなどによって次元の重みを調整する．

クラスタリング手法は大きく，次の5つの手法に分けられる．

- Hierarchical methods　階層的手法
- Partitioning methods　分割手法
- Density-based methods　密度に基づく手法
- Grid-based methods　格子に基づく手法
- Model-based methods　モデルに基づく手法

次の節にこの五つのクラスタリング手法についてを簡単に紹介する．最後に，本研究に用いたクラスタリング手法についてを説明する．

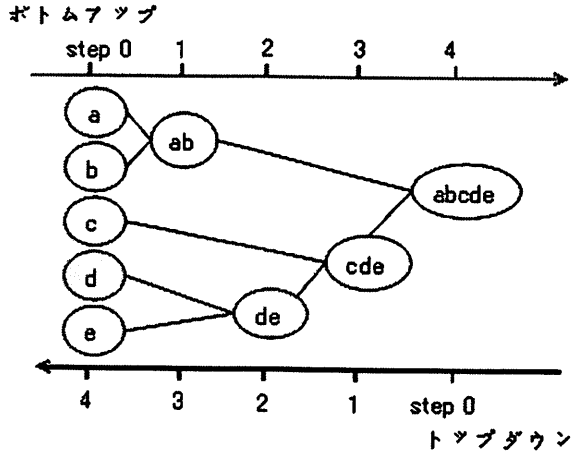
3.2 各種のクラスタリング手法

1 Hierarchical methods 階層的手法

階層的手法は，さらに分枝型(divisive)と凝集型(agglomerative)に分けられるが，ここでは後者のみを紹介する．また，分枝型については3.3に紹介する．

凝集型階層的手法は，N 個の対象からなるデータが与えられた時，1個の対象を1個のクラスタとする．つまり，データ数と同じ数のクラスタを初期状態とする．次に，対象 x1 と x2 の間の距離 D(x1,x2) (非類似度)からクラスタ間の距離 D(C1,C2) を計算し，最もこの距離の近い二つのクラスタを逐次的に併合する．そして，この併合を，全ての対象が一つのクラスタに併合されるまで繰り返すことで階層構造を構成する．階層構造は次の図3.1になる．

図3.1：クラスタの階層構造



クラスタ C1と C2 の距離関数 D(C1,C2)は以下のような手法が存在する．

・最短距離法 (nearest neighbor method) または 単連結法 (single linkage method)

最短距離法は，対象と対象の単純距離を調べ，お互いに一番近い対象から順にクラスタとして形成していき，最終的に階層構造を得る手法である．上の図1のボトムアップ方向はこの方法である．距離関数は以下の式で表す．

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

・最長距離法(furthest neighbor method) または 完全連結法 (complete linkage method)

最長距離法とは，最短距離法とは逆に，対象間の距離が一番長いものから，順に分割し

ていく方法である．つまり，距離が一番長い物は，同一クラスタではないと見なし，分割しながら，最終的に階層構造を形成する手法である．上の図3.1のトップダウン方向はこの手法である．また，距離関数で表すと次の式となる．

$$D(C_1,C_2)=\max_{x_1\in C_1,x_2\in C_2} D(x_1,x_2)$$

・群平均法（group average method）

群平均法は，最短距離法や最長距離法のように，極端に最大，最小という値に基づいて比較を行っている．この手法は，クラスタ間の距離を平均的な値で定義する考え方に基づいた方法のひとつである．具体的には，対象となる二つのクラスタがあるとすると，お互いが持っている対象すべての対の距離の平均により，両クラスタ間の距離を比較する方法である．距離関数は，以下の式より表される．

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

ここで，n_1,n_2はそれぞれクラスタC_1,C_2を構成する対象の数である．

・Ward法（Ward's method）

Ward法は各対象から，その対象を含むクラスタのセントロイド(重心)までのユークリッド距離を最小化する手法である．距離関数が次のように表れる．

$$D(C_1,C_2)=E(C_1\cup C_2)-E(C_1)-E(C_2)$$
$$(\text{ただし、}E(C_i)=\sum_{x\in C_i}(D(x,c_i))^2)$$

ここで，c_iはクラスタC_iのセントロイドで，xはクラスタC_iに含まれるデータである．また，式のように，E(A)は，Aのすべての点からAの重心までの距離の二乗の総和である．

2 Partitioning methods 分割手法

分割手法は，非階層的手法の他，partitional や optimization など多くの呼び方がある．この手法は，分割の良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．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(k平均法)はこの手法の代表である．

K-means法は，クラスタの重心点クラスタの代表点とし，重心から各データとの距離を

以下の式で評価する．

$$\sum_{i=1}^{k}\sum_{x\in C_i}(D(x,c_i))^2$$

以上の評価関数を最小化するようにデータを k 個のクラスタを分割する．最適解の探索アルゴリズムは次の図3.2に示した．このアルゴリズムは，対象を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ものである．また，この方法が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，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．また，クラスタリングのイメージを続いての図3.3に示した．

図3.2：k-means 法アルゴリズム

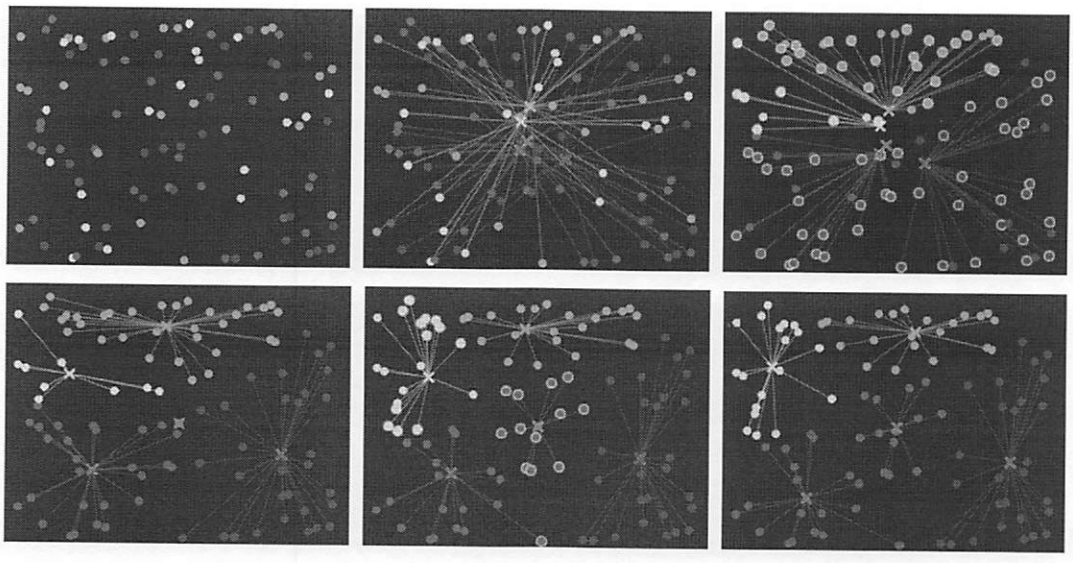
1　各データ$x_i(i=1\cdots n)$に対してランダムにクラスタを割り振る．

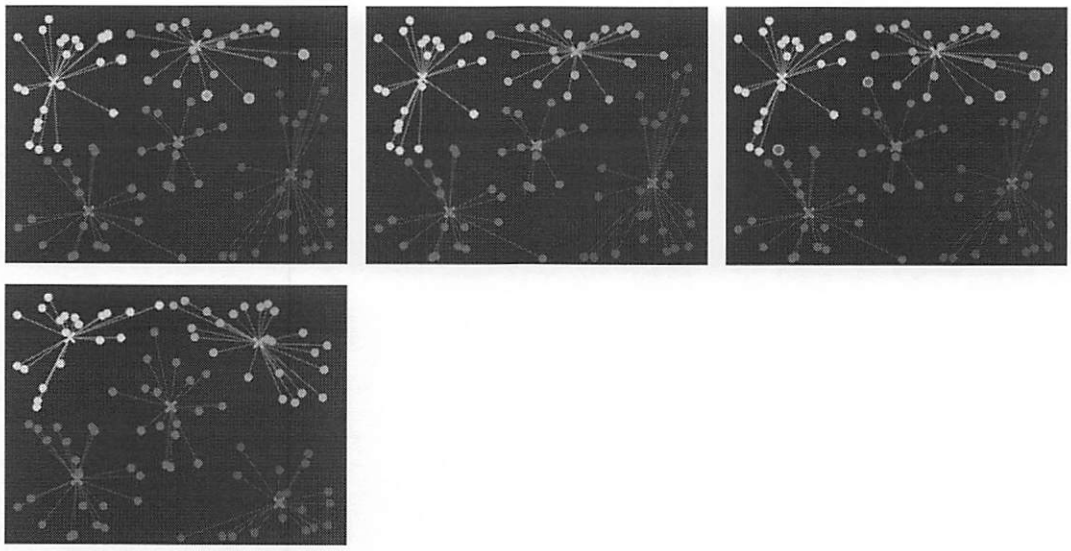
2　割り振ったデータをもとに各クラスタの中心$V_j(j=1\cdots K)$を計算する．計算は通常割り当てられたデータの各要素の平均が使用される．

3　各 x_i と各 V_j との距離を求め，x_i を最も近い中心のクラスタに割り当て直す．

4　上記の処理で全ての x_i のクラスタの割り当てが変化しなかった場合は処理を終了する．それ以外の場合は新しく割り振られたクラスタから V_j を再計算して上記の処理を繰り返す．

図3.3：K-means クラスタリングイメージ

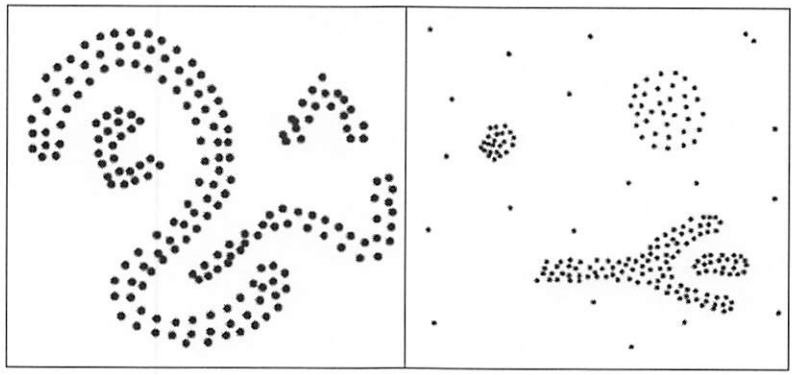




3　Density-based methods　密度に基づく手法

k-means 法などの多くのクラスタリング手法は,クラスタの分布形状が超球や超楕円体であることを仮定している．そのため，図3.4の高密度部分で構成されるような複雑な形状のクラスタは抽出できない.

図3.4：高密度で構成された複雑なデータ構造



この問題を解決する方法の一種としては密度に基づく手法である.この方法は一般的にデータ空間における低密度の領域によって分けられる高密度の領域のデータ要素をクラスタリングすることを目的とする.密度情報を用いる手法の利点は任意の形のクラスタを見つけることやノイズからクラスタを分けることができることなどである.欠点は入力パラメータを定めることが困難であることなどである.

上の図3.4のように,任意形状のクラスタを抽出する近年の代表的な研究に DBSCAN がある．この手法の Web ページが以下にある.

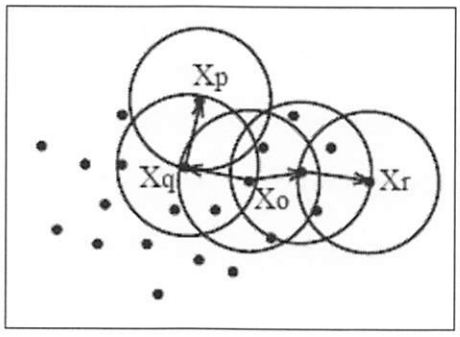
http://www.dbs.informatik.uni-muenchen.de/Forschung/KDD/Clustering/

DBSCAN は,距離のしきい値Eps と対象数のしきい値MinPts という二つのパラメータを用いる．これらのパラメータで決定される図3.5のような対象の接続関係を定義し，接続している対象を同じクラスタに分類する．この接続関係をdirectly densityreachable(DDR) と呼び，次の条件を満たすとき対象xpからxq へDDR であるという．

(1) $xq \in NEps(xp)$ ただし，$NEps(xp) = \{xq \in X \mid D(xp, xq) \leq Eps\}$

(2) $|NEps(xp)| \geq MinPts$　ただし，$Eps(xp) = \{xq \in X \mid D(xp, xq) \leq Eps\}$

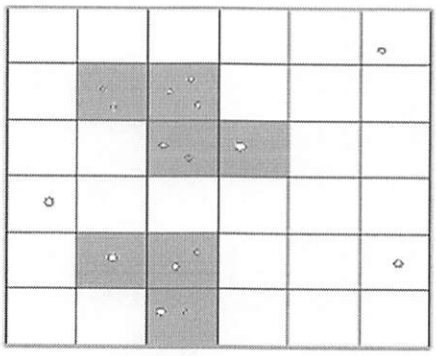
図3.5　DBSCAN 手法によるクラスタリング



4　Grid-based methods　格子に基づく手法

この手法は個々のデータ要素を結合させてクラスタを発見するのではなく,多重解像度の格子データ構造を用いてセル単位でクラスタを発見する手法である.格子構造を用いる手法では各次元における座標を K 個の区間に分割する．d 次元データに対しては，K^d 個のセルに分割し，類似した性質のセルを集めることでクラスタリングを行う手法である．図3.6はこの手法のイメージである．

図3.6　格子に基づくクラスタリング



5 Model-based methods モデルに基づく手法

この手法は与えられたデータと，ある数学的モデルとの適合性を最適化するものである．主に2つのアプローチがある．

- 統計的アプローチ (statistical approach)
- ニューラルネットワークアプローチ (neural network approach)

統計的アプローチは主に機械学習におけるクラスタリングの方法である．機械学習クラスタリングは最初に類似したオブジェクトのグループを特定し，その後，各グループの特徴を見つけ出すことである（この点が従来のクラスタリングと異なる）．各グループはクラスタと表す．クラスタを決定するには確率尺度を用いる．

ニューラルネットワークアプローチには各クラスタを見本(exemplar)として表す．見本はクラスタの“プロトタイプ”として振舞う．新たなオブジェクトは，何らかの距離尺度に基づいて，最も類似している見本を持つクラスタへ分配される．

3.3 本研究においてのクラスタリング手法

本研究においてはクラスタリングのアルゴリズムとしてk-wayクラスタリング法を用いる．この手法では，まず入力されたデータを2 つのクラスタに分類する．そして分割されたクラスタを再び2 つのクラスタに分割していく．こうした処理を必要なクラスタ数になるまで繰り返す．これらの分割処理は2-way クラスタリングが，以下に示す識別関数を最適化するように行われる．ここで，k はクラスタ数を，Si はi 番目のクラスタに属する文書集合を表す．

$$\text{maximize}\sum_{i=1}^{k}\sqrt{\sum_{v,u\in S_i} sim(v,u)}$$

また，ここでのsim(v, u)は文書間の類似度距離関数で，cosine関数を利用した．文書uおよびv 間の類似度sim(v, u) は，以下のように定義する．

$$sim(v,u)=\frac{\sum_{i=1}^{T} v_i \cdot u_i}{\sqrt{\sum_{i=1}^{T} v_i^2 \times \sum_{i=1}^{T} u_i^2}}$$

このk-way手法は3.2の階層的手法の分枝型に当たる．

本研究において，以上のようなクラスタリングを実現するために，ミネソタ大学で開発されているクラスタリングツールキットCLUTOを用いた．CLUTOは強力なクラスタリングツールであり，http://glaros.dtc.umn.edu/gkhome/views/clutoで公開されている．クラスタリング手法や類似度関数を様々に設定できるが，ここでは上記に紹介したK-way手法と類似度の距離関数cosineを用いた．

第4章　評価実験

この章では，作成した名詞のジャンルベクトルに対して評価実験を行う．従来型による文書クラスタリングと本研究で作成した名詞のジャンルベクトルによるクラスタリングとの比較を行う．また，本研究で作成した名詞のジャンルベクトルは名詞に対してもクラスタリング実験が効くため，ここでも実験を行う．その一方，従来型ではただの名詞に対しではクラスタリングができないため，文章のクラスタリングだけを行う．

文書クラスタリング実験で利用した文書はWebのニュース記事で，395個のファイルがある．これらは2003年11月25日から12月5日までの10日間でニュースサイトhttp://news.goo.ne.jp/に掲載されたニュース記事である．5個のカテゴリ(政治，経済，国際，社会，スポーツ）から集めた．

4.1 従来型の文書クラスタリング

従来型の文書クラスタリングは主に文書に存在した名詞とその頻度より文書間の類似度を測り，クラスタリングを行う．

文書間の類似度を測るためには，先ず文書内の単語をベクトル化しなければならない．ここの単語を主な場合には名詞を利用する．具体的な作業は次のようとなる．

1．先ず，形態素解析ツールより文書から名詞を取り出し，各名詞の数をカウントする．

2．二つの文書にそれぞれの名詞とその数は次のようになるとする．このように二つのファイルからトータル 15 個の名詞があり，それぞれのファイルに共通な名詞もあり，異なる名詞もある．u_i, v_i は *file1* と *file2* に i 個目の名詞の数を表す．

$$file1 = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$$
$$file2 = \{v_3, v_7, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}\}$$

3．これでも名詞の対象と次元(数)が違うため，類似度の計算ができない．　よって，両ファイルがすべての対象が揃うようにベクトルの修正が必要である．すると，次のようにファイル同士が参照しながら，ベクトルの次元を拡張していくことになり，最終的に各ファイルの中身が次の通りとなる．当然，実際のファイルの中身は修正する必要がまったくなく，類似度計算する際の処理でよい．

$$file1 = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}, 0, 0, 0, 0, 0\}$$
$$file2 = \{0, 0, v_3, 0, 0, 0, v_7, 0, 0, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}\}$$

ここで，ゼロの所が対応する i 番目の名詞がそのファイルに存在しないことを意味する．また，膨大なファイルを計算する場合，すべてのファイルから巨大な名詞のテーブルを作成する必要がなく，計算する二つのファイルの名詞だけのテーブルを作ればよい．

4．二つのファイルのベクトル化が完成し，それらの類似度を計算することの準備ができた．二つのファイルの類似度を次のように cosine 関数を用いて定義する．

$$sim(v,u) = \frac{\sum_{i=1}^{T} v_i \cdot u_i}{\sqrt{\sum_{i=1}^{T} v_i^2 \times \sum_{i=1}^{T} u_i^2}}$$

以上の１～４での手順より，すべての評価データの類似度を計算することができる．これで従来型のクラスタリング実験ための準備ができた．

実験にはCLUTOというクラスタリングツールキットを利用した．また，クラスタリング手法としては階層的手法の分枝型にあるk-wayという手法で，距離関数を上にあるcosine関数に指定した．

CLUTOによるクラスタリングの結果は，エントロピーは0.662で，純度は0.514 であった．

4.2 名詞ジャンルベクトルによるクラスタリング

4.2.1 名詞ジャンルベクトルによる名詞のクラスタリング

本研究では名詞のジャンルベクトルを作成しており，名詞に対するクラスタリングも可能となった．ここで，文章クラスタリングを行う前に，先ず名詞に対するクラスタリング実験を行い，本研究で作成した名詞のジャンルベクトルの精度を評価する．

この評価実験で利用した名詞のリストを次の表 4.1 に示した．

表 4.1：名詞クラスタリング用評価名詞リスト

「日本」「自民党」「野球」「リンゴ」「親切」「茨城」「予算」「テニス」「ご飯」「怒り」「日立」「献金」「バレー」「にんじん」「愛」「広島」「議員」「大相撲」「野菜」「思いやり」「アメリカ」「民意」「サッカー」「漬物」「努力」

表 4.1 の名詞が 25 あり，正解は次の五つのクラスタに分類される．

クラスタ 1	地名：	「日本」「茨城」「日立」「広島」「アメリカ」
クラスタ2	政治：	「自民党」「予算」「献金」「議員」「民意」
クラスタ3	スポーツ：	「野球」「テニス」「バレー」「大相撲」「サッカー」
クラスタ4	食べ物：	「リンゴ」「ご飯」「にんじん」「野菜」「漬物」
クラスタ 5	抽象名詞：	「親切」「怒り」「愛」「思いやり」「努力」

この 25 個の名詞に対し，本研究で作成したジャンルベクトルで表現し，名詞のクラスリングを行った．クラスタリングにも CLUTO を用いた．ここでもクラスタリングの手法としては k-way 手法で，類似度としては直接類似度行列(各名詞のジャンルベクトル)を与えて scluster で実験を行った．結果が次の表 4.2 に示した．

表 4.2：名詞クラスタリングの結果

クラスタ1 「大相撲」 クラスタ2 「自民党」「献金」「議員」「民意」 クラスタ3 「野球」「テニス」「バレー」「サッカー」 クラスタ4 「リンゴ」「ご飯」「にんじん」「野菜」「漬物」 クラスタ5 「日本」「茨城」「日立」「広島」「アメリカ」「親切」「怒り」「愛」「思いやり」「努力」「予算」

得られた結果のエントロピーは 0.256，純度は 0.760 であり，よい結果が得られていた．この結果から，本研究で作成したジャンルベクトルがある程度妥当であることがわかる．

4.2.2 名詞ジャンルベクトルによる文書のクラスタリング

前節で名詞ジャンルベクトルによる名詞のクラスタリング実験ではそれなりのよい結果を得られた. 続いて従来型のクラスタリングとの比較のため, 文書のクラスタリング実験を行う.

文書クラスタリングと名詞だけのクラスタリングとは違い, 文書内に多数の名詞が存在し, それら合計をそのファイルのベクトルに処理しなければならない. 一番単純なやり方は文章内にあるすべての名詞に対し, 同じ次元のものを合計してその次元の値にする方法である. こうすると, 文書のベクトル次元数も 362 と固定され, 名詞と同様なクラスタリング実験が行える.

このように各文書のベクトル化を行い, CLUTO を用いてクラスタリングを行った結果, エントロピーは 0.707 で, 純度は 0.506 となる. これは従来型の文書クラスタリングと比べて悪い結果となっていることを意味するが, 考えられる問題点を後の第 5 章の考察に指摘する.

ここで, 重みを調整し, 再実験を行った. 今度は, 従来の文書クラスタリング手法で作成した文書ベクトルを $s1$ とし, 本研究で作成した名詞のジャンルベクトルによる文書のベクトルを $s2$ とする. $s1$ と $s2$ との合計を文書のベクトルとする. 新たな実験の結果はエントロピーが0.601, 純度が0.539 であった.

このように本研究の名詞のジャンルベクトルと従来型手法の組合せにより, 文書のクラスタリング精度が従来型手法より改善されたことがわかる. これで本研究で作成した名詞のジャンルベクトルがある程度有効であることがわかる.

第5章　考察

5.1　問題点

第4章の評価実験で示したように，本研究で作成した名詞のジャンルベクトルがある程度有効であり，従来型手法と同様程度な結果を得られたが，十分とはいえないだろう．その問題点をここで簡単に指摘する．

先ず，本研究結果の精度と関わる一番考えられるものはコーパスの質である．基礎となるコーパスはバランスよく，各分野から集めてきたものが望ましいものと考えられる．しかし，本研究で使ったコーパスはそうでも言えない．著者選んだコーパスの大部分は小説などの文書で，科学技術，エンターテイメント，教育，経済政治など実は数少ない．従って，コーパスのバランスがよいものとは考えられない．よいコーパスを利用できれば，今度の実験結果より，よりよい結果が得られるのだろう．

次に，Webディレクトリの種単語に問題があると考える．そもそも種単語は属するジャンルと強い関係でなければならないが，実際はそうでもないのである．Webディレクトリのため，各ジャンルが人気があったり，なかったりなるのが事実である．人気のあるジャンルにはその下にサブジャンルが多数に存在し，幅が広く，そのジャンルの範囲を超えて，関係の薄いものが大勢に存在する．そうすると，そのジャンルの種単語が多い．その一方，あまり人気のないジャンルにはサブジャンルの数が少なく，結局のところ，種単語の数が少ないのである．各ジャンルに種単語数の差があり，単純に平均を取り，それをジャンルの次元値とするのが簡単だが，精度が落ちてしまうことになる．

最後に評価データにも問題あるではないかと考えられる．評価データはそれぞれ政治，経済，国際，社会，スポーツに分けられる．これらは機械的に判断されたものではなく，各記事の投稿者によりどのカテゴリにするかを決められた．人それぞれ投稿の分類に考え方が異なるが，多少誤ったところに投稿したのも考えられる．しかも，よく考えれば，この五つのクラスタには曖昧な境界が存在する．例えば，国際の文書は本当は政治にも，経済にも関わりやすいし，当然，社会に関するのも政治か経済かの問題と考えられる．このように，中身はっきりしないものには単語の共起が起こりやすく，当然クラスタリングしにくいだろう．

また，本研究で作成した名詞のジャンルベクトルは大きいジャンルと分類されても14個があるに対し，クラスタリングの場合は五つのクラスしか指定しないため，精度が落ちることが当然であり，簡単に比べられないと考えられる．その対策としてを次の5.2の改善点で紹介する．

5.2 改善点

前節の5.1ではコーパスの質，Webディレクトリの種単語の質，評価データセットの質の三つの問題を指摘した．ここで，それらの改善策を提案する．

先ず，コーパスの質の問題を挙げる．今度の研究で採用したコーパスの大部分が小説などの文書を中心とした．そうなると，各分野のバランスが取れていなかったと考えられる．コーパスのバランスをよく考慮するには，Webディレクトリとの相性も考え，コーパスをWebから取るのが望ましいと考える．多少面倒な作業になるが，コーパスの中身それぞれがあるジャンルに強く関係し，分類することができる．こうすれば，コーパスのバランスが保証でき，各分野の文書数まで正確に指定できるだろう．コーパスの質の改善は本研究における最重要な部分と考えられる．

次に，種単語の問題を説明する．本研究当初の生のWebディレクトリには各ジャンルの種単語の数がばらつき，あまり関係ないほどの種単語も多数存在した．そこで考えたのは種単語をその属するジャンルと強い共起関係を持つものだけにすることである．このような修正により，よりよい結果が得られることができた．種単語の修正より，名詞ジャンルベクトルの作成に精度よく，かつ実行時間が大幅に短縮できた．従って，この種単語の修正が妥当であることがわかる．

最後に評価データについてを述べる．評価データの問題点を前節で述べた．つまり，所謂の‘正解’は正解ではないこともあり得る．評価データを改善するには4.2.1の名詞の場合と同様に，明確に異なるクラスタに属するデータセットが必要であるだろう．

第6章　終わりに

本論文ではWebディレクトリを利用し，名詞のジャンルベクトルを作成した．作成した名詞のジャンルベクトルを用い，名詞のクラスタリング実験と文書のクラスタリング実験を行った．

作成した名詞のジャンルベクトルを測るため，従来型の文書クラスタリング実験も行った．クラスタリング実験比較の結果，本研究で作成した名詞のジャンルベクトルを用いた文書のクラスタリングの精度と従来型の文書のクラスタリングの精度と同等レベルであることがわかった．即ち，作成した名詞のジャンルベクトルが有効であることを示した．

しかし，期待と反して，従来型クラスタリングに比べ，はるかに良い結果を得ることができなかった．その原因を究明し，恐らく本研究の基本となるコーパスの質の問題が原因の一環となっているのだろう．今後同様な研究を行う際には，高質なコーパスを事前に用意することを勧める．その一方，本研究で提案したジャンルの種単語の修正案が精度にも実行効率にも有効であることを示した．

参考文献

[1] 新納浩幸，佐々木稔：Mcut + NMFによる文書クラスタリング
言語処理学会年次大会2007

[2] B.S.Everitt: Cluster Analysis, Edward Arnold, third edition (1993)

[3]大橋 靖雄: 分類手法概論，計測と制御，Vol.24, No.11, pp.999-1006 (1985)

[4]中村 朋健，上土井陽子，若林 真一，吉田 典可：FlexDice:高次元データ空間を持つ大規模データベースに対するリアルタイムクラスタリング

[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（2）－ 大規模データへの挑戦と次元の呪いの克服－

[6]村上浩司，野畑周，関根聡，井佐原均：新聞記事を対象にした，検索，分類，複数文書要約システムELIOTシステム

[7]橋本 力，黒橋 禎夫：基本語ドメイン辞書の構築と未知語ドメイン推定を用いたブログ自動分類法への応用

付録

(用例とプログラムソースコード)

コーパス用例

3　二人の生いたち

■アムンセンの生いたち

ローアル＝アムンセンは，一八七二年（明治５）にオスロ郊外の村に生まれたが，まもなくオスロ市内に引越した．探検家になろうと決心したのは，一五歳のときフランクリンの書いた本を読んだときである．さきに「一番悲惨な探検隊」として，一二九人全員が死んでしまった例を紹介したイギリス隊の隊長フランクリンは，それまでに二つの探検記を書いていた．いずれもアメリカ大陸の北極圏をさぐったときのものである．

フランクリンは探検記のなかで，氷や風雪とたたかい，飢えのため靴の皮まで食べるなどして耐えながら，死線をこえて生還する様子をくわしく書いている．アムンセンはそうした困難に耐えしのぶところにとくに感動し，自分もそのような未知の大自然で苦難に挑戦したいという衝動にかられた．

以来，アムンセンは酷寒の地への旅立ちにそなえて，体力づくりやスキー技術習得にはげんだ．真冬でも寝室の窓をあけっぱなしにして，寒さに耐えるように身体をきたえたので，母親はたいへん心配して注意したが，アムンセンは「新鮮な空気が好きだから」といいわけをしていた．父親はアムンセンが一四歳のときに死んでおり，母親はアムンセンを医者にしたいと考えていたため，アムンセンは探検家になる野心をかくしていたのである．

■すべては極地探検のために

母親の希望にしたがってアムンセンは大学の医学部にすすんだが，目的は探検家だったので，そのための鍛練や読書に熱中し，成績は普通以下だった．二一歳のとき，そんな息子の野心に気づかないまま母親は亡くなった．するとアムンセンは，もう母親の希望にしたがう必要もなくなったため，全力を探検のためにささげるべく大学を去った．

ノルウェーには兵役の義務があり，健康な青年は一定期間入営して軍事教練を受ける．アムンセンは将来の仕事に役立てるために，すすんで軍事教練を利用しようと思った．しかし近眼だったので，そのために合格できないおそれがある．アムンセンが身体検査ではだかになったとき，老軍医はその鍛練された頑健なからだに感心して，「なんて見事な体格だ！」と叫んだ．一五歳のときからきたえたおかげである．老軍医は，となりの部屋にいた将校たちを呼んでアムンセンのからだを見せているうちに，眼の検査をすっかり忘れてしまった．

二二歳の冬のこと，アムンセンは生涯でもっとも危険な目にあう．オスロの西方には高さ二〇〇〇メートルほどの高原がつらなっているので，極地旅行の訓練のつもりで友人と二人，スキーで横断して海岸側へ出ようとした．一一五キロほどを二日で越える計画だった．

しかし，第一日目の夜とまった牧場の小屋で吹雪になり，二日間とじこめられてしまった．三日目に歩きだしたところがまた雪になり，高原を横断できないうちに夜となって，雪の中でテントもなしに寝袋にはいるのだが，どうしたことか，このとき全食糧が紛失する．豪雪のなか，磁石をたよりに翌日さらに西へすすんだが，目的地の農家がどうしてもわからず，仕方なく東へ引き返した．

Yahoo！JapanのWebディレクトリ構造

エンターテインメント
　映画，ビデオ
　　ジャンル
　　SF，ファンタジー
　　アクション，アドベンチャー
　　アニメーション
　　インディペンデント
　　ウエスタン
　　コメディ
　　サイレント
　　サスペンス，ミステリー
　　時代劇
　　ショートフィルム
　　スポーツ
　　青春
　　戦争
　　伝記
　　ドキュメンタリー
　　ドラマ
　　任侠
　　ファミリー
　　ホラー
　　歴史
　　恋愛
　　ロードムービー
　　日本映画
　　SF，ファンタジー
　　アクション，アドベンチャー
　　インディペンデント
　　コメディ
　　サスペンス，ミステリー
　　時代劇
　　スポーツ
　　青春
　　ドキュメンタリー
　　ドラマ
　　任侠
　　ホラー
　　歴史
　　恋愛
　　ロードムービー
　　外国映画

noun_makefile.py(コーパス内の名詞を処理)

```
import os

#Translating orignal files from UTF-16 to Shift-JIS first,then excuse this file.
os.system('python26 make_mecab_files.py')
print 'Make mecab files done...'
os.system('python26 getsentence.py')
print 'Get sentences done...'
os.system('python30 noun_analyze.py')
print 'word,account analyzed...'
os.system('perl sort.pl noun_accounts.txt')
print 'oun_accounts.txt sorted...'
os.system('perl sort.pl pair_accounts.txt')
print 'pairt_accounts.txt sorted...'
print 'execute python30 and import calculate.py to start.'
```

make_mecab_files.py(コーパスのファイルをmecabより解析)

```
import os
import re

files = os.listdir('./files')
os.mkdir('./mecab_files/')
os.chdir('./mecab_files/')

for file in files:
    command = 'mecab '+'../files/'+file+'>'+file
    os.system(command)
```

getsentence.py(句を単位として，名詞を取り出す)

```
# coding: Shift-JIS

import os
import re

os.mkdir('./sentence')
os.chdir('./mecab_files')
files = os.listdir('./')

compiled_rule = re.compile('¥t名詞')
for file in files:
    f = open(file, 'r')
    lines = f.readlines()
    f.close()
    nouns = []
    for line in lines:
        if compiled_rule.findall(line):
            noun, rest = line.split('¥t')
            nouns.append(noun+' ')
        elif re.findall('. ', line) or (line == 'EOS¥n'):
            nouns.append('¥n')
    f = open('../sentence/'+file, 'w')
    f.writelines(nouns)
    f.close()
```

noun_analyze.py(コーパスにある名詞とそれらのペアーの計算)

```
# coding: Shift-JIS

import os
import re

os.chdir('./sentence')
files = os.listdir('./')
noun_counts = {}
pair_counts = {}
for file in files:
    f = open(file, 'r')
    sentences = f.readlines()
    f.close()
    for sentence in sentences:
        nouns = re.split(' ', sentence)
        l = len(nouns)
        pair_in_sentence = []
        list = []
        for i in range(l):
            if nouns[i] not in list and nouns[i] != '¥n':
                list.append(nouns[i])
        l = len(list)
        for i in range(l):
            noun = list[i]
            if noun not in noun_counts:
                noun_counts[noun] = 1
            else:
                noun_counts[noun] += 1
            j = i + 1
            while j<l:
                pair_in_sentence.append(list[i]+' '+list[j])
                pair_in_sentence.append(list[j]+' '+list[i])
                j += 1
        l = len(pair_in_sentence)//2
        for i in range(l):
            pattern1 = pair_in_sentence.pop()
            pattern2 = pair_in_sentence.pop()
            if pattern1 not in pair_counts and pattern2 not in pair_counts:
                pair_counts[pattern1] = 1
            elif pattern1 in pair_counts:
                pair_counts[pattern1] += 1
```

```
            else:
                pair_counts[pattern2] += 1

os.chdir('../')
f = open('noun_counts.txt', 'w')
for noun, count in noun_counts.items():
    f.write(noun+'¥t'+str(count)+'¥n')
f.close()

f = open('pair_counts.txt', 'w')
for pair, count in pair_counts.items():
    f.write(pair+'¥t'+str(count)+'¥n')
f.close()
```

sort.pl(名詞カウントソート)

```
#!/perl
while (<>){
    @noun_account = split / /, $_;
    $noun = @noun_account[0];
    $account = int(@noun_account[1]);
    $hash{$noun} = $account;
}
print 'read ok¥n';
@sorted = sort by_values keys %hash;
sub by_values {
    $hash{$b} <=> $hash{$a};
}
print 'sort ok¥n';
open out, '>./sorted_nouns.txt';
print out "$_¥t$hash{$_}¥n" foreach @sorted;
close out;
```

sort.py(名詞ペアーカウントのソート)

```
word_counts = []
f = open('./pair_counts.txt', 'r')
for line in f:
    word, count = line.split('¥t')
    word_counts.append([word, int(count)])
f.close()
print 'List maked...'

length = len(word_counts)
l, r = 0, length-1
def quicksort(l, r):
    if l < r:
        mid = (l + r)//2
        x = word_counts[mid][1]
        i, j = l, r
        while i <= j:
            while word_counts[i][1] > x:
                i += 1
            while word_counts[j][1] < x:
                j -= 1
            if i <= j:
                word_counts[j],word_counts[i] = word_counts[i],word_counts[j]
                i, j = i+1, j-1
        quicksort(l, j)
        quicksort(i, r)

quicksort(l, r)
print 'List sorted...'

f = open('./sorted_pairs.txt', 'w')
l = len(word_counts)
for i in range(l):
    f.writelines(word_counts[i][0]+'¥t'+str(word_counts[i][1])+'¥n')
f.close()
del word_counts
print 'File writed...'
```

calculate.py(関連度計算テスト)

```
# coding: Shift-JIS

word_count = {}
f = open('./noun_counts.txt', 'r')
for line in f:
    word,count = line.split('¥t')
    word_count[word] = int(count)
f.close()

pair_count = {}
f = open('./pair_counts.txt', 'r')
for line in f:
    pair,count = line.split('¥t')
    pair_count[pair] = int(count)
f.close()

def cal(word1,word2):
    if word1 not in word_count:
        print '¥"'+word1+'¥"が存在しないため，計算できない！'
        return
    if word2 not in word_count:
        print '¥"'+word2+'¥"が存在しないため，計算できない！'
        return
    c1 = word_count[word1]
    c2 = word_count[word2]
    pattern1 = word1+' '+word2
    pattern2 = word2+' '+word1
    if pattern1 in pair_count:
        c12 = pair_count[pattern1]
        print '¥"'+pattern1+'¥"¥t¥t存在¥t'+str(2.0*c12/(c1+c2))
    elif pattern2 in pair_count:
        c12 = pair_count[pattern2]
        print '¥"'+pattern2+'¥"¥t¥t存在¥t'+str(2.0*c12/(c1+c2))
    else:
        print '¥"'+word1+' '+word2+'¥"¥t¥t存在しない¥n'
        word_j = word1
        word_k = word2
        max_ij = 0
        for pair,count in pair_count.items():
            if (word_j in pair) and (count > max_ij):
                max_ij = count
                max_pair = pair
```

```
        print '¥"'+max_pair+'¥"¥t¥t'+str(pair_count[max_pair])
        pair = max_pair.split(' ')
        if pair[0] == word_j:
            word_i = pair[1]
        else:
            word_i = pair[0]

        max_km = 0
        for pair,count in pair_count.items():
            if (word_k in pair) and (count > max_km):
                max_km = count
                max_pair = pair
        print '¥"'+max_pair+'¥"¥t¥t'+str(pair_count[max_pair])
        pair = max_pair.split(' ')
        if pair[0] == word_k:
            word_m = pair[1]
        else:
            word_m = pair[0]

        pair_im1 = word_i+' '+word_m
        pair_im2 = word_m+' '+word_i
        if pair_im1 in pair_count:
            max_im = pair_count[pair_im1]
            pair_im = pair_im1
        elif pair_im2 in pair_count:
            max_im = pair_count[pair_im2]
            pair_im = pair_im2
        else:
            pair_im = ''
        if pair_im:
            print '¥"'+pair_im+'¥"¥t¥t'+str(max_im)
        else:
            print '¥"'+word_i+'¥"と¥"'+word_m+'¥"とが関連しないため，計算できな
い．'
            return
        c12 = 2.0 * max_im / (word_count[word_i]+word_count[word_m])
        print '¥"'+word1+' '+word2+'¥"¥t'+str(c12)
```

genre.py(ジャンルの修正)

```
import os

f = open("./yahoocategory.txt", "r")
lines = f.readlines()
f.close()

i = 0
genre_count = 0
n = len(lines)
genre = {}
f = open("./genre.txt", "w")
while i < n:
    if "¥t" not in lines[i]:
        level1 = lines[i].rstrip("¥n")
        level2 = lines[i+1].lstrip("¥t")
        key = level1 + " - " + level2
        f.write(key)
        i += 3
        genre_count += 1
        ws = []
    elif "¥t¥t¥t" in lines[i]:
        w = lines[i].lstrip("¥t")
        if w not in ws:
            ws.append(w)
    elif "¥t¥t" in lines[i]:
        pass
    elif "¥t" in lines[i]:
        fin = open("./in.txt", "w")
        fin.writelines(ws)
        fin.close()
        os.system("mecab in.txt > out.txt")
        ws = []
        fout = open("./out.txt", "r")
        for line in fout:
            if line == "EOS¥n":
                continue
            w, r = line.split("¥t")
            ws.append("¥t" + w + "¥n")
        fout.close()
        f.writelines(ws)
        ws = []
```

```
            level2 = lines[i].lstrip("¥t")
            key = level1 + " - " + level2
            f.write(key)
            genre_count += 1
        i += 1
f.close()
print genre_count
```

modified_genre.py(各ジャンルの種単語の修正)

```
#coding: shift-jis

import os

f = open("./yahoocategory.txt", "r")
lines = f.readlines()
f.close()

i = 0
genre_count = 0
n = len(lines)
genre = {}
f = open("./genre.txt", "w")
while i < n:
    if "¥t" not in lines[i]:
        level1 = lines[i].rstrip("¥n")
        level2 = lines[i+1].lstrip("¥t")
        key = level1 + " - " + level2
        f.write(key)
        i += 3
        genre_count += 1
        ws = []
    elif "¥t¥t¥t" in lines[i]:
        w = lines[i].lstrip("¥t")
        if w not in ws:
            ws.append(w)
    elif "¥t¥t" in lines[i]:
        pass
    elif "¥t" in lines[i]:
        fin = open("./in.txt", "w")
        fin.writelines(ws)
        fin.close()
        os.system("mecab in.txt > out.txt")
        ws = []
        fout = open("./out.txt", "r")
        for line in fout:
            if line == "EOS¥n":
                continue
            w, r = line.split("¥t")
            ws.append("¥t" + w + "¥n")
        fout.close()
        f.writelines(ws)
```

```
            ws = []
            level2 = lines[i].lstrip("¥t")
            key = level1 + " - " + level2
            f.write(key)
            genre_count += 1
        i += 1
f.close()
print genre_count
```

genre_calculate.py(各名詞のジャンルベクトル作成)

```
noun_count = {}
f = open('./noun_count.txt', 'r')
for line in f:
    k, v = line.split('¥t')
    noun_count[k] = int(v)
f.close()
print 'noun_count hash table done...'

genres = {}
f = open('./genre.txt', 'r')
for line in f:
    if '¥t' not in line:
        key = line.rstrip('¥n')
        genres[key] = ''
    else:
        genres[key] += line.rstrip('¥n')
f.close()
print 'genres hash table done...'

f = open('./sorted_paircount.txt', 'r')
pair_count = {}
for line in f:
        k, v = line.split('¥t')
        pair_count[k] = int(v)
f.close()
print 'pair_count hash table done...'

f = open('F:/genre_calculate_result2.txt', 'w')
for noun, count in noun_count.iteritems():
    f.write(noun + '¥n')
    for genreitem, wordlist in genres.iteritems():
        average = 0.0
        top15 = []
        wordlist = wordlist.strip().split('¥t')
        for genrenoun in wordlist:
            pattern1 = genrenoun + ' ' + noun
            pattern2 = noun + ' ' + genrenoun
            if pattern1 in pair_count:
                v = 2.0 * pair_count[pattern1] / (noun_count[genrenoun] + count)
                top15.append([pattern1, v])
            elif pattern2 in pair_count:
```

```
                v = 2.0 * pair_count[pattern2] / (noun_count[genrenoun] + count)
                top15.append([pattern2, v])
        if len(top15) > 15:
            n = len(top15)
            for i in range(n):
                max = top15[i][1]
                pos = i
                for j in range(i+1, n):
                    if top15[j][1] > max:
                        max = top15[j][1]
                        pos = j
                if pos != i:
                    top15[i], top15[pos] = top15[pos], top15[i]
            top15 = top15[:15]

        for pair, value in top15:
            average += value
        if len(top15) > 0:
            average /= len(top15)
        f.write(str(average) + '¥t')
    f.write('¥n')
f.close()
```

make_nouncount_in_testdata.py(テストデータの解析)

```
#coding: shift-jis

import os
import re

files = os.listdir('./data-sjis')
os.mkdir('./data-sjis-mecab')
os.chdir('./data-sjis-mecab')

for file in files:
    os.system('mecab ' + '../data-sjis/' + file + '>' + file)

pattern = re.compile('¥t名詞')
files = os.listdir('./')
os.mkdir('../noun-in-data-sjis')
os.mkdir('../noun-count')
for file in files:
    nounline = []
    noun_count = {}
    f = open(file, 'r')
    for line in f:
        if pattern.findall(line):
            nounline.append(line)
            noun, rest = line.split('¥t')
            if noun in noun_count:
                noun_count[noun] += 1
            else:
                noun_count[noun] = 1
    f.close()

    f = open('../noun-in-data-sjis/' + file, 'w')
    f.writelines(nounline)
    f.close()

    nounlist = list(noun_count)
    n = len(nounlist)
    for i in range(n):
        max = noun_count[nounlist[i]]
        pos = i
        for j in range(i+1, n):
            if noun_count[nounlist[j]] > max:
                max = noun_count[nounlist[j]]
```

```
            pos = j
        if pos != i:
            nounlist[i], nounlist[pos] = nounlist[pos], nounlist[i]

f = open('../noun-count/' + file, 'w')
for noun in nounlist:
    f.write(noun + '¥t' + str(noun_count[noun]) + '¥n')
f.close()
```

make_genrevalue_in_testdata.py(評価用データの作成)

```
import os

i = 0
noun_value = {}
f = open("./genre_calculate_result.txt", 'r')
for line in f:
    if i % 2 == 0:
        k = line.rstrip('¥n')
    else:
        v = line.rstrip().split('¥t')
        noun_value[k] = v
f.close()
print 'make hash table ok...'

genre_num = len(line.rstrip().split('¥t'))

files = os.listdir('./noun-count')
os.mkdir('./genrevalue_in_testdata')
os.chdir('./genrevalue_in_testdata')

sum = []
for i in range(genre_num):
    sum.append(0.0)

for file in files:
    noun_count = {}

    f = open('../noun-count/' + file, 'r')
    for line in f:
        noun, count = line.split('¥t')
        count = float(count)
        if noun in noun_value:
            values = noun_value[noun]
            for i in range(genre_num):
                sum[i] += float(values[i]) * count
        else:
            print noun + '¥t' + file

    f.close()

    f = open(file, 'w')
```

```
for i in range(genre_num):
    f.write(str(sum[i]) + '¥t')
f.close()
print file

for i in range(genre_num):
    sum[i] = 0.0
```