接尾辞配列とディリクレ過程混合モデルを用いたテキスト中の数値表現マイニング

吉田 稔[†1] 佐藤一誠[†1]
中川裕志[†1] 寺田昭[†2]

テキスト中には、「25 歳」「10000 円」等、数値としてデータベースに格納されているわけではないが、数字として表現された数値が存在する。本稿では、そのような「テキスト中に埋め込まれた数値」を対象とした、新しいマイニングシステムを提案する。提案システムは、「[200..800] 円」のような数値の範囲をクエリとして用いることができ、また、検索結果に現れる数値を、範囲表現として表示することができる。システムは、接尾辞配列を用いた文字列検索と、ディリクレ過程混合モデルを用いた数値クラスタリングにより実現される。また、応用として、数値範囲をクエリや結果に含む連接語抽出と同義語抽出の実行例を示す。

Mining Numbers in Text Using Suffix Arrays and Clustering Based on Dirichlet Process Mixture Models

Minoru Yoshida,[†1] Issei Sato,[†1] Hiroshi Nakagawa[†1]
and Akira Terada[†2]

We describe a text-mining system for interactively mining the relations between words and numbers. Texts such as newspaper articles or web documents have a lot of numbers references, such as "25 years old," "10,000 dollars," "31 May" etc. It is important to provide systems that can mine these types of numbers just as though they were numeric data stored in databases. Our system enables us to search for ranges of numbers, and use the results for text mining. Both queries and resulting strings can be both strings and numbers (e.g., "200..800 dollars"). The system is based on suffix arrays augmented to treat number information in a way that enables numbers to be searched for by using words and vice versa. Furthermore, the system performs clustering based on a Dirichlet process mixture of Gaussians to appropriately treat extracted collections of numbers.

1. 背　景

テキスト情報の中には、「25 歳」「10000 円」等、多くの数値表現が含まれている。通常、サーチエンジンやテキストへの索引付けツールでは、数値情報は、そのまま数字の文字列として取り扱うか、#や 0 などの数値を表す特殊記号に置き換えられるかであった。しかしながら、数値は、異なる数値間に順序を持つ（例えば、"200" は "100" より大きく、"101" は "100" と近い）等、他の文字列と異なる性質を持つ。こういった数値の特性を活用した検索を提供することは、テキスト中の数値からの知識発見に大きく貢献することが期待できる。

例えば、数字を全て同一に扱うシステムでは、"1", "2", "213", "215" という異なる数字文字列の区別をつけることができない。これに対し、「**類似する**数字（ここでは例えば "1" と "2"、"213" と "215"）を同一に扱い、そうでない数字（ここでは例えば "1" と "213"）を別に扱う」ことができれば、数値の性質をテキストマイニングに応用することができる。これは、**数字の範囲表現**（上の例では、例えば "[1..2]" と "[213..215]" という 2 つの異なる数値範囲）を用いることにより実現できる。我々は、与えられた数値コレクションから適切な**数値範囲**を導く機能、さらに、そのような数値範囲を**検索クエリ**として用いる検索機能を備えた新たなシステムを提案する。前者は数値範囲を**見つける**ための[*1]、後者は、見つけた数値範囲を**使う**ための機能であると言う事が出来る。例えばユーザは、「[100..200] 円」といったクエリを用い、「"円" の左側に 100 以上 200 以下の数値を含む文字列」を取り出すことができ、これにより、ユーザは数値の範囲を、通常の文字列と同様に用いることができるようになる。

このような柔軟な数値の取り扱いは、様々なテキストマイニングタスク、特に、数値文字列を、**文脈情報**として用いる場合に有用である。例えば、単語の類似度を測定する際、一般には、各単語 w に関連する語（w の周辺に出現する語、w と係り受け関係にある語、等）の出現分布情報を単語 w の**意味**を表す情報として捉え、それらの類似度を計算することにより単語同士の類似度を測る。我々のシステムでは、この文脈情報に、「数値の類似度」という観点を導入することができるようになる。例えば、「コーヒー」「紅茶」「定食」といっ

†1 東京大学
University of Tokyo
†2 (株) 日本航空
Japan Airlines Co., Ltd

*1 なお、本稿では、テキスト検索の結果として得られる数値コレクションのみを対象としているが、実際のアルゴリズムはあらゆる数値コレクションに適用可能である。

た3つの単語の類似度を測るとき、その周辺にそれぞれ「200円」「180円」「1000円」といった数値文字列が出現していた場合、「“200” と “180” を同一視し、“1000” は別に扱う」という処理が可能になることにより、「コーヒー」と「紅茶」が、「コーヒー」と「定食」よりも似ている、という知識を得ることができるようになる。

前者の「数値範囲を発見する」問題に対して、システムは、確率モデルDPM (Dirichlet Process Mixture, ディリクレ過程混合分布) を利用したクラスタリングを行う。DPMクラスタリングを用いることにより、クラスタ数を事前に指定せず、自動的に適切なクラスタ数が推定される。混合分布としては、DPMの例として一般的な混合正規分布を用いる[9]。DPMモデルによるクラスタ割り当ての問題に対しては、MCMC[7]、変分ベイズ法[2]、A*サーチ・ビームサーチ[6]等の近似解法が知られているが、我々の問題設定は、これら先行研究で議論されていた問題とは、**数値範囲を推定する**という点で異なっている。数値範囲の推定のためには、推定されるクラスタ割り当てに対し、「連続する領域への分割」という制約が必要になり、この制約をどのように既存の推定アルゴリズムに導入するかは自明ではない。このような領域分割へDPMクラスタリングを適用した先行研究は、我々の知る限りでは存在しない。これに対し本稿では、比較的少数の数値コレクションに対しては、最適な分割が、単純なCKY法により高速に求められることを示す。さらに、貪欲アルゴリズムを用いることにより、最適解に近い解を、遥かに高速に求められることも示す。提案アルゴリズムは、要素数100の数値コレクションに対するクラスタリングを、平均0.0002秒で行うことができる。これは、先行研究で示されている、一般的な数値クラスタリングタスクに対する推定時間[6]よりも遥かに高速であり、この速度により、数値クラスタリングを、テキストマイニングの部分処理として用いることが現実的になる。

後者の**数値範囲による検索**の実現のため、我々は、基本的な索引構造としてSuffix Array（接尾辞配列）を採用し、Suffix Arrayを利用して数値を検索するための手法を提案する。提案手法では、高速化のため、さらに **Number Array** という索引構造を追加する。Number Arrayのサイズは Suffix Arrayよりも大幅に小さいため、必要なメモリ領域の量は通常のSuffix Arrayとほぼ同一である。

本稿では、我々の提案システムを、**Number Suffix Array** と呼ぶ。Number Suffix Arrayでは、以下の2つの操作が可能である。

頻度計測： クエリ（数値範囲を含む）の頻度を測定する。

連接文字列取得： クエリ（数値範囲を含む）に連接する全文字列への索引を取得する。

ここでクエリは、数値範囲を複数含んでいても、数値範囲と通常の文字列が混在していてもよい。本稿では、これらの機能を用いて可能になるテキストマイニングの例を2つ紹介する。ひとつは「用例検索」であり、クエリに連接しやすい文字列を返すというタスクである。Number Suffix Arrayを用いることにより、「数値の範囲」をクエリ及び検索結果に含めることができるようになる。今ひとつは「同義語抽出」であり、クエリと類似した意味を持つ文字列を返すというタスクである。Number Suffix Arrayを用いてクエリや文脈に「数値の範囲」を含めることにより、同義語抽出精度の向上や、「数値の範囲」と類似した意味を持つ文字列の取得、等が行えるようになる。

数値の検索については、検索会社による既存のサービスが存在する。例えば、Fontouraら[5]は、文書に付随する数値メタ情報を用いて検索文書を限定するために、数値によるInverted Index（転置インデックス）を用いることを提案している。また、Google英語版では、オプション “..” を用いることにより、数値範囲検索が可能である。（例えば、クエリ “100..200 dollars” により、“125 dollars”, “147 dollars” といった文字列を含む文書が検索できる。）Inverted Indexにより、数値の存在する**位置情報**を得ることはできるが、これに対しNumber Suffix Arrayでは、位置情報に加えて、連接する文字列の集合を得ることができる。連接文字列情報は、単純な検索に限っても、連接文字列による検索結果の絞り込み（例えば、「数値＋ “円”」の文字列のみを取り出す）における高速化等に役立つが、加えて、前述の通り、様々なテキストマイニングに不可欠な情報であり、すなわち、Number Suffix Arrayは、通常のInverted Indexによる「検索」のための数値索引付けを、より「テキストマイニング」に適した形に拡張したものと言える。

現在の我々のシステムの限界としては、まず、「連続する数字」のみを（整数）数値文字列として定義していることが挙げられる。このため、例えば、“10,000” と “10000” や “10000.0”、さらには「一万」等を同一視することができず、「1.5キロメートル」と「1500メートル」のような単位の違いにも対応できない。これは「数字同義語」を抽出する問題であり、本稿の範囲を超えているため取り扱わないが、重要な将来課題の一つである。

また、索引構造としてSuffix Arrayを用いているため、Web全体のような大規模な文書集合に適用できないという問題もある。しかしながら、近年では、「特定サイトの文書集合（ある大学のWebページ全体、等）」や、「特定目的の文書集合（Wikipedia全体、等）」のように、「限定されたテキストコーパス」が、数ギガバイトのサイズに達することも多く、提案システムは、このような中規模文書集合に対するテキストマイニングを第一目的としている。

以上、本研究の主な貢献をまとめると、(1) 数値範囲を検索に用いることができる索引構

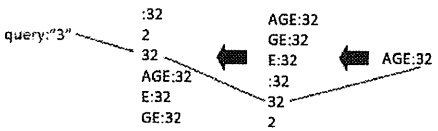

図 1　文字列 "AGE:32" に対する Suffix Array

造の提案、(2) 数値範囲分割における DPM モデルによる目的関数と、高速かつ精度の高いな推定アルゴリズムの提案、さらに、(3) 数値範囲を用いたテキストマイニングの提案、の３つである。

2. Suffix Array と Number Suffix Array

2.1 Suffix Array

Suffix Array[8] は、与えられた文字列のすべての接尾辞を辞書順にソートした整数配列である。与えられた文字列 S から、部分文字列 s を、$O(|s|log|S|)$ の時間計算量で発見することができる。(ここで $|x|$ は文字列 x の長さを示す。) Suffix Array は、文字列長と同じ長さの整数配列を必要とするため、整数を 32 ビット、各文字を 8 ビットで表現するとすると文字列の 4 倍、文字を 16 ビットで表現した場合は文字列の 2 倍のメモリを必要とするが、本研究では、文字列・Suffix Array ともにメモリ上に載っていることを仮定する。これは、将来的には、Compressed Suffix Array 等の省メモリの実装を用いることにより改善できると考えている。

図 1 に、S=AGE:32 に対する Suffix Array の構築例を示す。図中では、文字列の配列となっているが、実際には、構築された配列は、もとの文字列へのポインタの配列となっている。

また、$S[i..]$ で、i 文字目から始まる S の接尾辞（末尾までの部分文字列）を表す。さらに、Suffix Array sa に対し、sa の一部の値だけを、順序を保持したまま取り出した部分配列 sa' が定義できる。例えば、$sa' = [4, 5, 2]$ は、$sa = [4, 6, 5, 1, 3, 2]$ の部分配列となっている。部分配列に対しても、二分探索によって文字列の検索が行える（部分配列内にあるインデックスだけを対象に検索が行える）ことに注意されたい。

2.2 Number Suffix Array

我々の提案システム Number Suffix Array は、Suffix Array に、数値検索の機能を追加したインターフェースであり、以下の 2 種類の機能を持つ。

数値範囲クエリ　通常の文字列検索に加え、Number Suffix Array では、**数値範囲**をクエリに使用することができる。例えば、「[1000..2000]」というクエリ*1により、1000 以上 2000 以下の数値をすべて取得することができる。また、「[30..50] 人」のように、数値範囲と通常の文字列を混在させたクエリも可能である。

検索結果の数値範囲による表示　Number Suffix Array はさらに、複数の検索結果を、数値範囲を用いてまとめる機能を持つ。例えば、「300 円」「330 円」「350 円」という検索結果があったとき、これらをまとめて「[300..350] 円」という結果として表示することができる。どの範囲の数値をまとめて表示するかは、クラスタリングにより自動的に判別する。これにより、「1」と「1000000」をまとめて「[1..1000000]」となる、といった、不自然な数値範囲を避けることができる。

3. 数値範囲クエリによる検索

本節で、前者の数値範囲クエリによる検索の実現手法、次節で、後者の数値クラスタリング手法の解説を行う。

例えば、KDD-[2000..2005] というクエリを用いることにより、KDD-という文字列に 2000 から 2005 までの数値が連接した文字列を取得することができる。ここで問題となるのは、Suffix Array 上の文字列が辞書順に並んでおり、この辞書順が、数字文字列により表現される数値の大きさの順とは異なっているということである。このことは、数値範囲検索において、通常の二分探索法が使用できないことを意味する。例えば、"120" で始まる接尾辞は、"50" で始まる接尾辞よりも、Suffix Array において上方に配置される (図 2 参照) ため、"120" が「"50" より下の範囲」に含まれない。このことは、辞書順に基づく通常の二分探索が、数値の大小比較に基づく検索（この場合、"120" を、「"50" より大きい、すなわち下の位置にある数値」として取得すること）に適用できないことを意味する。従って、数値範囲検索においては、すべての数字文字列を調べ、与えられた範囲内にあるかどうかを調べる。

もう一つの問題として、数値範囲で検索した際、数値範囲に後続する文字列が整列されていないということが挙げられる。何故なら、後続する文字列は、その文字列自体ではなく、前方の数字の種類により整列されているからである。例えば、"[50..2000] 円" というクエリを扱う場合、まず最初に数値範囲 "[50..2000]" が検索され、その後それに連接する

*1 [と] で囲まれた部分は、「左の数値から右の数値までの数値範囲」を示す。

「円」という文字列を検索する。このとき、数値に後続する文字列のインデクスは、その左の数字文字列によりソートされており、後続文字列自体の辞書順には整列していないため、ここから二分探索で「円」を取得することができない。このため、数値範囲を検索した後、動的に後続文字列のソートを行う必要が生じ、このさい $O(a \log a)$ の時間計算量を要する（ここで、a は数値範囲検索結果のサイズ。）

以上の議論を踏まえ、数値検索アルゴリズムは以下のようになる。ここで、クエリが文字列 $s_1.[lb_1..ub_1].s_2.[lb_2..ub_2]...s_n$ （ただし、"." は文字列の連結、lb_k と ub_k は整数値とする。）例えば、クエリ q =KDD-[2000..2005] に対しては、s_1=KDD-, $lb_1 = 2000$, $ub_1 = 2005$, s_2="" (ヌル文字列) となる。（ただし、$n = 2$）. 索引配列 ca の初期値を $ca = sa$ (sa は入力文書全体に対する Suffix Array) と設定し、以下の処理を $k = 1, 2, ..., n$ に対して繰り返し行う。

(1) 配列 ca 内のインデックスから、文字列 s_k を検索する（s_k を接頭辞として持つインデックスを二分探索で求める）。結果の索引範囲を $[x \ldots y]$ とする。s_k に連接する文字列からなる新たな索引配列 sa_2 を $sa_2[i] = ca[x + i] + |s_k|$[*1]のようにして作成する。$ca = sa_2$ とする。

(2) s_k に連接するすべての数値を取得する。これは、ca において、文字 0 と文字 9 を検索し、結果の索引 i_1 と i_2 を用いて、$i_1 \leq j \leq i_2$ となるすべての j について、文字列 $S[sa_2[j]..]$（位置 $sa_2[j]$ を先頭とするコーパス S の部分文字列）の先頭の数字文字列を解釈することにより得られる。結果として得られた数値 d が $lb_k \leq d \leq ub_k$ を満たしていれば、数値の後続文字列への索引 i_3 を取得し、新たな索引配列 sa_3 に加える。

(3) 索引配列 sa_3 を、$S[sa_3[j]..]$ の辞書順に並べ、$ca = sa_3$ とする。((1) に戻る。)

3.1 Number Array

一般に、上記 (2) における範囲 $[i_1 \ldots i_2]$ はそれほど大きくならない（s_1 の後に数字が来る文字列検索結果のみが対象となるため）が、s_1 がヌル文字列の場合（すなわち、「[10..20] 歳」のように、クエリが数値範囲で始まる場合）には、文書中のすべての数字が対象となるため、範囲が非常に大きくなる。このとき、$[i_1 \ldots i_2]$ の範囲すべてについて解析を行うと非常に多くの時間を消費する。このため、我々は、Number Array という配列を新たに追加する。Number Array は、テキスト中の数字文字列のみを対象にインデクスを保持するが、その並びは、辞書順ではなく、**数値順**になっている。図 3 に、Number Array の例を

*1 $|s|$ は文字列 s の長さを表す。

```
...
100 people ...
1005 pieces ...
120 dollars ...
50 dollars ...
...
```

図 2 異なる長さの数値文字列の Suffix Array 上での並びの例

```
...
50 dollars ...
100 people ...
120 dollars ...
1005 pieces ...
...
```

図 3 Number Array 構築の例

示す。通常の Suffix Array(図 2 参照) と順序が異なっていることに注意されたい。

Suffix Array では、（接尾辞への）インデクスは、各接尾辞の辞書順にソートされており、文字列検索は、クエリの各文字を二分探索で発見することを繰り返すことにより行われる。数値範囲検索も、同様に二分探索で行う。すなわち、例えば数値範囲 [2000..2005] の検索では、「2000 より大きい数値のインデクス範囲」「2005 より小さい数値のインデクス範囲」を求め、その重複部分を取得することにより求める。数値が辞書順ではなく、数値そのものの値の順でソートされていれば、前者は数値「2000」を検索すれば、それよりも大きいインデクスが求める範囲となり、前者は数値「2005」を検索すれば、それよりも小さなインデクスが求める範囲となる。

コーパス S 上の Number Array na は、S 中のすべての連続する数字の最初のインデクスを保持しており、配列は、各連続数字文字列の表す数値の昇順でソートされている。すなわち、通常の Suffix Array の構築・検索が「接尾辞の辞書順での比較」に基づき行われるのに対し、Number Array na の構築・検索は、「接尾辞の数値順での比較」に基づき行われる。

4. 数値クラスタリング

Number Suffix Array は、クエリ中で数値範囲を使えるだけでなく、検索結果の中の数字文字列を数値範囲で表現する機能も持つ。これは、複数の数字が与えられた時、それを最大値と最小値、すなわち、例えば 20, 23, 30, 35, 42, 50 という数字を [20..50] と表

現することができるということである。ここで問題となるのは、単純に最小値と最大値をとるだけでは、与えられた数値コレクションの性質を表現できないことがあるということである。例えば、$< 1,2,3,4,1000,1001,1002 >$ という数値リストを [1..1002] と表現すると、1〜4,1000〜1002 という 2 つの範囲に数値が集中しているという情報を落とすことになる。また、少数の外れ値が、数値範囲に影響を与えるという問題もある。例えば、$< 1,50,51,52,53,54,1000 >$ という数値リストを [1..1000] と表現すると、実際には多くの値が 50〜60 付近であるにもかかわらず、範囲はそれよりも遥かに広くなってしまう。これらの問題は、数値のクラスタリングを行うことにより解決できる。

4.1 Dirichlet Process 混合モデルを用いた数値クラスタリング

提案システムが対象とする数値コレクションのクラスタリング問題では、適切なクラスタ数が数値コレクションによって変わるため、事前にクラスタ数を定めるアルゴリズムは適していない。

事前にクラスタ数を定める必要のない手法として、我々は DPM (Dirichlet Process Mixture)[1] クラスタリングを用いる。DPM は混合数が異なる混合モデル同士の確率を比較することのできる確率モデルであり、DPM により数値クラスタ集合をモデル化することにより、最適なクラスタリングをクラスタ数を事前に決めずに確率的な指標により求めることができるようになる。DPM に詳しくない読者は、本節は飛ばし、目的関数 (1) の形のみを確認されたい。

入力として与えられるのは、数値コレクション $x_1,\cdots,x_n$[*1] である。また、各数値 x_i には、隠れパラメータ θ_i が付随していると考える。

Dirichlet Process[4] は、確率分布の確率分布であり、与えられた集合（この問題においては、実数集合）上の離散分布を生成する。G を Dirichlet Process から生成される分布、θ_i が G から生成される実数とし、x_i は θ_i をパラメータとして持つ分布から生成されるとする。すなわち、

$$G \sim DP(\alpha, G_0)$$
$$\theta_i \sim G$$
$$x_i \sim f(\theta_i).$$

Dirichlet Process のパラメータは、基底分布 G_0 と集中度パラメータ α である。Dirichlet Process は、G を積分消去することにより、θ_i **のクラスタ**に確率を与えることができる。ただし、ここで θ_i と θ_j は、$\theta_i = \theta_j$ のときに同じクラスタに入っていると定義する。また、このとき、x_i と x_j も同じクラスタに入っていると定義する。C_j を、$\theta_{c_{j1}} = \theta_{c_{j2}} = \cdots = \theta_{c_{j|C_j|}}$ となるような**添字** $c_{j1}, c_{j2}, \cdots, c_{j|C_j|}$ のクラスタとする。また、C を、すべての C_j のコレクションとする。

我々は、正規分布の DPM（Infinit Gaussian Mixture[9] とも呼ぶ）を用いる。このモデルでは、G_0 と f は共に正規分布であると仮定する。前者のパラメータ（(平均, 分散)）は (μ_1, σ_1)、後者のパラメータを (θ_i, σ_2) と書く。すなわち、$G_0 = \mathcal{N}(\mu_1, \sigma_1), f_i = \mathcal{N}(\theta_i, \sigma_2)$. である。（実行速度の効率化のため、$\sigma_1$, σ_2, μ_1 はそれぞれ固定値に設定する。）これにより、$\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ と $\boldsymbol{\theta}$ の結合分布は

$$p(\boldsymbol{x}, \boldsymbol{\theta}) = \frac{\alpha^{|C|}}{\alpha^{(n)}} \prod_{j=1}^{|C|} (|C_j| - 1)! \frac{1}{\sqrt{2\pi}\sigma_1} exp\left\{ \frac{-(\theta'_j - \mu_1)^2}{2\sigma_1^2} \right\} \prod_{i=1}^{|C_j|} \frac{1}{\sqrt{2\pi}\sigma_2} exp\left\{ \frac{-(x_{c_{ji}} - \theta'_j)^2}{2\sigma_2^2} \right\}$$

となる。ここで、我々の目的はクラスタリングを求めることであるため、これを以下のように θ' を積分消去することで問題を単純化する。（$\mu_1 = 0$ と設定していることに注意されたい。）

$$\frac{\alpha^{|C|}}{\alpha^{(n)}} \frac{1}{\sqrt{2\pi}^n \sigma_2^n} \prod_{j=1}^{|C|} (|C_j|-1)! \frac{1}{\sqrt{1 + |C_j|(\frac{\sigma_1}{\sigma_2})^2}} exp\left\{ -\frac{1}{2\sigma_2^2} \left\{ (\sum_{i=1}^{|C_j|} x_{c_{ji}}^2) - \frac{\sigma_1^2}{\sigma_2^2 + |C_j|\sigma_1^2} (\sum_{i=1}^{|C_j|} x_{c_{ji}})^2 \right\} \right\}$$

これが最大化すべき目的関数（「クラスタリングの良さ」）となる。この関数を $f(C)$ と記述する。ここで、後の議論のため、$g(C_j)$ を

$$g(C_j) = (|C_j| - 1)! \frac{1}{\sqrt{1 + |C_j|(\frac{\sigma_1}{\sigma_2})^2}} exp\left\{ -\frac{1}{2\sigma_2^2} \left\{ (\sum_{i=1}^{|C_j|} x_{c_{ji}}^2) - \frac{\sigma_1^2}{\sigma_2^2 + |C_j|\sigma_1^2} (\sum_{i=1}^{|C_j|} x_{c_{ji}})^2 \right\} \right\}$$

と定義し、$f(C)$ を

$$f(C) = \frac{\alpha^{|C|}}{\alpha^{(n)}} \frac{1}{\sqrt{2\pi}^n \sigma_2^n} \prod_{j=1}^{|C|} g(C_j) \tag{1}$$

と書き換える。

4.2 クラスタ割り当ての貪欲法による発見

アルゴリズムは、目的関数 (1) を最大化させるクラスタリングを求める。ここで、式 (1)

*1 実際には、x_i としては、検索結果中の数字文字列を解釈したものの対数をとった結果を用いる。これは、数値の比較には、差よりも比率を用いたほうが適しているという事前の観察結果による。

```
for l = 0 to N − 1
  for i = 0 to N
    calculate p' = p(i, i + l)
    for j = i to i + l
      calculate p_j = α * ar[i, j] * ar[j + 1, i + l]
    endfor
    ar[i, i + l] ← the best value among p' and p_{i...i+l}
  endfor
endfor
```

図 4　ベースライン（CKY）法

は、各クラスタに対応する g を掛け合わせた形となっていることに注意されたい。各 $g(C_j)$ が、C_j 以外のクラスタ C_i s.t. $i \neq j$ の変化に影響されないため、この目的関数 f は「各クラスタ C_j 毎に $g(C_j)$ を計算し、それを後で掛け合わせる」ことで計算することができる。本稿では、この意味で、この目的関数を「文脈自由である」と呼ぶ。

最適なクラスタリングを発見するため、動的計画法によるボトムアップな最適解探索を適用することができる。本稿では、この手法を**ベースライン手法**あるいは、構文解析に用いられるアルゴリズムとの類似性から **CKY アルゴリズム**と呼ぶ。図 4 に詳細を示す。ここで $p(i, i+1)$ は $C =< x_i \ldots x_{i+l} >$ の場合の $g(C)$ の値、$ar[i, j]$ は、$C =< x_i \ldots x_j >$ の可能なクラスタリングの中で最大の g の値（の積）を格納する配列である。

ベースライン法では、n を数値の個数としたときに $O(n^3)$ の計算時間を要するため、n が大きい場合には多くの時間がかかる。本研究での数値クラスタリングは、検索結果の表示に用いるため、高速な応答が求められる。このため、我々は、アルゴリズムの高速化のため、貪欲計算法を用いた近似解法を提案する。これは、まったく分割のない「全要素が一つのクラスタ」となる状態から始め、各クラスタを、そのクラスタの g を最大化させるような 2 つの領域に再帰的に分割させていく（もし分割させても g が増加しなければ、そこで分割を停止する）という手法である。各クラスタの g は独立であるため、g を増加させることにより、目的関数 f も増加することになる。

具体的には、貪欲法によるクラスタリングは、A を与えられた数値全部のコレクションとするとき、以下に示す関数 $partition(A)$ を呼び出すことで実現される。(C がクラスタリング結果となる。)

$Partition(N)$:　$g(left(N))g(right(N))$ を最大化させる分割 $left'(N)$, $right'(N)$ を求める。もし

$$\alpha \cdot g(left'(N))g(right'(N)) \leq g(N),$$

ならば、N を C に加える (= N の分割を停止し、N を結果クラスタとして加える。).
さもなくば、$partition(left'(N))$ と $partition(right'(N))$ を再帰的に呼び出す。

ここで、α が $g(left'(N))g(right'(N))$ に掛けられているのは、クラスタを分割することによりクラスタ数 $|C|$ が増加し、目的関数 (1) 中、$\alpha^{|C|}$ の α の累乗数が増加するためである。

5. Number Suffix Array によるテキストマイニング

Number Suffix Array の応用として、「用例検索」と「同義語抽出」の二つのテキストマイニングに Number Suffix Array を適用した例を示す。

5.1 用例検索システム Kiwi

用例検索 Kiwi[10] は、与えられたクエリに「連接しやすい」文字列を提示するシステムである。ここで「連接しやすさ」は、Kiwi スコアと呼ばれるスコア関数で定義される。例えば、「*処理」というクエリ[*1]に対して、「情報処理」「知識処理」「自然言語処理」等、「処理」を右に含む文字列を検索結果として返す。Kiwi アルゴリズムの特徴は、「頻度と長さを考慮したスコア付け」と、「分岐数に基づく適切な切れ目の発見」にある。これらは、連接文字列 trie(木構造) を探索する問題であり、この木構造探索は、Suffix Array を用いてシミュレートすることができる。

Kiwi アルゴリズムを Number Suffix Array 上で実装することにより、数値範囲を用例の一部として抽出したり、数値範囲に連接し易い用例を取得したりすることが可能となった。毎日新聞コーパスを利用した用例検索の例を図 5 に示す。例えば「[3..8] 歳」というクエリに対して「息子」「娘」といった連接語が取得されるのに対し、「[20..40] 歳」というクエリに対しては「男性」「女性」「若者」といった連接語が取得されている。

5.2 同義語抽出

我々は、Suffix Array を用いたオンデマンド同義語抽出を提案している[11]。この手法では、クエリ q の文脈（q に連接する文字列）を Suffix Array を用いて取得[*2]、さらにその後、取得された文脈に連接する文字列を逆方向に検索して取得することで、q に類似した言

---

*1 ここで*は、検索したい文字列の位置を示す。

*2 このさい、左右両方向の文脈を取得するため、順方向と逆方向 (Reversed Suffix Array) の 2 つの Suffix Array を用いている。

クエリ：「[3..8] 歳の」
(1)[4..8] 歳のとき／ (2)[3..8] 歳のころ／ (3)[3..8] 歳の時に／ (4)[3..5] 歳の息子／ (5)[6..8] 歳の少女／ (6)4 歳のときから／ (7)[3..5] 歳の女の子／ (8)4 歳のときからピアノを習／ (9)[3..4] 歳のころから母／ (10)5 歳の外国産馬

クエリ：「[20..40] 歳の」
(1)[20..35] 歳の女性／ (2)[20..37] 歳の時に／ (3)[21..40] 歳の男性／ (4)[21..40] 歳の若さで／ (5)[26..36] 歳のベテラン／ (6)[20..40] 歳のとき／ (7)[20..35] 歳のころ／ (8)[20..25] 歳の若者／ (9)[20..37] 歳の誕生日／ (10)[20..31] 歳のころ

図 5 Number-Kiwi の出力例。(2 つのクエリに対する右側連接文字列の違い。)

葉を取得している。例えば、「U.S.」の右文脈として「army」が取得され、「army」の左に連接する文字列として「America's」が取得されることにより、「America's」が「U.S.」の同義語候補として取得される。この手法は、一般的なベクトル空間モデルを用いた同義語取得手法に比べやや劣る精度ながら、「単語のみ」「名詞のみ」といった従来の同義語抽出における候補語への制約を必要としない、適用範囲の広い手法となっており、また、通常の候補語ペア間の類似度を計算する手法に比べて高速に同義語を取得することができる。

我々は、このアルゴリズムを、Number Suffix Array を用いて拡張した。まず、文脈語として、数値範囲を利用できるようにする。たとえば、「10 名」「15 名」「20 名」といった文脈語を自動的に「[10..20] 名」のようにまとめることで、文脈および取得できる同義語候補のカバー率を向上させる。

6. 実　　験

Number Suffix Array のテキストマイニングへの有用性を示すため、同義語抽出の文脈抽出に Number Suffix Array を利用した場合の精度の変化を調べる。

6.1 ベンチマーク課題

実験には、(株) 日本航空インターナショナルの所蔵する航空安全情報文書を用いた。使用した文書集合のサイズは 6.1MByte であり、アルゴリズムの実装は Java, 実験は、Intel Core Solo U1300 (1.06GHz) プロセッサ＋ 2G バイトのマシン上で行った。Number Array のサイズ、順方向、逆方向それぞれの Number Suffix Array 毎に 60,766 であった。

正解データとしては、同様に (株) 日本航空インターナショナルにおいて制作された同義語辞書を使用し、性能を測定した。同義語の例としては、日本語と英語の表記の違いのほか、「LDG」と「landing」のような短縮形等がある。同義語辞書は、$(t, S(t))$ (t:用語、$S(t)$:t の同義語集合) というペアを列挙することで構成されている。((CAPT, {Captain})、(T/O, {Takeoff}) 等。) ペアの数は 404、各ペアにおける平均同義語数は 1.92 であった。精度測定においては、t をクエリとして用い同義語候補のランキングを取得し、$S(t)$ 内の言葉が高いランクに含まれているかどうかを見る。具体的には、各クエリ毎に平均精度 (average precision)[3] を計算し、全クエリの average precision の平均をとって全体の精度とする。システムの返した同義語候補ランキングを $\langle c_1, c_2, ..., c_n \rangle$ とし、正解同義語集合を $S = \{s_1, s_2, ..., \}$ とするとき、平均精度は、

$$\frac{1}{|S|} \sum_{1 \le k \le n} r_k \cdot precision(k),$$

として計算される。ここで $precision(k)$ は、ランキングの上位 k 候補における正解率 (正解と全体候補数の比率) であり、r_k は、k 番目の候補が正解 ($= S$ に含まれる) のとき 1 を、不正解のとき 0 をとる変数である。

表 1 同義語抽出実験の結果。σ_1 は 100.0 に設定。AvPrec は平均精度 (%)、Time はクエリ毎の平均実行時間 (秒) を示す。

Algorithm	AvPrec	Time
Baseline	40.66	2.294
No Clustering	41.30	10.272
Our Algorithm	41.68	7.837

6.2 結果:同義語抽出精度

提案手法の平均精度を、ベースライン (通常の Suffix Array を用いた場合) と、No-Clustering (Number Suffux Array を用いるが、数値クラスタリングを用いない場合) と比較する。

表 1 に結果を示す。Number Suffix Array を用いることにより 0.6 ポイントの、さらに数値クラスタリングを用いることで 0.4 ポイントの精度向上が確認された。

しかしながら、平均実行時間は、通常の Suffix Array を用いた場合と比べ、3.5〜4.5 倍必要となっている。このため、速度低下を防ぐために、数値範囲から始まる検索を制限する (これは例えば、「in 10,000」のような文脈を禁止し、「in 10,000 ft」でなければ採用しない等の制約を課することによって可能になる) 等の措置による速度向上が必要となると考えられる。

6.3 結果：システムの速度及び精度

次に、システム自体の精度・速度を測定する。検索の速度測定のため、前節の同義語抽出

表 2　数値範囲クエリに対する検索時間（秒）。NumStart は数値範囲を先頭とするクエリ、NotNumStart は数値範囲を先頭としないクエリを表す。

Algorithm	NumStart	NotNumStart
Baseline	169.501	0.711
w/ Number Arrays	12.87	0.632

表 3　数値クラスタリングの実行時間（秒）及び各クラスタ割り当ての log likelihood の合計。

Algorithm	Time	Total Log Likelihood
CKY (baseline)	102.638	-168021.7
Greedy	0.170	-168142.2

実験の際に Number Suffix Array へのクエリとして用いられた文字列を保存し、実験のためのクエリ集合とした。クエリ集合から、数値範囲を含むクエリをランダムに 400 個（右文脈用の 200 個と、左文脈用 200 個）選択し、それらの検索速度を測定した。各文脈の 200 個中、100 個を数値範囲を先頭とするクエリ (表中 "NumStart")、100 個をそれ以外のクエリ (表中 "NotNumStart")、となるよう選択した。

また、クラスタリングの性能測定のため、前節の同義語抽出実験中に実際にクラスタリングを行った数値コレクションを保存した。これらの数値コレクションのうち、サイズが 50 以上 1000 以下になるものからランダムに 1000 コレクションを選択した。平均サイズは 98.9 となった。上限の 1000 は、ベースラインアルゴリズム (CKY) の実行時間及び必要メモリが大きくなりすぎないための制限であり、下限の 50 は、少ない要素数では自明なクラスタリングが多くなり、精度の違いが不当に小さくなる可能性があるために設定した。

表 2 に実行時間の比較を載せる。Number Array を使用することにより、数値範囲を先頭とするクエリ (NumStart) の検索時間を大幅に短縮することに成功した。表からわかる通り、NumStart の検索時間は、NotNumStart の検索時間よりも大幅に長いため、Number Array を使用することによりこれを短縮することが数値範囲検索において非常に重要である。

また、表 3 に、クラスタリング結果の比較を載せる。貪欲法が、CKY 法に比べ極めて高速にクラスタリングを行うことができていることがわかる。また、全体の log likelihood を比較すると、両者のクラスタリング結果の差は非常に小さく、このため、貪欲法は、CKY 法に比べ、高速かつ精度のあまり落ちない手法として有用であることがわかる。

7. おわりに

テキスト中の数値表現に対するマイニングを行うための検索システム Number Suffix Array を提案した。提案システムは、Suffix Array による数値検索と、DPM による数値クラスタリングを機能として持つ。また、テキストマイニングへの応用として、数値範囲を用いた同義語抽出及び連接語抽出の例を示し、数値範囲を利用することにより同義語抽出精度が向上することを確認した。今後の課題としては、漢数字を含む数値同義語の扱い等が挙げられる。

謝辞　本研究の一部は、株式会社富士通研究所との共同研究の一環として行われた。

参考文献

1) Antoniak, C.E.: Mixtures of Dirichlet Processes with Applications to Bayesian Nonparametric Problems, *The Annals of Statistics*, Vol.2(6), pp.1152–1174 (1974).
2) Blei, D.M. and Jordan, M.I.: Variational Inference for Dirichlet Process Mixtures, *Bayesian Analysis*, Vol.1(1), pp.121–144 (2006).
3) Chakrabarti, S.: *Mining the Web : Discovering Knowledge from Hypertext Data*, Morgan-Kaufmann Publishers (2002).
4) Ferguson, T.S.: A Bayesian Analysis of Some Nonparametric Problems, *The Annals of Statistics*, Vol.1(2), pp.209–230 (1973).
5) Fontoura, M., Lempel, R., Qi, R. and Zien, J.Y.: Inverted Index Support for Numeric Search, *Internet Mathematics*, Vol.3(2), pp.153–186 (2006).
6) III, H.D.: Fast search for Dirichlet process mixture models, *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 83–90 (2007).
7) Jain, S. and Neal, R.M.: Splitting and merging components of a nonconjugate Dirichlet process mixture model, Technical Report 0507, Dept. of Statistics, University of Toronto (2005).
8) Manber, U. and Myers, G.: Suffix Arrays: A New Method for On-line String Searches, *Proceedings of the first ACM-SIAM Symposium on Discrete Algorithms*, pp.319–327 (1990).
9) Rasmussen, C.E.: The infinite Gaussian mixture model, *Advances in Neural Information Processing Systems, 13th Conference, NIPS 1999*, pp.554–560 (2000).
10) Tanaka-Ishii, K. and Nakagawa, H.: A Multilingual Usage Consultation Tool based on Internet Searching —More than search engine, Less than QA, *Proceedings of the 14th International World Wide Web Conference (WWW2005)*, pp.363–371 (2005).
11) Yoshida, M., Nakagawa, H. and Terada, A.: Gram-Free Synonym Extraction via Suffix Arrays, *Proceedings of the Fourth Asia Information Retrieval Symposium (AIRS 2008) (LNCS 4993)*, pp.282–291 (2008).