対話的テキストマイニングのためのソフトウェアアーキテクチャー

吉田 一星　宅間 大介

Software Architecture for Interactive Text Mining

Issei Yoshida　Daisuke Takuma

ユーザーとの対話的テキストマイニングにおいて, 応答速度に優れ, かつマイニングの特性を考慮したソフトウェアアーキテクチャーを提案する. テキスト文書の性質を活かしたマイニングを実現する上では, 文書中の「高頻度キーワードの頻度分布（ランキング）」の計算コストや, 数値データを扱うデータマイニング手法とテキスト解析との効率的な統合が課題となる. これに対し, ランキング計算のための索引構造, および, データの高速な操作に特化した層とマイニングの様々なタスクを記述する層とを分離する手法を提案する. 実データを用いた検証の結果, ランキング計算時間を従来手法と比べて大幅に短縮するとともに, 提案手法の一応用として検索エンジンの対話的マイニングへの統合の可能性を示した.

We propose new software architecture for interactive text mining which has rapid response speed and which reflects the characteristics of text mining. The major challenges are calculating the costs of " frequency distribution (rankings) of high-frequency keywords "in a document set and integrating data mining and text analysis. We formulate index structures to efficiently calculate rankings, and we define layer architecture that separates the fast manipulation of data from the description of various mining tasks. Experimental evaluation shows that our implementation significantly outperforms conventional methods. Also, as an application of our architecture, we show the potential of integrating search engines with interactive mining.

Key Words & Phrases： テキストマイニング, 高速化, アーキテクチャー, 索引構造
text mining, speeding up, architecture, index structure

1. はじめに

大量のテキスト文書（以下, 文書）から新しい知見を得る「テキストマイニング」の分野において, 言語処理技術によって単語（名詞, 動詞など）・固有表現（人名, 住所など）・係り受け（「パソコンが故障する」といった, 名詞と動詞の関係など）をキーワードとして抽出し, それらを文書の属性と見なして統計計算などの「データマイニング」を適用するアプローチ［1］［2］が, 現在多くのテキストマイニングシステムに採用されている.

本論文では, 対話的テキストマイニング, すなわちユーザーの問い合わせに対してデータマイニングの結果をリアルタイムに計算して返す機能について論じる. ユーザーとの対話性において良く似たタスクに, Web検索に代表される情報検索が挙げられる. 主にキーワードによる検索問い合わせに対して, 適合する文書のリストを返すこのタスクについては, SMART［3］以来非常に多くの手法が研究されてきた. 本論文で述べるマイニングは, 検索などの手段で絞り込まれた文書集合に特徴的なキーワードを「発見」するタスクである. 一般に, 対話的でないマイニングでは, 数時間あるいはそれ以上の時間を要するバッチ処理で計算した結果を閲覧する. これに対し, 対話的マイニングでは情報検索に近いレスポンスタイムで結果が得られ, マイニングの結果をすぐ次のマイニングに活かすことができるため, 発見のための思考が中断しない. 典型的な例として, 発見したキーワードを検索条件に追加して, 直ちにマイニングを続行できる.

対話的マイニングにおいては, 以下の二点が問題となる. 第一に, データ量の増大に伴う応答時間の増大である. 文書集合の特徴を知る上で最も基本的なマイニング機能と考えられる「多くの文書に共通して出現する, 高頻度キーワードの頻度分布（ランキング）」について, システムが扱うデータ量は情報検索のそれと大きく異なる. 情報検索とマイニングが対象とす

提出日：2006年8月31日　　再提出日：2006年12月17日

るデータ構造は, いずれもキーワードと文書の対応行列(キーワード文書行列)[4]である. 情報検索とは, この行列の1行(AND/ORなどの複合条件の場合は, 構成する最小要素の個数分の行)を抜き出すタスクに他ならない. これは, 大規模な文書集合であっても計算機の主記憶上で容易に保持・計算ができる.

一方, ランキングではこの行列全体, すなわち, 多数あるキーワードのそれぞれに対して文書のリストが付随する構造全体を分析しなければならない. しかし, 大規模な文書集合に対しては, この行列を主記憶上に載せることはできない. 処理すべきデータ量のこの本質的な差異が, 情報検索に比べてマイニングの対話的性能の実現を難しくしている. ランキングについては文書サンプリング[5]やSQLを用いた計算手法[6]が提案されているが, いずれも対話的マイニングの要件を満たすためには課題がある.

第二に, テキスト処理の結果をデータマイニングに適用する際, 統計計算などのデータマイニング手法の殆どは数値データを対象としているため, 言語処理が出力した文字列情報をそのまま用いることは難しい. このため, 文字列を数値に変換したデータに対してデータマイニング手法を適用するのは自然であるが, データマイニングと対話的テキストマイニングの効率的な統合について, 大規模データに対する実証を伴って論じた文献は, 筆者の知る限りまだ存在しない.

この論文において, 筆者は以下の二点を主張する. 1つは, キーワードの頻度順にソートされた索引構造を用いることにより, 頻度の単調性(詳細は後述, [7])を利用して, 解析対象文書集合中のランキングを効率的に計算する手法を考案し実装した. これにより, 従来のサンプリング手法やSQLに比べて, ランキング計算時間を飛躍的に向上させた.

もう1つは, データマイニングの手法を高速に実現するため, データの高速な操作に特化した層(索引API)と, マイニングの様々なタスクを記述する層(マイニングAPI)を明確に分離したアーキテクチャーである. 索引APIは, キーワード文書行列への高速なアクセス手段であり, ランキングの計算効率も考慮して設計されている. また, マイニングAPIは, 時系列解析や相関分析といったタスクを, 索引APIの機能を組み合わせて提供する. このアーキテクチャーによって, 新しいマイニングをタスク単位でプラガブルに追加可能となった. 言語処理技術においては, このような柔軟性を持つアーキテクチャーとして, IBMが推進するUIMA [8] や, GATE [9]が知られている. 本論文の提案は, 言語処理の出力結果の解析を念頭に置いた, データマイニング側のアーキテクチャーであると言える. 本論文ではこのアーキテクチャーの一応用として, 既存の検索エンジンとランキングとの統合を考察する.

筆者は, 上記二点を備えたJava™ライブラリを実装し, 本提案の有用性を検証した. このライブラリは, 東京基礎研究所が研究開発しているテキストマイニングシステムTAKMI [1] [4]の最新バージョンに用いられている.

2. 対話的テキストマイニング

2.1 対話的テキストマイニングの概要

本論文で対象とするテキストマイニングは, ユーザーの問い合わせに対してリアルタイムに計算結果を返すものを指す. 現在, [1]をはじめとする多くのテキストマイニングシステムが, **図1**に示すような「前処理」と「対話的処理」の2つから成る仕組みを採用している.

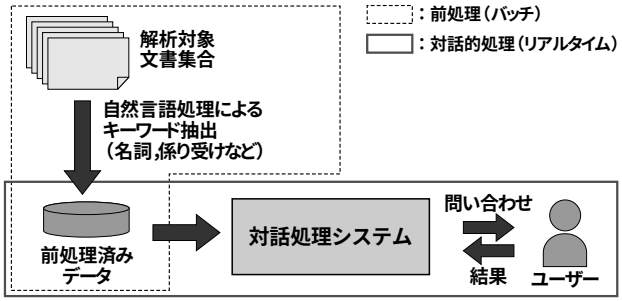


図1. 対話的テキストマイニングの概要

前処理では, 解析対象の文書を, 自然言語処理の技術を用いて解析する. 解析対象の文書集合として, コールセンターのコールログ, 医療文献, Webページなど, テキストを含む任意のデータが解析対象となりうる. 本論文の提案は解析対象分野に依存しないため, 以下コールログを例にして説明する. 先ず, 文書集合の各文書に対し, 文書中のテキストを, 自然言語処理の技術を用いて解析する. この解析の出力結果として以下のようなものが挙げられる[10].

- 単語：名詞, 動詞など
- 固有表現：住所, 人名など
- 係り受け：例「ソフト…アンインストールする」

これらの出力結果は, 各文書に対する付加情報と呼べるものである. 本論文ではこれらをまとめて「キーワード」と呼ぶこととする.

各文書から抽出されたキーワードは, 文書とキーワードとの対応を保持する何らかのデータ構造に, 前処理済みデータとして格納される. **図2**に, 保持される内容の一例を示す. 通常, 前処理は, 文書集合全体に対するバッチ処理として実施される.

表の各行は, 各文書から抽出されたキーワードの

リストを示す. また, [名詞] [係り受け] といったラベルは, 各キーワードの種別(カテゴリ)を示す.

文書1	[名詞] インターネット	[名詞] メール	[動詞] 受信する
文書2	[名詞] インターネット	[名詞] モデム	[動詞] 接続する
文書3	[名詞] ハードディスク	[係り受け] ソフト... アンインストールする	
文書4	[名詞] インターネット		
文書5	[名詞] ハードディスク	[動詞] 接続する	

図2. 文書とキーワードの対応表の例

前処理の完了後, 対話的処理において, この対応表に, 相関ルール検出, クラスタリングなどのデータマイニングの既存手法を適用する. 一般にこれらの手法は高い計算コストを要するが, テキストマイニングで要求される機能の多くは, 文書集合におけるカテゴリ別キーワードの, 高頻度キーワードの頻度分布(ランキング)の計算に帰着できるため, ランキングの対話的性能を高速化できれば良い. **表1**に, 図2の文書集合例に対する, [名詞] カテゴリのランキングの結果を示す. 各キーワードの出現文書数を頻度と呼び, キーワードは頻度の降順に並ぶ. 頻度による順位だけでなく, 頻度の値そのものも分析上有用であるため, 本論文では頻度の値をキーワードのランクと見なす.

なお, この頻度の定義から, ある一文書中に同一キーワードが複数回現れても1回とカウントしている.

ユーザーは, 処理結果を見てから別の処理依頼を行う. ランキングを例にとると, " ハードディスク "を含む文書だけを検索し, 検索結果として得られる文書集合に対して [動詞] カテゴリのランキングを求める, という処理が考えられる. これにより, ハードディスクと関連の強い動詞(例えば,「増設する」「認識する」)を得ることが期待される. このような処理を対話的に繰り返すことによって, 解析対象の性質を効率良く調査することが可能となる.

表1. ランキングの例

[名詞カテゴリーのキーワード]	頻度
インターネット	3
ハードディスク	2
メール	1
モデム	1

2.2 従来研究

大規模なデータに対する対話的処理として, 公共医療文献データベースMEDLINEを対象としたマイニングが実現されている [5]. ランキングなどの計算には対象文書のサンプリングを行っている. しかし一般に, サンプリングでは検索のための索引構造に対するランダムアクセスが発生するため, 計算コストが文書数に線型になるだけでなく, 1文書あたりの計算コスト(主に入出力コスト)が大きくなる. 従って, サンプル文書数を大きくするとリアルタイム性を保てなくなる.

また, リレーショナルデータベース(RDB)に図2の対応表と同等な情報を格納し, SQLを用いてサンプリングなしで計算を行う手法が提案されている [6]. しかしこの手法には, 計算の過程で得られる有用な中間情報を得るのが難しいという課題がある. **図3**は, 3個のキーワードをすべて含む文書を検索したときの, 中間情報つきの検索結果の例である. 各キーワードが, 総文書(324,677文書)中" リカバリー ","" バックアップ ","" 再起動 "の順に多く出現すること, これらをすべて含む文書はいずれかを含む文書に比べて非常に少ない割合であること, などが見て取れる. SQLでこれを実現するには, 各キーワードに対する件数を求める count 文を発行する必要があるため, 効率が大幅に低下する.

文書数: 324677		
検索条件: 該当文書数: 83件		
すべて含む 83件	キーワード : リカバリー を含む	17173件
	キーワード : 再起動 を含む	5356件
	キーワード : バックアップ を含む	7311件

図3. 検索の中間結果の例

3. 高速化のためのアーキテクチャー

本章では, 2章で述べた対話的マイニングを実現するため, データの高速な操作・マイニングのタスクの記述・アプリケーションを明確に分離したソフトウェアアーキテクチャーを提案する. 3.1節では, ランキング計算のための索引構造およびその実装を, 3.2節では, データの高速な操作を提供するインタフェースを, 3.3節では, 上記アーキテクチャーの全体像について述べる.

3.1 索引構造の設計と実装

本節では, 1つのカテゴリに対してランキングを効率良く計算するための索引構造を述べる. 3.3節に示すように, この構造は多くのマイニングに対して有用である.

先ず, ランキングを形式的に定義する. 先ず, 2章で述べた解析対象文書集合をD_{ALL}と書くことにし, D_{ALL}内の各文書から言語処理によってキーワードが抽出済みであるとする. ランキングは,「与えられたカテゴリC, D_{ALL}の任意の部分集合D, および正整数kに対し, Cのキーワードを Dにおける頻度降順で上位k個取得すること」と定義される. ここで, $D = \{d_1, d_2, ..., d_m\}$ を文書集合, $T(C) = \{t_1, t_2, ..., t_n\}$ をカテゴリCのキーワード全体の集合とする. また, $t \in T(C)$ に対

し, Doc(t, D):= {d∈D | 文書dがキーワードtを含む} と定義する. ランキングとは, T(C)の要素を#Doc(t, D)すなわちDoc(t, D)の要素数でソートしたときの, 上位k個を取得することに他ならない. このモデルにおいて文書集合Dを1つ与えることは, ユーザーが検索条件を1つ与えることに対応する. すなわち, 検索で見つかった文書集合中に頻出するキーワードを求めることの定式化となっている.

あるスコアに対して上位k件を取得するという一般的なタスクに対し, スコア #Doc(t, D)の単調性を利用する手法が知られている[7]. ここで単調性とは, キーワードtを任意に1つ固定したとき, 文書集合D, D'に対して「$D \subset D' \Rightarrow$ #Doc(t, D)≦ #Doc(t, D')」が常に成立することをいう, 筆者はこれをランキング計算に応用するとともに, 他のマイニングにも用いるための索引構造「ランキング索引」を設計した. ランキング索引は, 情報検索で一般的に用いられる転置索引[4]と同様の「与えられたキーワードを含む文書を高速に検索する」ためのデータ構造であるが, ランキング索引ではリストをキーワードの頻度順にソートすることによって, 以下の高速なアルゴリズムを可能にしている.

図4に, ランキング索引の概念図を示す. この索引は, ランキング計算に必要なキーワードと文書の対応を, 予め計算済みの頻度情報とともに保持している. この索引は各カテゴリCに対し, ランク表と分布表の2つの表から成る. ランク表は, CのすべてのキーワードをD_{ALL}内の頻度降順にソートし, キーワードtと頻度 f := #Doc(t, D_{ALL})のペア(t, f)を並べた構造, および各ペアから分布表へのポインタを持つ. 分布表は, ランク表の各ペア(t, f)に対し, Doc(t, D_{ALL})に含まれる各文書の文書ID(文書を一意に指定する整数で, 予め付与しておく)のリストを持つ. 図4の例では, #Doc(" インターネット ", D_{ALL}) = 105672であり, キーワード" インターネット "は文書IDが0, 1, 3, 4, 7, ..., の文書に含まれる. 以下, 文書と文書IDを同一視して, Doc(t, D) = {dt_1, dt_2, ..., dt_p}の各要素 dj は文書IDであるとする. この索引を, 前処理時に作成しておく.

次に, 対話的処理時にこの索引を用いるランキング計算のアルゴリズムを, **図5**に示す. ランク表と分布表を上から順に読んで, D内の頻度順で上位k個が求まったか, またはランキング索引のすべてのキーワードを読み終えた時点で停止する.

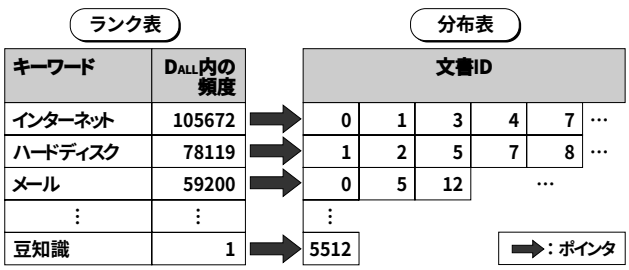


図4. ランキング索引の概念図

```
入力: k, D={d1, ..., dm}
出力: TopK = { (t1, f1), ..., (tk, fk) | f1≧f2≧...≧fk }

TopK = φ // (t, f)をfの降順に整列して保持するリスト
for each (t, f) in ランク表 do  // 頻度の降順に取り出す
 if #TopK<k then TopK = TopK ∪{(t, #(Doc(t, DALL)∩D))}
 else
  (tk, fk) = (TopK の要素のうち f が最小な任意の要素)
  if f < fk then return TopK  // @
  else TopK = TopK ∪{ (t, #(Doc(t, DALL)∩D)) }−{(tk, fk)}
end for
return TopK
```

図5. ランキング計算アルゴリズム

上で述べた文書集合についての頻度の単調性により, ある時点で既にk個の候補が得られ, かつTopK内の頻度最小の要素(tk, fk)の頻度fkよりも今読み込んだランク表の要素(t, f)の頻度fのほうが小さい場合は, ランク表をそれ以上読む必要がないため停止する(図5中の@).

実装では, 各カテゴリのランク表や分布表をファイルにシリアライズする. 分布表内では, 集合演算∩の効率を高めるために, 各Doc(t, D_{ALL})内の文書IDを昇順にソートしておく. ランク表から分布表へのポインタは, ランク表の累積頻度と1文書IDあたりの記憶容量(本実装では4bytes)からリアルタイムに計算できる. このため, 実装では頻度の代わりに, 先頭のキーワードからの累積頻度をランク表に記録している. これらの結果, ランキングの計算はランク表と分布表への順次アクセスだけで実行される. 単調性による停止条件と合わせて, ランダムアクセスを伴う文書サンプリングに比べて一般的に非常に高いパフォーマンスが得られる. 情報検索における転置索引は頻度順でソートされていないため, 停止条件が使えないか, 何らかの停止条件を与えたとしても低速なランダムアクセスを避けられない. 情報検索のタスクは, 1個のキーワードに対して1回のランダムアクセスで文書のリストを取得できるのに対し, ランキングでは他のキーワードとの関連を見ながら全体の結果を計算しなければならないことに注意されたい.

例外的に, Dの文書数が極めて小さく, かつキーワードの異なり数(ランク表に含まれる(t, f)の個数)が極端に大きい場合に, 図5の終了条件@に到達するまでに膨大な量のランク表および分布表を読むことがありうる. この場合に備え, 各文書IDから, その文書に含まれるキーワードのリストを求める別の索引構造「文書索引」も用意し, 文書索引にランダムアクセスを行って計算を行う. 例外的な場合かどうかの判

定にも興味あるアルゴリズムを用いているが, その紹介は別の機会に譲る.

以上の索引構造は, 前処理時に文書集合全体に対してバッチ処理で構築される. ソート済みの索引を構築するコストは単純な転置索引のそれよりも高いが, 本論文で述べた対話的マイニングはランキングをはじめとするリアルタイム計算の実行時間を最重視している.

3.2 索引API

3.1節で述べた図5のアルゴリズムで, キーワードの頻度fの代わりに文書IDリストを出力結果TopKに保持することによって, キーワードから文書IDリストへのポインタも得られる. また, 文書索引は文書IDからキーワードリストへのポインタそのものである. 従って, 3.1節で論じた索引構造は, キーワード文書行列[4]に相当する双方向のマッピングを提供する. 一方で, 3.1節で述べた索引構造は一実装であり, 他の実装, 例えば更新などのデータ管理の点で有利なRDBを使用可能にする余地を残すべきである.

これを踏まえ, ランキングをベースにして, 文書とキーワードの情報を返す索引API(Application Programming Interface)を定義した. 索引APIが提供する機能の例を**図6**に示す. キーワードも索引内では数値IDで記録されるため, 元の文字列とのマッピングを提供するキーワード変換APIが用意されている.

索引APIによって文書とキーワードを数値IDとして操作できるため, 既存のデータマイニングの手法を比較的容易に適用可能である. 図6で,「文書IDつきランキング」は, ランキング結果の各キーワードに対してそれを含む文書IDのリストも結果に含まれている. これを用いて, 引き続き文書IDに関する操作, 例えば別のカテゴリに対するランキングを用いてキーワードの共起計算などを行うことができる.

「キーワード検索」は, 入力キーワードに対する検索機能そのものである. 筆者の提案する, マイニングにおける検索機能とは,「検索キーワードに対してすべての適合文書を高速に返す機能」である. 戻り値を数値のリストとするAPIを提供することによって, 対話的マイニングに必要な高速な実装が実現しやすくなる.

キーワード検索は, ランキング索引の他に, キーワードからランク表上の対応位置へのポインタを保持することで可能となる. 与えられたキーワードに対し, このポインタへのアクセス, ランク表へのアクセスは定数時間であるため, 分布表内の文書IDリストを高速に取得できる.

2.2節で述べたような検索の中間結果の取得に対しても, 本索引APIを用いると容易に実現可能である. 集合演算を用いて最終検索結果を求める際に, 中間結果の文書IDの数を保持しながら計算すれば良い. ここで, 文書IDを数値で定義しているため, 集合演算を効率良く実行できること, および, 戻り値が数値のリストであることが高速化のために本質的である. 検索エンジンなどの従来の情報検索システムが文書IDを返す際のAPIは, 文書を単位とするイテレーター(呼び出し毎に次のIDを返す)であることが多いが, 実装に依存して大きなオーバヘッドを生むため適切でない.

「全文検索」も同様に文書IDのリストを返すが, キーワード検索と異なり, 検索条件を表す文字列を入力とする. これは, 他の検索エンジン実装と連携する場合でも, その検索エンジンの照会構文をそのまま記述できるようにするためである. ただし, 従来の検索エンジンが提供するAPIはデータマイニングとの連携を想定していないため, 文書IDが数値でなく(一般にURL などの不定長文字列), かつスコア順で上位固定件数の文書しか返さない. このことは「検索」の索引APIを適用する上での課題となる. 4章で, 検索エンジンとの連携について詳述する.

ランキング	入力	C: カテゴリ, D: 文書IDの集合, k: 正整数
	出力	Dにおける頻度順で上位k個のキーワードIDとその頻度
文書IDつきランキング	入力	C: カテゴリ, D: 文書IDの集合, k: 正整数
	出力	Dにおける頻度順で上位k個のキーワードIDとそのキーワードを含む文書IDのリスト
キーワード検索	入力	C: カテゴリ, kid: キーワードID
	出力	カテゴリ C 内でキーワード kid を含む文書IDのリスト
キーワード変換	入力	kid: キーワードID
	出力	kidに対するキーワードの文字列表現
全文検索	入力	S: 検索条件を表す文字列
	出力	検索条件に適合する文書IDのリスト

図6. 索引APIが提供する機能の例

3.3 マイニングアーキテクチャー

3.2節で述べた索引APIはデータの基本的な操作を提供するが, マイニングのタスクはより複雑な処理を行うものが多い. 頻繁に用いられるタスクについては, その組み合わせを1つの上位機能として提供することによって, アプリケーションがタスクを容易に利用することができる.

そこで, マイニングのタスクを記述するための層「マイニングAPI」を新たに定義し, 索引API・マイニングAPI・アプリケーションの三層から成るアーキテクチャーを提案する. **図7**に, 三層の関係を整理した. この階層構造を, 対話的テキストマイニングのためのマイニングアーキテクチャーと呼ぶ.

一例として, マイニングAPIの1つである「カテゴリビュー」について説明する. これは, 与えられたカテ

ゴリに対し, 与えられた検索条件を満たす文書集合内のランキングを表示するという機能であり, 以下の処理を順に実行して結果を得る.

(1)検索条件内の個々のキーワードで検索し, 文書IDのリストを取得
(2)文書IDのリストに対する集合演算(AND/OR/NOT)で, 検索結果の文書集合を取得
(3)検索結果の文書集合中のランキングを取得
(4)ランキングで得られたキーワードIDを文字列表現に変換

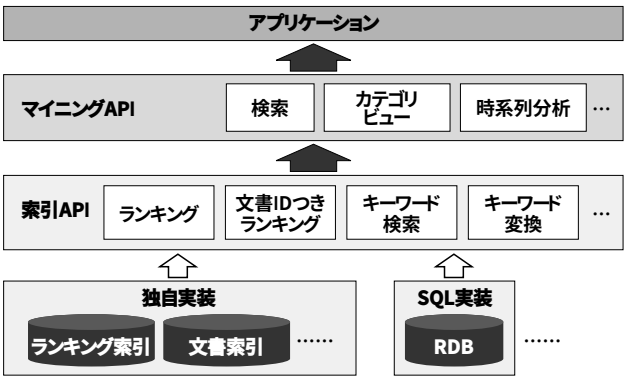


図7. マイニングアーキテクチャー

この処理中, (1)(3)(4)はそれぞれ索引APIのキーワード検索・ランキング・キーワード変換機能によって実現している. (2)はマイニングAPIのレベルで実装している. 実際には, (2)の機能は検索APIとして独立したマイニングAPIとなっており, 他のマイニングAPI, 例えば時系列解析などでも用いられている. 同様にして, 任意の索引APIの組み合わせから成る処理を, 新たなマイニング機能としてマイニングAPIにプラガブルに追加できる.

また, 3.2節で述べたように個々の索引APIが高速なため, 各タスクの高速な実装が可能なことも, 三層構造の大きな利点である.

4. 評価

4.1 ランキング索引のパフォーマンス

3.1節で提案したランキング索引のパフォーマンスを測定し, サンプリング手法およびRDB実装と比較した. サンプリングは, 解析対象文書集合からランダムに10,000文書(対象文書がそれ以下の場合はすべて)を選び, 2.2節で述べた文書索引を用いてキーワードの頻度分布を計算する. また, RDB実装は[6]に拠った. 実験に用いた計算機は標準的なPCである(CPU:Pentium® R 2.13GHz, Memory:1.5GB).

IBM PCコールセンターのコールログ324,677文書に対し, [一般名詞][ソフトウェア][機種名]の各カテゴリについて, 頻度順で上位1000個のキーワードを求めるランキングの計算時間を測定した. 解析対象文書は,“ PC ”(高頻度語, 全文書中19,810文書に出現),“ キーボード ”(中頻度語, 同4,810文書),“ 煙 ”(低頻度語, 同4文書)の各キーワードの検索結果を用いた. **図8**に実験結果を示す. この結果から, 提案手法がサンプリング, RDB実装に比べて数倍から数十倍高速なことがわかる.

表2. 比較実験に用いたランキング計算手法

手 法	特 徴
提案手法	キーワード索引+単調性
サンプリング	文書索引+サンプリング
RDB実装	RDB索引+SQL

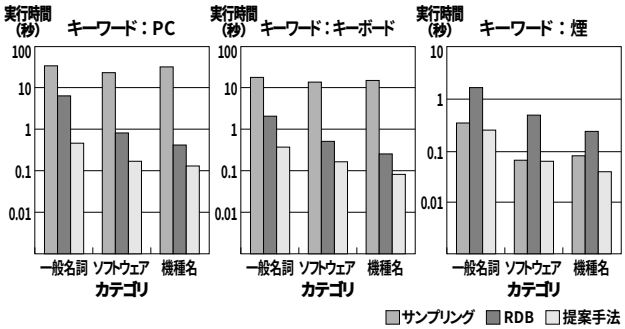


図8. ランキング計算時間の比較(実行時間軸は対数目盛り)

4.2 検索エンジンとの統合について

3.3節で述べたマイニングアーキテクチャーの自然な応用として, 索引APIの「全文検索」機能の実装に, 汎用の検索エンジンを用いることが考えられる. IBMの検索エンジンミドルウエアOmniFind™ [11]をはじめとする検索エンジンの多くは, 単純なキーワード検索より強力な検索手段を提供している. 例えば, 部分一致検索・特定の文書フィールド(日付, タイトルなど)に対する検索・数値の範囲検索が挙げられる. これがマイニングの対象となる文書集合を特定するために有用であることは明白であろう.

本提案のマイニングアーキテクチャーにおいて, 図6で示した全文検索の索引APIを検索エンジンで実装するためには, 検索エンジンから検索条件に適合する文書IDをすべて取得するようにすれば良い. 具体的には, 各文書を検索エンジンの索引に取り込む際に, 予め採番しておいた本アーキテクチャーの文書IDを文書のメタデータとして保管するようにする. 検索エンジンは一般に, 検索に適合した文書のメタデータを返す機能を持っているので, 文書IDをメタデータとして返すことができる.

しかしこの方式を用いた場合, 文書IDの取得に要するオーバヘッドは, 本アーキテクチャーのランク表

および分布表を用いて文書IDを取得するのに比べて大きい. この違いは, メタデータなどデータ構造のオーバヘッドの有無などに由来する. 1文書あたりの文書ID取得コストは, 何らかのスコアに対して上位の少数の文書だけを対象とする検索タスクでは, 検索エンジンを使った場合でも問題にならないほど微小である. しかし, 対象文書数が大きくなると, このコストが対話性を保つ上で無視できなくなる. 例えば, 1000文書あたりの文書ID取得時間が1秒であったとする. これは検索の上位100件を表示するタスクでは問題ないが, 4.1節の実験でPCを検索した場合には, 文書ID取得だけで20秒近くかかってしまうことになる.

文書IDの取得コストは検索エンジンの個々のアーキテクチャーや実装に大きく依存するため統一的な議論は難しいが, 安定して対話的にマイニングを行うためには, 検索適合文書の数に上限を設定する必要がある.

5. おわりに

本論文では, 文書集合に対する高頻度キーワードの頻度分布(ランキング)の計算の高速化と, そこから発展してデータマイニングを対話的テキストマイニングと統合するためのソフトウェアアーキテクチャーを論じた. 高速化では, キーワードの頻度降順にソートされた索引を用いて, 頻度の単調性を用いた効率的な探索を可能にした. これにより従来手法に比べ飛躍的に速度を向上させた. また, 数値データ操作のための索引APIと, マイニングのタスクを記述するマイニングAPIとを分離することによって, マイニングのタスクの追加が容易となった.

大規模キーワード文書行列に対する研究は数多い. 代表的なものの1つに, Web文書集合の行列から, リンク構造を基にして各文書のスコアを計算する手法PageRank [12]が挙げられる. PageRank が各文書に静的な, すなわち, 検索条件などで変化しないスコアを付与するのに対し, 本論文で議論したマイニングは, ユーザーからの問い合わせに対して, そのキーワードで対象文書集合を絞った上で, ランキングを動的に計算している点が異なる.

ランキングの関連研究として, 複数のカテゴリからの検索を同時に実行し, 頻度分布をユーザーに提示する「ファセット検索」がある[13][14]. これは本論文におけるキーワードを, テキスト処理に無関係なメタデータ(データが画像であれば, テーマや撮影者, 媒体種類など)に置き換えたものと機能的には等価である. しかし, ファセット検索のキーワードの種類は, 数十から数百と仮定されていることが多い. これに対し, 本論文で述べた手法は, キーワード種類が数万に達するようなカテゴリ(「名詞」「化合物名」など)にも適用可能である.

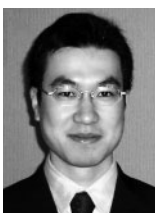
高速化については, 複数マシンを利用した分散環境上でのアルゴリズムの研究が報告されている[15][16]. 各マシン上での計算は本論文の手法が自然に応用できるが, 一般に分散環境では正しい解を得るためのコストが高くなるため, 実用上は近似解を求めることが多い.

検索エンジンとの統合は, 検索エンジンが文書集合を絞り込むための強力な手段であるため, 応用範囲が広いタスクである. 検索条件に適合する全文書のIDを返すという検索APIによって, 検索エンジンとマイニングが統合可能なことを示した. ただし, 現状では大規模データにおいて「検索条件の記述能力と速度とのトレードオフ」が生じる. この課題は, ユーザーの検索要求に適合する上位文書を提示すれば良いという検索のタスクと, 巨大な対象データを統計処理するというマイニングのタスクを融合していく上で, 今後も重要になると考える.

参考文献

[1] Tetsuya Nasukawa and Tohru Nagano:“ Text Analysis and Knowledge Mining System,” *IBM Systems Journal*, Vol.40, No.4, pp.967-984 (2001).
[2] Haralampos Karanikas, Christos Tjortjis and Babis Theodoulidis:“ An approach to Text Mining using Information Extraction,” *4th European Conference on Principles and Practice of Knowledge Discovery in Databases* (2000).
[3] Gerard Salton:“ The SMART Retrieval System : Experiments in Automatic Document Processing,” Prentice-Hall, Inc. (1971).
[4] Naohiko Uramoto, Hirofumi Matsuzawa, Tohru Nagano, Akiko Murakami, Hironori Takeuchi and Kohichi Takeda:“ A text-mining system for knowledge discovery from biomedical documents,” *IBM Systems Journal*, Vol.43, No.3, pp.516-533 (2004).
[5] Akihiro Inokuchi and Kohichi Takeda:“ SQL-based Aggregation for Text Mining,” *IBM Research Report* (2005).
[6] 北 研二, 津田 和彦, 獅々堀 正幹:情報検索アルゴリズム, 共立出版, ISBN4-320-12036-1 (2002).
[7] Ronald Fagin, Amnon Lotem and Moni Naor:“ Optimal aggregation algorithms for middleware,” *Journal of Computer and System Sciences*, Vol.66, No.4, pp.614-656 (2003).
[8] UIMA Java Framework: http://uima-framework.sourceforge.net/ (2006.11.22).

［9］GATE: http://gate.ac.uk/（2006.11.22）.
［10］長尾真：自然言語処理, 岩波講座ソフトウェア科学15, ISBN4-000-10355-5（1996）.
［11］IBM WebSphere Information Integrator OmniFind Edition: http://www.ibm.com/jp/software/websphere/ii/omnifind/（2006.11.22）.
［12］Larry Page, Sergey Brin, Rajeev Motwani and Terry Winograd:“ The PageRank Citation Ranking: Bringing Order to the Web ”, http://dbpubs.stanford.edu:8090/pub/1999-66（1998）.
［13］Ka-Ping Yee, Kirsten Swearingen, Kevin Li and Marti A. Hearst:“ Faceted metadata for image search and browsing, ”*Proc. Conference on Human Factors in Computing Systems（CHI 2003）*, pp.401-408（2003）.
［14］Catherine Plaisant, Ben Shneiderman, Khoa Doan and Tom Bruns:“ Interface and Data Architecture for Query Preview in Networked Information Systems, ”*ACM Trans. Inf. Syst.（TOIS）*, Vol.17, No.3, pp.320-341（1999）.
［15］Sebastian Michel, Peter Triantafillou and Gerhard Weikum:“ KLEE: A Framework for Distributed Top-k Query Algorithms, ”*Proc. 31st International Conference on Very Large Data Bases（VLDB 2005）*, pp.637-648（2005）.
［16］Pei Cao and Zhe Wang:“ Efficient top-K query calculation in distributed networks, ”*Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing（PODC 2004）*, pp.206-215（2004）.



日本アイ・ビー・エム株式会社
東京基礎研究所
I&I．ナレッジ・インフラストラクチャー
副主任研究員
吉田 一星 Issei Yoshida

［プロフィール］
2001年，日本IBM入社．テキストマイニング・システム IBM TAKMI Text Analysis and Knowledge Mining の設計・開発を担当．データマイニングとテキスト解析の統合のための，ソフトウェアアーキテクチャーおよびアルゴリズムの研究に従事．
issei@jp.ibm.com

日本アイ・ビー・エム株式会社
東京基礎研究所
I&I．ナレッジ・インフラストラクチャー
副主任研究員
宅間 大介 Daisuke Takuma

［プロフィール］
テキストや音声など，非定型データを分析するためのインデックス技術を研究している．特に，テキストデータについては，実用的な分析シナリオに沿ったアプリケーション開発も担当し，企業に蓄積された顧客の声データから，商品の不具合や要望，FAQを見つけ出すためのテキストマイニング・ツールを開発した．
ta9ma@jp.ibm.com