頻出言語パターンのマイニング技術とその応用

工藤　拓

単語や文節，係り受け関係といった特定の言語パターンの頻度を数え個々の表現の分布を調査することは，自然言語処理でもっとも基本的な処理であることは疑いの余地はなく，その応用範囲は多岐にわたる．処理の内容はいたって単純であるが，大規模データを扱えるようアルゴリズムのスケーラビリティーを確保することは容易ではない．本稿では，頻出言語パターンマイニングアルゴリズムを概観するとともに，マイニングアルゴリズムを応用した機械学習手法について紹介する．

キーワード：頻出パターンマイニング，SE-Tree, PrefxSpan，最右拡張，機械学習，ブースティング

1．はじめに

自然言語処理における「頻出言語パターンのマイニング」とは，係り受け解析・構文解析済みの半構造テキストデータから頻出する部分パターン（例えば部分木）を効率よく列挙する技術の総称である．マイニングの結果は，そのまま人間が理解できる形式であるため，人間の意思決定をサポートするデータとして用いられる．特に近年，ブログといったWeb上のテキストからユーザの主観的な嗜好性を抽出する目的で係り受け構造を部分パターンとしたテキストマイニング手法がさかんに研究されるようになった．

頻出言語パターンマイニングの問題設定そのものは単純であるが，大規模データを扱えるようマイニングアルゴリズムのスケーラビリティーを確保することは容易ではない．もっとも単純なマイニングアルゴリズムは言語パターンを事前にすべて枚挙し，各パターンの頻度を数えることであるが，この手法がスケールしないことは容易に想像できるだろう．例えば，ある構文解析済みの木から部分木を列挙した場合，その部分木の数は木のノード数に対し指数的に増加する．一般にこのような完全枚挙の方法はNP困難であることが知られている．しかし，頻出するパターンのみを枚挙する場合には，頻出しないパターンを早期に枝狩りすることで，枚挙すべきパターンを劇的に減らすことが可能となる．頻出するという事前知識をいかに利用し，探索空間を小さくするかがマイニングアルゴリズムの腕の見せ所である．

本稿では，頻出言語パターンマイニングアルゴリズムについて概観するとともに，頻出言語パターンマイニングの言語処理，機械学習への応用を紹介する．

2．頻出言語パターンマイニング

2.1　問題設定

頻出言語パターンマイニングの工学的定義は以下のようになる．

問題1　ある構造化（もしくは半構造化）された事例（＝文）の集合（＝テキスト，コーパス）が与えられたときに，ξ個以上の事例に含まれる部分言語構造（＝部分言語パターン）tを漏れなくすべて列挙せよ．□

データマイニングの分野では，ξを最小サポート(minimum support)，tが出現した事例の個数をtのサポートと呼ぶ．

事例は集合，系列，木，グラフといった離散データ構造で表現される．系列や木といったデータ構造は，自然言語を計算機上で表現する方法として一般的に用いられる．具体的には以下のような対応関係が考えられる[1]．

1．集合：単語の集合
2．系列：単語の並び
3．木：単語の木構造（構文解析済みデータなど）
4．グラフ：単語グラフ（単語の参照関係など）

くどう　たく
グーグル㈱
〒 150-8512 渋谷区桜ヶ丘町 26-1

[1] 単語を要素とすることが一般的であるが，文節や固有表現にしてもかまわない．

例えば，自然言語処理における「頻出部分系列パターンマイニング」とは，文の集合が与えられ，各文は単語の並びとして表現されているとき，文集合から，ξ個以上の文に含まれるすべての単語の並びを漏れなく抽出するタスクのことをいう．

以下に，大量の新聞記事，および小説，『我輩は猫である』から抽出された頻出言語パターンのいくつかを紹介する．これらがどれだけ有用で興味深いかの客観的な評価は行えないが，文としての構造を崩していないため，それ自身で何かしらの意味をもつようなパターンが抽出されている．

・頻出する単語の並び（上：新聞，下：小説）

ロシア　南部　チェチェン　共和国　の　首都　グロズヌイ

これ　が　鈴木　君　の　心　の　平均　を　破る　第

・頻出する部分木（上：新聞，下：小説）

((ついて　述べ,)(記者会見で，明らかにした.))

(休養を　(また　(吾輩は　要する.)))

頻出言語パターンマイニングアルゴリズムとは，上記の例のような言語パターンを大量のテキストデータ，コーパスから高速に発見する技術の総称である．マイニングの結果は，人間の意思決定をサポートするデータとして用いられたり，統計的自然言語処理の素性として用いられたりする．

2.2　マイニングアルゴリズム

これまで多くの頻出部分構造マイニングアルゴリズムが提案されてきたが，それらは以下の3つの観点から考察することができる．

(1) 部分構造（部分パターン）を枚挙する正準的な探索空間の定義

(2) 探索方法：幅優先もしくは深さ優先

(3) 頻出するという条件を用いた探索空間の効率的な枝狩り

(1)はデータ構造に依存する．集合は集合の，木は木の探索空間を個別に定義しなければならない．一般に構造が複雑になればなるほど探索空間の定義も複雑になる．探索空間は，空集合からはじまり，サイズ1のパターンサイズ2のパターンといった具合に小さいパターンから順に枚挙できるように定義する．

(2)は，応用に依存するが，粗データの場合は深さ優先，密データのときは幅優先のほうがよいことが知られている．粗データの場合，探索空間が大きくなり，幅優先ではメモリの使用量が大きくなってしまう．言語データは一般に粗データであるため，深さ優先のほうが望ましい．

(3)は「パターンAが頻出でないならば，Aを部分集合として含むすべてのパターン（Aの上位パターン）は，頻出ではない」という性質を用いる．例えば単語「京都」がコーパス中に100回出現した場合，京都と共起するフレーズ「京都大学」の頻度は必ず100回以下となる．「京都」が頻出でなければ，「京都大学」も頻出ではない．言われてみれば当たり前のことであるが，この性質は枝狩りにおいて最も重要な条件となる．探索空間を辿りながら，あるパターンが頻出しないとわかると，その部分空間（そこを起点とする空間）は枝狩り可能である．この性質は，幅優先，深さ優先にかかわらず用いられる．

さらに，幅優先のときに限り別の枝狩り条件を使うことができる．サイズNのパターンが頻出であるためには，サイズ$N-1$のすべての部分パターンもすべて頻出でなければならない．幅優先の場合，サイズNのパターンを枚挙しているときには，サイズがN未満のすべての頻出構造はすでに取得済みであるため，上記のチェックが容易に行える．幅優先に基づく代表的な頻出部分集合マイニングアルゴリズムにApriori[2]がある．本稿は言語パターンマイニングがテーマであるため，幅優先に基づく手法は深く言及せず，主に深さ優先に基づく手法にフォーカスを当てる．

図1に頻出言語パターンマイニングアルゴリズムの(構造に依存しない)擬似コードを示す．関数Projectは再帰的に呼ばれ，深さ優先探索を実現している．

以下では，集合，系列，順序木の深さ優先に基づく頻出パターンマイニングアルゴリズムについて概観する．部分構造を枚挙する探索空間がどのように定義されるのか，どういう共通点，相違点があるか比較されたい．

2.3　頻出部分集合マイニング

集合の部分集合を完全に枚挙する探索空間に，Bayardoらによる Set Enumeration Tree (SE-Tree)[3]がある．図2にSE-Treeの例を示す．

SE-Treeでは，集合の要素に順序（例えば辞書順）を与え，集合の枚挙の順番をその要素順に限定することで，正準的な探索空間を構築する．

実際のマイニングの動作過程については，2.2節の頻出部分系列マイニングで説明する．

2.4　頻出部分系列マイニング

Peiら[4]は，深さ優先探索で頻出部分系列を抽出する手法PrefixSpanを提案している．PrefixSpanが用いる探索空間は，SE-Treeとほぼ同じである．

```
Algorithm: FindFrequentPatterns
begin
    function project (t, S)
        if |S| ≥ ξ then
            t を出力
        else
            return // 頻出パターンではない, 枝狩り
        end

        foreach t' ∈ { t の子となる全パターン }
            S|t' ← t' を含む文集合 (S から作成)
            project(t', S|t')
        end
    end

    foreach t ∈ { 文集合中にあるサイズ 1 のパターン }
        S|t ← t を含む文集合
        project (t, S|t)
    end
end
```

図 1　頻出言語パターンマイニングアルゴリズム

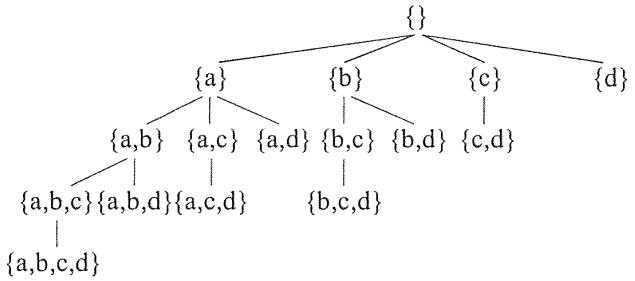

図 2　集合 {a, b, c, d} に対する SE-Tree

SE-Tree は，マイニング対象が集合であったため，あらかじめ集合の要素を辞書順に整列していたが，系列マイニングでは出現順序をそのまま用いる．

具体例として，図 3 に，ξ=2 とした場合の PrefixSpan の動作過程を示す．まず，サイズ数 1 の頻出系列 a, b, c を抽出する．d は，出現頻度が 1 であるため，頻出系列ではない．頻出系列は，これら 3 種類あるが，そのうち a について考え，a からはじまる空間を展開する（図 3 中央上）．展開された部分探索空間から，頻出系列 b, c を生成する．以下再帰的にこれらの作業を繰り返すことで，すべての頻出系列を抽出することができる．

2.5　頻出部分木マイニング

Abe と Zaki は，順序木からその部分木を，完全に重複なく列挙するアルゴリズム最右拡張をそれぞれ独

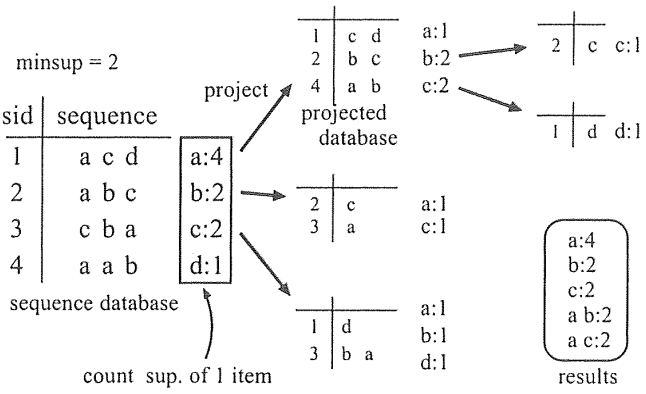

図 3　PrefixSpan の動作過程

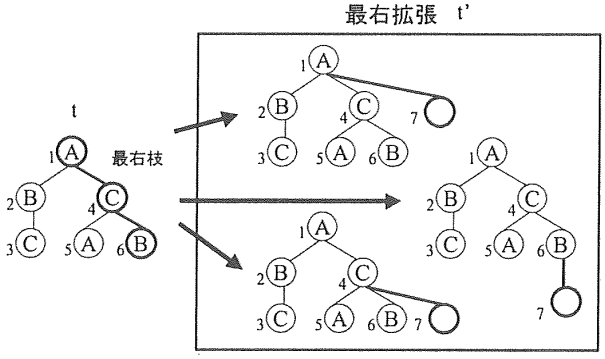

図 4　最右拡張

立に提案している[1][5]．最右拡張では，まず，サイズ 1 の木が列挙される．そして，サイズ $(k-1)$ の木に 1 つのノードを追加することでサイズ k の木を構築する．この手続きを再帰的に適用することで全部分木を列挙する．しかし，任意の位置にノードを追加すると重複する木を生成してしまうため，ノードの追加に一種の制限を設ける．この制限が最右拡張の鍵となるアイデアである．以下に最右拡張の定義を与える．

定義 1　最右拡張[1]

木 t に 1 つのノード s を追加して得られる順序木 t' を，木 t の最右拡張と定義する．ただし，ノードの追加には以下の制約がある．(1)ノードは，t の最右枝 (ルートから常に右端のノードを葉まで選ぶ) に追加される．(2)追加されるノードは，最右の兄弟（末弟）である．□

図 4 に最右拡張の例を示す．便宜上，木 t は前順序 (pre-order) でノードに順序が付与されているものとする．図 4 の例では，位置 7 のノードが最右拡張で追加されるノードとなる．また，追加される位置の候補は 3 種類（位置 1, 4, 6）あるが，ラベルの候補集合を考えると，ラベルの種類数×位置数（図 4 では，

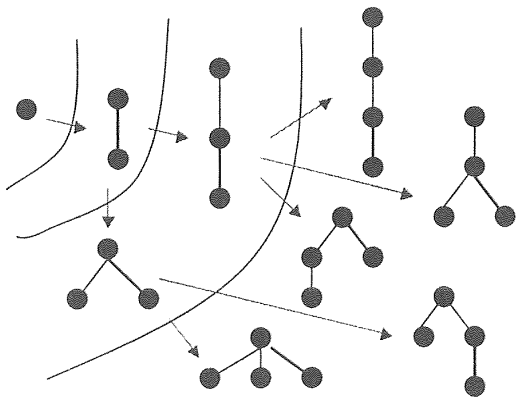

図5　最右拡張の再帰（探索木）

$|\{A, B, C\}|\times 3=9$) 通りの木 t' が，最右拡張より生成される．

さらに，最右拡張を，再帰的に適用することで，探索木を構築することができる．図5に，その探索木の例を示す．図5中では，便宜的にラベルの種類を1種類のみとしている．この探索木を探索することで，全部分木を，完全に重複なく列挙できる．

3.　頻出言語パターンマイニングの応用

頻出パターンマイニングの問題設定は以下のように一般化できる．

問題2　ある構造化（もしくは半構造化）された事例（＝文）の集合（＝テキスト，コーパス）が与えられたときに，ある基準 $\pi(t)$ をみたすような有意な t を漏れなくすべて列挙せよ．□

2節では，基準 $\pi(t)$ を頻度の観点から定義していた．この場合，ある個数 ξ 以上の事例に含まれるようなすべての部分構造 t を漏れなく列挙せねばならない．

基準 $\pi(t)$ を機械学習の観点から再定義すれば，何か面白いことができそうである．例えば，大量の離散データが与えられたときに，ある特定の分類クラスに偏って出現する部分構造は何か？　という問に答えるには，偏りに関する基準を定義すればよいだろう．

言語パターンマイニングアルゴリズムは，パターン抽出の基準 $\pi(t)$ の定義をタスクに応じて再設計することで，応用の幅が広がる．3節ではその具体例の1つを紹介する．

3.1　文の構造を考慮した機械学習手法

自然言語処理の代表的な応用に文書分類がある．文書分類では，一般に文書を単語の集まり（bag-of-words, BOW）によって表現する．その後，各単語を手がかり（素性，特徴量）とし，SVMといった機械学習手法を用いて文書の内容を政治，経済，スポーツといった粒度の粗い分類クラスに分類する．個々の単語がこれらのカテゴリを特徴付けるのに有効な情報を与えるため，BOWのような単純な素性を用いるモデルによっても高い精度を達成することができる．

一方で，テキストマイニングの分野では，Web上の製品レビューサイトやアンケート結果などから製品に対する要望や不満などの有用な情報を効率よく入手する要素技術が求められている．このようなタスクでは，意見が主観的に述べられているのか客観的に述べられているのか，あるいは，ある製品をほめているのかけなしているのかなど，書き手の意図に関する分類が求められる．つまり，このようなタスクは，分類するのは文が漠然と表す意味内容ではなく，文そのものがもつクラスであったり，文書中での文の役割についての分類になる．

図6に，想定するタスクで分類すべき文例を示す．評判分類はWebの掲示板に投稿された携帯電話のレビューを良い点と悪い点に分類するタスク，主観性判定は，車のレビューサイトにおいて投稿者が車に対して下している評価のうち，それが投稿者自身の主観的な評価かそうでないかを分類するタスクの具体例である．「パワーが足りないっていう人もいます」という文は，製品の評価にはなっているが，本人の評価ではないため，主観的な評価文とは判断していない．

文の意味内容だけでなく発話の意図を量るこのようなタスクでは，単純なBOWではなく，単語のつながりや文体を素性とする必要があるだろう．しかし，どのような単語のつながりや文体が分類に有効な素性となるか事前に分からないことが多い．人手により有効な素性を事前に調査をすることは可能であるが，思いがけない単語のつながりや文体を見落とす可能性がある．

筆者は，データマイニングの手法を用いて自動的に有効な手がかり（素性，特徴量）を発見し，それを用いて分類を行う手法を提案した[6]．従来法との決定的な違いは，文を単語の集合ではなく，木構造として表現する点にある．分類に使う素性は，任意の部分木となる．これは任意の単語のつながりや文体に対応する．

本手法では，学習，分類アルゴリズムの土台としてDecision Stumpsを用いる．Decision Stumpsは，1つの素性の有無に基づき分類を行う単純なアルゴリズ

評判分類	良い点：メールを送受信した日付，時間が表示されるのも結構ありがたいです．
	悪い点：なんとなく，レスポンスが悪いように思います．
主観性判定	主観的 ：エンジンパーツが豊富で安い．
	主観的以外：パワーが足りないっていう人もいます．

図6　分類対象文の例

ムである．分類器 $h(\mathbf{x})$ は具体的には以下のように定義される．

$$h_{<t,y>}(\mathbf{x}) \stackrel{\text{def}}{=} \begin{cases} y & t \subseteq \mathbf{x} \\ -y & otherwise. \end{cases}$$

ただし，$\mathbf{x}$ は分類対象の木である．分類器のパラメータは，木 t とラベル $y \in \{\pm 1\}$ の組 $\langle t, y \rangle$ で与えられる．もし分類対象 $\mathbf{x}$ が木 t を部分木として含む場合は分類結果は y となり，それ以外は $-y$ となる．

Decision Stumps の学習は，学習データ $T = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{L}$ に対するエラー率を最小にする（正解率を最大にする）最適パラメータ $\langle \hat{t}, \hat{y} \rangle$ を導出することで実現できる．学習自身は一見単純に見えるが，木 t の候補には制限を設けず，可能なすべての木を対象としているために，完全列挙に基づく手法は困難となることに注意されたい．この問題は，頻出部分木マイニングアルゴリズムを一部変更することで解決できる．具体的には以下の2つの手続きとなる．

1．木の集合から，全部分木を完全に重複なく列挙するための正準的な探索空間を定義する．
2．1の空間を探索し，最適パラメータを発見する．このとき，分枝限定法を用いて探索空間を枝刈りする．

(1)に対しては，前節で述べた最右拡張をそのまま用いる．手続き(2)では，ある部分木 t に対し，その全上位木 $t' \supset t$ の正解率の上限 $\mu(t)$ を求める．もし，既に発見された準最適部分木の正解率が $\mu(t)$ より大きい場合，t から先の空間は枝刈りできる．具体的な $\mu(t)$ の導出方法については，論文を参照されたい．

さて，Decision Stumps だけでは，たった1つの木の有無のみで分類が決定されるので性能が悪い．そこで，Decision Stumps を弱学習器として Boosting アルゴリズムを実行し全体の精度を向上させる．Boosting を適用することで，それまでの学習器にとって分離困難な事例に集中して学習した弱学習器が次々に作られる．最終的な文分類は，これらの弱学習器の重み付き多数決によって行われる．

本手法では，精度の高い統合的な分類器が得られるだけでなく，各弱学習器が用いている属性を直接観察することができるので，どのような部分構造が文分類に有効に働いたかを知ることができるという利点がある．

表1　文分類実験の結果（F 値）

	評判分類	主観性判定
構造なし	76.0	77.4
構造あり	78.7	81.6

表1に，実験結果を示す．評判分類，主観性判定それぞれ5,700，7,500 文のタグ付きデータを用い，5分割交差検定によって得られたF値（精度と再現率の調和平均）を示している．「構造なし」は個々の単語を個別属性として用いる従来法，「構造あり」は文を係り受け木とみなし，その任意の部分木構造（部分木）を素性として用いる提案法である．この結果より，単語個別の情報では，文分類のタスクが充分な精度で行えないことがわかる．

評判分類の実験によって得られた素性（部分木）の例を表2に示す．数値は，各部分木が分類にどのように寄与しているかを表す重みである．正の数値が「良い点」の分類に肯定的に寄与する属性，負の数値はその逆を示す．例えば，「切れる」が「にくい」に係る構造は「良い」カテゴリにプラスに働いているのに対し，「にくい」を含むその他の表現（「にくくなった」「読みにくい」など）は，「悪い」カテゴリを分類するのに寄与している．「使う」を含む構造では，「使いたい」「使ってる」「使いやすい」などが正の重みをもつのに対し，「使いやすかった」「を使ってた」のように過去形になったり，「方が使いやすい」のように比較になると負の重みをもつことがわかる．また，同じ「充電時間」を含む場合でも，これが「短い」に係るか「長い」に係るかで，評価が正反対になっている．このような属性は，BOW に基づくモデルでは用いることのできない情報である．

表2　サポート素性の一部

キーワード	重み λ_t	部分木 t (サポート素性)
A. にくい	0.00402	切れる にくい
	-0.00055	にくい なる た
	-0.00056	にくい
	-0.00069	読む にくい
	-0.00073	にくい なる
	-0.00076	使う にくい
	-0.00170	にくい
B. 使う	0.00273	使う たい
	0.00015	使う
	0.00013	使う てる
	0.00007	使う やすい
	-0.00010	使う やすい た
	-0.00076	使う にくい
	-0.00085	は 使う づらい
	-0.00188	方 が 使う やすい
	-0.00233	を 使う てる た
C. 充電	0.00280	充電 時間 が 短い
	-0.00410	充電 時間 が 長い

4. おわりに

本稿では自然言語処理における，頻出部分パターンマイニングの問題定義および種々の手法を概観した．さらに，機械学習への頻出部分パターンマイニングの応用例について紹介した．

本稿で紹介したアルゴリズムの実装はオープンソースソフトウェアとして作者のホームページから入手可能である[2-4]．

参考文献

[1] Kenji Abe, Shinji Kawasoe, Tatsuya Asai, Hiroki Arimura and Setsuo Arikawa. Optimized substructure discovery for semi-structured data. In *Proc. of PKDD*, 2002.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487-499. Morgan Kaufmann, 12-15, 1994.

[3] Roberto J. Bayardo. Efficiently mining long patterns from databases. In *Proc. of SIGMOD 1998*, 1998.

[4] Jian Pei, Jiawei Han et al. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proc. of International Conference of Data Engineering*, pages 215-224, 2001.

[5] Mohammed Zaki. Efficiently mining frequent trees in a forest. In *Proc. of KDD*, pages 71-80, 2002.

[6] 工藤拓, 松本裕治. 半構造化テキストの分類のためのブースティングアルゴリズム. 情報処理学会論文誌, 45(9), 2004.

[2] http://chasen.org/~taku/software/freqt
[3] http://chasen.org/~taku/software/prefixspan
[4] http://chasen.org/~taku/software/bact