言語情報を利用したテキストマイニング

工藤 拓　山本 薫　坪井 祐太　松本 裕治
奈良先端科学技術大学院大学情報科学研究科
{taku-ku,kaoru-ya,yuuta-t,matsu}@is.aist-nara.ac.jp

1 はじめに

データマイニングとは，膨大なデータの中から有用，あるいは興味のあるパターンを明示的な知識として発掘する方法についての科学的研究である．

その対象をテキストデータとしたテキストマイニングの分野は，決して目新しいものではない．例えば，語と語の共起関係や相関関係を調べたり，語のクラスタリングといった処理は，古くから研究されてきている．これらの研究の多くは，形態素解析といった比較的「浅い」言語処理技術のみを利用し，テキスト全体を単語の「集合」とみなして処理を行っている．しかし，このようなデータ構造は，機械処理に向いているといはいえ，各単語のテキスト中の役割，単語間の係り受け関係などを考慮に入れないため，文がもつ本当に意味のある構造をうまくとらえらないという問題がある．

その一方で，Text Chunking，係り受け解析 (構文解析)，固有名詞抽出といった高度な自然言語処理ツールが近年利用できるようになってきた．これらのツールにより，テキスト中の単語の役割がより明確になり，テキストの具体的な内容をよりよく反映したデータ構造に変換することができる．しかし，自然言語処理ツールを利用することで，生のテキストは，ある構造を持ったデータ (半構造テキストデータ) に変換される．複雑なデータ構造になればなるほど，単純に単語の集合を前提としていたようなアプローチは有効に機能しなくなってくる．今後，このような半構造化されたテキストデータからいかに有用なパターンを発見するかが大きな課題となってくるであろう．

本稿では，このような流れを受け，種々の自然言語処理ツールの結果から得られた半構造化されたテキストデータに対するマイニング手法提案する．具体的には，まず，半構造テキスマイニングを与えられた半構造テキストデータから頻出する部分構造を発見する問題と定式化する．さらに，シーケンシャルパターンのマイニング手法である PrefixSpan アルゴリズム [2] を拡張し，この問題を効率よく解決するアルゴリズムを提案する．

2 シーケンシャルパターンマイニング

Agrawal[1] らは，シーケンシャルパターンマイニングを以下のように定式化している [1]．

$I = \{i_1, i_2, \ldots i_n\}$ を，アイテム 集合とする．系列とは，アイテムの列であり，$\langle s_1, s_2, \ldots, s_n \rangle$，と表記する．$s_k$ は任意のアイテムである．ある系列 α 中のすべてのアイテムが，別の系列 β の中にに存在し，その順番が保持されている場合，系列 β は系列 α を「含む」と言い，$\alpha \sqsubseteq \beta$ と表記する．系列データベース S とは，系列 id sid と系列 s のタプル $\langle sid, s \rangle$ の集合である．さらに，系列 α の系列データベース S におけるサポート $support_S(\alpha)$ とは，S 中のすべての系列のうち，系列 α を含むタプルの数と定義される．

シーケンシャルパターンマイニングとは，系列データベース S と 任意の整数 (**最小サポート値 (minimum support)** ξ) に対し，$support_S(\alpha) \geq \xi$ となるような系列 α を全て列挙するタスクの事を指す．

3 PrefixSpan

Pei[2] らは，シーケンシャルパターンマイニングに対する効率的な手法 PrefixSpan を提案している．

PrefixSpan を説明する前に，そのアルゴリズムの核となる**射影**という操作を説明する．ある系列 $s = \langle a_1, a_2, \ldots a_m \rangle$，アイテム a に対し，$a_1 \neq a$，$a_2 \neq a, \ldots, a_{j-1} \neq a$，$a_j = a$ となるような整数 $j (1 \leq j \leq m)$ が存在する場合，系列 $\langle a_1, a_2, \ldots a_j \rangle$ を s の a に対する **prefix** ($prefix(s,a)$) と定義する．また，系列 $\langle a_{j+1}, \ldots, a_m \rangle$ を s の a に対する **postfix** ($postfix(s,a)$) と定義する．もし j が存在しない場合は，prefix,postfix は未定義 (ε) となる．ある系列データベース S に対し，アイテム a によって **射影 (project)** し，**射影データベース** $S|a$ を作成するとは，S 中のそれぞれ系列 s に対し，$postfix(s,a)$ を作成し，それらを改めて系列データベースとする操作と定義される．

図 1 に，$\xi = 2$ とした場合の PrefixSpan の動作過程を示す．まず，アイテム数 1 の多頻度系列 a, b, c を抽出する．d は，サポートが 1 であるため，多頻度系列ではない．多頻度系列は，これら 3 種類あるが，そのうち a について考え，S の a による 射影データベース $S|a$ を作成する (図 1 中央上)．このデータベースから，多頻度系列 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，すべての多頻度系列を抽出することができる．PrefixSpan の擬似コードを 図 3 に示す．実際には，系列データベース S が，メモリに納まる場合は，系列へのポインタと postfix の先頭位置を使って射影 (擬似射影) を行う．

3.1 集合を単位とする PrefixSpan

部分的に集合を考慮するような系列データベースに対しても，PrefixSpan は適用可能である．具体的には，まず，**エレメント** という単位を導入する．エレメントは，アイテムの集合である．エレメント中のアイテムは集合となるため，それらの順番は考慮せず，あらかじめ辞書順にソートしておく．図 2 にエレメントが含まれる系列データベースの例を示す．'()' で囲まれるアイテムがエレメントである．

[1] この定式化は，エレメントという単位が無いため厳密ではない．エレメントについては，次節で扱う

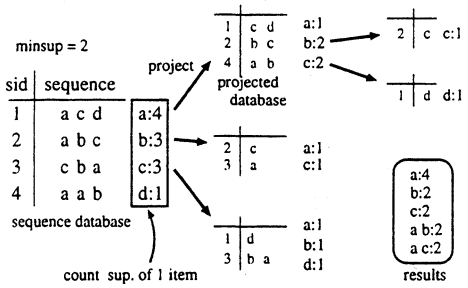

図 1: PrefixSpan の動作過程

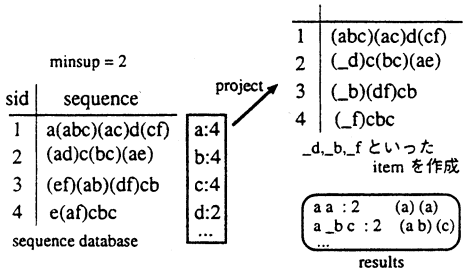

図 2: PrefixSpan + エレメント

```
call PrefixSpan(ε, S)
procedure PrefixSpan (α, S|α)
begin
  B ← {b|(s ∈ S|α, b ∈ s) ∧ (support_S|α(⟨b⟩) ≥ ξ)}
  foreach b ∈ B
  begin
    # パターンとその頻度を出力
    print αb, support_S|α(⟨b⟩)
    # 射影データベースの作成
    (S|α)|b ← {⟨sid, s'⟩|(⟨sid, s⟩ ∈ S|α)
              ∧ (s' = postfix(s, b)) ∧ (s' ≠ ε)}
    call PrefixSpan (αb, (S|α)|b)
  end
end
```

図 3: PrefixSpan の擬似コード

図 2 を例に, 実際の PrefixSpan の動作過程を説明する. まず, アイテム数 1 の多頻度系列 a, b, c, d, e, f が抽出される. 次に, 前節と同様にアイテム a の 射影データベースを作成する. その際, a と同じエレメントにあるアイテムに対し, '_' という prefix を付け, 別のアイテムに変換し, 射影データベースを作成する. 同一エレメントにエレメントが存在するかどうかは, '_' によって区別される.

4 PrefixSpan の拡張

4.1 関係関数

ここで, 関係関数 という関数を用い, PrefixSpan を拡張する. 関係関数とは, 系列データベース S, 系列 id sid, 整数 $i, j (i < j)$ を引数とするような関数 $Relation(S, sid, i, j)$ と定義され, $seq(S, sid)$ 中のアイテム $item(S, sid, i), item(S, sid, j)$ の関係を返す.

```
S = {⟨sid_1, s_1⟩, ⟨sid_2, s_2⟩, ..., ⟨sid_n, s_n⟩}
P = {⟨sid_1, 0⟩, ⟨sid_2, 0⟩, ..., ⟨sid_n, 0⟩}
function Relation(S, sid, i, j)
call PrefixSpan(ε, P)
procedure PrefixSpan (α, P|α)
begin
  # B は, タプルをキー,
  # タプルの集合を値とするマップ
  foreach ⟨sid, l⟩ ∈ P|α
  begin
    # C は, タプル の集合
    C ← {}
    foreach i ← l + 1 to |seq(S, sid)|
    begin
      b ← item(S, sid, i)
      r ← Relation(S, sid, l, i)
      if (r ≠ ε and C ∉ ⟨b, r⟩)
        B[⟨b, r⟩] ← B[⟨b, r⟩] ∪ ⟨sid, i⟩
      C ← C ∪ ⟨b, r⟩
    end
  end
  foreach ⟨b, r⟩ ∈ keys of B
  begin
    # ξ 未満のものは考慮しない
    if (|B[⟨b, r⟩]| < ξ) continue
    # パターンとその頻度を出力
    print α ⟨b, r⟩, |B[⟨b, r⟩]|
    call PrefixSpan (α⟨b, r⟩, B[⟨b, r⟩])
  end
end
```

図 4: 拡張 PrefixSpan の擬似コード

この関係関数を用い, PrefixSpan の擬似コード (擬似射影版) を, 図 4 のように改める. ただし, S 中の sid k に対応する系列を $seq(S, k)$ と表記し, $seq(S, k)$ の 先頭から i 番目 のアイテムを,$item(S, k, i)$ と表記する. 通常の PrefixSpan では, 各アイテムが射影の対象となるが, 拡張後は, 関係関数が返す値とアイテムのタプルが射影の対象となる. また, 関係関数が ε を返す場合は, それ以上の射影を行わないと定義する.

関係関数は, それぞれの目的に従って設計されるが, 個々のデータ構造に対し, 関係関数をいかなる方法で一般的な形で与えるかが大きな問題となる. 与えられた任意の構造に対し一般的な形で関係関数を与える事は, 今後の課題とし, 本稿では, 自然言語で頻繁に使用されるデータ構造のうち, エレメント, N-gram, チャンク, そして係り受け関係に対する具体的な関係関数の実装方法についての議論を行う.

4.2 エレメント

集合を単位とする PrefixSpan の拡張は, 関係関数を用いても実現可能である. 具体的な関係関数は, 図 5 中 $ElementRelation$ のようになる. 前節では, エレメント内のアイテムを, '_' という prefix を付けることで区別していたが, エレメント内と外に対し, 別の関係名 (IN/OUT) を返すことで実現できる.

4.3 N-gram

N-gram のマイニングは, シーケンシャルパターンマイニングに含まれるため, N-gram をパターンとするようなマイニングの実現は極めて容易である. 具体的な

関係関数は, 図 5 中の *NGramRelation* のようになる.

4.4 チャンク

チャンク構造は, 図 6 上のように, チャンク名とそのチャンクの中に含まれるアイテムから構成される. チャンク構造に対しては, まず, 図 6 下のように, アイテムの列に続いて, チャンク名を別のアイテムとして挿入する. この時, 各アイテムの タイプ **(type)** を以下のように定義する. $type(a) = NT$(a がチャンク名の場合), $type(a) = T$(それ以外). チャンク構造に対する関係関数は, 図 5 の *ChunkRelation* ようになる. この関数により, 基本的には, チャンク名の単位でマイニングが行われるが, 個々のチャンクがどうのようなアイテムから構成されるか同時に考慮することができる.

4.5 係り受け

語順が比較的自由である日本語のような言語では, 係り受け解析の結果を利用することで, 意味的に同じ文だが語順の違う場合, それらの違いを吸収できる可能性がある. まず, 語順を正規化する目的で, 係り受け解析の結果を, 以下の手続きで正規化された係り受け系列に変換する (図 7).

1. 文の head となる文節を配列に追加 する.
2. head 係る文節を辞書の降順に列挙する.
3. それら一つ一つに対し, 配列に追加し, その文節を head とみなし (2) へ戻り再帰的に繰り返す.
4. 配列を逆順にし, 出力する.

係り受け構造についての関係関数を, 図 5 中の *DepRelation* のように定義する. この関数は, $item(S, sid, i)$ の係り先が, $item(S, sid, j)$ の $k-1$ ($k \geq 1$) 代目の祖先である場合, 関係名が k となる. それ以外は, 未定義 (ε) となる. 図 8 にアイテム a から見た, 他のアイテムの関係名を示す.

5 実験

5.1 実験設定

以下の 2 つの実データを用い実験を行った.

- 新聞記事 (京都大学コーパス 3.0 全文, 約 38,000 文)
- 小説 (夏目漱石の「我輩は猫である」, 約 9,300 文 [2])
  ChaSen[3] 及び CaboCha[4] を用いて形態素解析 (文/単語認定), 構文解析を実施した.

テキストの構造として以下の 3 つの実験を行った.

- N-gram (アイテムは, 単語の表層)
- チャンク (2 段階のみの係り受け)
  文節の表層をチャンク名, チャンクの中にあるアイテムをチャンク名に係る文節の表層とする. チャンク名とチャンク中のアイテムは集合とみなし辞書順にソートする. 松澤 [3] と同一の構造である.
- 係り受け (アイテムは文節の表層)

実験には, Linux (XEON 2.2 GHz のプロセッサ, 3.5GB の主記憶) の WS 使用した. 係り受けは Perl を, それ以外は C++ を用いて実装した[5].

[2] http://www.aozora.gr.jp/
[3] http://chasen.aist-nara.ac.jp/
[4] http://cl.aist-nara.ac.jp/~taku-ku/software/cabocha/
[5] http://cl.aist-nara.ac.jp/~taku-ku/software/prefixspan/

```
function ElementRelation(S, sid, i, j)
begin
  x ← item(S, sid, i)
  y ← item(S, sid, j)
  if (x, y が同一 element 内に存在) return IN
  else return OUT
end

function NGramRelation(S, sid, i, j)
begin
  if (i + 1 = j) return NGRAM
  else return ε
end

function ChunkRelation(S, sid, i, j)
begin
  x ← item(S, sid, i)
  y ← item(S, sid, j)
  if (type(x) = NT) return CHUNK
  if (type(x) = T and type(y) = T and
      x, y が同一 チャンク 内にある) return CHUNK
  if (type(x) = T and type(y) = NT and
      チャンク y の中に x が含まれる ) return CHUNK

  return ε
end

function DepRelation(S, sid, i, j)
begin
  x = item(S, sid, i)
  y = item(S, sid, j)
  if (x の係り先位置 < j) return ε

  r ← 0
  while (x の係り先 ≠ y)
  begin
    r ← r + 1
    y ← y の係り先
  end
  return r
end
```

図 5: 関係関数の具体例

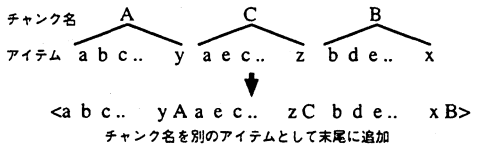

図 6: チャンク の表現

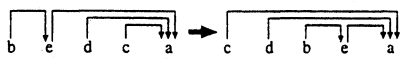

図 7: 係り受けの正規化

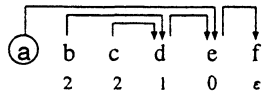

図 8: アイテム a から見た関係名

表 1: 実験結果

minsup	抽出時間 新聞記事/小説 (秒)		
	N-gram	チャンク	係り受け
2	2.2/0.46	N/A/0.74	355.6/7.8
5	2.0/0.43	3.2/0.60	26.7/5.8
10	1.9/0.41	3.0/0.57	24.0/5.2
15	1.9/0.41	2.8/0.55	22.9/4.8
20	1.9/0.41	2.9/0.55	22.1/4.6

5.2 実験結果

表 1 に最小サポート値 (minsup.) と実行時間 (秒) の関係を示す. 新聞記事に対しチャンクの構造を適用した時以外は, どれも現実的な時間で終了している. ただし, 新聞記事に関しては, 最小サポート値が小さくなるにつれ実行時間が大きく増加している. この理由としては, 新聞記事では言い回しが小説に比べ固定されており, それらが頻出するためであると考えられる.

以下に, それぞれの構造において抽出された頻出パターンのいくつかを紹介する. これらがどれだけ有用か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持つようなパターンが抽出されている.

・N-gram (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

・チャンク (上:新聞, 下:小説)
(震度は {各地の}) (通り {次の, 震度は})
(ないから、{吾輩は, 仕方が})

・係り受け (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。)))

6 応用例

6.1 機械学習の素性抽出

機械学習を運用する際, 相互情報量などの基準を用い, どういった素性が分類に有効か, 素性の選別を行う事で精度向上に繋がる場合がある. もし, 素性の組み合わせや, 素性どうしの関係を新な素性 (パターン) して定義すれば, 同様な方法でそらの新しい素性を選別することが可能であろう.

ここで, 本稿で提案した マイニングの手法が適用可能ではないかと考える. 具体的には, クラス y_i と 素性の系列 $\mathbf{x}_i$ を単純に連結し, 新たな系列を作成する. 関係関数としては, アイテム y_i からは, $\mathbf{x}_i$ の任意のアイテムに射影可能な関数を, また, $\mathbf{x}_i$ 内のアイテムどうしの関係は, 半構造化データ $\mathbf{x}_i$ を記述するための 関係関数を用いる. これにより, クラスの独立頻度, パターンとクラスの共起頻度, パターンの独立頻度を算出し, 相互情報量などを計算することができる.

6.2 対訳パターンの抽出

対訳コーパスから, 対訳パターンを抽出するタスクにおいても, PrefixSpan は有効であると考える. まず, 各々の言語を Chunker や係り受け解析器を用い, 半構造化テキストデータに変換する. 次に, それぞれのデータを単純に連結し, 1 つの系列を作成する. 単言語間では, その言語を表現するための関係関数をそのまま使用し, 2 言語間のアイテムをまたぐ射影では, 任意の射影が許可されるように関係関数を設計する.

このような系列データベースを用い, PrefixSpan を実行することで, 頻出する共起部分パターン及び, 単言語のみから構成されるパターンが抽出される. それらの頻度も抽出されるため, 共起頻度と独立頻度のみで計算を行う dice 係数や, 相互情報量の算出は容易に行える.

簡単な例として, 日英の対訳コーパス 9268 文を用い, 対訳表現抽出を行った. 構造として, 系列及び N-gram を対象とした. ただし, あらかじめ機能語相当の単語はアイテムから取り除いている. 最小サポート値を 2 とした設定で PrefixSpan を実行した結果, 候補生成に要した時間は, 系列 52 分, N-gram 7 秒となり, 十分実用的な時間で終了した. 抽出された翻訳パターンの良し悪しを評価する事は, 本稿の主旨では無いので, 詳細な評価は行わないが, 系列に基づく手法において, 以下のような不連続な翻訳パターンが抽出されたことは大変興味深い.

earliest convenience	都合 つき 次第
let ... know	お知らせ
thank ... letter	手紙 ありがとう

7 まとめと今後の課題

本稿では, 種々の自然言語処理ツールの結果から得られた半構造化されたテキストに対するマイニング手法提案した. 具体的には, シーケンシャルパターンのマイニング手法である PrefixSpan アルゴリズムに対し, 関係関数を導入することで, チャンクや係り受けといったデータ構造を考慮できるように拡張した.

本稿では, 抽出されたパターンの客観的な有効性に関する評価は行っていない. 今後は, 対象とする構造, 関係関数の違いにより, 抽出されるパターンにどのような違いが現われ, それらが具体的な応用においてどれほどの有意差を生じるか評価を行いたいと考える. さらに, 木構造, グラフ構造といった一般的な構造が関係関数で記述できるか, また, それらの完全性や健全性についても議論していきたいと考える.

参考文献

[1] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee L. P. Chen, editors, *Proc. 11th Int. Conf. Data Engineering, ICDE*, pp. 3-14. IEEE Press, 6-10 1995.

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

[3] 松澤裕史. 大規模データベースからの頻出構造化パターンの抽出. 情報処理学会論文誌, Vol. 42, No. 8, 2001.