自 然 言 語 処 理 148－10
（２００２．３．５）

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

〒 630-0101 奈良県生駒市高山町 8916-5

{taku-ku,kaoru-ya,yuuta-t,matsu}@is.aist-nara.ac.jp

テキストマイニングの分野は決して新しいものではなく, 単語の共起やクラスタリングといった多くの手法が現在まで提案されている. しかし, それらの多くは, テキストを単語の集合という簡単な構造で表現しており, 単語間の関係や構文的な係り受け関係を無視してしまっている. このような構造では, 多くの場合, 文がもつ本当に意味のある構造をうまくとらえらない. 本稿では, チャンキングや係り受け解析といった自然言語処理ツールを使用し, テキストデータを具体的な内容をよりよく反映した半構造化データに変換し, そこから有益なパターンを抽出する手法を提案する. 具体的には, テキストマイニングを半構造化されたデータから頻出する部分構造の抽出ととらえ, 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.

キーワード: テキストマイニング, シーケンシャルパターンマイニング, 半構造化データ, PrefixSpan

Text Mining using Linguistic Information

Taku Kudo　　Kaoru Yamamoto　　Yuta Tsuboi　　Yuji Matsumoto

Graduate School of Information Science, Nara Institute Science and Technology

8916-5 Takayama, Ikoma Nara 630-0101 Japan

{taku-ku,kaoru-ya,yuuta-t,matsu}@is.aist-nara.ac.jp

Text mining has gained the focus of attention recently, in particular, the success in word clustering have been reported. However, many of these bag-of-word approaches ignore the implicit dependency relations between words completely which are critical to understanding of the original text. In this paper, we apply NLP tools, such as chunking and dependency analysis, to convert a raw text into a semi-structured text from which useful patterns are extracted. We extend the PrefixSpan algorithm, one of an efficient algorithms for sequential pattern mining, to efficiently extract sub-structures from a linguistic data annotated by NLP tools.

Keywords: Text Mining, Sequential Pattern Mining, Semi-structured Data, PrefixSpan

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」もしくは「並び」とみなして処理を行っている. しかし, 単語の集合といった単純なデータ構造は, 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. これらのツールの応用分野として, テキストマイニングを取りあげることはごく自然な発想であろう. 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. 木, グラフといった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 [5] 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.

1.1 関連研究

松澤 [8] は, コールセンターのデータを対象に, 半構造化されたテキストデータから頻出する部分構造を効率良く発見する手法を提案している.

例えば,「音は良いが, 映像が悪い」といった文を, 単純な単語の集合とみなすと, 音, 映像のどちらが良く, どちらが悪いのか区別ができなくなる. 実際に, 体言と用言が同一文書中に共起しても, 文法的な係り受け関係にあるものはそのうち 4 割程度にすぎないという結果が得られており, 係り受け解析の認識が必要なのは自明であろう. 彼は, 上記の例に対し 係り受け解析を実行し, {(良い { 音 }), (悪い { 映像 })} のような, 述語 (用言) と 引数 (述語に係る体言の集合) のタプルというデータ構造で表現し, そこから頻出するパターンをマイニングする手法を提案している. その結果, 従来の単語の集合による手法に比べ有益なパターンが発見できたと報告している.

しかし, このデータ構造では, 3 項以上の係り受け関係を考慮できない. これは,「何がどうなった結果どうなったのか」といったパターンが発見できない事を意味する.

安部 [6] らは, 半構造データとして, 木構造のデータをとり挙げ, その集積から頻出する部分構造を効率良く発見する手法を提案している.

2 シーケンシャルパターンマイニング

Agrawal[2] 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. $S = \{\langle sid_1, s_1 \rangle, \langle sid_2, s_2 \rangle, \ldots, \langle sid_2, s_n \rangle\}$. さらに, 系列 α の系列データベース S における**サポート** 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.

$$support_S(\alpha) = \left|\{\langle sid, s \rangle | (\langle sid, s \rangle \in S) \wedge (\alpha \sqsubseteq s)\}\right|$$

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.

自然言語処理において最も単純な応用は, アイテムを **単語**, 系列 を**文**, 系列データベースを **テキスト** とすることであろう. この場合, シーケーシャルパターンマイニングとは, テキスト中に ξ 回以上出現する単語の列 (それらは連続している必要はない) を, 漏れなく完全に抽出する事を意味する.

3 PrefixSpan

シーケンシャルパターンマイニングに対する最初の研究としては, **候補生成-テスト** に基づく Apriori ア

[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

ルゴリズム [1] がある．これは, あるパターンが多頻度パターンであるためには, その部分集合も多頻度でなければならないというヒューリスティックを用い, アイテム数が少ないパターンから順に候補を生成しながら, 頻度を数えあげていく手法である．しかし, アイテム集合が非常に巨大でかつ疎な場合[2], 非常に多くの候補パターンを生成しなければならず効率が悪い．

Pei[5] らは, Apriori の 候補生成-テストとは異なり, 深さ優先探索で多頻度パターンを抽出する手法 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．

$$S|a = \{\langle sid, s' \rangle | (\langle sid, s \rangle \in S) \wedge (s' = postfix(s,a)) \wedge (s' \neq \varepsilon)\}$$

図 3 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 3 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 2 に示す．この手続きは, 順に系列データベースを分割, 生成していくため, 分割統治に基づくマイニング手法だと言える

実際には, 射影データベースは生成するのではなく, 系列へのポインタと postfix の先頭位置さえ記憶しておくだけで十分である．一般に, 系列データベース S が, メモリに納まる場合は, ポインタと位置を記憶する **擬似射影** が使用される．図 3 にその擬似コードを示す．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．$|seq(S,k)|$ は, $seq(S,k)$ のアイテム数である．

[2]単語集合はこのようなアイテム集合である．

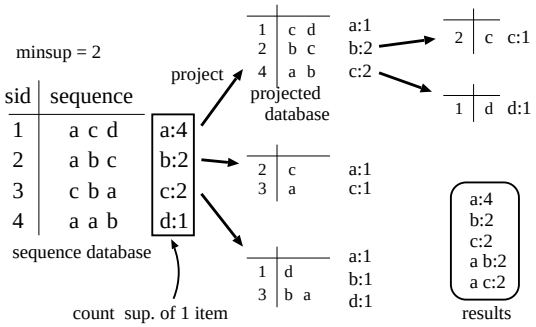

図 1: PrefixSpan の動作過程

```
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 2: PrefixSpan の擬似コード

```
S = {⟨sid_1, s_1⟩, ⟨sid_2, s_2⟩, ..., ⟨sid_n, s_n⟩}
P = {⟨sid_1, −1⟩, ⟨sid_2, −1⟩, ..., ⟨sid_n, −1⟩}
call PrefixSpan(ε, P)

procedure PrefixSpan (α, P|α)
begin
  # B は, アイテムをキー,
  # タプル の集合を値とするマップ
  foreach ⟨sid, l⟩ ∈ P|α
  begin
    # C は, アイテムの集合
    C = {}
    for i ← l + 1 to |seq(S, sid)|
    begin
      b ← item(S, sid, i)
      if (C ∉ b)
        B[b] ← B[b] ∪ ⟨sid, i⟩
      C ← C ∪ b
    end
  end

  foreach b ∈ keys of B
  begin
    # ξ 未満のものは考慮しない
    if (|B[b]| < ξ) continue
    # パターンとその頻度を出力
    print αb, |B[b]|
    call PrefixSpan (αb, B[b])
  end
end
```

図 3: 擬似射影 による PrefixSpan

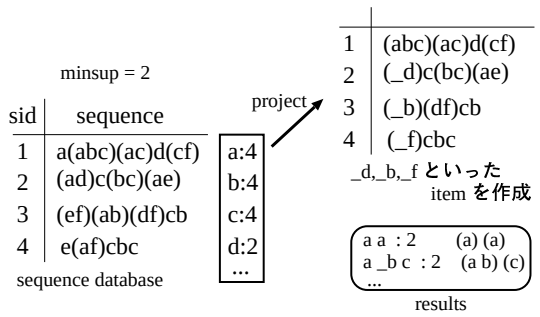

図 4: PrefixSpan + エレメント

3.1 集合を単位とする PrefixSpan

実際の自然言語処理のタスクにおいては, アイテムとしての単語は複数の素性の集合 (例えば, 表層, 品詞, 活用など) によって表現されることが多い. このように, 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[3].

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. 前節までの系列データベースは, 各エレメントが, 1 つのアイテムで構成される特別な場合といえる. 図 4 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.

図 4 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. 抽出されるパターンには, アイテムに '_' が付与されるアイテムと, されないアイテムがある. 前者は, 直左のアイテムと同一のエレメントにアイテムが存在することを示し, 後者は, 別のエレメントに存在することを示す. 図 4 右下に, 実際に抽出されるパターンの例を示す.

4 PrefixSpan の拡張

4.1 関係関数

PrefixSpan はそのアルゴリズムの簡潔さから, 容易に拡張が可能である. 例えば, N-gram のパターンの場合, 前回射影したアイテムと, これから射影しようとするアイテムが隣接してる場合のみ射影するような制約を加えれば良い. また, あるアイテムどうしに, なんらかの関係 (係り受け関係等) が定義できる場合, 関係名 +アイテムというタプルで射影すれば, 直感的には, 関係の違いをパターンの違いに反映することが可能となるだろう.

この考えを一般化したものが **関係関数**である. 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 $Relation(S, sid, i, j)$ と定義される. 具体的には, 関係関数は $seq(S, sid)$ 中のアイテム $item(S, sid, i), item(S, sid, j)$ の関係を返す.

この関係関数を用い, PrefixSpan の擬似コードを, 図 5 のように改める. 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. 実際に, 半構造化データの多頻度部分パターンをどのように定義するかは, 各データ構造に依存するため, 与えられた構造から関係関数を一般的な形で決定することは困難である. 個々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, 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. この限定は一般性に欠けるが, 自然言語処理の結果を利用したテキストマイニングという事を考えれば, これらの関係及びそれらの組み合わせで十分であろう.

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.

具体的には, 関係関数は, 図 6 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. これらの 2 つは, 表現の違いだけで本質的には同一の処理である. しかし, 提案手法により, 関係関数という形でより一般化されてることに注意されたい.

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 7 上のようになる. i, j が隣接する場合のみ射影が許可される.

さらに, ある固定長の Window size の中のみ射影を許可したり (7 下), N-gram の大きさ N に制約を設けたりすることも容易に実現可能である.

[3]エレメントを含む構造に対するマイニングが, 厳密な意味でのシーケンシャルパターンマイニングである. 本稿では, 後の拡張との対応を明確にするために, 別に掲載した.

```
S = {⟨sid_1, s_1⟩, ⟨sid_2, s_2⟩, ..., ⟨sid_n, s_n⟩}
P = {⟨sid_1, −1⟩, ⟨sid_2, −1⟩, ..., ⟨sid_n, −1⟩}
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 5: 拡張 PrefixSpan の擬似コード

```
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
```

図 6: 関数 Element Relation

```
# 連続する場合のみ
function NGramRelation(S, sid, i, j)
begin
   if (i + 1 = j)
      return NGRAM
   else
      return ε
end

# 距離 C 以下離れているアイテムのみを考慮
function NGramRelation(S, sid, i, j)
begin
   if (j − i < C)
      return NGRAM
   else
      return ε
end
```

図 7: 関数 N-gram Relation

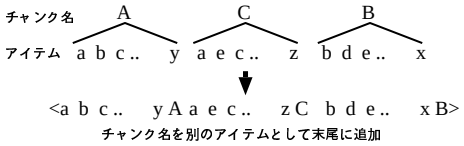

図 8: チャンク の表現

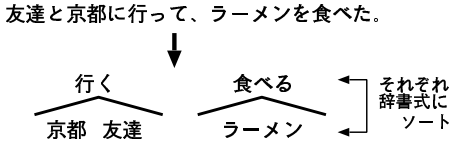

図 9: 係り受けのチャンク表現

```
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
```

図 10: 関数 Chunk Relation

4.4 チャンク

チャンク構造は, 図 8 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. 図 9 のように, 各チャンク名を用言, チャンクの中のアイテムを用言に係る体言とし, チャンク名とアイテムを辞書式にソートしておけば, 松澤が提案する 2 段階の係り受けに対するマイニング問題と同一になる.

チャンク構造に対しては, まず, 図 8 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 10 のようになる. 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, (アイテムのみから構成される パターンはマイニングされない) 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. さらに, チャンクの N-gram のみを考慮する拡張や, 前置詞句の連続は考慮しないといった言語制約の投入等は, 容易に実現可能である.

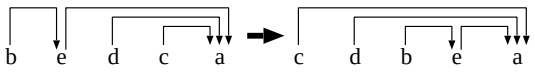

図 11: 係り受けの正規化

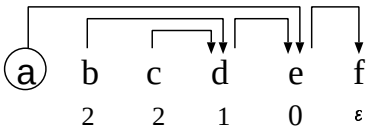

図 12: アイテム a から見た関係名

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, 以下の手続きにて, 係り受け解析の結果を, 正規化された係り受け系列に変換する[4]. これは, ある文節に係る文節の出現順番が変わっても文全体が伝える情報にはあまり変化が無いというヒューリスティックスに基づく処理である.

(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.

図 11 に変換前後の 2 つの係り受け系列 を示す.

次に, 係り受け構造についての関係関数を, 図 13 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 12 にアイテム a から見た, 他のアイテムの関係名を示す.

```
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
    else
    begin
        r ← 0
        while (x の係り先 ≠ y)
        begin
            r ← r + 1
            y ← y の係り先
        end
        return r
    end
end
```
図 13: 関数 DepRelation

[4]兄弟の順番を考慮しない木構造に変換する処理ともいえる

5 実験

5.1 実験設定

本アルゴリズムの評価を行うため, 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 文を利用した. このコーパスは, 95 年 1 月の 1 週間分の毎日新聞記事に対し, 形態素解析, 構文解析が施されたものである.
- 小説
  電子化された小説として夏目漱石の「我輩は猫である」を利用した[5]. 形態素解析 (文/単語認定), 構文解析には, ChaSen[6] 及び CaboCha[7] を使用した.

テキストの構造として以下の 3 つの実験を行った.

- N-gram
  それぞれのパターン中のアイテムは隣接しなければならないとう制約を付与する. アイテムは, 単語の表層である
- 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 [8] と同一の構造である.
- 係り受け
  関係関数を用い係り受けを考慮する. アイテムは文節の表層である.

実験には, Linux (XEON 2.2 GHz のプロセッサ, 3.5GB のメモリ) の WS を利用した. 係り受けは, Perl を, それ以外は C++ を用いて実装した[8].

5.2 実験結果

表 1,2 に最小サポート値 (minsup.) と実行時間 (秒) の関係を示す.

N-gram は, 最小サポート値にあまり影響されず実用的な時間で抽出が完了している. 係り受けに関しても実用的な時間で抽出が完了した. 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, 新聞記事では使用される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. チャンクに関しては, 新聞記事を使用し, 最小サポートが 2 の場合は, 終了しなかったが, それ以外は 3 秒程度で終了した.

[5]http://www.aozora.gr.jp/
[6]http://chasen.aist-nara.ac.jp/
[7]http://cl.aist-nara.ac.jp/~taku-ku/software/cabocha/
[8]http://cl.aist-nara.ac.jp/~taku-ku/software/prefixspan/

表 1: 実験結果 (新聞記事)

minsup	抽出時間 (秒)		
	N-gram	チャンク	係り受け
2	2.2	N/A	355.6
5	2.0	3.2	26.7
10	1.9	3.0	24.0
15	1.9	2.8	22.9
20	1.9	2.9	22.1

表 2: 実験結果 (小説)

minsup	抽出時間 (秒)		
	N-gram	チャンク	係り受け
2	0.46	0.74	7.8
5	0.43	0.60	5.8
10	0.41	0.57	5.2
15	0.41	0.55	4.8
20	0.41	0.55	4.6

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

2 種類の異なるデータを連結し, その相関関係を同時にマイニングするという試みがデータマイニングの分野で定着しつつある [3]. このような考え方を応用した興味深い例として, 機械学習の素性抽出と対訳コーパスからの対訳パターン抽出を挙げる.

6.1 機械学習の素性抽出

機械学習に基づく自然言語処理がその精度, 頑健性から注目を浴びている. 機械学習を適用するには, まず最初に素性集合を定義する必要がある. しかし, 自然言語処理において, 素性集合の多くは発見的に設定されている. さらに, 半構造化テキストデータに対し, そのクラスを推定するようなタスクにおいても, その構造をどういう素性集合で表現するかは恣意的に定義されている[9].

一方で, 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することが可能となる. 最後に, こら頻度から, 相互情報量などを計算し, 有効なパターンを新たな素性として選択する.

このような考えに基づいた手法として, 坪井 [7] らは, 著者判定のタスクにおいて, テキスト中に頻出する単語の列を素性として追加するとことで, 精度向上に繋がったと報告している.

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. 対訳パターン抽出のタスクは, 大きく 3 つのサブタスクから構成される.

(1) 対訳パターンの候補生成
(2) dice 係数, 相互情報量などを用いた類似度計算
(3) (2) を用いた候補選択

PrefixSpan は, (1) の候補生成に有効である.

例として, 文対応があらかじめとれている英語, 日本語の対訳コーパスを考える. まず, 英語, 日本語を 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 つの系列データベースを作成する. さらに, 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ような射影の場合は, 任意の射影が許可されるように関係関数を設計する (図 14).

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.

[9] 文節係り受け解析など. 文節は複数の形態素から構成されるが, 恣意的に定義された主辞や機能語といった形態素のみを素性としている場合が多い.

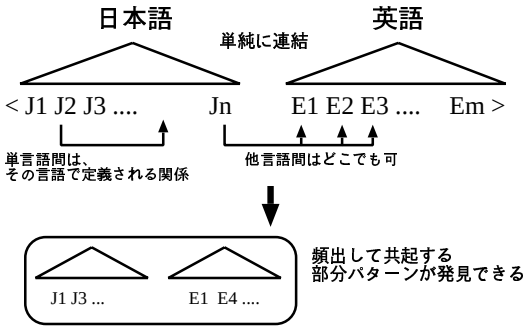

図 14: 対訳表現抽出

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. 構造としては, 以下の 2 つを用いた. 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.

- 系列
  単言語のパターン中のアイテムは, その語順のみ考慮される. これは, オリジナルの PrefixSpan と同じ設定である.
- N-gram
  単言語のパターン中のアイテムは連続している必要がある. ただし, N の大きさには制限は無い. N の大きさに制限が無いことを除けば北村 [4] らと同じ設定である.

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 秒であった. これらの実行時間は十分実用的であると考える. 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

また, 北村 [4] らは, 言語的に意味があるパターンが抽出されにくい事と, 計算コストの関係から, $N \leq 10$ のみの N-gram に限っている. しかし, PrefixSpan を用いた場合は, 長さ N に制限を設けなかったにもかかわらず, 7 秒という極めて高速な時間で候補生成が行えた事は興味深い.

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. 実際に, 新聞記事と小説から頻出部分パターンを抽出する実験を行い, 現実的な時間で, 効率良く頻出パターンを抽出できることを確認した. また, 機械学習や対訳パターンの抽出に本手法が提案できる可能性を示した.

本稿では, 実際のテキストデータから頻出するパターンを抽出する実験のみに留まり, 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. また, 本稿では, 個々の関係について個別に関係関数を定義したが, 個々の構造について一意の関係関数が定義できるかどうかの厳密な議論は行っていない. 今後は木構造, 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. 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*, pp. 487–499. Morgan Kaufmann, 12–15 1994.

[2] 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.

[3] Helen Pinto and Jiawei Han and Jian Pei and Ke. Wang and Qiming and Chen and Umeshwar Dayal. Multi-Dimensional Sequential Pattern Mining. In *Proc. 10th ACM International Conference on Information and Knowledge Management (CIKM'01)*, November 2001.

[4] Mihoko Kitamura and Yuji Matsumoto. Automatic Extraction of Word Sequence Correspondences in Parallel Corpora. In *Proc. 4th Workshop in Very Large Corpora*, pp. 79–87, 1996.

[5] 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.

[6] 安部賢治, 川副真治, 浅井達哉, 有村博紀, 坂本比呂志, 有川節夫. 頻出順序木パターン発見に基づくウェブマイニング. 人工知能学会研究会資料 SIG-FA/KBS-J22, pp. 133–139, 2001.

[7] 坪井祐太, 松本裕治. Authorship identification for heterogeneous documents. 第 148 回 情報処理学会 自然言語処理研究会 NL-148.

[8] 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.