系列パターンマイニングを用いた有効な素性の組み合わせの発見

工藤 拓　　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,matsu}@is.aist-nara.ac.jp

近年, Support Vector Machine を中心とする Kernel 法が注目され, 自然言語処理においても良い結果を示している. Kernel 法により, これまで巧妙に選択する必要があった「組み合わせ素性」を一般性や計算量を落とすことなく取り入れることができる. しかし, Kernel を用いた場合には, 素性の組み合わせは陰に展開されるため, 有効な素性の分析が難しく, さらに, 解析時の計算量が大きくなる問題がある. 本稿では, マイニングアルゴリズムの 1 つである PrefixSpan を用い, サポートベクターの集合から有効な素性の組み合わせを発見し, 解析の速度向上を試みる. 日本語係り受け解析における実験では, Kernel を用いた通常の解析器に比べ, 約 30 倍の速度向上に繋がった.

キーワード: Support Vector Machine, Kernel 法, Data Mining, PrefixSpan, 係り受け解析

Mining Effective Feature Combinations from Support Vectors

Taku Kudo　　Yuji Matsumoto

Graduate School of Information Science,Nara Institute of Science and Technology

8916-5 Takayama, Ikoma Nara 630-0101 Japan

{taku-ku,matsu}@is.aist-nara.ac.jp

The kernel method (e.g. Support Vector Machines) has gained the focus of attention recently. In the field of Natural Language Processing, many successes have been reported. The merit of the kernel method is that the *effective feature combinations*, which have been manually selected in the previous approaches, are implicitly expanded without loss of generality and computational cost. However, the kernel-based classifier is usually too slow to apply to large-scale text analysis. In this paper, we extend the PrefixSpan algorithm, one of efficient algorithms for sequential pattern mining, to explicitly extract effective feature combinations from a set of support vectors and make a classification speed faster. Experimental results on Japanese dependency analysis show that our new classifier(parser) is about 30 times faster than standard kernel-based classifier(parser).

Keywords: Support Vector Machines, Kernel Method, Data Mining, PrefixSpan, Dependency Analyisis

1 はじめに

近年, 学習理論に基く自然言語処理が急激に進展しており, 多くの研究者から注目されている. 得に, Support Vector Machine[12] は, テキストチャンキング [8], 固有名詞抽出 [13, 4], 構文解析等 [14] の自然言語処理タスクに応用され, いずれにおいても高い精度を示している. 今後もそのタスクの裾野を広げて行くであろう.

パターン認識における従来の方法論は, 与えられた学習データを分類に寄与するであろうできるだけ小さな素性集合で表現し[1], その後に, 分類誤差が小さくなるような識別関数を構成するものであった. 前処理の素性選択こそが, 汎化誤差を小さくする鍵となる. いっぽう, SVM では, Kernel 関数を使い, 超高次元空間上にデータを写像し, そこでの線型分離を考える. このとき, データを表現する素性空間は非常に大きいのにもかかわらず, 分離平面はマージン最大化により決定されるため, 結果的に汎化誤差を小さくすることができる.

この方法論の違いは, 非常に大きい. 従来の機械学習を用いた自然言語処理の多くは, 前者の方法論であったため, 素性集合の設計がひとつの腕の見せどころであった. もちろん, 基本素性としてどのような素性集合を仮定すればよいかが問題となるが, 多くの自然言語処理タスクでは, 1 つの素性のみで決定できることが少ないため[2], それらの基本素性の任意の個数の組み合わせも同時に考慮しなければならない. 結果として, 候補となる素性集合が膨大になり, この中から有効な素性を選択する必要があった.

SVM では, 巧妙な素性選択をせずとも, 従来手法で巧妙に選択した場合と比較して, 少なくとも同等か場合によってはそれ以上の認識率が得られることが実験的に知られている. また, 素性の組み合わせに関しても, Polynomial Kernel を用いることで, 計算量や一般性を落とすことなく考慮することが可能である.

しかし, SVM に問題点が無いわけではない. まず, Kernel 関数は, 陰に素性の組み合わせを展開するために, どのような素性の組み合わせが (一般には事例の部分構造が) 分類に寄与しているのかが分からない. 分類に有効な素性は, 我々が知らない一種の知識と考えられるため, 有効な素性そのものを分析対象したいことがある. もう一つの問題とは分類速度である. Kernel 関数を用いた場合の分類速度は, サポートベクターの数 (時として数万になる) に依存するため, テキストマイニングといった大規模テキストデータの解析が必要な場合, 適用が難しい.

本稿では, この 2 つの問題を解決すべく, データマイニングの手法を使い, 学習後のサポートベクターの集合から, 分類に寄与する有効な素性の組み合わせ (一般には事例の中の部分構造) を抽出, 提示する手法を提案する. また, この結果を用い, 分類の高速化を試みる. 日本語係り受けにおける実験では, Kernel を用いた従来の分類器に比べ, 30 倍の速度向上に成功した.

2 Kernel Method

2.1 Support Vector Machines

正例, 負例 の二つのクラスに属す学習データのベクトル集合を,

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_L, y_L) \qquad \mathbf{x}_i \in \mathbf{R}^N,\ y_i \in \{+1, -1\}$$

とする. SVM では, 二値分類を仮定しているために, y_i は, クラスに応じて +1 もしくは, -1 をとるものとする. この時, SVM の分離関数は, 次式で与えられる.

$$y = \mathrm{sgn}\Big(\sum_{i=1}^{L} y_i \alpha_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) + b\Big) \tag{1}$$

ただし,

(1) ϕ は, $\mathbf{R}^N$ から $\mathbf{R}^H$ (一般に $N \ll H$) への写像関数であり,

(2) $\alpha_i, b \in \mathbf{R},\ 0 \leq \alpha_i \leq C$ である.

式 (1) は, テストデータを高次元空間に写像したデータ $\phi(\mathbf{x})$ と l 個の学習データを写像したデータ $\phi(\mathbf{x}_i)$ との内積を全事例についてそれぞれ計算し, 重み $y_i\alpha_i$ 線形結合した形となっている.

ここで, 高次元空間への写像関数 ϕ は, すべての事例が, $\mathbf{R}^H$ における超平面で分離できるようにユーザが設計する. 通常, H は, N よりもはるかに高い次元になるために計算量が大きくなる問題がある. この問題を解決する手段として Kernel 関数がある.

式 (1) より, 実際に SVM を構成する場合には, 写像関数は, $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x})$ という内積の形で出現する. 本稿では, α_i, の最適化手法について言及しないが, SVM における学習の式も, 写像関数の内積のみで表現できる. したがって, 高次元空間 R^H 上での内積 $\phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$ が効率良く計算できれば, R^H が高次元空間であることの計算量的問題は解決できる. このように, 写像された高次元空間上での内積を表現するための関数 $K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2)$ を Kernel 関数と呼ぶ. Kernel 関数を用いると, 式 (1) は以下のように書き直すことができる.

[1] 人手で巧妙に素性集合を表現したり, PCA 等の次元圧縮手法を用いて素性集合を選択する

[2] 係り受けタスクは, 係り元の係り先の基本素性 (単語, 品詞, 活用など) の組み合わせが必要である.

$$y = \mathrm{sgn}\Big(\sum_{i=1}^{L} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b\Big) \qquad (2)$$

式 (2) より, SVM の学習, 分類には, 陽に表現された素性の集合は必ずしも必要ではなく, 事例間の類似度 (内積) を与える Kernel 関数のみ定義できればよいことが分かる[3]. この「分類したいデータ構造に関する情報を Kernel 関数と形で学習アルゴリズムとは独立に設計できる」という性質こそが, SVM を中心とする Kernel 法が急激に脚光を浴びることとなった 1 つの要因と考えられる.

実際に, 近年, 素性の集合として表現するのが困難である離散データの分類のための Kernel 関数の提案が活発である. 文字列間の内積を与える String Kernel[10], 順序木の内積を与える Tree Kernel[5, 3], さらに, 無向グラフのノード間の内積を与える Defffusion Kernel [6] などが提案されている.

2.2 Kernel 法の問題点

Kernel 法は, 非常にエレガントではあるが, 実際の応用においては, 以下の 2 つの問題があると我々は考えている.

- 有効な素性の分析が困難
  Kernel 法では, 素性空間が陰に表現されてしまうため, いったいどのような素性 (部分構造) が分類に寄与したのか, 分析が困難である.
- 分類時の計算量
  式 (2) から分かるように, Kernel を用いた場合の分類の計算量は, サポートベクターの数に比例する. サポートベクターの数は, 数万を超える事もあるために, 分類にかかる計算コストが非常に大きく, 実際の応用に耐えられない場合が多い.

この 2 つは, 自然言語処理において特に深刻な問題点である. かつて, 自然言語処理において決定木が爆発的に使用された理由として, ルールが木という形で表現され, その結果を内省と照らし合わせるといった分析が可能であったことがある. このような分析は, 自然言語処理ではごく一般的に行われている. SVM においては, 分類に有効な事例がサポートベクターという形で表現されるが, その事例のどの部分が有効に機能しているのかは与えてくれない.

また, 近年大量のテキストデータから知識を発掘するといった, Text Mining の研究が活発になっている. これらのタスクにおいては, 大規模なテキストデータを高速に解析する必要があり, 通常の SVM を用いた分類器では遅くて実用に耐えない.

[3]実際には Kernel として満たすべき制約 (Mercer's condition) が存在する

3 マイニングアルゴリズムによる有効な部分構造の発見

本稿では, SVM によって求められたサポートベクターの集合から, 分類に寄与する部分構造を発見することを試みる.

3.1 基本的なアイデア

式 (1) を以下のように書きかえる.

$$\begin{aligned} y &= \mathrm{sgn}\Big(\sum_{i=1}^{L} y_i \alpha_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) + b\Big) \\ &= \mathrm{sgn}\Big(\mathbf{w} \cdot \phi(\mathbf{x}) + b\Big) \qquad (3) \\ \text{ただし } \mathbf{w} &= \sum_{i=1}^{L} y_i \alpha_i \phi(\mathbf{x}_i) \end{aligned}$$

$\mathbf{w}$ は, $\phi(\mathbf{x})$ と同じ H 次元のベクトルであり, その要素は, 写像後の各素性が分類にどれだけ寄与するかを表現する. もし, $\mathbf{w}$ が計算できると, 前節で取りあげた問題点は解決できると考える. 以下にその理由を述べる.

- 有効な素性の分析
  $\mathbf{w}$ の各要素について絶対値の大きい軸は, 分類に寄与する軸だと考えられる. すなわち, 事例 x の中で分類に寄与する部分構造を目に見える形で提示することが可能である.
- 分類時の計算量
  分類は, 式 (3) で実行される. 式 (3) は, サポートベクターの数に依存せず, 写像空間中での単純な内積を計算すればよいので, 一般に高速になる. しかし, 写像そのものにコストが必要な場合はその限りではない.

ここでの問題は, いかにして $\mathbf{w}$ を求めるかにある. 事例が少なく, 写像した空間も計算機で扱える程度の次元の場合は, 式 (3) を直接計算可能である. これは, 磯崎らが 2 次の polynomial kernel に限り, 素性中のすべての 2 つまでの組み合わせを全展開した手法と本質的に同一である [4].

しかし, 3 次や 4 次の polyinomial Kernel, String Kernel, Tree Kernel といった Kernel に対し, 素性の組み合わせや事例中の部分構造を全て展開することは, 部分構造の数が, 指数的に増えるために非常に困難である. さらに, RBF Kernel といった関数は, 写像後の空間がヒルベルト空間 ($H = \infty$) になるため, そもそも全展開ができない.

3.2 写像空間上での近似解 $\mathbf{w}'$

$\mathbf{w}$ の近似解 $\mathbf{w}'$ を次のように定義する.

定義 1 ($\mathbf{w}$ の近似解 $\mathbf{w}'$) $\mathbf{w} \in \mathbf{R}^H$ の各要素 $w_j (j = 1, \ldots, H)$ について, $\sigma_{neg} < w_j < \sigma_{pos}$ ($\sigma_{neg} < 0$, $\sigma_{pos} > 0$, σ_{neg}) となる場合は, $w_j = 0$ に近似する. このようにして作成されたベクトルを $\mathbf{w}$ の近似解 $\mathbf{w}'$ と定義する.

$\mathbf{w}'$ は, 0 付近の微少な範囲 $(\sigma_{neg}, \sigma_{pos})$ 内にある重みをもつ部分構造は分類に寄与しないという仮定の基での近似解である. また, たとえ $\mathbf{w}$ がヒルベルト空間になっても, $\mathbf{w}'$ の非 0 の要素は有限となる[4].

$\mathbf{w}'$ の導出の最も単純な方法として, 部分構造を全展開し, 定義どおりに直接 $\mathbf{w}'$ を作成することである. しかし, この手法は全展開を必要とするため, 前節で述べた計算量的問題点の解決にはならないことに注意されたい. ここで, $\sigma_{neg} < w_j < \sigma_{pos}$ となる場合は, その素性 (部分構造) の軸が無いものとみなせることに注意すると, $\mathbf{w}'$ の導出は, 次に述べるマイニングタスクと本質的に同一である.

3.3 有効部分構造のマイニング

定義 2 (有効部分構造のマイニング) サポートベクターの集合 $(y_1\alpha_1, \mathbf{x}_1), \ldots, (y_L\alpha_L, \mathbf{x}_L)$, 写像関数 ϕ が与えられた時に, $w_j \geq \sigma_{pos}$ または $w_j \leq \sigma_{neg}$ となるような, 部分構造 p_j と, その重み w_j を網羅的に求めるタスクを有効部分構造のマイニングと定義する.

この問題に対し, 本稿では, データマイニングアルゴリズムを適用する. データマイニングアルゴリズムとは, 任意の構造 (集合, 系列, 順序木, グラフなど) の集合から, 頻出する部分構造 (部分集合, 部分系列, 部分木, 部分グラフ) を高速に, 効率良く発見, 抽出するアルゴリズムの総称である. これらのアルゴリズムの特徴として以下が挙げられる.

- 部分構造を全展開できないような大規模なデータを想定しいる.
- 種となる小さな部分構造から始め, 深さ優先, もしくは幅優先で部分構造を広げていきながら, 頻出する部分構造のみを抽出する.

Kernel 関数が持つ射影関数 ϕ が, どのような部分構造の空間にデータを射影するかによって, 選択するデータマイニングアルゴリズムが決定される.

以下に, 有効部分構造マイニングの具体的な適用方法の手順を示す. 通常のマイニング手法と異なる点は, 正例, 負例, 個別に頻出パターンをマイニングし, 最後にその結果をマージする点にある.

[4]もし, $w_j < \sigma_{neg}$ または $w_j > \sigma_{pos}$ となる軸が無限個ある場合, 自分自身との内積が発散してしまい, 内積が定義できない.

(1) サポートベクターの集合 $(y_1\alpha_1, \mathbf{x}_1), \ldots, (y_L\alpha_L, \mathbf{x}_L)$, を, 正例の集合 $\Gamma_{pos} = \{(y_i\alpha_i, \mathbf{x}_i) | y_i = +1\}$ と負例の集合 $\Gamma_{neg} = \{(y_i\alpha_i, \mathbf{x}_i) | y_i = -1\}$ に分割する.

(2) $y_i\alpha_i$ を事例 $\mathbf{x}_i$ の頻度とみなす

(3) 正例の集合 Γ_{pos} に対し, 頻度 σ_{pos} 以上の部分構造と頻度のタプルの集合をマイニングアルゴリズムを使って抽出する. 抽出された部分構造と頻度の集合を $\Omega_{pos} = \{\langle p_j, \eta_j \rangle\}$ とする.

(4) 負例の集合 Γ_{neg} に対し, 頻度 (最小サポート) σ_{neg} 以下の部分構造と頻度のタプルの集合をマイニングアルゴリズムを使って抽出する. 抽出された部分構造と頻度の集合を $\Omega_{neg} = \{\langle p_j, \mu_j \rangle\}$ とする.

(5) 正例負例の各部分構造 p_j について, $w_j = \eta_j + \mu_j \geq \sigma_{pos}$ または, $w_i = \eta_j + \mu_j \leq \sigma_{neg}$ となるような部分構造 p_i を抽出し, それらを結果集合 $\Omega = \{\langle p_j, w_j \rangle\}$ とする.

σ_{pos} および σ_{neg} は, どれくらいの近似精度を要求するかを決定する変数である. 一般に, 正例と負例の事例数には, 偏り (bias) がある. その偏りを考慮し, ユーザは閾値 $\sigma(> 0)$ のみを与え, それを以下のように正例と負例の数で線型分配して $\sigma_{pos}, \sigma_{neg}$ を決定する.

$$\sigma_{pos} = \sigma \cdot \Big(\frac{\text{サポートベクターの正例数}}{\text{全サポートベクター数}} \Big)$$

$$\sigma_{neg} = -\sigma \cdot \Big(\frac{\text{サポートベクターの負例数}}{\text{全サポートベクター数}} \Big)$$

4 具体例: 部分集合 Kernel

前節で, Support Vector からの部分構造マイニングの一般的な手法について述べた. 本節では, 部分集合 Kernel という種類の Kernel を取り挙げ, その具体的な適用方法について述べる.

4.1 バイナリ素性

自然言語処理タスクにおける事例は, その事例にマッチする任意の条件の集合という形で表現されることが多い. これは, 言語データそのものが離散データあることに起因する. このような離散データは, 素性空間を以下のようなバイナリで表現することと等価である.

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_L, y_L) \qquad \mathbf{x}_i \in \{0,1\}^N,\ y_i \in \{+1, -1\}$$

本稿では, 応用として自然言語処理を念頭に置いているため, 上記のようなバイナリ素性で表現されるデータのみを考える.

4.2 部分集合 Kernel

バイナリ素性として表現される事例 $\mathbf{x} \in \{0,1\}^N$ を, 事例 $\mathbf{x}$ の k 個 ($k = \{0,1,\ldots,N\}$) のすべての「部分集合」の「集合」(k 個までのすべての素性の組み合わせの「集合」) の空間に射影するような 射影関数 ϕ を持つ Kernel を**部分集合 Kernel** と呼ぶこととする. ただし, r 個 で構成される部分集合には, **部分集合重み** $c_r \in \mathbf{R}$ が付加される. 部分集合重み c_r は, r 個で構成される部分集合をどれだけ重要視するかを与える事前分布となる. 個々の部分集合 Kernel は, 固有の部分集合重み分布を持ち, その分布が Kernel としての性質を与える.

$$
\begin{aligned}
\phi(\mathbf{x}) \quad = \quad (& \underbrace{c_0}_{\text{0 個の部分集合 (組み合わせ)}}, \\
& \underbrace{c_1 x_1, c_1 x_2, \ldots, c_1 x_N}_{\text{1 個の部分集合 (組み合わせ)}}, \\
& \underbrace{c_2 x_1 x_2, \ldots, c_2 x_1 x_N, \ldots, c_2 x_{N-1} x_N}_{\text{2 個の部分集合 (組み合わせ)}}, \\
& \ldots, \\
& \underbrace{c_k x_1 x_2 .. x_k, \ldots, c_k x_{N-k} x_{N-k+1} .. x_N}_{k \text{ 個の部分集合 (組み合わせ)}})
\end{aligned}
$$

$$
\text{ただし}, \mathbf{x} = \{x_1, x_2, \ldots, x_N\},
$$
$$
x_j \in \{0,1\}, c_k \in \mathbf{R}, k = \{0,1,\ldots,N\}
$$

この部分集合 Kernel に族す Kernel として, 本稿では, 頻繁に使用される Polynomial Kernel と RBF Kernel を取り挙げる.

4.2.1 Polynomial Kernel

$\mathbf{R}^2$ から $\mathbf{R}^6$ への写像関数 ϕ を

$$
\phi((x_1, x_2)) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)
$$

とする. この時, 高次元空間上での内積は

$$
\phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) = (1 + \mathbf{x}_1 \cdot \mathbf{x}_2)^2 \tag{4}
$$

となる. つまり $\mathbf{R}^6$ での内積 (左辺) が $\mathbf{R}^2$ の空間の内積 (右辺) のみで計算できることを意味する. バイナリ素性の場合は, $x_1^2 = x_1, x_2^2 = x_2$ となることに注意すると射影関数は

$$
\begin{aligned}
\phi((x_1, x_2)) &= (1, \sqrt{3}x_1, \sqrt{3}x_2, \sqrt{2}x_1 x_2) \\
&\text{ただし } x_1, x_2 \in \{0,1\}
\end{aligned}
$$

となる. ϕ で表現される 4 番目の軸 $\sqrt{2}x_1 x_2$ は, 素性 x_1, x_2 の**組み合わせ**を表現している. 以上より, 式 (4) で表現される 2 次の polynomial kernel の部分集合重みは $c_0 = 1, c_1 = \sqrt{3}, c_2 = \sqrt{2}$ となることが分かる. 以上の議論を, 一般の d 次の polyinomial kernel に拡張する.

$$
K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) = (1 + \mathbf{x}_1 \cdot \mathbf{x}_2)^d \tag{5}
$$

$\mathbf{x}_1 \in \{0,1\}^N, \mathbf{x}_2 \in \{0,1\}^N$ に対し, d 次の polynomial Kernel は二項定理より以下のように展開できる.

$$
\begin{aligned}
K(\mathbf{x}_1, \mathbf{x}_2) &= (1 + \mathbf{x}_1 \cdot \mathbf{x}_2)^d \\
&= \sum_{l=0}^{d} {}_dC_l (\mathbf{x}_1 \cdot \mathbf{x}_2)^l
\end{aligned}
$$

$(\mathbf{x}_1 \cdot \mathbf{x}_2)^l$ において, r 個で構成される部分構造の個数を $\tau(l,r)$ と表現すると[5], d 次の polynomial Kernel は, 部分構造重み $c_r = \sqrt{\sum_{l=r}^{d} {}_dC_l \cdot \tau(l,r)}$ を持つ (ただし $r = \{0,1,\ldots,d\}$). 3 次の polynomial kernel の場合は, $c_0 = 1, c_1 = \sqrt{7}, c_2 = \sqrt{12}, c_3 = \sqrt{6}$ となる.

4.2.2 RBF Kernel

$\mathbf{x}_1 \in \{0,1\}^N, \mathbf{x}_2 \in \{0,1\}^N$ に対し, RBF Kernel (分散パラメータ $s \in \mathbf{R}, s > 0$) は, 以下のように漸近展開できる.

$$
\begin{aligned}
K(\mathbf{x}_1, \mathbf{x}_2) &= \exp(-s||\mathbf{x}_1 - \mathbf{x}_2||^2) \\
&= \exp(-s||\mathbf{x}_1||^2) \exp(-s||\mathbf{x}_2||^2) \exp(2s(\mathbf{x}_1 \cdot \mathbf{x}_2)) \\
&= \exp(-s||\mathbf{x}_1||^2) \exp(-s||\mathbf{x}_2||^2) \cdot \\
&\quad \Big(\sum_{l=0}^{\infty} \frac{(2s)^l (\mathbf{x}_1 \cdot \mathbf{x}_2)^l}{l!} \Big)
\end{aligned}
$$

$(\mathbf{x}_1 \cdot \mathbf{x}_2)^l$ において, r 個で構成される部分構造の個数を $\tau(l,r)$ と表現すると, RBF Kernel は, 部分構造重み $c_r = \exp(-s||\mathbf{x}||^2) \sqrt{\sum_{l=r}^{\infty} \frac{(2s)^l \tau(l,r)}{l!}}$ を持つ (ただし $r = \{0,1,\ldots,n\}$).

4.3 PrefixSpan

部分集合 Kernel に対するマイニングアルゴリズムは, 素性集合から頻出する部分集合を枚挙するようなアルゴリズムとなる. 代表的なアルゴリズムとして, Apriori[1] があるが, 実行速度や実装の手軽さを考えて, 本稿では PrefixSpan[11] を用いる. PrefixSpan は, もともと系列データのためのマイニングアルゴリズムであるが, 集合に対し, 個々の要素を辞書順にソートしたものを系列とみなせば, 集合のマイニングも可能となる.

PrefixSpan は, 系列の集合 (系列データベース) から, ユーザが与えた最小サポート値, ξ 回以上の系列

[5] $\tau(l,r)$ の具体的な値は, 付録 A を参照のこと

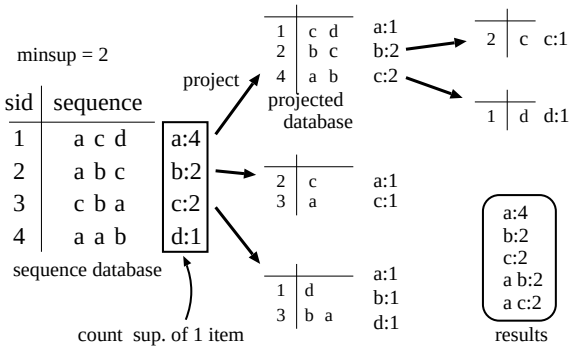

図 1: PrefixSpan の動作過程

に含まれるすべての部分系列を列挙するアルゴリズムである. このアルゴリズムは, 分割統治法に基づき, パターンを深さ優先で広げていきながらマイニングを行う. 具体的には, まず, 1 つの要素のみで構成される系列のうち, 最小サポート値以上の部分系列を列挙する. 次に, 頻出パターンが出現する系列の postfix を集め, それをあらたな部分系列集合とする. この部分系列集合を, 改めて系列データベースとみなし, 同じ処理を再帰的に繰り返す.

図 1 に, $\xi = 2$ とした場合の PrefixSpan の動作過程を示す. まず, サイズ 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.

以上が, オリジナルの PrefixSpan であるが, 部分集合 Kernel におけるマイニング問題に適用する場合は, 以下の 3 点を変更する必要がある.

- 頻度 $y_i\alpha_i$
  系列データベースの各系列が, サポートベクター $\mathbf{x}_i$ に対応する. サポートベクター $\mathbf{x}_i$ は, それに対応する重み $y_i\alpha_i$ を持つ. PrefixSpan を動作させる時には, $y_i\alpha_i$ をその系列の**頻度**とみなす.
- サイズの制限
  d 次の polynomial kernel の場合は, d 個 までの部分系列を列挙するだけで十分である. そのため, $d+1$ 個の部分系列への射影は行わない.
- 部分集合重み c_r
  サイズ r の部分系列の頻度を c_r 倍する. しかし, 大きい部分系列に対する重みが, 小さい部分系列に対する重みより大きい場合, 射影が行われない場合が発生し, パターンを取りこぼしてしまう可能性がある. そのため, 便宜的に, c_r のうち最大の値 $m(= \max(\{c_1, c_2, \ldots, c_n\}))$ を選び, 部分系列のサイズに関わらず, m 倍した頻度によって射影をコントロールする. 一方, 実際の部分系列の頻度計算には, 重み c_r を用いる.

4.4 TRIE による部分集合の格納

マイニング後, 部分集合 p_i と, それに付与される重み w_j のタプルの集合 $\Omega = \{\langle p_j, w_j \rangle\}$ が抽出される. 部分集合 p_j のそれぞれは, 多くの共通した部分構造を持つため, それを単純な Hash 等を使って格納するのは記憶効率が悪い.

その問題を解決するために, 共通部分を共有する TRIE を作成する. 具体的には, 部分集合の要素を辞書順にソートし, 要素の左から順に TRIE に格納する. TRIE の各ノードには, その ノードに対応する部分構造が持つ重み w_j が格納される.

事例 $\mathbf{x}$ を分類する時は, $\mathbf{x}$ の部分構造を深さ優先に列挙しながら, TRIE の ノードを探索していけばよい. TRIE の実装として, Double-Array [2, 7] を用いた.

5 実験

5.1 設定

提案手法の有効性を示すために, 日本語係り受け解析タスクに対し実験を行った. 採用した係り受け解析モデルは, Cascaded Chnking Model[14] と呼ばれるモデルで, 直後に係るか係らないかのみで動作し, その判定に SVM を用いている.

実験に用いたコーパスは京大コーパス（Version 2.0）[9] の一部で, 学習には 1 月 1 日から 8 日分（7958 文）, テストには 1 月 9 日分の （1246 文）を用いた. これは, 文献 [14] で用いたデータと同一である. また, 実験には, 3 次の polynomial kernel を用いた. これも, 文献 [14] と同一の設定である. RBF Kernel については, 分散パラメータの値に精度が敏感に反応し, 本タスクでは実用に向かなかったので, 実験を行っていない. すべての実験は, XEON 2.4GHz 4.0Gbyte Memory の Linux 上で実験を行った. マイニング, 解析のプログラムは, すべて C++ で実装した.

表 1 に, 実験データのサイズ等の情報, それぞれの解析精度 (係り受け精度), および式 (2) の通常の Kernel を用いて解析を行った場合の解析時間を示す.

5.2 結果

表 2 に, σ を 0.1 から 0.0001 まで変化させた時の, 解析時間, 解析精度, モデルサイズをまとめる. $\sigma = 0.0005$ の時に, テストデータを通常の Kernel を用いた分類器と同じ精度で解析することができた. この

- {係り元, 主辞, 品詞細分類, 普通名詞} and
  {係り元, 機能語, 表層, と} and {係り先, 主辞, 品詞細分類, 普通名詞}
  (「普通名詞　と　普通名詞」並列構造)
- {係り元, 機能語, 表層, を} and {係り先, 主辞, 表層, 中心} and {係り先, 機能語, 表層, に}
  (「〜を 中心に」 頻出言い回し)
- {係り元, 機能語, 表層, から} and
  {係り元, 機能語, 品詞, 助詞} and {係り先, 機能語, 表層, まで}
  (「〜から 〜まで」 頻出言い回し)

図 2: 取り出された 3 つ組み部分構造の一部

時の解析速度は, 通常の Kernel を用いた分類器に比べ, 約 1/30 (0.0097/0.2848) となった.

図 2 に, 実際抽出された 3 つ組パターンのうち, 値の大きいものを 3 つ列挙した.「名詞 と 名詞」「〜を中心に」「〜から〜まで」といった, 頻出する言い回しを適度に一般化した素性の組みが抽出されている.

5.3 頻度によるフィルタリング

実験より, 通常の Kernel に基づく解析器と同程度の解析精度を得るためには, 少なくとも $\sigma = 0.0005$ 程度に設定する必要があることが分かる. しかし, この時のパターン数は, 約 826 万 で, モデルのファイルサイズは, 391MB にも及び, 非常に大きく, あまり実用的ではない.

一般に, Ω のサイズが小さくなれば, TRIE を探索する空間が減少するために, 速度向上に繋がる. そこで, 不必要な部分構造を削除するために, サポートベクターの集合から, Ω 中の各部分構造の出現頻度を計算し, 高頻度の部分構造のみを残し, 残りを削除するという単純なフィルタリング実験を行った. 各部分構造の頻度は, PrefixSpan を用いて効率良く計算できる. σ=0.0005 にし, 頻度閾値 $\xi(= \{1, 2, 3, 4\})$ 回以上のパターンのみを残した実験結果を表 3 に示す.

結果 $\xi = 2$ とした場合, サイズを約 1/3 にすることができた.　また, 若干ながら精度向上 (89.29%→89.34%) が確認できた. これは, 低頻度にもかかわらず, 高い重みを持つ部分構造が削除されたからと考える. このような部分構造は, 学習データの誤りや特殊な事例と考えられ, 過学習の原因になりやすい.

6 今後の課題とまとめ

本稿では, データマイニングの手法を使い, 学習後のサポートベクターの集合から, 分類に寄与する有効な素性の組み合わせ (一般には事例の中の部分構造) を抽出, 提示する手法を提案した. 具体的には, 部分集合 Kernel という種類の Kernel に注目し, その Kernel の 部分構造の展開に データマイニングアルゴリズムの 1 つである PrefixSpan を用いた. 日本語係り受けの実験では, Kernel を用いた従来の分類器に比べ, 30 倍の速度向上に成功した.

今後は, 部分集合 Kernel のうち RBF Kernel に関する実験, 係り受け解析以外のタスクにおける実験を行いたいと考える. さらに, 部分集合 Kernel 以外の Kernel (Tree Kernel, String Kernel) についても 部分木 や 部分系列のマイニングアルゴリズムを用い, 提案手法の有効性を検証したいと考える.

付録 A

$$\tau(l, r) = \sum_{m=0}^{r} {}_rC_m (-1)^{r-m} \cdot m^l$$

[証明]

$$\mathbf{x} = \{x_1, x_2, \ldots, x_N\},\ \mathbf{y} = \{y_1, y_2, \ldots, y_N\} \quad (\text{ただし } x_j, y_j \in \{0, 1\})$$

とする. この時, 多項定理を用いると

$$\begin{aligned} (\mathbf{x} \cdot \mathbf{y})^l &= \Big(\sum_{j=1}^{N} x_j y_j\Big)^l \\ &= \sum_{k_l \geq 0}^{k_1+\ldots+k_N=l} \frac{l!}{k_1! \ldots k_N!} (x_1 y_1)^{k_1} \ldots (x_N y_N)^{k_N} \end{aligned}$$

となる. $x_j^{k_j} \in \{0, 1\}$ に注意しながら同類項をまとめ, さらに対称性に注意すると, r 個 $(r = \{0, 1, \ldots, l\})$ の部分構造の個数は, $(x_1 y_1 x_2 y_2 \ldots x_r y_r)$ の定数項となる. よって,

$$\begin{aligned} \tau(l, r) &= \sum_{k_l \geq 1, l=1,2,\ldots,r}^{k_1+\ldots+k_r=l} \frac{l!}{k_1! \ldots k_r!} \\ &= r^l - {}_rC_1 (r-1)^l + {}_rC_2 (r-2)^l - \ldots \\ &= \sum_{m=0}^{r} {}_rC_m (-1)^{r-m} \cdot m^l \quad \blacksquare. \end{aligned}$$

表 1: 係り受け実験データとベースライン結果

学習データ数	SV 数 (正例/負例)	次元数	平均非 0 次元数	解析時間 (秒/文)	係り受け正解率 (%)
110,355	34,996 (17,528/17,468)	43,637	17.64	0.2848	89.29

表 2: 実験結果

σ	解析時間 (秒/文)	係り受け正解率 (%)	$\|\Omega\|$	TRIE ファイルサイズ (MB)
0.1	0.0013	82.02	7,289	1.8
0.05	0.0020	86.27	30,523	3.1
0.01	0.0050	88.91	73,9147	37.2
0.005	0.0067	89.05	1,924,221	93.4
0.001	0.0092	89.26	6,686,737	317.5
0.0005	0.0097	89.29	8,262,428	391.1
0.0001	0.0101	89.29	9,846,092	464.9
通常 Kernel	0.2848	89.29		

表 3: 頻度によるフィルタリング結果

ξ	解析時間 (秒/文)	係り受け正解率 (%)	$\|\Omega\|$	TRIE ファイルサイズ (MB)
1	0.0097	89.29	8,262,428	391.1
2	0.0074	**89.34**	2,450,855	118.0
3	0.0069	89.31	1,360,019	66.4
4	0.0065	89.25	945,286	46.8

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] Junichi Aoe. An efficient digital search algorithm by using a double-array structure. *IEEE Transactions on Software Engineering*, Vol. 15, No. 9, 1989.

[3] Michael Collins and Nigel Duffy. Convolution kernels for natural language. In *NIPS*, 2001.

[4] Hideki Isozaki and Hideto Kazawa. Efficient support vector classifiers for named entity recognition. In *Proceedings of COLING-2002*, 2002.

[5] Hisashi Kashima and Teruo Koyanagi. Svm kernels for semi-structured data. In *Proc. 19th Int. Conf. Machine Learning*, 2002.

[6] Risi Imre Kondor and Jon Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proc. 19th Int. Conf. Machine Learning*, 2002.

[7] Taku Kudo. Darts: Double-ARray Trie System.

[8] Taku Kudo and Yuji Matsumoto. Chunking with support vector machines. In *Proceedings of the Second Meeting of the NAACL*, 2001.

[9] Sadao Kurohashi and Makoto Nagao. Kyoto University text corpus project. In *Proceedings of the ANLP*, pp. 115–118, 1997.

[10] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, Vol. 2, , 2002.

[11] 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.

[12] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[13] 山田寛康, 工藤拓, 松本裕治. Support vector machine を用いた日本語固有表現抽出. 情報処理学会論文誌, Vol. 43, No. 1, pp. 44–53, 2002.

[14] 工藤拓, 松本裕治. チャンキングの段階適用による日本語係り受け解析. 情報処理学会論文誌, Vol. 43, No. 6, pp. 1834–1842, 2002.