組み合わせ素性に基づく分類器の高速化と係り受け解析への適用

吉永 直樹
東京大学 生産技術研究所
ynaga@tkl.iis.u-tokyo.ac.jp

喜連川 優
東京大学 生産技術研究所
kitsure@tkl.iis.u-tokyo.ac.jp

1 はじめに

機械学習に基づく分類器 [2, 12] は，多くの自然言語処理タスクでその有効性が実証されているが，一般に高精度の分類には組み合わせ素性（例: 単語 n グラム）を含む大量の素性を必要とする場合が多く，分類器の精度・速度を両立させることが難しい．そのため，膨大な Web 文書を処理場合や，実時間システムでは，素性の組み合わせの次数を下げたり，素性選択 [13] を徹底するなどして，精度を犠牲にして速度を保障することも多い．

本稿では，組み合わせ素性に基づく分類器の一般的な高速化手法を提案する．以下では，組み合わせ素性と明示的に区別するために，他の素性の組み合わせとして表現されない素性を**基本素性**と呼ぶこととする．本稿で提案する分類アルゴリズムにより，組み合わせ素性に基づく分類器は，理想的には基本素性のみからなる分類器と同じ計算オーダで分類を行うことができる．

アルゴリズムの基本的なアイデアは，タスクで頻出する基本素性ベクトルの重みを予め計算し，実際に素性ベクトルの重みを計算するときの部分結果として用いることである．具体的には，高速化対象の分類器を用いてタスクの入力データを解析し，生成された基本素性ベクトル（とその重み）をトライに保存し，分類する素性ベクトルに対して高速に（疑似）最長部分基本素性ベクトルとその重みを得る．このようにして得られた部分重みに，差分の組み合わせ素性の重みを足し合わせることで最終的な重みを得る．

提案手法を，三次の多項式カーネルを用いたサポートベクタマシンに基づく日本語係り受け解析の分類器に適用し，約 3-6 倍の高速化が得られることを確認した．

2 背景知識

本稿で提案する手法は，素性の重みの線形和でスコア（確率）を計算する一般的な分類器を高速化の対象としており，自然言語処理で典型的に用いられる対数線形モデル [2] はもちろん，組み合わせ素性を多項式カーネルで考慮するサポートベクタマシン（以下 SVM）[12] などについても，スコアの計算を素性の線形和に変換するカーネル展開 [6, 9] と組み合わせることで適用可能である．本節では，まず分類器を用いた自然言語処理について簡単に述べる．その後，実験で用いた SVM と，SVM で組み合せ素性を表現する多項式カーネルのための現状最速の分類手法であるカーネル展開について説明する．

多くの自然言語処理タスクは，一連の分類ステップとしてモデル化することができる．例えば品詞タグ付けタスクであれば，各分類ステップで各**サンプル**（文中の各単語）に適切な**クラスラベル**（品詞タグ）を割り当てる．各分類ステップ（サンプル）は，サンプルの特徴を表現する**素性ベクトル**として表現され，ベクトル中の個々の次元 x_i は，あるサンプルの一つの特徴を表現する**素性（関数）** $f_i \in \mathcal{F}$ の値である．本稿では，特定の文脈が現れたときに 1 を出力する二値素性関数 $f_i(\boldsymbol{x}) \in \{0, 1\}$ のみを考える（素性ベクトル $\boldsymbol{x}$ は二値ベクトルとなる）．以後，$x_i = f_i(\boldsymbol{x}) = 1$ のとき，素性 f_i がサンプル $\boldsymbol{x}$ で**発火する**と呼ぶ．また，$|\boldsymbol{x}|$ は $\boldsymbol{x}$ で発火した素性の数，すなわち $|\boldsymbol{x}| = |\{f_i | f_i(\boldsymbol{x}) = 1\}|$ とする．

2.1 サポートベクタマシン (SVM)

SVM は二値分類器 [12] である．二値素性関数のみを考える場合，二値分類問題の学習データは，

$$(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_L}, y_L)\ \boldsymbol{x_i} \in \{0, 1\}^n, y_i \in \{+1, -1\},$$

と表現される．ただし，$\boldsymbol{x_i}$ はサンプルの素性ベクトルであり，y_i は対応するクラスラベルである．SVM の学習結果は，以下で定義される決定関数 $y(\boldsymbol{x})$ である．

$$\begin{aligned} y(\boldsymbol{x}) &= \mathrm{sgn}(g(\boldsymbol{x}) + b) \\ g(\boldsymbol{x}) &= \sum_{\boldsymbol{x_j} \in \mathcal{SV}} y_j \alpha_j \phi(\boldsymbol{x_j}) \cdot \phi(\boldsymbol{x}). \end{aligned} \tag{1}$$

ここで，$b \in \mathbf{R}$, $\phi : \mathbf{R}^n \mapsto \mathbf{R}^H$, また $\boldsymbol{x_j} \in \mathcal{SV}$ は**サポートベクタ**（学習サンプルの部分集合）であり，$\alpha_j \in \mathbf{R}$ は各サポートベクタ $\boldsymbol{x_j}$ に対応する重みである．以後，$g(\boldsymbol{x})$ を**スコア関数**と呼ぶ．

SVM では，学習サンプルを非線形写像 ϕ を用いて高次素性空間 $\mathbf{R}^H (H \gg n)$ に写像することで，サンプルを線形分離可能とする．また，直接高次素性空間上での内積を計算する**カーネル関数** $k(\boldsymbol{x_j}, \boldsymbol{x}) = \phi(\boldsymbol{x_j}) \cdot \phi(\boldsymbol{x})$ を用いて効率的に $\mathbf{R}^H$ での内積を計算することができる．

基本素性 $f_j \in \mathcal{F}$ の組み合わせを考慮するためには，多項式カーネルと呼ばれるカーネル関数 $k_d(\boldsymbol{x_j}, \boldsymbol{x}) = (\gamma \boldsymbol{x_j} \cdot \boldsymbol{x} + c)^d$ を用いれば良い．式 1 から多項式カーネルに基づくスコア関数は以下となる．

$$g(\boldsymbol{x}) = \sum_{\boldsymbol{x_j} \in \mathcal{SV}} y_j \alpha_j (\gamma \boldsymbol{x_j} \cdot \boldsymbol{x} + c)^d. \tag{2}$$

ここで，x_i が基本素性が発火しているか否かの二値であることから，d 次の多項式カーネルは $\boldsymbol{x}$ から個々の素性の d 次以下の全組み合わせを考慮した $H = \sum_{k=0}^{d} \binom{n}{k}$ 次元のベクトル $\phi_d(\boldsymbol{x})$ への写像 ϕ_d を意味している[1]．

[1]例えば，素性ベクトル $\boldsymbol{x} = (x_1, x_2)$ とサポートベクタ $\boldsymbol{x'} = (x'_1, x'_2)$ が与えられた場合，二次の多項式カーネル $k_2(\boldsymbol{x'}, \boldsymbol{x}) = (\boldsymbol{x'} \cdot \boldsymbol{x} + 1)^2$ が返す値は $k_2(\boldsymbol{x'}, \boldsymbol{x}) = (x'_1 x_1 + x'_2 x_2 + 1)^2 = 3x'_1 x_1 + 3x'_2 x_2 + 2x'_1 x_1 x'_2 x_2 + 1$ ($\because\ x'_i, x_i \in \{0, 1\}$) となる．この関数は写像 $\phi_2(\boldsymbol{x}) = (1, \sqrt{3}x_1, \sqrt{3}x_2, \sqrt{2}x_1 x_2)$ を意味している．以下では，簡単のため写像された素性空間での定数項は定数 b に含まれているとみなし議論を進める．

カーネル展開　式 2 のスコア関数 $g(\boldsymbol{x})$ の計算量は $O(|\boldsymbol{x}|\cdot|\mathcal{SV}|)$ であり，一般に自然言語処理タスクの分類器ではサポートベクタの数が非常に大きい [9] ($|\mathcal{SV}| > 10,000$) ため，計算コストも非常に高くなってしまう．これに対し，Isozaki と Kazawa は式 2 を写像された高次素性空間での素性の重みの線形和に変換する**カーネル展開**と呼ばれる手法を提案している [6].

$$g(\boldsymbol{x}) = \boldsymbol{w}\cdot\boldsymbol{x}^d. \tag{3}$$

$\boldsymbol{x}^d$ は各要素 x^d_i が $(\phi_d(\boldsymbol{x}))_i > 0$ のとき 1 となる二値素性ベクトルであり，$\boldsymbol{w}$ は写像された高次素性空間 $\mathcal{F}^d$ の素性ベクトル $\boldsymbol{x}^d$ に対する重みベクトルである．重みベクトルは，サポートベクタ $\boldsymbol{x_j}$ とその重み α_j から前もって計算される．$\boldsymbol{w}$ の具体的な計算方法は，文献 [9] を参照されたい．式 3 の計算量は $O(|\boldsymbol{x}^d|)$ となり，$\boldsymbol{x}^d$ で発火している素性の数に比例する．

近似カーネル展開　カーネル展開は，写像された高次素性空間に含まれる組み合わせ素性に対する重みを保存するために膨大なメモリが必要となる．これに対し，Kudo らは，重みが一定以下の素性を無視するヒューリスティックスを提案している [9][2]．このフィルタリングにより，$|\mathcal{F}^d|$ を小さくすることができる．また，分類時に考慮する素性を絞ることで，分類器の精度が低下する恐れがあるものの，$|\boldsymbol{x}^d|$ を小さくすることもできるため分類速度を高速化できることが報告されている．

カーネル展開を用いた SVM では，高次の組み合わせ素性を考慮しようとすると，発火する（組み合わせ）素性 $|\boldsymbol{x}^d|$ が爆発的に増えるため，より低次のモデルに比べて大幅に解析器の速度が落ちる（この点については実験の節でも確認する）．

3　提案手法

節 2 でみたように，組み合わせ素性を考慮した分類器のスコア関数を計算するには，高次素性空間 $\mathcal{F}^d$ で素性の重みの線形和を計算する必要がある．本節では，組み合わせ素性（あるいは多項式カーネル）を用いて学習された分類器を，基本素性のみから学習された分類器と理想的には同程度まで高速化する手法を提案する．なお，以下では，d 次の多項式カーネルに基づく分類器の高速化を考えるが，同様の議論は組み合わせ素性の重みを直接推定する最大エントロピー法などにも適応できる[3]．

以下の議論では，簡単のため二値素性ベクトル $\boldsymbol{x}$ を発火した素性 $\{f_i|f_i(\boldsymbol{x}) = 1\}$ またはそのインデクス $\{i|f_i(\boldsymbol{x}) = 1\}$ の集合として表現する．従って，$\boldsymbol{x}$ は基本素性の集合 $\mathcal{F}$ の冪集合 $2^{\mathcal{F}}$ の要素 ($\boldsymbol{x} \in 2^{\mathcal{F}}$) である．

3.1　アイデア

まず，式 3 のスコア関数 $g(\boldsymbol{x})$ が基本素性ベクトル $\boldsymbol{x} \in 2^{\mathcal{F}}$ を $w \in \mathbf{R}$ に写像する関数であったことを思い出されたい．もし，$W_{\boldsymbol{x}} = g(\boldsymbol{x})$ を予め全ての $\boldsymbol{x}$ に対して前もって計算できるのであれば，$g(\boldsymbol{x})$ は単に $|\boldsymbol{x}|$ の要素をチェックするだけで，すなわち，発火する基本素性に比例する時間で得られることになる．しかしながら，$\boldsymbol{x}$ の値域 $|\{\boldsymbol{x}|\boldsymbol{x} \in 2^{\mathcal{F}}\}| = 2^{|\mathcal{F}|}$ は非常に大きい（自然言語処理タスクではしばしば $|\mathcal{F}| > 10,000$）ためこの

[2]正確には，重みが正のサポートベクタと重みが負のサポートベクタの数の割合を考慮し，これらに対して異なる閾値を用いる．

[3]素性ベクトル $\boldsymbol{x}$ が組み合わせ素性 $f \in \mathcal{F}^d$ を含む場合には，自明な縮退写像 $\phi_d^{-1} : \mathcal{F}^d \mapsto \mathcal{F}$ で $\boldsymbol{x}$ を写像して得られた $\boldsymbol{x}'$ に対して，スコア関数 $g'(y|\boldsymbol{x}') = g(y|\boldsymbol{x})$ を考えれば良い．

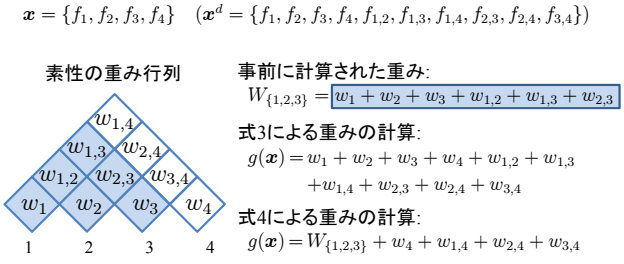

図 1: スコア関数 $g(\boldsymbol{x})$ の効率的な計算

計算は現実的でない．そこで我々は，$W_{\boldsymbol{x}'} = g(\boldsymbol{x}')$ を $2^{\mathcal{F}}$ の部分集合 $\mathcal{V}_c$ の要素 $\boldsymbol{x}' \in \mathcal{V}_c$ に対して計算しておき，$\boldsymbol{x} \notin \mathcal{V}_c$ に対するスコア関数 $g(\boldsymbol{x})$ を $\boldsymbol{x_c} \subseteq \boldsymbol{x}^4$ に対して予め計算された重み $W_{\boldsymbol{x_c}}$ を用いて以下のように計算することを考える．

$$g(\boldsymbol{x}) = W_{\boldsymbol{x_c}} + \bar{g}(\boldsymbol{x}, \boldsymbol{x_c}) \tag{4}$$

ここで，$\bar{g}(\boldsymbol{x}, \boldsymbol{x_c})$ は式 3 を用いて差分の重みを計算する関数であり，以下のように定義される:

$$\bar{g}(\boldsymbol{x}, \boldsymbol{x_c}) = \sum_{f_j \notin \boldsymbol{x}^d_{\boldsymbol{c}} \wedge f_j \in \boldsymbol{x}^d} w_j \tag{5}$$

直感的に言えば，この関数は $\boldsymbol{x}$ でのみ発火している基本素性 $f \in \mathcal{F}$ の重みと，この素性 f 含む組み合わせ素性 $f' \in \mathcal{F}^d$ の重みの和を計算している．

図 1 に $d = 2$ のときのこの計算の例を示す．図では，4 つの基本素性 f_1, f_2, f_3, f_4 が発火している素性ベクトル $\boldsymbol{x}$（$\boldsymbol{x}^d$ ではこれらに加えて二次の組み合わせ素性が 6 つ発火している）に対し，予め計算された重み $W_{\{1,2,3\}}$ を用いて $g(\boldsymbol{x})$ を計算している．まず，$W_{\{1,2,3\}}$ を得るために，3 つの素性 f_1, f_2, f_3 をチェックし，さらに残りの 4 つの素性 f_4, $f_{1,4}$, $f_{2,4}$, $f_{3,4}$ の重みを加える．一方で，この重みを式 3 で計算すると，10 の素性の重みを足し合わせる必要がある．

式 4 の計算量は $O(|\boldsymbol{x_c}|)+O(|\boldsymbol{x}^d|-|\boldsymbol{x}^d_{\boldsymbol{c}}|)$ となる．$|\boldsymbol{x_c}|$ が $|\boldsymbol{x}|$ に近ければ近いほど，この値は $O(|\boldsymbol{x}|)$ に近づく．すなわち，この計算コストを最小化するには，$\boldsymbol{x}$ に対して $\mathcal{V}_c$ から $\boldsymbol{x_c}$ を以下のように選べば良い．

$$\boldsymbol{x_c} = \underset{\boldsymbol{x}' \in \mathcal{V}_c, \boldsymbol{x}' \subseteq \boldsymbol{x}}{\operatorname{argmin}} \; (|\boldsymbol{x}'| + |\boldsymbol{x}^d| - |\boldsymbol{x}'^d|). \tag{6}$$

以上を踏まえて上記の計算で分類器を高速化するには，以下の要件を満たす必要がある．まず，メモリの制限から可能な素性ベクトルのほんの一部（つまり $|\mathcal{V}_c| \ll 2^{|\mathcal{F}|}$）についてしか重みを保持することができないので，$\mathcal{V}_c$ をどのように選ぶかということが非常に重要となる．次に，入力の $\boldsymbol{x}$ に対して，$\mathcal{V}_c$ から最適な $\boldsymbol{x_c}$ を高速に見つける必要がある．この問題は，素性を文字と考えた最長共通部分文字列問題 [3] と等価である．

まず最初の問題を解くために，自然言語処理データの偏りを利用する．つまり，自然言語処理タスクでは個々の基本素性には依存関係がある場合が多く，実データでは偏って出現している．例えば品詞タグ付けで前の単語の品詞を素性とする場合，異なる複数の品詞の素性が同時に発火することはない．また，品詞細分類（例: 固有名詞）のような他の素性（ここでは品詞，例:

[4]$\boldsymbol{x_c}$ で発火する素性が全て $\boldsymbol{x}$ でも発火することを意味する．

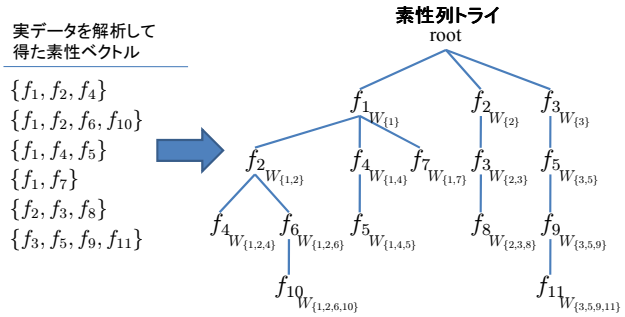

図 2: 素性列トライと接頭素性列の重みの補完

入力: 素性ベクトル $\boldsymbol{x} \in 2^{\mathcal{F}}$,
重みベクトル $\boldsymbol{w} \in \mathbf{R}^{|\mathcal{F}|}$, 素性列トライ T
出力: 重み $W = g(\boldsymbol{x}) \in \mathbf{R}$

```
sort(x)
⟨x_c, W⟩ ← prefix_search(T, x)          (1)
foreach feature j ∈ x^d − x_c^d          (2)
  W ← W + w_j
end foreach
return W
```

図 3: 素性列トライを用いた分類アルゴリズム

名詞）を細分類した，従ってその素性と必ず共起する素性もある．そこで，実データから素性ベクトルを列挙することを考える．要するに，高速化対象の分類器をそのタスクの入力データに適用し，その過程で得られる素性ベクトルを利用することとする．

次に，二番目の問題を，素性ベクトルで発火している素性に順序付けを与えソートし，最長共通接頭素性列を見つける問題として近似的に解く．ここでは素性ベクトルを発火している素性のインデクスで順序づけられた素性列だとみなし，素性ベクトルとその重みをトライに格納することとする．

トライを用いることで，入力の素性ベクトルの最長共通接頭素性列 $\boldsymbol{x_c}$ を $|\boldsymbol{x_c}|$ に比例する時間で得ることができる．以下で，素性ベクトルとその重みを保存するトライを**素性列トライ**と呼ぶこととする．図 2 は 6 つの素性ベクトルから構築された素性列トライを表している．素性列トライでは，根ノードからトライ中の任意のノードへのパスが一つの素性ベクトルを表現するため，構築時に用いた素性ベクトル（素性列）の接頭辞（一部の部分素性ベクトル）についてもトライのサイズを増やすことなく重みを自然に保存できることに注意されたい．図では，素性列トライは入力の素性ベクトルに加え，9 の部分素性ベクトルとその重みも同時に保存している．

この利点を最大限に利用するため，我々は素性列トライに保存する素性ベクトル中の素性をそのベクトルの獲得元のデータ中での出現頻度の降順でソートする．こうすることで，同時に保存される部分素性ベクトルは，元の素性ベクトルから頻度のより低い素性を除きより一般化した素性ベクトルとなる．語彙素性や，他の素性に依存する素性（品詞に対する品詞細分類など）は，素性ベクトルの後半に並べられるので，素性ベクトルを得ることができると期待される．同時に，素性列トライのサイズも頻出する共通接頭辞を共有することで最小化される（図 2 では $\{f_1, f_2\}$ など）．

図 3 に素性列トライを用いた分類アルゴリズムを示す．入力の素性ベクトル $\boldsymbol{x}$ を発火している素性のインデクスの集合 $i \in \mathbf{N}$ とする．まず，入力の素性ベクトル中の素性を素性列トライ T の獲得元のデータ中での出現頻度の降順でソートする．次に，素性列トライから得られた素性列に対する最長共通接頭素性列（部分素性ベクトル）$\boldsymbol{x_c}$ を得る（図 3 の (1)）．得られた部分重み $W_{\boldsymbol{x_c}}$ に，残りの素性に対する重みを加えて出力する（図 3(2)）．

前／後文節	主辞見出し，主辞品詞，主辞品詞細分類，主辞活用，語形見出し，語形品詞，語形品詞細分類，語形活用，括弧の有無，句読点の有無，文節の位置（文頭，文末）
文節間	距離（1,2-5,6 以上），括弧，助詞，句読点の有無

表 1: 使用した基本素性

4 評価実験

本節では，我々の提案する分類アルゴリズムを日本語係り受け解析タスクで評価する．

4.1 実験設定

日本語係り受け解析は，文中の文節の間の適切な係り受けを求めるタスクである．係り受け解析器は，特定の文節ペア（係り元と係り先の文節）に対して素性ベクトルを生成し，係るか係らないか（あるいは係る確率）を分類器により求める．我々の提案する分類アルゴリズムは，特定の係り受け解析アルゴリズムに依存しないため，本稿では Sassano による Shift-Reduce 型の係り受けアルゴリズムを用いる [11]．このアルゴリズムは，日本語係り受け解析で最も高速でかつ最高精度に匹敵する精度が得られることが報告されている [7]．

素性セットとしては，日本語係り受け解析で用いられる標準的な素性セットを用いた（表 1）．ここでは，文節間素性など，他の素性と独立に出現する素性が含まれていることに注意されたい．品詞タグ付けのような局所的な素性が大半を占めるタスクに比べると，素性ベクトルの自由度が高いため提案手法で高速化することが難しいタスクとなっている．

評価には，京都大学コーパス (Version 4.0) [10] を用いた．学習，開発，評価データは以下の通りである．

学習 24,283 文，234,685 文節（1 月 1 日，1 月 3-11 日の全記事と 1 月-8 月の社説）

開発 4833 文，47,571 文節（1 月 12-13 日の全記事と 9 月の社説）

評価 9284 文，89,874 文節（1 月 14-17 日の全記事と 9 月-12 月の社説）

以下の構文解析実験は，Intel® Xeon™ 3.20-GHz CPU のサーバ上で行った．SVM とトライの実装としては Kudo による TinySVM[5] とダブル配列 [1] Darts[6] を用いた．また，素性列トライとしては，3 つの異なるサイズ，fstrie_S, fstrie_M, fstrie_L を用意した．具体的には，JUMAN と KNP[7] で形態素解析・文節区切りを行った 1991 年の毎日新聞の記事 1000, 20,000, 400,000 文を，高速化対象の係り受け解析器でそれぞれ解析し，解析時に生成された全素性ベクトルを利用した．

[5] http://chasen.org/~taku/software/TinySVM/
[6] http://chasen.org/~taku/software/darts/
[7] http://www-nagao.kuee.kyoto-u.ac.jp/nl-resource/

SVM d	SVM σ	構文解析精度 (%)	近似カーネル展開 メモリ (MB)	近似カーネル展開 解析時間 (ms/文) 分類時間（全時間）	提案手法 (fstrie$_S$) +メモリ (MB)	提案手法 (fstrie$_S$) 解析時間 (ms/文) 分類時間（全時間）	提案手法 (fstrie$_M$) +メモリ (MB)	提案手法 (fstrie$_M$) 解析時間 (ms/文) 分類時間（全時間）	提案手法 (fstrie$_L$) +メモリ (MB)	提案手法 (fstrie$_L$) 解析時間 (ms/文) 分類時間（全時間）
1	0	88.28	0.8	**0.010 (0.034)**	+1.4	0.016 (0.040)	+21.0	0.020 (0.045)	+263.2	0.025 (0.050)
2	0	90.66	25.3	0.053 (0.079)	+1.4	**0.037 (0.063)**	+20.9	0.038 (0.064)	+261.3	0.038 (0.064)
3	0	90.94	408.7	0.370 (0.399)	+1.4	0.217 (0.246)	+20.9	0.166 (0.194)	+262.2	**0.122 (0.149)**
3	0.001	90.94	207.6	0.372 (0.399)	+1.4	0.203 (0.231)	+20.4	0.151 (0.178)	+253.6	**0.109 (0.135)**
3	0.003	90.85	12.6	0.326 (0.353)	+1.3	0.140 (0.166)	+18.4	0.094 (0.120)	+218.7	**0.065 (0.091)**
3	0.005	90.54	3.6	0.292 (0.317)	+1.2	0.107 (0.132)	+16.6	0.068 (0.094)	+188.2	**0.048 (0.073)**

表 3: 構文解析結果

SVM d	SVM σ	素性 $\lvert\mathcal{F}\rvert$ ($\times 10^3$)	素性 $\lvert\mathcal{F}^d\rvert$ ($\times 10^3$)	素性 $\lvert\boldsymbol{x}\rvert$	素性 $\lvert\boldsymbol{x}^d\rvert$	構文解析精度 (%)
1	0	39.8	39.8	27.3	27.3	88.28
2	0	38.1	1496.6	27.3	380.7	90.66
3	0	39.0	26,331.4	27.3	3287.4	90.94
3	0.001	32.0	13,265.6	27.2	2729.7	90.94
3	0.003	11.6	794.7	26.9	1883.6	90.85
3	0.005	5.3	221.8	26.6	1361.3	90.54

表 2: 学習された SVM の詳細

4.2 実験結果

表 2 に学習で得られた SVM の詳細を示す．学習時のソフトマージン C は，$C = 0.1, 0.01, 0.001, 0.0001$ を試み，開発データで最高精度を示した値を用いた ($d = 1$ で $C = 0.1$，$d = 2$ で $C = 0.01$，$d = 3$ で $C = 0.0001$)．得られた精度 90.94% ($d = 3$) は日本語係り受け解析の最高精度に近く [7]，学習で得られた SVM は我々のタスクを評価するのに十分現実的（言い換えると複雑）であると言える．表から分かるように，組み合わせの次数を上げると発火する（組み合わせ）素性の数の平均 $|\boldsymbol{x}^d|$ は急激に増加していることが分かる．また，SVM ($d = 3$) については，組み合わせ素性の重みの閾値 σ を変え，$|\mathcal{F}^d|$ を小さくすることを試みたが，精度を犠牲にせず発火する素性の数 $|\boldsymbol{x}^d|$ を小さくすることは困難であった．

従来の分類器と素性列トライに基づく分類器を用いた係り受け解析器の構文解析実験結果を表 3 に示す．$|\boldsymbol{x}^d|$ の増加から予想される通り，組み合わせ素性を用いた分類器は基本素性のみからなる分類器 ($d = 1$) に比べ，5 倍 ($d = 2$) から 30 倍 ($d = 3$) 遅く，処理時間の大半が分類に費やされることが分かった．三次の組み合わせを考慮したモデルは確かに 低次のモデルに比べて高精度だが，その効率の差は非常に大きい．

一方，我々の提案するアルゴリズムで，fstrie$_L$ を用いることで，三次の分類器 ($\sigma = 0$) を約 3.0 倍 (=0.370/0.122)，fstrie$_S$ を用いることで二次の分類器を約 1.4 (=0.53/0.37) 倍高速化できており，構文解析時間を大幅に削減できることが分かった．最高精度を達成した三次の係り受け解析器 ($\sigma = 0.001$) の分類器は約 3.4 倍 (=0.372/0.109) 高速化され，解析速度は 7407 文/秒である．

ここで，素性列トライのサイズは主に保存する素性ベクトルを生成する際に処理した文数に比例し，組み合わせの次数 d とほぼ関係ないことに注意されたい[8]．また，最小の fstrie$_S$ でも高速化は顕著である．また，高速化は高次の分類器に適用したときにより顕著であり，提案手法は高次の組み合わせ素性に基づく分類器にも有効であると言える．

[8]基本素性のみの分類器 ($d = 1$) の速度が劣化しているのは，素性のソートのコスト（図 3）と素性列トライの肥大化による．

もう一つ注目するべき点は，素性空間 $|\mathcal{F}^d|$ を疎にすることで高速化の効果がさらに上がることである．高速化は三次の分類器で 6.1 (=0.292/0.048, $\sigma = 0.005$) 倍に達している．提案手法により，異なる次数の組み合わせ素性を用いた分類器の間の速度差は縮まり，高次の組み合わせ素性を用いた高精度の分類器をより現実的に利用することができるようになったと言える．

5 おわりに

本稿では，自然言語処理タスクで頻繁に用いられる組み合わせ素性を用いた分類器の一般的高速化手法を提案した．部分素性ベクトルに対する重みをトライに保存・利用することで，高次の組み合わせ素性を用いた分類器を大幅に高速化できることを日本語係り受け解析の実験で確認した．提案手法は，素性空間が疎のとき特に効果が大きく，これは L1 正則化 [8] などと非常に相性が良いことを意味する．また SVM の部分カーネル展開 [5] と組み合わせることも可能である．

今後の課題としては，様々なタスクで提案手法の有効性を検証すること，また，簡潔データ構造 [4] によるトライ実装を利用して，重みを格納する素性ベクトルを増やすことなどが上げられる．

参考文献

[1] J. Aoe. An efficient digital search algorithm by using a double-array structure. *IEEE Transactions on Software Engineering*, Vol. 15, No. 9, pp. 1066–1077, Sep. 1989.

[2] A. Berger, S. D. Pietra, and V. D. Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, Vol. 22, No. 1, pp. 39–71, Mar. 1996.

[3] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *Proc. SPIRE*, pp. 39–48, 2000.

[4] O. Delpratt, N. Rahman, and R. Raman. Engineering the LOUDS succinct tree representation. In *Proc. WEA, LNCS 4007*, pp. 134–145, 2006.

[5] Y. Goldberg and M. Elhadad. splitSVM: Fast, space-efficient, non-heuristic, polynomial kernel computation for NLP applications. In *Proc. ACL, Short papers*, pp. 237–240, 2008.

[6] H. Isozaki and H. Kazawa. Efficient support vector classifiers for named entity recognition. In *Proc. COLING*, pp. 1–7, 2002.

[7] M. Iwatate, M. Asahara, and Y. Matsumoto. Japanese dependency parsing using a tournament model. In *Proc. COLING 2008*, pp. 361–368, 2008.

[8] J. Kazama and J. Tsujii. Evaluation and extension of maximum entropy models with inequality constraints. In *Proc. EMNLP*, pp. 137–144, 2003.

[9] T. Kudo and Y. Matsumoto. Fast methods for kernel-based text analysis. In *Proc. ACL*, pp. 24–31, 2003.

[10] S. Kurohashi and M. Nagao. Building a Japanese parsed corpus while improving the parsing system. In *Proc. LREC*, pp. 719–724, 1998a.

[11] M. Sassano. Linear-time dependency analysis for Japanese. In *Proc. COLING*, pp. 8–14, 2004.

[12] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[13] Y.-C. Wu, J.-C. Yang, and Y.-S. Lee. An approximate approach for training polynomial kernel svms in linear time. In *Proc. ACL Poster and Demo Sessions*, pp. 65–68, 2007.