組み合わせ素性に基づく分類器の効率的学習法

吉永 直樹 † 喜連川 優 †
† 東京大学 生産技術研究所

1 はじめに

本稿では，自然言語処理などの分野で有用な組み合わせ素性を用いた分類器を，効率的に学習する手法を提案する．具体的には，逐次学習の一つである Passive-Aggressive アルゴリズム [1] に Goldberg と Elhadad が分類のために提案した分割多項式カーネル [2] を組み込み，訓練例に頻出する素性組み合わせのみを陽に考慮することで，線形学習の時間効率とカーネル学習の空間効率とを両立した理想的な学習を実現する．提案手法は厳密解法であり，高速化のために組み合わせ素性の数やサポートベクタ数を制限する学習手法 [3, 4] に比べ，得られる分類器の分類精度の点で有利である．

提案手法を用いて，組み合わせ素性を考慮した日本語係り受け解析器の分類器 [5] の学習実験を行った．その結果，提案手法が多項式カーネルを利用した従来手法に対して，約 30-330 倍高速であることを確認した．

2 既存手法

2.1 Passive-Aggressive アルゴリズム

Passive-Aggressive (PA) アルゴリズム [1] は，各訓練例に対し，パラメタをそのつど更新する逐次学習法の一つである．なお，以後は二値素性のみを扱い，簡単のため二値分類に限って議論を進める．

アルゴリズム 1 は，PA にパラメタの更新速度を制御する変数 C を導入した PA-I アルゴリズムで，カーネル関数 k を用いた分類器を学習する手順を示している．例えばここでカーネルとして多項式カーネル $k_d(\boldsymbol{x}, \boldsymbol{s}) = (\boldsymbol{x} \cdot \boldsymbol{s} + 1)^d$ を用いれば，d 次以下の (重み付きの) 組み合わせ素性を陰に考慮することができ[*1]，素性空間を爆発させない空間効率の良い学習が可能である．

PA-I では，4, 6 行目から $l_t > 0$，すなわち，訓練例 $\boldsymbol{x}_t$ を充分なマージン $|\sum_{\boldsymbol{s}_i \in \mathcal{S}_{t-1}} \alpha_i k(\boldsymbol{s}_i, \boldsymbol{x}_t)| \geq 1$ で正しく分類できないとき，その訓練例をサポートベクタ $\mathcal{S}_{t-1}$ に追加する．カーネルを用いた学習器では，第 5 節でも確認するように，訓練例数が多いとサポートベクタの数も多くなり，学習速度が極端に低下してしまう．

2.2 分割多項式カーネル

Goldberg と Elhadad は，全素性集合 $\mathcal{F}$ を，サポートベクタ中で高頻度で発火する素性 $\mathcal{F}_c$ とそうでない素性 $\mathcal{F} \setminus \mathcal{F}_c$ に予め分類し，高頻度の素性の組み合わせのみを陽に考慮して分類 (マージンの計算) を高速化する分割多項式カーネルを提案している [2]．所与の例 $\boldsymbol{x}$ のマージンは以下のように計算される．

$$\begin{aligned}
&\sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i k_d(\boldsymbol{s}_i, \boldsymbol{x}) \\
&= \sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i k_d(\boldsymbol{s}_i, \boldsymbol{x}_c) + \sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i \{k_d(\boldsymbol{s}_i, \boldsymbol{x}) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_c)\} \\
&= \sum_{k=0}^{d} \sum_{f_i^k \in \boldsymbol{x}_c^k} w_i f_i^k + \sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i \{k_d(\boldsymbol{s}_i, \boldsymbol{x}) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_c)\} \quad (1)
\end{aligned}$$

ただし，$\boldsymbol{x}_c$ は $\boldsymbol{x}$ で $\mathcal{F}_c$ に含まれない低頻度の素性の次元を 0 にした素性ベクトル，f_i^k は $\boldsymbol{x}_c$ 中で発火する素性の k 次の組み合わせを表し，w_i は $\boldsymbol{\alpha}$ からカーネル展開 [6, 7] により計算される f_i^k の重みである．式 1 右辺の第二項は，$k'_d(\boldsymbol{s}_i, \boldsymbol{x}) = k_d(\boldsymbol{s}_i, \boldsymbol{x}) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_c)$ が $\boldsymbol{s}_i \cdot \boldsymbol{x} \neq \boldsymbol{s}_i \cdot \boldsymbol{x}_c$，すなわち $\boldsymbol{x}$ で発火する低頻度の素性が $\boldsymbol{s}_i$ で発火しないとき 0 となることを利用し，効率的に計算できる．

アルゴリズム 1 PASSIVE-AGGRESSIVE-I WITH KERNEL

INPUT: $C > 0, k : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$
OUTPUT: $\mathcal{S}_T, \boldsymbol{\alpha}$
1: Initialize: $\mathcal{S}_0 = \phi, \boldsymbol{\alpha} = \mathbf{0}$
2: **for all** $t = 1, 2, \ldots T$ **do**
3: Receive instance $\langle \boldsymbol{x}_t, y_t \rangle : \boldsymbol{x}_t \in \mathbb{R}^n, y_t \in \{-1, +1\}$
4: Suffer loss: $l_t = \max\{0, 1 - y_t \sum_{\boldsymbol{s}_i \in \mathcal{S}_{t-1}} \alpha_i k(\boldsymbol{s}_i, \boldsymbol{x}_t)\}$
5: Update:
6: 1. set: $\tau_t = \min\{C, \frac{l_t}{\|\boldsymbol{x}_t\|^2}\}$
7: 2. update: $\alpha_t = \alpha_t + \tau_t, \mathcal{S}_t = \mathcal{S}_{t-1} \cup \{\boldsymbol{x}_t\}$
8: **end for**
9: **return** $\mathcal{S}_T, \boldsymbol{\alpha}$

3 提案手法

本研究では，PA-I で分割多項式カーネルを利用することで組み合わせ素性に基づく分類器を効率的に学習する．具体的にはまず，アルゴリズム 1 の 4 行目のカーネル関数 k を式 1 に置き換える．なお学習の前に，訓練データ中における各素性の出現頻度を数え，組み合わせを陽に展開する上位 N 素性 $\mathcal{F}_c$ を決定しておく．

次に，7 行目で訓練例 $\boldsymbol{x}_t$ をサポートベクタに追加する際は，$\boldsymbol{x}_t$ で発火する素性のうち，組み合わせを展開する素性 $f \in \mathcal{F}_c$ について，d 次以下の組み合わせ素性を陽に列挙しその重みを更新する．なお，k 次の組み合わせ素性 f_i^k の重みには，τ_t に d と k に依存する重み係数 c_d^k をかけた値 $c_d^k \tau_t$ が加算される (c_d^k の算出方法については [7] を参照; 例えば脚中*1 から $c_2^2 = 2$).

学習アルゴリズムの変更は以上であるが，実際に高速な学習を可能とするには，式 1 の第一項の計算をいかに効率良く行うかが鍵となる．まず，空間効率を向上するため，既存のカーネル展開 [7] と同様に，トライを用いて組み合わせ素性の重みを管理する．次に，重みが 0 の組み合わせ素性の列挙を可能な限り回避する

Efficient Training of Classifiers with Conjunctive Features
YOSHINAGA Naoki† and KITSUREGAWA Masaru†
† Institute of Industrial Science, University of Tokyo

[*1] 例えば，訓練例 $\boldsymbol{x} = (x_1, x_2)$ とサポートベクタ $\boldsymbol{s} = (s_1, s_2)$ に対して，$k_2(\boldsymbol{s}, \boldsymbol{x})$ が返す値は $k_2(\boldsymbol{s}, \boldsymbol{x}) = (s_1 x_1 + s_2 x_2 + 1)^2 = 3 s_1 x_1 + 3 s_2 x_2 + 2 s_1 x_1 s_2 x_2 + 1$ ($\because \quad x_i, s_i \in \{0, 1\}$) となる．この関数は組み合わせ素性を含む高次空間への写像 $\phi_2(\boldsymbol{x}) = (1, \sqrt{3}x_1, \sqrt{3}x_2, \sqrt{2}x_1 x_2)$ を意味している．

ため，低頻度の素性から順に素性の組み合わせを列挙し，トライを探索する．さらに，重みを保存するトライのノード数を可能な限り少なくするため，高頻度の素性のバイト表現がより短くなるように，素性 $f \in \mathcal{F}_c$ を訓練データ中の頻度で整列し，出現順位を可変長バイト符号で圧縮したバイト列を素性の表現としてトライに格納する．

さらに，PA-I のパラメタ更新条件 $l_t > 0$ から，式 1 は $y_t|\sum_{\boldsymbol{s}_i \in \mathcal{S}_{t-1}} \alpha_i k(\boldsymbol{s}_i, \boldsymbol{x}_t)| >= 1$ となるとき正確な値を求める必要がないため，計算の枝刈りが可能である．具体的には，発火素性を一つ追加したときのマージンの変化量の上限を，分割多項式カーネルを利用して見積もり，低頻度の素性から順に累積して枝刈りに利用する．$\boldsymbol{x}'$ に素性 f を一つ追加した $\boldsymbol{x}$ のマージンの正負の変化量の上限 $\widehat{\Delta}(m^+), \widehat{\Delta}(m^-)$ は以下のようになる．

$$\widehat{\Delta}(m^+) = \sum_{\boldsymbol{s}_i \in \mathcal{S} \wedge f \in \boldsymbol{s}_i \wedge \alpha_i > 0} \alpha_i \hat{k}'_d + \sum_{\boldsymbol{s}_j \in \mathcal{S} \wedge f \in \boldsymbol{s}_j \wedge \alpha_j < 0} \alpha_j \check{k}'_d$$

$$\widehat{\Delta}(m^-) = \sum_{\boldsymbol{s}_i \in \mathcal{S} \wedge f \in \boldsymbol{s}_i \wedge \alpha_i > 0} \alpha_i \check{k}'_d + \sum_{\boldsymbol{s}_j \in \mathcal{S} \wedge f \in \boldsymbol{s}_j \wedge \alpha_j < 0} \alpha_j \hat{k}'_d$$

ただし，$\hat{k}'_d = (|\boldsymbol{x}'| + 2)^d - (|\boldsymbol{x}'| + 1)^d$，$\check{k}'_d = 2^d - 1$．この枝刈りは，式 1 の第 1 項，第 2 項の両方で行う．

4 評価実験

提案手法の有効性を検証するために，日本語係り受け解析のタスクの訓練データ [8] を用いて評価実験を行った．訓練例数は 296,776 (正例 150,064，負例 146,712) であり，有効素性数は 64,493, 平均発火素性数は 27.6 である．素性の詳細は [8] を参照されたい．組み合わせ素性の重みを管理するトライの実装としては，動的ダブル配列 [9] を用いた．以後の実験は，Intel® Xeon™ 3.2 GHz CPU と主記憶 32GB を備えたサーバ上で行った．

提案手法 PA-I SPLIT$_d$ を，多項式カーネル k_d を用いた PA-I (PA-I k_d; PKI [7] で高速化) とサポートベクタマシン (SVM k_d)[*2]，さらに d 次以下の全組み合わせ素性の重みを陽に推定する L1 正則化付対数線形モデル (ℓ_1-LLM$_d$)[*3] と比較した．精度比較のため，組み合わせ素性を用いない PA-I (PA-I LINEAR) と SVM (LIBLINEAR)[*4] でも学習を行った．それぞれ既定の設定で学習を行い，開発データでハイパーパラメタを調整した．PA-I の繰り返し回数は 20 回とし，PA-I SPLIT$_d$ では訓練データ中の出現頻度の上位 $N = 125 \times 2^n (0 \leq n \leq 9)$ 素性の組み合わせを陽に考慮し，それぞれ分類器を学習した．

表 1 に実験結果を示す．まず，多項式カーネル k_d に基づく学習手法は，線形学習に比べて，極端に時間がかかることが確認出来る．一方，組み合わせ素性を全展開すると，学習は高速化されるが考慮する素性数が爆発してしまう (3 次以下の全組み合わせを考慮する場合，PA-I k_3 で約 90MB，PA-I SPLIT$_3$ $(n = 9)$ で約 1.3GB のメモリを消費)．提案手法は展開する素性を制限しても，有意に高速に高精度の分類器を学習できている．

最後に，提案手法による展開素性数と学習速度の関係を図 1 に示す．図から，全体の約 1/250 程度 $(n = 1)$ の素性の組み合わせを展開すれば，全展開する場合と同程度の学習速度が得られることが分かる．PA-I SPLIT$_3$ $(n = 1)$ で陽に展開した組み合わせ素性数は 697,817 であり，全展開する場合 $(n = 9)$ の約 1/10 のメモリ消費 (130MB) で，より高速に学習が可能であった．

[*2] http://chasen.org/ taku/software/TinySVM/
[*3] http://www-tsujii.is.s.u-tokyo.ac.jp/~tsuruoka/maxent/
[*4] http://www.csie.ntu.edu.tw/~cjlin/liblinear/

学習手法	解析精度	訓練時間	有効素性数
PA-I LINEAR	88.39%	1s	44,753
LIBLINEAR	88.18%	6s	53,471
ℓ_1-LLM$_2$	90.61%	8115s	33,109
SVM k_2	90.76%	29807s	(1,478,096)
PA-I k_2	90.64%	9029s	(1,821,244)
PA-I SPLIT$_2$ $(n = 5)$	90.65%	**27s**	375,537
ℓ_1-LLM$_3$	90.71%	102295s	129,548
SVM k_3	90.93%	25869s	(26,194,241)
PA-I k_3	90.87%	7885s	(30,915,981)
PA-I SPLIT$_3$ $(n = 2)$	90.86%	**258s**	1,751,861

表 1: 分類器の訓練時間 (括弧内は展開時の素性数)

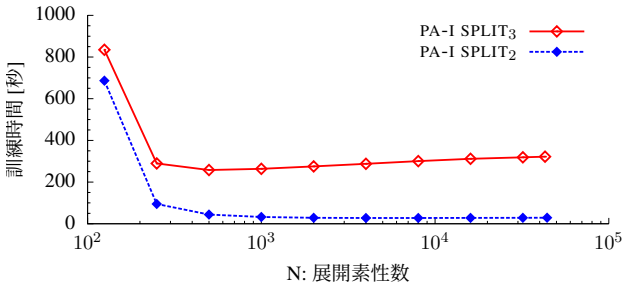

図 1: 展開素性数と訓練時間のトレードオフ

5 まとめ

本稿では，組み合わせ素性を用いた分類器の学習を，逐次学習 [1] と分割多項式カーネル [2] を用いて効率的に行う手法を提案した．提案手法を用いて日本語係り受け解析の分類器を学習したところ，多項式カーネルを用いる場合の約 30-330 倍の速度で学習ができるだけでなく，素性の組み合わせを陽に全展開する場合と比べて約 1/10 のメモリ消費で学習が可能であった．

参考文献

[1] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *JMLR*, Vol. 7, pp. 551–585, March 2006.

[2] 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.

[3] D. Okanohara and J. Tsujii. Learning combination features with L_1 regularization. In *Proc. HLT-NAACL, Short Papers*, pp. 97–100, 2009.

[4] O. Dekel, S. Shalev-Shwartz, and Y. Singer. The forgetron: A kernel-based perceptron on a budget. *SIAM Journal on Computing*, Vol. 37, No. 5, pp. 1342–1372, January 2008.

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

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

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

[8] N. Yoshinaga and M. Kitsuregawa. Polynomial to linear: Efficient classification with conjunctive features. In *Proc. EMNLP*, pp. 1542–1551, 2009.

[9] 矢田晋, 田村雅浩, 森田和宏, 泓田正雄, 青江順一. ダブル配列による動的辞書の構成と評価. 第 71 回情報処理学会全国大会講演論文集, pp. 1263–1264, 2009.