木構造カーネルの高速化とノード関係分類への応用

風間 淳一 (kazama@jaist.ac.jp)　鳥澤 健太郎 (torisawa@jaist.ac.jp)
北陸先端科学技術大学院大学 情報科学研究科

1　概要

木構造カーネルは, 全ての部分構造を考慮して木構造間の類似度を計算する方法であり, 近年様々な自然言語処理への応用が期待されている. しかし, 従来提案されている動的計画法 (DP) を用いた計算方法ではコストが高く, 応用の際の障壁となっている. 本研究では, マイニングの技術を使って木構造をあらかじめベクトルに変換することで, 学習時の木構造カーネルの計算を大幅に高速化する方法を提案する. 実例として, 意味役割付与 (semantic role labeling; SRL) などのノード間の関係を分類するタスクのための新しい木構造カーネル (マーク付きラベル順序木に対する木構造カーネル) を提案し, このカーネルを用いた SVM による意味役割分類器の学習が, 提案手法によって数十倍高速になることを示す. また, この高速化を利用して提案カーネルの性能を比較したので報告する.

2　背景

木構造カーネル (Collins and Duffy 2001; Kashima and Koyanagi 2002) は,

$$K(T_1, T_2) = \sum_{S_i} W(S_i) \cdot \#_{S_i}(T_1) \cdot \#_{S_i}(T_2), \qquad (1)$$

のように定義され, 部分構造 (部分木)S_i の木 T_j の中での出現回数 $\#_{S_i}(T_j)$ を次元としたベクトルの (重み付き) 内積で木の類似度を測るものである. 名前からも分かる通り, カーネル関数の要件を満たすので SVM などのカーネル学習法と組み合わせて用いることができる.

木構造カーネルの利点は, 従来の素性ベクトルを用いる方法では考慮されてこなかった大域的な部分構造も含め, 全ての部分構造を考慮しながら, その際問題となるスパース性の問題を SVM などの汎化能力の高いカーネル学習法によって回避できる点にある. 自然言語処理では木構造が重要な役割を担うことが多いため, 木構造カーネルは構文解析 (Collins and Duffy 2001), 情報抽出 (Kashima and Koyanagi 2002), 意味役割付与 (Moschitti 2004) などの様々な自然言語処理に応用されてはじめている.

木構造カーネルの計算は, 式 (1) 通りに計算すると指数関数的数の可能な部分構造に関して和をとらなければならず, 現実的ではない. 木構造カーネルの効率的な計算法としては, 従来, DP を用いた方法が提案されており (Collins and Duffy 2001; Kashima and Koyanagi 2002), $|T_j|$ を T_j のノード数とすると $O(|T_1||T_2|)$ の時間で計算することができるが, 現実的なコストは依然として高いと言わざるを得ない.[*1] これは, 学習時間・処理時間の増大に直結するため, 応用の際の障壁となってきた.

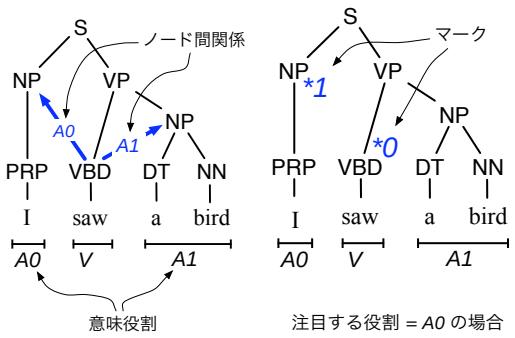

図 1　左: ノード間関係分類 (SRL), 右: マーク付きラベル順序木による表現

本研究では, 特に, 学習時のカーネル計算を高速化して, 学習を高速化することを目指す. 一般に, 機械学習では学習器のハイパーパラメータを様々に変えて学習を繰り返し行ってその最適な値を求めた上でないと手法の真の性能を評価できないため, 学習の高速化は非常に重要である.

3　マーク付きラベル順序木に対するカーネル

ここでは, 本研究で実例として取り上げる SRL タスクとそのために本研究で提案する新しい木構造カーネルを説明しておく. SRL タスクは, 動詞の各意味役割を担っている単語列 (句) を認識するタスクである. SRL タスクに対してはこれまで様々な手法が提案されているが, 近年, 構文木情報の利用が注目されている (Carreras and Màrquez 2005).

本研究では, 図 1 の左で示すように, 構文木中の動詞に対応するノードと他のノードとの関係を適切な意味役割に分類する「ノード関係分類」の一つとして SRL を捉える. このような定式化において木構造カーネルを用いる場合, なんらかの方法でノード間の関係を表現する必要がある. そのため, 例えば Moschitti (2004) は, 動詞ノードと分類すべきノードにしたがって定まる木構造の一部を取り出した上で既存の木構造カーネルを適用し, SRL のための分類を行っている. 本研究では, このような取り出しは ad-hoc である, また, 分類ごとに木構造が変化するため高速化に向かないなどの理由から, ノード関係分類一般に適用できる方法として, マーク付きラベル順序木とそれに対する木構造カーネルを提案する.

マーク付きラベル順序木は, 各ノードがラベル以外にマークを持つことができるラベル順序木である. このマーク付き木を用いると, ノード間関係は図 1 の右のように表現することができる.[*2]

このようなマーク付き木に対してカーネルを定義し, SVM

[*1] 例えば, 我々が実験で用いたデータでは木の平均ノード数は 80 程度もある. また, 動的計画法のコストの定数部分も C を子ノードの数とすると $O(C)$(Collins and Duffy 2001) から $O(C^2)$(Kashima and Koyanagi 2002) と大きい.

[*2] ここでは 2 ノード間の関係としているが, 本稿で述べるアルゴリズムなどは一般の k ノード間関係に応用できる.

表 1　カーネルの値の傾向. 3,136 個の木からなる学習データにおいて, $W(S_i) = 1$ として, $K(T_i, T_j) = 10^4$ を閾値として分析したもの. (A) は組 (T_i, T_i), (B) は (T_i, T_i) 以外の同じ文から生成された木の組, (C) は異なる文から生成された木の組.

		K_{lo}^r		K_c^r	
		組の数	カーネルの平均	組の数	カーネルの平均
$\geq 10^4$	(A)	3,121	1.17×10^{52}	3,052	2.49×10^{32}
	(B)	7,548	7.24×10^{48}	876	1.26×10^{32}
	(C)	6,510	6.80×10^{9}	28	1.82×10^{4}
$< 10^4$	(A)	15	4.19×10^{3}	84	3.06×10^{3}
	(B)	4,864	2.90×10^{2}	11,536	1.27×10^{2}
	(C)	9,812,438	1.82×10^{1}	9,818,920	1.84×10^{-1}

などで分類することで SRL を行うことができる. 本研究で提案するカーネルは, 通常の木構造カーネル (Collins and Duffy 2001; Kashima and Koyanagi 2002) に, (1) ノードのラベルの代わりにラベルとマークの組をラベルとみなす (マークを持たないときはラベルをそのまま使う), (2) 少なくとも 1 つのノードがマークを持つ部分木に関してのみ和をとる, という変更を行うことで得られる. (2) は, マークを全く持たない部分木はマークの付け方の良さを判定する際に有用ではないという考えからである. (2) の条件のため多少複雑になるが, このカーネルに対しても $O(|T_1||T_2|)$ の DP を構成することができる (Kazama and Torisawa 2005). しかし, 前述の計算コストの問題は残る.

本研究では, Kashima and Koyanagi (2002) をマーク付き木に拡張したカーネルを K_{lo}^r, Collins and Duffy (2001) を拡張したものを K_c^r と呼ぶことにする. Collins and Duffy (2001) と Kashima and Koyanagi (2002) では部分木の定義が異なり, 前者ではあるノードの子ノードを含むならば全ての子ノードを含まなければならないのに対して, 後者では子ノードが抜けてもよいため, より多くの部分木が考慮される.

4　高速化の基本的アイデア

本研究の高速化では, 学習データの観察から得られた「学習データ中のごく少数の木の組のみが極端に多い（指数関数的数の）共通の部分木を持ち, それ以外の組はごく少数の共通部分木しか持たない」という性質を利用する. 本研究では前者の組を「危険ペア (malicious pairs)」後者を「非危険ペア (non-malicious pairs)」と呼ぶことにする. 表 1 は, この性質を木構造カーネルの値を用いて定量的に分析したものである.

非危険ペアに共通して出現する部分木のみを次元とするベクトルに木を展開しておくことで, 非危険ペアのカーネルの計算はそれらのベクトルの内積で置き換えることができる. 上記の観察から, 各ベクトルのサイズ (非ゼロの要素の数) は小さい数でおさえられると期待され, その場合, DP で計算するよりも大幅に高速になる. ごく少数の危険ペアに関しては, ベクトルへの展開は現実的ではないので従来通り動的計画法を用いるが, 危険ペアの割合はごく小さいので全体としては大幅な高速化が期待できる. もちろん, ベクトルへ展開するコストはかかるが, 同じ学習データに対して何度も学習を行う場合には無視できると考えられる.

実際には, これだけではベクトルの大きさが学習データサイズにしたがって大きくなるため, 高速化がスケールしない. ただし, (1) SVM などの学習では, ある学習サンプルに対して他の全ての学習サンプルとの間のカーネルの値が一度に必要になる, (2) それが, (カーネルが内積の場合には), 素性が出現するサンプルを列挙した転置リストを用いて効率的に計算できる (Kudo and Matsumoto 2003), ということを利用すると, 一組あたりのカーネル計算のコストを, r_m を危険ペアの割合, Q を組で共通する部分木の数として, $O(c_1 Q + r_m c_2 |T_i||T_j|)$ に抑えることができる. Q や r_m は学習データサイズに依存しない小さい値のため, スケーラブルな高速化が可能になる.

アルゴリズム **5.1:** FREQTM(D, R)

procedure SEARCH$(F_i, \underline{precheck})$
if $|F_i| \geq D$ **then** REGISTERMAL(F_i) **return** (**false**) –**(P)**
$(\mathcal{C}, \underline{suc}) \leftarrow$ GENERATECANDIDATE(F_i)
if not suc **then** REGISTERMAL(F_i) **return** (**false**) –**(S)**
for each $F_k \in \mathcal{C}$ **do**
  if $malicious(F_k)$ **then** goto next F_k ——**(P2)**
  $\underline{suc} \leftarrow$SEARCH$(F_k, precheck)$
  if not suc **and** $|\underline{sup(F_i)}| = |sup(F_k)|$ **then**
    return (**false**) ——**(P1)**
if not $precheck$ **and** $marked(F_i)$ **then**
  REGISTERSUBTREE(F_i) ——**(F)**
return (**true**)

main
$\mathcal{M} \leftarrow \phi$ (a set of malicious pairs)
$\mathcal{F}^1 \leftarrow \{F_i | |F_i| = 1$ and $|sup(F_i)| \geq \underline{2}\}$
for each $F_i \in \mathcal{F}^1$ **do** SEARCH$(F_i,$ **true**) ——**(PC)**
for each $F_i \in \mathcal{F}^1$ **do** SEARCH$(F_i,$ **false**)
$\mathcal{M} \leftarrow \mathcal{M} \cup \{(i,i) | 1 \leq i \leq L\}$
return $(\underline{\mathcal{M}}, \{V(T_i)\}, \underline{\{W(f_i)\}})$

以上の高速化の実現には, 学習データ中の危険ペアを検出すること, また, 非危険ペアのみに共通して表れる部分木 (素性部分木) を列挙することが必要になる. 危険ペアの最大の要因は同じ文から生成された組であるが, (特に K_{lo}^r の場合), 異なる文から生成された組も危険ペアになる可能性があるため, 危険ペアの検出は自明ではない. また, 素性部分木の列挙自体も自明ではない. 本研究では, これらを実現するために, 次節で述べる FREQTM アルゴリズムを開発した.

5　FREQTM アルゴリズム

FREQTM は FREQT と呼ばれる木構造マイニングアルゴリズム (Asai et al. 2002) を基にしている. FREQT アルゴリズムは, 最右拡張と呼ばれる操作で候補部分木を生成しながら, データ中にある回数 ($minsup$ 回) 以上現れている部分木を効率的に探索・列挙するアルゴリズムであり, 近年, 部分木を素性とするような自然言語処理で応用されはじめている (Kudo and Matsumoto 2004).

危険ペアは, 単純には DP でカーネルを計算すれば検出できるが, L を学習データの数として, L^2 組について計算する必要があるためコストが大きい. そこで, 本研究では, 危険ペアの定義を探索アルゴリズムに適した定義に変える. 具体的には, 探索中に候補木 F_i の大きさ (ノード数) が大きくなりすぎた時 (閾値 D 以上になった時) に, F_i が現れている学習データ中の木 ($sup(F_i)$ として保持されている) の全てのペアを危険ペアと見なすことにする. 大きな部分木を共通に持つことは, その部分木の, 最悪の場合指数関数的数の全ての部分木を共通に持つことを意味するからである. また, 同じサンプルの

組 (T_i, T_i) は必ず危険ペアであると定義して, $minsup = 2$ の FREQT を修正することで FREQTM が得られる.

アルゴリズム 5.1 に FREQTM の概略を示す. FREQT から変更された部分は下線で示した. 修正点は, (1) 検出された危険ペアを $\mathcal{M}$ に保持しておく, (2) 上で述べた危険ペアの検出を行って $\mathcal{M}$ に登録する, (3) $sup(F_i)$ 中の全ての木の組が, 直接的あるいは間接的に危険ペアであると分かる場合に F_i より先の探索を枝刈りする[*3], の 3 点である. (3) の枝刈りは, (P), (P1), (P2) の位置で行われる. 探索は深さ優先なので, 探索の早い段階で危険ペアを検出することができ, 効率的な枝刈りが実現できる. また, 上の枝刈りだけでは, 一つの木から同じ候補木が大量に生成されるような場合に探索の速度が大幅に低下してしまうので, これを防ぐため, R をパラメータとして, (F_i の出現回数/F_i が現れた木の数) $> R$ のときに (S) の位置で枝刈りを行う.

素性部分木の列挙は, 枝刈りが行われなかった場合にのみ手続きの最後の位置 (F) で行われる. これにより, 非危険ペアに表れる部分木のみを列挙できる. 正確には, 後から危険ペアであることが判明する組に出現する部分木を列挙してしまう可能性があるので, 以上の探索を 1 回目は危険ペアを検出するためだけに実行し (PC), 2 回目の探索で実際に列挙する. (F) で列挙する際に部分木 F_i に ID を与え, それを $sup(F_i)$ に従って出現する木に分配することで木をベクトルに展開していくことができる.

最後に, 提案手法では, 二つの部分木 F_k と F_l は出現パターン (出現する木と回数のパターン) が同じ時には, $W(f_i) = W(F_k) + W(F_l)$ の重みを持つ 1 つの素性 f_i としてまとめられることを利用して, 素性の圧縮を行う.[*4] これにより, さらなる高速化が実現される.

6 他手法との比較

木構造カーネルの計算の高速化手法として, suffix tree を利用した高速化 (コストは $O(|T_1| + |T_2|)$) が提案されている (Vishwanathan and Smola 2004). しかし, この手法は文字列で木を表現したときにその連続した部分になるような部分木を扱う場合にしか適用できないため, Collins and Duffy (2001) と Kashima and Koyanagi (2002) のどちらのタイプの木構造カーネルも高速化することができない.

ラベルからノードへの対応を各木が持つようにすることで DP を高速化することもできる. 予備実験の結果, この実装は通常の DP より約 2 倍高速であることがわかったので, 実験ではこの高速化された DP を比較用に用いることにする.

7 実験

7.1 実験設定

CoNLL05 SRL shared task(Carreras and Màrquez 2005) のために提供されているデータ中の学習データから, この実験用の学習セット・開発セット・評価セット (それぞれ 23,899, 7,969, 7,967 文) を生成した. 構文木としては, 上記データに付属する Charniak パーザによる構文木を用いた. また, 単語は全て小文字化し, 品詞情報を利用するため単語ノードの親ノードとして品詞をラベルとして持つノード挿入した (図 1 参照). 意味役割や動詞で, その範囲が構文木中のどのノードともマッチしないもの (およそ 3.5%) は無視した. 範囲が一致するノードが複数ある場合には一番上のノードに意味役割がアノテーションされているとして扱った.

カーネル学習器としては SVM を用い, TinySVM[*5]に K^r_{lo} と K^r_c を実装して用いた. FREQTM は, 工藤による FREQT の実装[*6]を基に実装した.[*7]

木構造カーネルの部分木の重みは, 既存研究 (Collins and Duffy 2001; Kashima and Koyanagi 2002) と同様, $W(S_i) = \lambda^{|S_i|}$ $0 \leq \lambda \leq 1$ (K^r_c の場合には, $W(S_i) = \lambda^{(\#\text{ of CFG productions in } S_i)}$) を用いた. この重み付けは, カーネルの値が極端に大きくなるのを防ぐために用いられている. λ はハイパーパラメータの一つとして最適値の探索の対象とする. また, カーネルは $K(T_i, T_j)/\sqrt{K(T_i, T_i) \times K(T_j, T_j)}$ と正規化して用いた ($K(T_i, T_i)$ はあらかじめ計算しておけるので学習時間にはほとんど影響しない).

この実験では, 意味役割を持つノードを認識してからそのノードを意味役割に分類するという二段階の SRL を仮定して, 後者の分類にのみ注目する. 分類には one-vs-rest 法を用い, その際の各々の 2 値分類器の性能に注目する. 意味役割を図 1 の右のように二つのマークで表現した木をその意味役割の正例とし, 同じ動詞の他の意味役割を表現した木を負例とする. 実験では, このように生成されたデータを用いて学習と各々の分類器の性能評価を行う.[*8]

7.2 学習の高速化

意味役割 A2 のデータを用いて, FREQTM による変換の諸元, DP を用いる場合と提案手法によるベクトルを用いる場合[*9]の SVM の学習時間を調べた. FREQTM のパラメータは, $D = 20$, $R = 20$ を用いた. SVM の学習は, 収束許容度 0.001(TinySVM の-e オプション), ソフトマージンコスト $C = 10^3$, 最大反復数 10^5, キャッシュサイズ 512MB, $\lambda = 0.2$ という設定で行った. 時間の計測は, 2.4GHz Opteron を搭載したマシン上で行った.

表 2 が結果である. まず, 提案手法により SVM の学習が大幅に高速化されていることがわかる. 期待した通り, 学習データが大きくなっても高速化の倍率は減少せず, スケーラブルな高速化が実現されている. また, SVM の学習コストはおよそ $L^{2.0}$ に比例すること, FREQTM のコストは, K^r_{lo} の場合およそ $L^{1.7}$, K^r_c の場合およそ $L^{1.3}$ に比例することが表から計算でき, FREQTM による変換は SVM の学習に比べてスケーラブルであることがわかる. また, 提案手法は同じデータで繰り返し学習する場合に最も効果的であるが, FREQTM による

[*3] F_i より先で最右拡張により生成される候補木 F_k は F_i より大きく $sup(F_k) \subseteq sup(F_i)$ であり, $sup(F_k)$ 中の全ての組は危険ペアであるから探索を進めても素性部分木はあり得ない.

[*4] それまでに現れた出現パターンを記憶しておき, (F) の位置でまとめられるかのチェックを行う.

[*5] http://chasen.org/~taku/software/TinySVM

[*6] http://chasen.org/~taku/software/freqt

[*7] http://www.jaist.ac.jp/~kazama/freqtm.html より入手可能.

[*8] 多値分類としての評価ではなく, 意味役割 S の分類器の評価を S を含む文から生成された評価データで評価する点で, 実際の SRL よりもあまい評価になっている.

[*9] 数値計算による多少の違いはあるが, DP を用いた場合とほぼ同じ SVM モデルが得られることが確認されている.

表 2　FREQTM による変換および SVM 学習の高速化

	K^r_{lo}				K^r_c			
サイズ (正例数)	2,000	4,000	8,000	12,000	2,000	4,000	8,000	12,000
サンプル数	6,246	12,521	25,034	34,632	6,246	12,521	25,034	34,632
素性部分木数 ($\times 10^4$)	2,427.3	6,542.9	16,750.1	26,146.5	9.009	21.867	52.179	78.440
素性部分木数 (圧縮後)　($\times 10^4$)	67.3	207.2	585.9	977.0	1.437	3.426	8.128	12.001
ベクトルサイズの平均 (圧縮後)	866.5	1,517.3	2,460.5	3,278.3	14.0	17.9	23.1	25.9
危険ペアの割合 $r_m(\%)$	0.711	0.598	0.575	1.24	0.0891	0.0541	0.0370	0.0361
変換時間 (秒)	629.2	1,921.1	6,519.8	14,824.9	8.7	20.4	46.5	70.4
SVM 学習時間 (DP+lookup) (秒)	1,716.2	4,526.4	79,800.7	92,542.2	1,263.5	5,893.3	53,055.5	47,089.2
SVM 学習時間 (提案手法) (秒)	68.6	186.4	1,721.7	2,531.8	25.7	119.5	982.8	699.1
高速化 (倍)	**25.0**	**24.3**	**46.4**	**36.6**	**49.1**	**49.3**	**53.98**	**67.35**

表 3　K^r_{lo} と K^r_c のいくつかの意味役割に対する性能の比較

		学習サイズ (正例数) = 8,000		
		最良設定 λ, C	F_1 (dev)	F_1 (test)
A1	K^r_{lo}	$0.25, 8.647$	**89.80**	**89.81**
	K^r_c	$0.2, 17.63$	87.93	87.96
A2	K^r_{lo}	$0.20, 57.82$	**87.94**	**87.26**
	K^r_c	$0.15, 10^3$	87.37	86.23
AM-ADV	K^r_{lo}	$0.15, 45.60$	**86.91**	**87.01**
	K^r_c	$0.20, 2.371$	84.34	84.08
AM-LOC	K^r_{lo}	$0.15, 20.57$	**91.11**	**92.92**
	K^r_c	$0.15, 13.95$	89.59	91.32

変換にかかる時間をみると, 学習を一回しか行わないとしても高速化として意味があることがわかる. また, 素性の圧縮の効果が非常に大きいこともわかる. この圧縮がなければ, 必要とされるメモリ・ファイル容量ともに許容できないほど大きくなっていたと考えられる. また, この圧縮は高速化にも大きく貢献している. 例えば, K_{lo} の場合, 圧縮なしでは高速化がおよそ 1/5 になることが観察された.

7.3　カーネルの性能比較

次に, いくつかの意味役割に対する K^r_{lo} と K^r_c の性能を, 高速化された SVM 学習を利用して詳しく調べた. λ と C を変化させ, 開発セットに対する F_1 値を最大にする設定を探索した. このとき, 開発セットに対する分類を高速に行うため, 開発セットは学習セットと一緒に FREQTM により変換しておく. λ を 0.1, 0.15, 0.2, 0.25, 0.3 と変化させ, 各々について $[0.5 \ldots 10^3]$ の範囲の 40 個の C の値について実験を行い (合計 200 回の実験), 最良の設定を求めた. 最後に, 最良設定での評価セットに対する F_1 値を求めた.

表 3 が結果である. どの意味役割に対しても K^r_{lo} の精度が K^r_c の精度を上回った. これは, 前の実験からもわかるが, K^r_c では部分木の制約が強すぎて十分な数の有用な素性部分木が得られていないためであると推測される.

8　まとめと今後の課題

本研究では, FREQTM で木をベクトルに展開することで木構造カーネルを用いた SVM の学習を数十倍高速化できることを示した. しかし, 展開されたベクトルは学習データ中の木の間のカーネル計算にしか使えないため, 実際の分類を高速化することはできない.[*10] これについては, タスクの性質を利用して DP を高速化するなどして今後対処する必要があるが, クラスタリングなど分類を必要としない枠組みにおいては, 提案手法はそれだけで多いに役立つと考えられる.

本研究では, SRL を実例としてマーク付き木に対する木構造カーネルを用いた学習の高速化を示したが, 提案手法は通常の木構造カーネルに対しても適用できる. しかし, 提案手法は, 学習データが危険ペアと非危険ペアにきれいに分けられるという性質に大きく依存しているため, この性質が SRL 以外のタスクでどの程度一般的であるかを調べる必要がある.

FREQTM に関しては, 今回, 実験的に時間コストを示したが, 実験からも示唆される通り, コストはカーネルの種類によって大きく変わる. その点も含めたコストの理論的な解析が今後の課題である. また, より高速なマイニングアルゴリズムの利用, あるいは, マーク情報を利用した枝刈りなどによって変換コストをさらに小さくすることも考えられる.

実験では, K^r_{lo} が K^r_c より高精度であることを示したが, これは SRL の一部である分類のしかもあまい評価によるものであり, 今回提案したマーク付き木とそれに対する木構造カーネルを用いる方法で SRL 全体を行ったときに, どの程度の精度が得られるかは今後検証していく必要がある.

参考文献

Asai, T., K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa (2002). Efficient substructure discovery from large semi-structured data. In *SIAM SDM'02*.

Carreras, X. and L. Màrquez (2005). Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *CoNLL 2005*.

Collins, M. and N. Duffy (2001). Convolution kernels for natural language. In *NIPS 2001*.

Kashima, H. and T. Koyanagi (2002). Kernels for semi-structured data. In *ICML 2002*, pp. 291–298.

Kazama, Jun'ichi and Kentaro Torisawa (2005). Speeding up training with tree kernels for node relation labeling. In *EMNLP 2005*.

Kudo, T. and Y. Matsumoto (2003). Fast methods for kernel-based text analysis. In *ACL 2003*.

Kudo, T. and Y. Matsumoto (2004). A boosting algorithm for classification of semi-structured text. In *EMNLP 2004*.

Moschitti, A. (2004). A study on convolution kernels for shallow semantic parsing. In *ACL 2004*.

Vishwanathan, S. V. N. and A. J. Smola (2004). Fast kernels for string and tree matching. *Kernels and Bioinformatics*.

[*10] ただし, 開発セットのように変化しないことが分かっている場合には, 学習データと同時に変換すれば高速化できる.