3B1-R-2-3

木構造の垂直方向の構造に基づいた線形時間木カーネル

A Linear Time Subpath Kernel for Unordered Trees

木村 大翼　　鹿島 久嗣
Daisuke Kimura　　Hisashi Kashima

東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

The kernel method is a promising approach to analyzing structured data such as sequences, trees, and graphs; however, unordered trees have not been investigated extensively. Kimura et al. proposed a kernel function for unordered trees on the basis of their subpaths, which are vertical substructures of trees responsible for hierarchical information in them. They propose a linear-time computation algorithm for the kernel by exploiting an enhanced suffix array for trees and it shows practically good performance in terms of accuracy and speed. In this paper, we present an efficient prediction algorithm whose running time depends only on the size of the input tree by extending that of Teo and Vishwanathan. The time complexity of the algorithm is $O(|T|^2)$ in the worst case where T is a newly coming tree in the prediction phase. Experimental results show that the proposed algorithms are quite efficient in practice.

1. はじめに

現実世界において様々なデータが配列, 木構造, あるいはグラフ構造といった非ベクトル型のデータとして表現される. 例えば配列データとしては, バイオインフォマティクス [Durbin 98] の分野における DNA や RNA や自然言語処理 [Manning 99] におけるテキストがあり, 木構造としては同分野の構文解析木や Web データの HTML や XML [Abiteboul 00] といった半構造データがある. さらにグラフ構造 [Kramer 01] として表現されるものとしては化合物がある. これら配列, 木構造, グラフ構造といったいわゆる構造データを対象とした解析手法が近年盛んに研究されている.

これら構造データを対象とした解析手法における有望なアプローチの一つにカーネル法 [Schölkopf 02] がある. カーネル法は, データにアクセスする際に単体ではなく, 必ず 2 つのデータの内積の形でアクセスする. これはすなわち, 特徴空間の次元がどんなに高くとも, なんらかの方法によって特徴ベクトルの内積だけを高速に計算することができるならば, 学習器の訓練にも予測にも特徴空間の次元が明示的に現れることがないことを意味する. この内積を与える関数は「カーネル関数」と呼ばれ, カーネル法はカーネル関数を用いることで, 高次元の特徴空間においても効率的に働くことができる. 構造データに対するカーネル関数の設計の代表的な枠組みとして畳み込みカーネル [Haussler 99] がある. 畳み込みカーネルにおいては構造データは陰に部分構造に分解され, カーネル関数はこれら部分構造間のカーネル関数の和として定義される. 構造データを対象としたカーネル関数の設計においては, 部分構造の表現力とカーネル関数の効率的な計算アルゴリズムのバランスを上手くとることが必要不可欠であり, 様々なカーネル関数が文字列 [Lodhi 02, Leslie 02], 木構造 [Collins 01, Kashima 02, Aiolli 09], グラフ構造 [Kashima 03, Gärtner 03] などに対して提案されてきた.

1.1 木カーネル

本稿でわれわれは特に木カーネルに着目する. 最初の木カーネルは構文解析木を対象として提案され [Collins 01], のちに一般の順序木に対して拡張された [Kashima 02]. 以降近年に至るまで順序木を対象とした様々な木カーネルが提案されてきたが [Moschitti 06, Kuboyama 06, Aiolli 09, Sun 11], これら既存の木カーネルにおいて無順序木 (図 1(a)) を扱うことができるものは少ない. このような状況において Vishwanathan と Smola は無順序木を対象とした効率的な木カーネルを提案した [Vishwanathan 03]. 彼らの木カーネルは完全部分木を特徴として用いており (図 1(b)), 入力の木構造を文字列に変換し, 接尾辞木を用いることで木のサイズについて線形時間で計算することができる. さらに予測時における計算量は新たな入力の木のサイズのみに依存し, サポートベクトルの数に依存しない高速な計算が可能である. またこの木カーネルについてはのちに文字列に対する拡張接尾辞配列を用いた省メモリな計算アルゴリズムが提案されている [Teo 06].

一方 Kimura らは部分パス (図 1(c)) と呼ばれる木構造の垂直方向の特徴を用いた無順序木カーネルを提案している [Kimura 11]. 彼らの木カーネルは階層構造を表現する木の垂直方向の部分構造を捉えることができる.

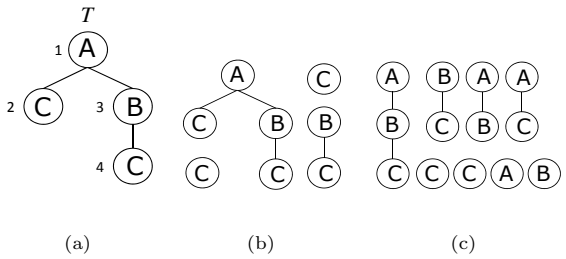


図 1: (a) 無順序木. (b) Vishwanathan らの完全部分木. (c) Kimura らの部分パス.

連絡先: Daisuke_Kimura@mist.i.u-tokyo.ac.jp

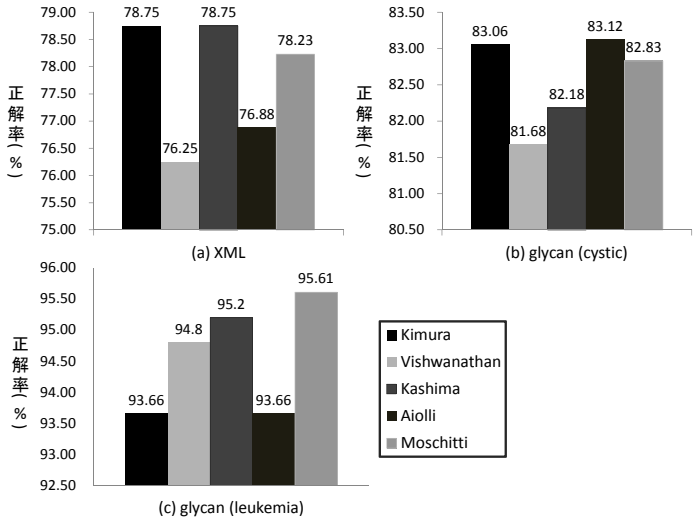


図 2: 3 つのデータセットに対する 5 つの木カーネルの正解率比較. 部分パス木カーネル [Kimura 11] は他の木カーネル [Vishwanathan 03, Kashima 02, Moschitti 06, Aiolli 09] と同等の正解率を示している.

図 2 は 3 つのデータセット (XML [Zaki 06] と糖鎖 [Hashimoto 03, Doubet 92]) に対して Kimura らの部分パス木カーネルと既存の他のカーネルの予測精度を比較した実験結果である*1. ここで Kimura らによる木カーネルと Vishwanathan らによる木カーネル以外の 3 つの木カーネル [Kashima 02, Moschitti 06, Aiolli 09] は順序木を対象としているので, 木構造に順序の情報が含まれるデータセットを用いた. 実験結果から Kimura らの部分パス木カーネルは他の木カーネルと同等以上の予測精度をもち, なおかつ Vishwanathan らの完全部分木カーネルと相補的に働くことがわかる.

Kimura らは木構造に対する拡張接尾辞配列を用いることで部分パス木カーネルを入力の木のサイズについて線形時間で計算するアルゴリズムを提案し, 実際に Vishwanathan らの線形時間木カーネルより高速に働くことを示した [Kimura 11]. しかし予測時においては彼らの木カーネルと異なり, 新たな入力の木構造とすべてのサポートベクトル間のカーネル関数を計算する必要がある. これは大量のデータを利用する際に深刻な速度低下を引き起こす. そこで本稿では Vishwanathan らの計算アルゴリズムを拡張することで, 彼らの木カーネルと同様に予測時においてサポートベクトルの数に依存しない高速な計算アルゴリズムを示す.

2. 部分パス木カーネル

Kimura らは木構造の階層構造を表現した垂直方向の特徴を捉えるために, 部分パスに基づいた木カーネルを提案した [Kimura 11]. 部分パスとは木の根から各葉までのパスの部分文字列として定義される. 部分パスの例を図 1(c) に示す. この部分パスを用いて 2 つの木 T_1 と T_2 間における部分パス木カーネルは以下のように定義される.

$$K(T_1, T_2) \equiv \sum_{p \in P} \lambda^{|p|} \mathrm{num}(T_{1p}) \mathrm{num}(T_{2p}) \qquad (1)$$

ここで P は T_1 と T_2 における全部分パス集合であり, $\mathrm{num}(T_{1p})$ と $\mathrm{num}(T_{2p})$ はそれぞれ T_1 と T_2 における部分パス $p \in P$ の出現回数である. また λ $(0 < \lambda \leq 1)$ は減衰パラメータである.

Kimura らは部分パスと木の接尾辞の接頭辞の間に一対一の対応関係が存在することを示した. 彼らはこの対応関係を利用し, 部分パス木カーネルの計算の際には式 (1) を直接計算せずに以下の式を計算した.

$$K(T_1, T_2) = \sum_{s \in S} \lambda^{|s|} num(T_{1s}) num(T_{2s}) \qquad (2)$$

ここで S は T_1 と T_2 における接尾辞の全接頭辞の集合である. 彼らは上記の式を木構造に対する拡張接尾辞配列を利用することで $\mathrm{O}(|T_1| + |T_2|)$ 時間で計算するアルゴリズムを提案した.

3. 木構造に対する接尾辞木と拡張接尾辞配列

本節では部分パス木カーネルの高速な計算アルゴリズムに必要不可欠なデータ構造である木構造に対する接尾辞木と拡張接尾辞配列について述べる.

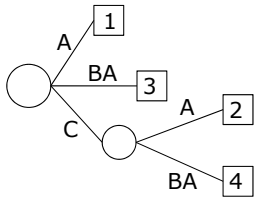
T を n ノードから構成される根付き木とする. ここで各ノードのラベルはサイズが $\sigma = |\Sigma|$ であるアルファベット集合 Σ からなるものとする. ノード i から根ノードまでのパスを T の i 番目の接尾辞 S_i と呼ぶ. また文字列 s について, 先頭から始まる部分文字列を s の接頭辞と呼ぶ. 木 T の接尾辞木とは T の全ての接尾辞に対するパトリシア木を指す (図 3(a)). 接尾辞木において共通する接尾辞の接頭辞は接尾辞木の根ノードからある内部ノードまでのパスに対応する. 接尾辞木によって木の接尾辞への高速なアクセスが可能となるが, 一般に接尾辞木は大量のメモリを消費するため接尾辞木は中規模以上のデータに対して用いることが困難である.

接尾辞木と同等の操作が可能であり, より省メモリであるデータ構造として拡張接尾辞配列 [Abouelhoda 04] があり, 現実の多くの場合で接尾辞木の代わりとして用いられている. 木 T に対する拡張接尾辞配列とは接尾辞配列と高さ配列の 2 つのデータ構造からなる. 接尾辞配列 $\mathrm{SA}[1, |T|]$ とは各接尾辞を辞書順に並べた際の先頭ノードのインデックスを保持した配列であり, 高さ配列 $\mathrm{LCP}[1, |T|]$ とは接尾辞配列において隣接する接尾辞の最長共通接頭辞長を保持した配列である. つまり $1 \leq i < |T|$ においては $\mathrm{LCP}[i] \equiv \mathrm{lcp}(S_{\mathrm{SA}[i]}, S_{\mathrm{SA[i+1]}})$ であり, $\mathrm{LCP}[|T|] \equiv -1$ である. ここで $\mathrm{lcp}(s, t)$ は 2 つの文字列 s と t の最長共通接頭辞長を表す. 高さ配列は接尾辞木の内部ノードの高さに対応している. 図 3(b) に接尾辞配列と高さ配列の例を示す.

4. 予測時における高速な計算アルゴリズム

カーネル法を大規模なデータセットに適用する際の深刻な問題の一つに, 予測時において新たな入力データ T と全てのサポートベクトル T_i $(i = 1, ..., m)$ との間でカーネル関数を評価する必要があるいうことがある. 部分パス木カーネルにおいては, 予測時において木 T を評価する際には以下の式を計算す

*1 学習器としては SVM を用いた. 実装は LIBSVM [Chang 01] を用いた. 正解率は 10 分割交差検定によって測定し, 木カーネルにおいてパラメータが存在するものについては同様に交差検定によってパラメータを決定した.



(a)

i	SA	LCP	*Si*
1	1	0	A
2	3	0	BA
3	2	1	CA
4	4	-1	CBA

(b)

図 3: (a) 図 1 (a) における木 T に対する接尾辞木 (b) 木 T に対する拡張接尾辞配列. 接尾辞配列と高さ配列から成る.

る必要がある.

$$f(T) = \sum_{i=1}^{m} \alpha_i K(T_i, T) = \sum_{s' \in PS_i} \sum_{s \in PS} \left(\sum_{i=1}^{m} \alpha_i \mathrm{num}(T_i s) \right) \lambda^{|s|} \delta(s', s) \quad (3)$$

ここで PS_i と PS はそれぞれ T_i と T における全接尾辞の接頭辞集合である. 式 (3) は $\{T_i\}_i$ と T がともに文字列である際には Teo と Vishwanathan らによる線形時間木カーネル [Teo 06] の予測時における評価式と完全に同一となる. 彼らは計算時間がサポートベクトルの数に依存せずに $\mathrm{O}(|T|)$ 時間で働く洗練された高速な予測アルゴリズムを提案した. 以降ではそのアルゴリズムを簡潔に述べる. 彼らは式 (3) の計算が入力の文字列 T の各接尾辞 S_i と, サポートベクトルに対応する文字列をひとつに結合させた “マスター文字列” 間の最長共通接頭辞 (lcp) を求める問題に帰着されることを示した. 彼らのアルゴリズムは, まずマスター文字列に対して拡張接尾辞配列を構築し, 入力の文字列 T に対し, 拡張接尾辞配列を利用することで接尾辞木のトップダウン巡回を模倣し, S_i とマスター文字列間の lcp を求める. この際に $l_i \geq l_{i-1} - 1$ が成立し, アルゴリズムはこの性質を利用する. ここで l_i は S_i とマスター文字列間の lcp である. この性質は l_i を求める際に, 最初の l_{i-1} 文字を比較する必要がないことを示しており, この性質によって全体で $\mathrm{O}(|T|)$ 時間のアルゴリズムで計算することができる.

われわれはこの文字列に対するアルゴリズムを木構造に対して拡張する. まずサポートベクトルに対応する木構造を一つにまとめ, この結合後の木構造に対して拡張接尾辞配列を構築する. 実際の予測時においては入力の木構造 T に対して結合後の木構造と S_i の lcp を先の拡張接尾辞配列を利用して求める. 文字列の場合との違いは比較せずにスキップできる長さである. 文字列の場合と異なり, 木構造においてノード i はひとつ以上の子ノードを持ちうるため $l_i \geq \max l_{Ch(i)} - 1$ が成立する. ここで $Ch(i)$ はノード i の子ノードの集合である. これは l_i を求める際に, 最初の $\max l_{Ch(i)} - 1$ ノードを比較する必要がないことを示している. 以下の定理は入力の木構造の T について予測時における最悪計算量が $\mathrm{O}(|T|^2)$ であることを保証している.

定理 1 予測時において木 T について式 *(3)* の計算量は高々 $\mathrm{O}(|T|^2)$ である.

証明 1 サポートベクトルに対応する木構造を一つに結合した木と各 S_i 間の *lcp* を全て求める際の接尾辞木の全巡回長を評価する.

$$\begin{aligned} &\sum_{i:\max l_{Ch(i)}=0} (l_i + 1) + \sum_{i:\max l_{Ch(i)}\neq 0} (l_i - \max l_{Ch(i)} + 2) \\ &\leq 2|T| + \sum_{i:l_i \neq \max l_{Sib(i)}} l_i \\ &\leq 2|T| + (\mathrm{L} - 1)\mathrm{H} \leq \mathrm{O}(|T|^2). \end{aligned}$$

$\max l_{Ch(i)} = 0$ を満たすノード i に関しては, *lcp* を求める際にノードをスキップすることができない. この場合は接尾辞木の根から最大マッチング長を求める *(1* 行目の第 *1* 項*)*. $\max l_{Ch(i)} \neq 0$ を満たすノード i に関しては, $\max l_{Ch(i)} - 1$ 個のノードの比較をスキップすることができる *(1* 行目の第 *2* 項*)*. また *2* 行目の第 *2* 項における $Sib(i)$ はノード i の兄弟ノード集合である. このノード集合の大きさは $L - 1$*(*ここで L は T における葉の数*)* であり, また l_i は高々 $\mathrm{H}(= \max \mathrm{depth}(T))$ であるから, 予測時における計算量は高々 $\mathrm{O}(|T|^2)$ である.

5. 実験

本節では提案する予測時における高速な計算アルゴリズム (‘Prediction’ と表記する) を実験的に評価する. 実験では XML データセット [Zaki 06] を用いてサポートベクトルの数と入力の木構造のサイズが予測時の計算時間に与える影響を調べた. この XML データセットの統計量を表 1 に示す. また提案アルゴリズムの計算時間を式 (3) を直接計算するアルゴリズム (‘Direct’) と既存の線形時間木カーネル*2 [Teo 06] (‘Vishwanathan’) と比較した. 実験において式 (2) の減衰パラメータは $\lambda = 1$ で固定した. また全てのサポートベクトルに対して一様に $\alpha_i = 1$ を与えた. 全ての実験は CPU が Intel Core2 Duo 2.40GHz, 主記憶装置 4GB, OS は Windows Vista という計算機環境で行った.

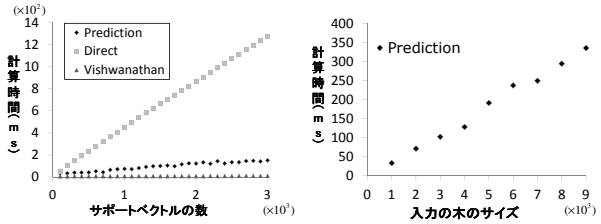
表 1: XML データセットの統計量

データ数	ノードラベル数	平均ノード数	平均次数	平均の深さ
3183	9097	14.3	1.9	6.6

最初に実験ではサポートベクトルの数が提案アルゴリズムに与える影響を調査した. ここでは XML データセットのうちの 100 個を選び, それらをひとつの木構造としてまとめて入力データとし, 残りをサポートベクトルとして用いてその数を変化させた. 図 4 (a) に実験結果を示す. 図の縦軸が計算時間であり, 横軸がサポートベクトルの数である. 図から ‘Direct’ の計算時間がサポートベクトルの数に関して線形に増加する一方で, ‘Prediction’ と ‘Vishwanathan’ の計算時間はサポートベクトルの数に依存しないことが確かに確認できる. また計算時間そのものについては ‘Vishwanathan’ が最も高速であった.

次に’Prediction’ アルゴリズムにおいて入力の木構造のサイズが計算時間に与える影響を調査した. ここではサポートベクトルの数を 100 で固定し, 入力の木構造のサイズを変化させた. 図 4 (b) に実験結果を示す. 図の縦軸が計算時間であり, 横軸が入力の木構造のサイズである. 提案アルゴリズムの理論的な最悪計算量は入力の木構造のサイズの 2 乗であるが, 実際の計算時間は線形に増加していることが確認できる.

*2 http://users.cecs.anu.edu.au/~chteo/SASK.html



(a) (b)

図 4: (a) サポートベクトルの数を変化させた場合の実行時間の変化. (b) 入力の木構造のサイズを変化させた場合の実行時間の変化.

6. まとめ

本稿でわれわれは木村らによる部分パス木カーネル [Kimura 11] に着目し, Vishwanathan らの線形時間木カーネルの計算アルゴリズム [Teo 06] を拡張することで予測時における高速な計算アルゴリズムを提案した. 提案アルゴリズムは予測時において新たな入力の木構造 T に関して高々$O(|T|^2)$ 時間で計算することができ, サポートベクトルの数に依存しない. さらに実験を通して提案アルゴリズムの計算時間が, 実際には入力の木構造のサイズに関して実際には線形に増加し, また確かにサポートベクトルの数に依存しないことを確認した.

参考文献

[Abiteboul 00] Abiteboul, S., Buneman, P., and Suciu, D.: *Data on the Web: from relations to semistructured data and XML*, Morgan Kaufmann (2000)

[Abouelhoda 04] Abouelhoda, M. I., Kurtz, S., and Ohlebusch, E.: Replacing Suffix Trees With Enhanced Suffix Arrays, *J. Discrete Algorithms*, Vol. 2, No. 1, pp. 53–86 (2004)

[Aiolli 09] Aiolli, F., Martino, G. D. S., and Sperduti, A.: Route Kernels for Trees, in *ICML*, pp. 17–24 (2009)

[Chang 01] Chang, C.-C. and Lin, C.-J.: *LIBSVM: a library for support vector machines* (2001)

[Collins 01] Collins, M. and Duffy, N.: Convolution Kernels for Natural Language, in *NIPS*, pp. 625–632 (2001)

[Doubet 92] Doubet, S. and Albersheim, P.: CarbBank, *Glycobiology*, Vol. 2, No. 6, p. 505 (1992)

[Durbin 98] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G.: *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press (1998)

[Gärtner 03] Gärtner, T., Flach, P., and Wrobel, S.: On Graph Kernels: Hardness Results and Efficient Alternatives, in *COLT* (2003)

[Hashimoto 03] Hashimoto, K., Hamajima, M., Goto, S., Masumoto, S., Kawashima, M., and Kanehisa, M.: GLYCAN: The database of carbohydrate structures, *In GIW*, Vol. 14, pp. 649–650 (2003)

[Haussler 99] Haussler, D.: Convolution Kernels on Discrete Structures, Technical Report UCSC-CRL-99-10, University of Calfornia in Santa Cruz (1999)

[Kashima 02] Kashima, H. and Koyanagi, T.: Kernels for Semi-Structured Data, in *ICML*, pp. 291–298 (2002)

[Kashima 03] Kashima, H., Tsuda, K., and Inokuchi, A.: Marginalized Kernels between Labeled Graphs, in *ICML*, pp. 321–328 (2003)

[Kimura 11] Kimura, D., Kuboyama, T., Shibuya, T., and Kashima, H.: A Subpath Kernel for Rooted Unordered Trees, in *PAKDD*, pp. 62–74 (2011)

[Kramer 01] Kramer, S. and De Raedt, L.: Feature Construction with Version Spaces for Biochemical Application, in *ICML*, pp. 258–265 (2001)

[Kuboyama 06] Kuboyama, T., Hirata, K., Aoki-Kinoshita, K. F., Kashima, H., and Yasuda, H.: A Gram Distribution Kernel Applied to Glycan Classification and Motif Extraction, in *GIW*, pp. 25–34 (2006)

[Leslie 02] Leslie, C., Eskin, E., and Noble, W. S.: The spectrum kernel: a string kernel for SVM protein classification, in *PSB*, pp. 564–575 (2002)

[Lodhi 02] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C.: Text classification using String Kernels, *J. Mach. Learn. Res.*, Vol. 2, pp. 419–444 (2002)

[Manning 99] Manning, C. D. and Schütze, H.: *Foundations of Statistical Natural Language Processing*, The MIT Press (1999)

[Moschitti 06] Moschitti, A.: Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees, in *ECML*, pp. 318–329 (2006)

[Schölkopf 02] Schölkopf, B. and Smola, A. J.: *Learning with Kernels*, MIT Press (2002)

[Sun 11] Sun, J., Zhang, M., and Tan, C. L.: Tree Sequence Kernel for Natural Language, in *AAAI*, pp. 921–926 (2011)

[Teo 06] Teo, C. H. and Vishwanathan, S. V. N.: Fast and space efficient string kernels using suffix arrays, in *ICML*, pp. 929–936 (2006)

[Vishwanathan 03] Vishwanathan, S. V. N. and Smola, A.: Fast Kernels for String and Tree Matching, in *NIPS*, pp. 569–576 (2003)

[Zaki 06] Zaki, M. J. and Aggarwal, C. C.: Xrules: An effective structural classifier for XML data, *Mach. Learn.*, Vol. 62, No. 1-2, pp. 137–170 (2006)