社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

素性の組み合わせを実現する Power Set Kernel とその高速化

工藤　　拓†　　松本　裕治†

† 奈良先端科学技術大学院大学　〒 630-0192 生駒市高山町 8916-5
E-mail: †{taku-ku,matsu}@is.aist-nara.ac.jp

あらまし　近年, Support Vector Machine を中心とする Kernel 法が注目され, 多くの分野に応用されている. 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, まず, 素性の組み合わせを実現する Power Set Kernel を定式化する. 次に, バスケットマイニングアルゴリズムを用いて, 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, 単純な線型分類器に変換することで, 解析の速度向上を試みる. 日本語わかち書き, 及び, 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 30-280 倍の速度向上に成功した.

キーワード　Support Vector Machine, Kernel 法, Power Set Kernel, Data Mining

Power Set Kernel for Feature Combination: Data Mining approach for its fast classifiers

Taku KUDO † and Yuji MATSUMOTO†

† Graduate School of Information Science,Nara Institute of Science and Technology
8916-5, Takayama-Cho, Ikoma-shi Nara, 630-0192 Japan
E-mail: †{taku-ku,matsu}@is.aist-nara.ac.jp

Abstract　The kernel method (e.g., Support Vector Machines) attracts a great deal of attention recently. The merit of the kernel method is that the *effective feature combination*, which has been manually selected in the previous approaches, is implicitly expanded without loss of generality and computational cost. However, the kernel-based approach is usually too slow to classify large-scale test data. In this paper, we fist formulate a Power Set Kernel which gives a dot product of two *sets*. Then, we extend the *Basket Mining* algorithm to convert a kernel-based classifier into a simple and fast linear classifier. Experimental results on Japanese Word Segmentation and Japanese Dependency Parsing show that our new classifier is about 30-280 times faster than the standard kernel-based classifier.

Key words　Support Vector Machines, Kernel Method, Power Set Kernel, Data Mining

1. はじめに

近年, Support Vector Machine [13] を中心とする Kernel 法が多くの分野に応用され, いずれにおいても高い精度を示している. 例えば, 自然言語処理においては, 文書分類 [5], テキストチャンキング [9], 固有名詞抽出 [4], [14], 構文解析等 [16] のタスクに応用されている.

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(注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. 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, 汎化誤差を小さくすることができる. Kernel を用いた 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.

しかし, SVM に問題点が無いわけではない. まず, Kernel 関数は, 素性空間を陰に表現するために, 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, 大規模なテストデータの分類は困難である.

本稿では, まず, Power Set Kernel という集合の内積を与える Kernel を定式化する. この Kernel は, 自然言語データなどの離散的データを分類するための, Kernel である. 次に, Power Set Kernel に基づく分類器の高速化について PSKI, PSKE 2 つの手法を提案する. 日本語わかち書き, 及び, 日本語係り受け解析における実験では, Kernel を用いた従来の分類器に比べ, PSKI で約 3-12 倍, PSKE で約 30-280 倍の速度向上に成功した.

2. Kernel Method

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_j 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.

$$y = \mathrm{sgn}\Big(\sum_{j=1}^{L} y_j \alpha_j \phi(\mathbf{x}_j) \cdot \phi(\mathbf{x}) + b\Big) \qquad (1)$$

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） $\alpha_i, b \in \mathbf{R},\ 0 \leqq \alpha_j \leqq C$ である.

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, $\phi(\mathbf{x}_j) \cdot \phi(\mathbf{x})$ という内積の形で出現する. 本稿では, α_j, 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.

$$y = \mathrm{sgn}\Big(\sum_{j=1}^{L} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}) + b\Big) \qquad (2)$$

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(注2). 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 [3], [6], さらに, 無向グラフのノード間の内積を与える Deffusion Kernel [7] などが提案されている.

3. Power Set Kernel

ここで, 分類したい事例 $\mathbf{x}$ が, **集合**として表現される場合を考える. 自然言語処理においては, このように事例を集合で表現することが多い. 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.

まず, 素性集合 $F = \{1, 2, \ldots, N\}$ を与える. すべての事例 $X_j (j = \{1, 2, \ldots, L\})$ は, F の部分集合とする $(X_j \in F)$. この時, 集合 X_j は, 素性空間を 2 値ベクトル $\mathbf{x}_j = \{x_{j1}, x_{j2}, \ldots, x_{jN}\}$ (ただし $i \in X_j$ の時, $x_{ji} = 1$, それ以外の時に $x_{ji} = 0$). で表現することと等価であり, 実数値の素性ベクトルの特殊形となる. Kernel 法の考えに従えば, SVM を適用するには, 集合を分類するための「適切な」 Kernel を設計すればよい. 本稿では, そのような Kernel の一種として, **Power Set Kernel (PSK)** を与える.

(注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

(注2)：実際には Kernel として満たすべき制約 (Mercer's condition) が存在する

［定義 1］ **Power Set**
集合 X (要素の数を $|X|$ と表記) の Power Set $P(X)$ とは, X の**すべての部分集合の集合**である.
［例 1］ $X = \{a, b, c\}$
$P(X) = \{\phi, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

［定義 2］ **Power Set Kernel** $K(X, Y)$
集合 X, Y の内積を与える Kernel $K(X, Y)$ が Power Set Kernel であるための必要十分条件は,

$$K(X,Y) = \sum_{r=0}^{|X \cap Y|} c_r \cdot |\{s | s \in P(X \cap Y), |s| = r\}|$$

となる $c_r \in \mathbf{R}^*$ が存在することである.
c_r は, サイズ r の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. 本稿では, c_r を, **部分集合重み**と呼ぶ. 個々の PSK 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.
［例 2］ $X = \{a, b, c\}$, $Y = \{a, b, d\}$
$c_0 = 1, c_1 = 2, c_2 = 4, c_3 = 1$
$K(X, Y) = 1 \cdot 1 + 2 \cdot 2 + 4 \cdot 1 + 1 \cdot 0 = 9$

3.1 Power Set Kernel の周辺定理

［補題 1］ d を任意の正の整数とする時, $K(X, Y) = |X \cap Y|^d$ は, Power Set Kernel ある.
［証明 1］ 付録 A を参照. 本稿では, $K(X, Y) = |X \cap Y|^d$ の部分集合重みを $c_r = \tau(d, r)$ と記す. ■
［補題 2］ g を, n 回微分可能で, $g^{(n)}(0) \geqq 0$ なる実関数とするとき, $K(X, Y) = g(|X \cap Y|)$ は, Power Set Kernel である.
［証明 2］ $K(X, Y) = g(|X \cap Y|)$ は,

$$K(X,Y) = g(|X \cap Y|) = \sum_{l=0}^{\infty} \frac{g^{(l)}(0)|X \cap Y|^l}{l!}$$

のように漸近展開できる. $|X \cap Y|^i$ は, PSK であることに注意すると, 部分集合重みは, $c_r = \sum_{l=r}^{\infty} \frac{g^{(l)}(0)\tau(l,r)}{l!}$ となる. よって, $K(X, Y) = g(|X \cap Y|)$ は, PSK である. ■
［例 3］ 多項式 Kernel
集合 X, Y に対する, d 次の 多項式 Kernel は, 次式で与えられ, 補題 2 より, PSK となる.

$$K(X,Y) = (1 + |X \cap Y|)^d = \sum_{l=0}^{d} {}_dC_l |X \cap Y|^l$$

また, 上式より, d 次の 多項式 Kernel は, 部分構造重み $c_r = \sum_{l=r}^{d} {}_dC_l \cdot \tau(l, r)$ を持つ. 例えば, 3 次の 多項式 kernel の場合は, $c_0 = 1, c_1 = 7, c_2 = 12, c_3 = 6$ となる.
［例 4］ Quasi-RBF Kernel
集合 X, Y に対する, RBF Kernel (分散パラメータ $a \in \mathbf{R}^+$) は, 次式で与えられる.

$$K(X,Y) = \exp(-a(|X|^2 + |Y|^2 - 2|X \cap Y|))$$
$$= \exp(-a|X|^2)\exp(-s|Y|^2)\exp(2a|X \cap Y|)$$

ここで $|X| = |Y| = N$(定数) (集合の要素数は一定) という仮定をおくと, RBF Kernel は, 補題 2 より, PSK となる. この時, RBF Kernel は, 部分構造重み $c_r = \exp(-aN^2) \sum_{l=r}^{\infty} \frac{(2s)^l \tau(l,r)}{l!}$ を持つ.

```
function PSKI_classify(X)
  r = 0 # 配列，値を 0 に初期化
  foreach i ∈ X
    foreach j ∈ h(i)
      r_j = r_j + 1
    end
  end
  result = b #戻り値，実数
  foreach j ∈ (1 .. L)
    result += y_j α_j · g(r_j)
  end
  return sgn(result)
end
```

図 1 PSKI の擬似コード

4. Power Set Kernel の高速化

本節では, 次式で与えられる分類関数について, その高速化手法を述べる. ただし, 関数 g は, 補題 2 をみたす関数である.

$$y = \mathrm{sgn}\Big(\sum_{j=1}^{L} y_j \alpha_j \cdot g(|X_j \cap X|) + b\Big) \qquad (3)$$

便宜的に, 本稿では, 式 (3) の分類手法を **PSKB** (Baseline) と呼ぶ. PSKB の分類計算量は, 事例 X とサポートベクター X_j との共通要素を列挙するのに $O(|X|)$, さらに, サポートベクターが L 個あるため, 全体として $O(|X| \cdot L)$ となる.

4.1 PSKI (Inverted Representation)

X 中の各要素 $i \in X$ が, どのサポートベクターに出現するかが事前に分かっていれば, 共通要素の数を全事例について列挙する必要はなくなる. これは, 情報検索の転地インデックスと同じ考えである. 図 (1) にアルゴリズム PSKI の擬似コードを示す. $h(i)$ は, 要素 i を含むサポートベクターを返す関数 (テーブル) である.
PSKI の計算量は, $O(|X| \cdot B + L)$ である (B は, $|h(i)|$ の平均値). PSKI は, B が小さい時, すなわち, 素性空間が疎の場合に特に高速となる.

4.2 PSKE (Expanded Representation)

4.2.1 基本的なアイデア

PSK の定義より, 関数 g には, 固有の部分集合重み c_r が存在する. c_r を用いると, 式 (3) は, 以下のようになる.

$$y = \mathrm{sgn}\Big(\sum_{j=1}^{L} y_j \alpha_j K(X_j, X) + b\Big) \qquad (4)$$

ただし,

$$K(X_j, X) = \sum_{r=0}^{|X_j \cap X|} c_r \cdot |\{s | s \in P(X_j \cap X), |s| = r\}$$

ここで, すべての部分集合 $s \in P(F)$ について,

$$w(s) = \sum_{i=1}^{L} y_j \alpha_j c_{|s|} I(s \in P(X_j)) \qquad (5)$$

を事前に求めておけば (ただし, $I(\cdot)$ は, 引数が真のときに 1 それ以外は 0 を返す関数である), 式 (4) は, 式 (6) のような簡単な形になる.

$$y = \mathrm{sgn}\Big(\sum_{s \in P(X)} w(s) + b \Big) \qquad (6)$$

式 (6) による分類アルゴリズムを, 本稿では PSKE と呼ぶ. PSKE の計算量は, $O(|P(X)|)$ となり, サポートベクター数 L に依存しない. さらに, 部分集合 s について $|w(s)|$ の大きい s は, 分類に寄与する部分集合となる. つまり, 分類に寄与する部分集合を陽に提示することが可能となる.

4.2.2 データマイニングに基づく PSKE

PSKE を適用するには, まず, ベクトル

$$\mathbf{w} = \{w(s_1), w(s_2), \ldots, w(s_{|P(F)|})\} \qquad (7)$$

を事前に算出する必要がある. 2 次の多項式 Kernel のように, サイズ 2 までの部分集合のみを列挙すればよい時は(注3), 式 (5) を用い, 直接算出できる. これは, 磯崎らが 2 次の 多項式 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 次の多項式 Kernel の場合, 組み合わせの数が指数的に増え列挙が困難である. そこで, 以下のような近似解 $\mathbf{w}'$ を求め, 代用する.

［定義 3］ **$\mathbf{w}$ の近似解 $\mathbf{w}'$**

$\mathbf{w} = \{w(s_1), w(s_2), \ldots, w(s_{(|P(F)|)})\}$ について, $\sigma_{neg} < w(s) < \sigma_{pos}$ ($\sigma_{neg} < 0$, $\sigma_{pos} > 0$) となる場合は, $w(s) = 0$ に近似する. 結果, 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集合は分類に寄与しないという仮定の基での近似解である. $\mathbf{w}'$ 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.

［定義 4］ **有効部分集合のマイニングタスク**

サポートベクターの集合 $(y_1\alpha_1, \mathbf{x}_1), \ldots, (y_L\alpha_L, \mathbf{x}_L)$, 部分集合重み $c_0, c_1, \ldots$ が与えられた時に, $w(s) \geqq \sigma_{pos}$ または $w(s) \leqq \sigma_{neg}$ となるような, 部分集合 s と, その重み $w(s)$ を網羅的に求めよ.

有効部分集合のマイニングタスクは, **バスケットマイニングタスク**の変形とみなすことができ, Apriori [1] をはじめとするマイニングアルゴリズムを用いて効率良く算出できる. 実際には, オリジナルのアルゴリズムに対し以下の 4 点の変更を与える必要がある.

- サイズの制限

d 次の 多項式 kernel の場合は, d 個 までの部分集合を列挙するだけで十分である.

- 部分集合重み c_r

サイズ r の部分集合の頻度は c_r 倍される. 一般にマイニングアルゴリズムの多くは, 小さい部分集合から, 集合のサイズを深さ優先または幅優先で大きくしながら, 頻出部分集合を列挙する. そのため, 大きい部分集合に対する重み c_r が, 小さい部分集合に対する重み c_r より大きい場合, パターンを取りこぼしてしまう可能性がある. そのため, 最悪のケースを考え, c_r のうち最大の値 $c_{max}(= \max(\{c_1, c_2, \ldots, c_n\}))$ を選び, 部分集合のサイズに関わらず, c_{max} 倍した頻度によってマイニングアルゴリズムを動作させる. 一方, 実際の部分集合 s の頻度は, マイニングによって得られた頻度を $c_{|s|}/c_{max}$ 倍して求める.

- 頻度 $c_{max} y_j \alpha_j$

データベース中の各トランザクションが, サポートベクター X_j に対応する. マイニングアルゴリズムを動作させる時には, $c_{max} y_j \alpha_j$ をそのトランザクションの頻度とみなす.

- 正例/負例

各トランザクザクションは, $-\infty - \infty$ の範囲の頻度を持つ. このため, 正例 ($y_j > 0$), 負例 ($y_j < 0$) のトランザクションに分け, それぞれ独立にマイニングを行い, 結果をマージする.

- 最小サポート $\sigma_{pos}, \sigma_{neg}$

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.

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

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

マイニング後, 部分集合 s と, それに付与される重み $w(s)$ のタプルの集合 $\Omega = \{\langle s, w(s) \rangle\}$ が抽出される. この集合を効率良く保持するために, 共通部分を共有する TRIE を作成する. TRIE の各ノードには, そのノードに対応する部分集合が持つ重み $w(s)$ が格納される. TRIE の実装として,

(注3): 例 3 の結果より, d 次の多項式 Kernel における部分集合重みは, $c_r = 0$ ($r > d$) となる. すなわち, d 次の多項式 Kernel においては, 元の素性空間 F は, F^d の空間 ($P(F)$ の部分空間) のみに射影される

Double-Array [2], [8] 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, 日本語わかち書きタスク (JWS), 日本語係り受け解析タスク (JDP) に対し実験を行った. それぞれの詳細は文献 [12], [15], [16] に譲る. Kernel 関数には予備実験の結果に基づき, 3 次の多項式 kernel を用いた. 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. バスケットマイニングアルゴリズムには, PrefixSpan [11] を用いた(注4). 表 1 に, 実験データのサイズ等の情報を示す. 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.

5.2 結　　果

表 2,3 に, PSKI 及び PSKE において 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(注5) , 部分集合のサイズ $|\Omega|$ をまとめた.

PSKI は, JWS で約 1.7 倍 (0.85 → 0.49), JDP で約 12 倍 (0.28 → 0.022), の速度向上が確認された. JDP では, B ($|h(i)|$ の平均値) が JWP に比べ小さく, 素性空間が疎となっているため, このような結果が得られたと考察される.

PSKE は, JWP で約 280 倍 ($\sigma = 0.005$, 0.85 秒/文 → 0.0033 秒/文), JDP で約 30 倍 ($\sigma = 0.0005$, 0.28 秒/文 → 0.0097 秒/文) の速度向上が確認された.

5.3 頻度によるフィルタリング

実験より, PSKE において, PSKB と同等の解析精度を得るためには, JWS においては $\sigma = 0.005$ 程度, JWD においては $\sigma = 0.0005$ 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 232 万 (JWD)/826 万 (JWP) で, 非常に大きく, 実用的ではない(注6).

一般に, $|\Omega|$ 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, 不必要な部分集合を削除するために, サポートベクターの集合から, Ω 中の各部分集合の出現頻度を計算し, 高頻度の部分集合のみを残し, 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. JWD では, σ=0.005, JDP では σ=0.0005 に固定し, 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 4,5 に示す.

結果, JDP において, $\xi = 2$ とした場合, TRIE のサイズを約 1/3 にできた. さらに, 速度向上 (0.0097 秒/文 → 0.0074 秒/文) も確認できた. また, 若干ながらの精度向上 (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. このような部分集合は, 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.

一方, JWP では, $\xi = 2$ の場合, TRIE のサイズを 約半分にできたが, 解析精度の低下 (97.94%→97.83%) が確認された. これは頻度 1 の素性も分類に寄与していることを示唆している.

6. 今後の課題とまとめ

本稿では, まず, Power Set Kernel という集合の内積を与える Kernel を定式化した. さらに, Power Set Kernel に基づく分類器の高速化について PSKI, PSKE 2 つの手法を提案した. 日本語わかち書き, 及び, 日本語係り受け解析における実験では, Kernel を用いた従来の分類器に比べ, PSKI で約 3-12 倍, PSKE で約 30-280 倍の速度向上に成功した.

今後は, 他の離散 Kernel (Tree Kernel, String Kernel) について, 部分木 や 部分系列のマイニングアルゴリズムを用いた高速分類手法を提案したいと考える.

付　録　A

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

[証明] 集合 X, Y は, 集合 $F = \{1, 2, \ldots, N\}$ の部分集合とする. この時, 共通要素の個数 $|X \cap Y|$ は, 以下の 2 値ベクトルの内積 $\mathbf{x} \cdot \mathbf{y}$ と同値である.

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

x_j, y_j は, 集合 X, Y が要素 j を含む時に 1, それ以外に 0 となる. この時, 多項定理を用いると

$$(\mathbf{x} \cdot \mathbf{y})^d = \left(\sum_{j=1}^{N} x_j y_j \right)^d = \sum_{k_n \geqq 0}^{k_1 + \ldots + k_N = d} \frac{d!}{k_1! \ldots k_N!} (x_1 y_1)^{k_1} \ldots (x_N y_N)^{k_N}$$

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 ($r = \{0, 1, \ldots, d\}$) の部分集合の個数は, $(x_1 y_1 x_2 y_2 \ldots x_r y_r)$ の係数となる. よって,

$$\tau(d, r) = \sum_{k_n \geqq 1, n=1,2,\ldots,r}^{k_1 + \ldots + k_r = d} \frac{d!}{k_1! \ldots k_r!}$$

(注4) : PrefixSpan は, 頻出部分系列を列挙するアルゴリズムであるが, 集合は, 要素を辞書式にソートしておくことで, 系列とみなすことができる.

(注5) : 現実的な評価のため, 1 文あたりの解析時間とした.

(注6) : この時の TRIE のサイズは,111MB(JWD)/391MB(JDP) であった.

表 1　実験データの詳細

データセット	JWS	JDP
事例数	265,413	110,355
SV 数	57,672	34,996
SV の正例数	28,440	17,528
SV の負例数	29,232	17,468
$\|F\|$ (素性集合のサイズ)	11,643	28,157
$\|X_j\|$ の平均	11.73	17.63
B ($\|h(i)\|$ の平均))	58.13	21.92

表 2　実験結果 (日本語わかち書き JWS)

PSKE σ	time (秒/文)	acc.(%)	$\|\Omega\|$
0.1	0.0009	96.09	21,118
0.05	0.0014	97.36	84,409
0.01	0.0028	97.93	1,228,035
0.005	0.0033	97.95	2,327,599
0.001	0.0039	97.94	4,392,993
0.0005	0.0041	97.94	4,820,714
0.0001	0.0042	97.94	5,206,639
PSKI	0.4989	97.94	
PSKB	0.8535	97.94	

表 3　実験結果 (日本語係り受け JDP)

PSKE σ	time (秒/文)	acc.(%)	$\|\Omega\|$
0.1	0.0013	82.02	7,289
0.05	0.0020	86.27	30,523
0.01	0.0050	88.91	73,9147
0.005	0.0067	89.05	1,924,221
0.001	0.0092	89.26	6,686,737
0.0005	0.0097	89.29	8,262,428
0.0001	0.0101	89.29	9,846,092
PSKI	0.0226	89.29	
PSKB	0.2848	89.29	

表 4　頻度による枝刈り (日本語わかち書き JWS)

ξ	time (秒/文)	acc.(%)	$\|\Omega\|$
1	0.0033	97.95	2,327,599
2	0.0030	97.83	954,840
3	0.0030	97.83	591,793
4	0.0029	97.83	421,463
PSKI	0.4989	97.94	
PSKB	0.8535	97.94	

表 5　頻度による枝刈り (日本語係り受け JDP)

ξ	time (秒/文)	acc.(%)	$\|\Omega\|$
1	0.0097	89.29	8,262,428
2	0.0074	89.34	2,450,855
3	0.0069	89.31	1,360,019
4	0.0065	89.25	945,286
PSKI	0.0226	89.29	
PSKB	0.2848	89.29	

$$= r^d - {}_rC_1(r-1)^d + {}_rC_2(r-2)^d - \ldots$$
$$= \sum_{m=0}^{r} {}_rC_m(-1)^{r-m} \cdot m^d \quad \blacksquare.$$

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] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, No. 1398, pp. 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

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

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

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

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

[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] Manabu Sassano. An empirical study of active learning with support vector machines for japanese word segmentation. In *Proceedings of the Second Meeting of the ACL*, pp. 505–512, 2002.

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

[14] 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.

[15] 新納弘幸. 決定リストを弱学習器としたアダブーストによる日本語単語分割. 自然言語処理, Vol. 8, No. 2, 2001.

[16] 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.