2008－NL－187 (9)
2008／9／25

全ての部分文字列を考慮した文書分類

岡野原 大輔 † 辻井 潤一 †‡§

† 東京大学情報理工学系研究科コンピュータ科学専攻
‡ SORST, 科学技術振興事業団 § School of Informatics, University of Manchester
{ hillbig, tsujii }@is.s.u-tokyo.ac.jp

本稿では，全ての部分文字列が素性として利用される文書分類モデル，及びその効率的な学習，推定手法を提案する．文書分類に有効な部分文字列は，単語と異なる場合や，署名やテンプレートなど，非常に長くなる場合が少なくない．しかし，部分文字列の種類数は文書長の二乗に比例するため，それらを素性として直接用いて学習することは，計算量的に困難だった．本稿では，テキスト長に比例する個数のみ存在する極大部分文字列に関する統計量を扱うことで，有効な部分文字列を漏れなく求めることができることを示す．また，拡張接尾辞配列を用いることで，これらを効率的に列挙可能であり，全文書長に比例した時間で学習可能であることを示す．さらに L_1 正則化を適用することで，コンパクトな学習結果が得られ，高速な推定が可能であることを示す．このモデルは，形態素解析結果や TF/IDF などの統計量と組み合わせられることを示し，従来の単語ベースの Bag of Words 表現と比較し，精度が向上することを示す．

Text Categorization with All Substring Features

Daisuke Okanohara † Jun'ichi Tsujii †‡§

†Department of Computer Science, University of Tokyo
‡School of Informatics, University of Manchester §National Center for Text Mining, UK
{ hillbig, tsujii }@is.s.u-tokyo.ac.jp

This paper presents a novel document classification method using all substrings as features. Although an effective substring for a document classification task is often different from tokanized words, the number of all candidate substrings is the quadratic of the length of a document, and a learning using all these substrings as features requires a prohibitive computational cost. We show that we can compute all effective substrings exhaustively by check only maximal substrings, which can be enumerated in linear time by using enhanced suffix arrays. Moreover, we use L_1 regularization to obtain a compact learning result, which makes a inference effient. We show that many prior weights (tf, idf, other tokenization result) can be included in this method naturally. In experiments, we show that our model can extract effective substrings, and more accurate than that of word-base BOW representation.

1 はじめに

文書分類タスクは，与えられた文書に対し，その文書の意味に基づいたラベルを付与するタスクである．この分類には，従来，人手により作成されたルールに基づく分類がされていたが，近年では，サポートベクトルマシン (SVM) やロジスティック回帰モデル (LR) など，機械学習による分類 (例 [1]) が，頑健かつ高精度な分類を達成できるため，主流となってきた．

機械学習を用いる手法では，多くの場合，各文書 d は，その中に出現する単語により決定される素性ベクトル $f(d) \in R^m$ で表わされる．これは，各単語の順序や位置は無視されるため，Bag of Words (BOW) 表現と呼ばれる．文書分類タスクでは，特定単語の出現がラベル推定に貢献する場合が多いため，単純化された BOW 表現でも，高い精度を達成する場合が多い．

この BOW 表現への変換は，日本語など，分かち書きされてない言語の場合は，文書に対し形態素解析を行い，得られた形態素列や，N 文字以下 (N は 2, 3 を使う場合が多い) の全ての部分文字列を単語として扱い，また，英語など分かち書きされている言語では，区切られた部分文字列をそのまま単語として扱う場合が多い．しかし，これらの変換には少なくとも次の三つの問題点がある．

一つ目は，単語に変換する際の解析エラーである．現在，日本語の形態素解析の精度は新聞コーパスなどにおいては 9 割近くに達するが，未知語が頻出するブログやメールの解析など他分野の文書に適用した場合，解析精度は低くなる．特に，文書分類に重要な固有表現（人名

や商品名など）の多くが未知語であるため，これらで形態素解析が失敗し，素性として利用できない．

二つ目は，単語の定義が困難な場合である．十分な精度の形態素解析器が無い言語や，今回は対象としていないが塩基配列などの生物情報配列，ログ情報においては，どのような単位で単語を定義するかが自明でない．

三つ目は，文書分類タスクにおいて効く単位と，形態素解析単位が違う場合が多いことである．例えば，映画タイトルは，カテゴリ推定に重要な情報だが，形態素単位に分けられてしまうと，この情報は利用できない．さらにスパムメール判定などでは署名やテンプレートなどが重要となるが，こうした非常に長い部分文字列情報は単語ベースの BOW 表現では利用できない．

このような BOW 表現の問題点を解決した手法として文字列カーネル [2] が知られている．これは二つの文書間 d_1, d_2 のカーネル値を

$$K(d_1, d_2) = \sum_{s \in \Sigma^*} d_s s(d_1) s(d_2) \qquad (1)$$

として定義する，ただし Σ^* はアルファベット Σ からなる全ての長さの部分文字列の集合であり，d_s は s に依存する重みパラメータ（学習では決定できない），$s(d)$ は部分文字列 s の d 中の出現頻度である．

この文字列カーネルを利用し，SVM などの学習を行いカテゴリ分類を行なうことで，全ての部分文字列を考慮して分類することができる．接尾辞配列などのデータ構造を利用することにより，$O(|d_1| + |d_2|)$ 時間でカーネル値を計算でき，テスト文書 d に対する推定結果も $O(|d|)$ 時間で行なうことができる [3]．

しかし，こうした高速化を用いても文字列 カーネルを利用した場合，推定時にも作業領域量が訓練文書の約 19 倍必要という問題点もあるため，大規模の訓練データを用いる場合では実用的ではない．

さらに，一般のカーネル法と同様，どの訓練データを重くみるかということしか，パラメータを調整することしかできず，部分文字列毎に異なる重みを与えることができない．一般に文書分類に貢献するような部分文字列は非常に少ないため，文字列カーネルではノイズの影響が非常に受けやすい．そのため，長さで打ち切るなどのヒューリスティックスを使わなければならず，全ての部分文字列を考慮することは現実的には難しい．

本稿では文字列カーネルと同様に，全ての部分文字列を考慮しながらも，カーネル法を使わずに，全ての部分文字列が素性として明示的に表されている線形識別モデルを考える．これにより，部分文字列毎に違う重みを与えることができる．更に，学習時に L_1 正則化を適用することにより，学習後の有効な素性数（重みが 0 でない素性数）は非常に少なく，分かりやすい学習結果と少ない作業領域量を達成できる．

この提案手法では，カーネル法を用いないため，訓練時間は訓練数ではなく，素性種類数に依存する．部分文字列種類数は文書長の二乗個存在するため，直接学習することは計算量的に困難である．最近の研究 [4] でも部分文字列を素性として利用した Logistic 回帰モデルによる文書分類を提案しているが，有効な部分文字列を greedy に探索する方法であり網羅性は保障できない．

本稿では，極大部分文字列のみを扱えば有効な部分文字列が漏れなく全て列挙できることを示す．さらに，これら極大部分文字列は拡張接尾辞配列を利用することにより，文書長に比例する時間で効率よく列挙することができ，有効な部分文字列を効率よく抽出可能であることを示す．また，L_1 正則化の Grafting と組み合わせることにより，訓練データ長に比例する時間で学習可能であることを示す．推定時にも，学習の結果得られた有効な部分文字列より構成された trie データ構造を利用することにより，テスト文書長に比例する時間で処理可能なことも示す．

文書分類タスクの実験結果では，本手法が適切な部分文字列を効率的に抽出可能であり，精度も単語ベースの BOW 表現による最大エントロピーモデルや SVM より高いことを示す．

2 L_1 正則化付 Logistic 回帰モデル

本稿では文書分類器として多クラス Logistic 回帰モデル (LR)[1] を利用する．LR は入力 x, 出力 $y \in \{1, \ldots, k\}$ から定義される素性ベクトル $\phi(x, y) \in R^m$ を用いて，次のように定義される

$$\begin{aligned} p(y|x; \boldsymbol{w}) &= \frac{1}{Z(x)} \exp(\boldsymbol{w}^T \phi(x, y)) \qquad (2) \\ Z(x) &= \sum_{y'} \exp(\boldsymbol{w}^T \phi(x, y')) \end{aligned}$$

但し重みベクトル $\boldsymbol{w} \in R^m$ は，モデルパラメータである．確率が最大となるラベルを選ぶ場合は，重みベクトルと内積が最大となるラベル y^* を選べば良い．

$$y^* = \arg\max_y p(y|x; \boldsymbol{w}) = \arg\max_y \boldsymbol{w}^T \phi(x, y) \qquad (3)$$

訓練データ (x_i, y_i) $(i = 1, \ldots, n)$ を利用し，重みベクトルを最尤推定して求めた場合，次の通りになる，

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} \sum_i \log p(y_i|x_i; \boldsymbol{w})$$

この最尤推定は，パラメータ数が訓練データ数に比べ多い場合，過学習を引き起こす．例えば，ある素性 $\phi_k(x, y)$ が訓練データでのみで発火する場合，それに対応する重み w_k は $+\infty$ に発散する．本稿では重みベクトルに対し，次の L_1 正則化を学習時に適用する (L_1-LR),

[1] このモデルは素性ベクトルが入力から決定される素性ベクトルと出力ラベルの直積である場合の最大エントロピーモデルに一致する．

$$\boldsymbol{w}^* \quad = \quad \arg\max_{\boldsymbol{w}} \sum_i \log p(y_i|x_i;\boldsymbol{w}) - C|\boldsymbol{w}|_1. \quad (4)$$

但し，$|\boldsymbol{w}|_1 = |w_1| + |w_2| + \ldots, |w_m|$ は $\boldsymbol{w}$ の L_1 ノルム，$C \in R$ はトレードオフパラメータである．C が大きい場合は，正則化が重視され，C が小さい場合は学習データが重視される．この L_1 正則化は重みベクトルの推定に，Laplace 分布を事前分布として仮定した場合の事後確率最大化を利用した場合と考えられる．

L_1 正則化を適用した学習結果の特徴として，多くの重みが 0 になることが知られている．例えば Gao [5] らは，多くの自然言語処理タスクにおいて L_1 正則化による学習結果は，多く用いられている L_2 正則化 ($|\boldsymbol{w}|_2 = \boldsymbol{w}^T\boldsymbol{w}$) と比較し，ほぼ同等の精度を達成しながら，有効な素性数（重みが 0 ではなかったもの）は，L_2 の 1/10 から 1/20 であったと報告している．

L_1-LR の学習は，$|\boldsymbol{w}|_1$ が $w_i = 0$ の時に微分不可能なため，一般の L-BFGS などの勾配法を適用できない．本稿では OWL-QN [6, 7] を適用し学習する．

さらに，L_1 正則化の特徴を生かし，次の Grafting [8] を利用し，学習を効率化する（Algorithm 1）．Grafting では，初めに有効な素性候補を管理する H を空集合に，重みベクトル $\boldsymbol{w}$ を 0 に初期化する．次に，対数尤度項の $\boldsymbol{w}$ に関する微分を $\boldsymbol{v} \in R^m$ とする．

$$\boldsymbol{v} \quad = \quad \sum_{i,y} (I(y = y_i) - P(y|x_i;\boldsymbol{w}))\, \phi(x_i, y),$$

但し $I(a)$ は a が真の時 1，偽の時 0 となる関数である．このとき最も絶対値が大きい v_k を H に追加し H に含まれる素性に関してのみ最適化を行なう．これを繰り返し，全ての素性に関し $v_k < C$ となった時に終了する．この時，得られた重みベクトルは最適解に一致する．

この Grafting は，素性の候補数が非常に大きい場合でも $\arg\max_k |v_k|$ さえ，効率的に求めることができれば，有効な素性数のみ依存した計算量で最適化が可能である．

Algorithm 1 L_1-LR の Grafting による学習

入力: 訓練データ (x_i, y_i) $(i = 1, \ldots, n)$，トレードオフパラメータ C
$H = \{\}, \boldsymbol{w} = \boldsymbol{0}$
loop
  $\boldsymbol{v} = \frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}$
  $k^* = \arg\max_k |v_k|$
  ($\boldsymbol{v}$ は対数尤度項の勾配)
  if $|\boldsymbol{v}_{k^*}| < C$ **then**
    break
  end if
  $H = H \cup k^*$
  H に含まれる素性についてのみ，$\boldsymbol{w}$ を最適化
end loop
$\boldsymbol{w}$ と H を出力

3 拡張接尾辞配列

本稿では，有効な部分文字列を効率的に求めるために，拡張接尾辞配列を用いる．以下に簡単に説明するが詳しくは [9] を参照されたい．

入力文字列が $T[1, \ldots, s], T[i] \in \Sigma$ で与えられ，末尾は便宜上，入力中には出現しない文字で辞書式順序で一番小さい文字であるとする（$T[s] = \$$）．

この時，$S_i = T[i, \ldots, s]$ $(i = 1, \ldots, s)$ を T の接尾辞と定義する．これら全ての接尾辞を辞書式順序の小さい順に並べた時の添え字を並べた配列 $SA[1, \ldots, s]$ を T の接尾辞配列 (Suffix Arrays) と呼ぶ（$S_{SA[i]} < S_{SA[i+1]}$）．例を図 1（列 SA) に挙げる．

この接尾辞配列は入力長が s の時，$s \lg s$bit [2] で保存することができ，O(s) 時間で構築できる [10].

この接尾辞配列を用いて，任意の部分文字列 $q[1, \ldots, t]$ が何回出現したか，どこに出現したかは SA 上の二分探索を二回行うことで，それぞれ O($t \log s$)，O($t \log s + occ(q)$) 時間で求めることができる，但し $occ(q)$ は q の出現回数である．例えば，$q = abra$ の出現位置については，SA 中の $i \in [3, 4]$ が $abra$ が出現しており，これらの出現場所は $SA[3] = 8, SA[4] = 1$ と分かる．

接尾辞木は全ての接尾辞から構成された Trie データ構造であり，枝分かれが無い枝は全て一つの枝に縮退されたデータ構造である．これにより，内部節点数は多くても $s - 1$ 個で抑えられる．接尾辞木を用いることで最長一致列を線形時間で求められるなど，接尾辞配列のみでは解けない様々な文字列問題が効率的に処理可能であることが知られている [11].

しかし，接尾辞木を利用した場合，その作業領域量は少なくとも 20NByte であることが知られており，現実的ではない．拡張接尾辞配列 [9](ESA) は接尾辞木がサポートする多くの操作を同様の計算量でサポートする．これは接尾辞配列に加え，次に述べる最長共通接頭辞配列 (LCP) を組み合わせたものであり，作業領域量は $9N$bit と少なくて済む．

入力 T とその接尾辞配列 SA が与えられた時，最長共通接頭辞配列 $LCP[1, s]$ は $LCP[i] = T_{SA[i]}$ と $T_{SA[i+1]}$ の共通な接頭辞の長さと定義する，ただし $LCP[s] = 0$ である．この配列は $s \lg s$ bits で保存でき，O(s) 時間で構築可能である [12]．例を図 1(列 L) に挙げる．

また，Burrows Wheeler's 変換（BWT) は次のように定義される．入力 T とその接尾辞配列 SA が与えられた時，BWT 後の文字列 B は $B[i] = T[SA[i] - 1]$，但し $SA[i] = 1$ の時 $B[i] = T[s]$ である．例を図 1（列 B) に挙げる．

[2] lg は底が 2 の log．一般に $s < 2^{32}$ の時，$s \lg s$ bit = $4s$ byte

i	SA	L	B	suffix
1	12	0	a	$
2	11	1	r	a$
3	8	4	d	abra$
4	1	1	$	abracadabra$
5	4	1	r	acadabra$
6	6	0	c	adabra$
7	9	3	a	bra$
8	2	0	a	bracadabra$
9	5	0	a	cadabra$
10	7	0	a	dabra$
11	10	2	b	ra$
12	3	0	b	racadabra$

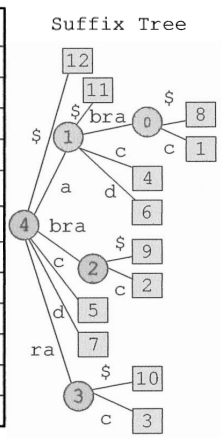

図 1: 入力 $T = abracadabra$ に対する接尾辞配列（列 SA), 最長共通接頭辞（列 L), Burrows-Wheeler 変換列 (列 B), とそれらに対応する接尾辞 (列 suffix) を示す. 右にそれに対応する接尾辞木を示す.

4 極大部分文字列

本章では，文書を，その中に出現する全ての部分文字列を単語とみなした，BOW によって表現する．

文書 d の長さを $|d|$ で表す時，d 中に出現する部分文字列種類数は最大 $|d|(|d|-1)/2$ であり，文書長の二乗に比例する．これを直接，素性ベクトルとして表現し学習，推定することは計算量的に困難であり，部分文字列長の長さを打ち切る，ヒューリスティクスなどにより採用する部分文字列を制限することが考えられる．

本章では，極大部分文字列とそれを拡張接尾辞配列を用いて効率的に列挙することにより全ての有効な部分文字列を効率的に求められることを示す．

初めに極大部分文字列を定義する．入力 T と，部分文字列 q が与えられた時，q の T 中の出現回数を $occ(q)$，q の T 中の出現場所の集合を $P(q) = \{p_1, p_2, \ldots, p_{occ(q)}\}$ と定義する．また，$P(q)$ と $c \in R$ が与えられた時，$P(q) - c = \{p_1 - c, p_2 - c, \ldots, p_{occ(q)} - c\}$ と定義する．

二つの部分文字列 q_1, q_2 が，$q_1 = \alpha q_2 \beta$ であり (α, β は空文字を含む部分文字列)，$P(q_1) - |\alpha| = P(q_2)$ を満たす時，$q_1 =_P q_2$ と定義する．直感的には，q_1 と q_2 の T 中の出現位置は全て同じであることを意味する．例えば，$T = abracadabra$ の場合，$ab =_P abr$，$bra =_P abra$ だが，$a \neq_P ab$ である．この時，T 中に出現する全ての部分文字列は $=_P$ の関係により，いくつかの集合に分割される．この各集合に属する部分文字列の中で，一番長い文字列[3]を極大部分文字列と呼ぶ．これらのうち出現回数が二回以上の極大部分文字列の集合を M_T で表すとする．図 1 の例の場合，$M_T = \{a, abra\}$ である．

[3]各集合の極大部分文字列は必ず一意に決まる

訓練データ (x_i, y_i) に対し，それら入力を元の入力には出現しない特殊文字 $\$_i$ を間に挟んでつなげた配列 $x_1\$_1x_2\$_2 \ldots x_n\$_n$ を T と置く．

この時，T の二つの部分文字列 q_1, q_2 が，$q_1 =_P q_2$ を満たすならば，q_1 と q_2 の各文書での出現位置は全く同じであり，同様に出現回数も同じである．よって，BOW 表現の場合，$=_P$ の関係を持つ部分文字列の集合は全て同じ統計量を持ち極大部分文字列を代表とする各集合のみになる．よって，極大部分文字列のみを部分文字列の素性として最適化すればよいことになる．

この極大部分文字列は前章述べた拡張接尾辞配列を用いることで，効率的に列挙できる．まず，極大部分文字列は必ず，接尾辞木における内部節点または，葉に対応する部分文字列である．特に，出現回数が 2 回以上の極大部分文字列は必ず，内部節点に対応する．しかし全ての内部節点が極大部分文字列であるとは限らない．例えば，図 1 では，ra は内部節点だが極大部分文字列ではなく，$abra$ がその集合に対する極大部分文字列である．極大部分文字列の必要十分条件は，その部分文字列が内部節点に対応し，かつ，その節点に対応する BWT 配列が二種類以上の異なる文字を持つ場合である．

接尾辞木中の内部節点の個数は最大でも $s-1$ 個であり，これより，極大部分文字列を代表とする集合の数は多くても $s-1$ 個である．

拡張接尾辞配列を利用することで，接尾辞木の全ての内部節点は線形時間で列挙することができる [12] (図 2)．また，その中の BWT が全て一致するかは BWT に関する rank 辞書 [13] を構築することで定数時間でチェックできる．

この学習に必要な作業領域量は $|T| < 2^{32}$ としたとき，$10|T| + \mathrm{O}(n)$bits である．

5 極大部分文字列を用いた学習

前章では，極大部分文字列の線形時間の列挙方法を示した．これにより明示的に BOW 表現による素性ベクトルを作ることもできるが，本章では，訓練時に明示的に素性ベクトルを表現しない方法を説明する．

前章で述べたように Grafting（図 1）では，勾配の絶対値が最大となる素性（$k^* = \arg\max_k v_k$）さえ求まれば学習ができる．

本章では補助データ構造を利用することにより多くの種類の素性値の場合で，この k^* が効率よく求められることを示す．

はじめに，各極大部分文字列は，(l, r, d) の三つ組で表現する，この三変数の意味は，この極大部分もい z 列が，$SA[l, \ldots, r]$ 中に出現し，長さが d であるという意味である．次にこの極大部分文字列を素性として用いた場合の対数尤度の勾配を $culcGrad(l, r, d)$ と表すことに

Algorithm 2 ESA を用いた勾配が最大となる素性の求め方

```
入力： L[0, n], SA[0, n], B[0, n]
S: (pos: SA 中の開始位置，len: 部分文字列長) からなるスタック
v*_k = 0 : 勾配の絶対値の最大値
for i = 0 to n + 1 do
  cur = (i, L[i])
  cand = top(S)
  while cand.len > cur.len do
    if B[cand.pos, ..., i] が 2 種類以上の文字を含む then
      v_k = calcGrad(cand.pos, i, cand.len)
      // 素性値の勾配を求める．5 章参照
      v*_k = max(v*_k, |v_k|)
    end if
  end while
  if cand.len < cur.len then
    push(S, v)
  end if
  push(S,(i, n-SA[i] + 1))
end for
```

する．これは素性を明示的に表した場合は式 (5) で求められる．

はじめに素性値として，tf を用いた場合を考える．$D[i]$ を $SA[i]$ が含まれる文書番号とし，$P_{cum}[0,\ldots,s]$ を $P_{cum}[i] = \sum_{j=0}^{j-1} P(y_{D[j]}|x_{D[j]}; \boldsymbol{w})$ と定義する．また，$\mathbf{rank}(y', j)$ を $y_1, \ldots, y_j$ 中に y' が出現した個数とする．このとき，素性 k が極大部分文字列 (l, r, d) でラベル y に対応するものであれば，その勾配値 culcGrad(l, r, d) は，

$$\mathbf{rank}(y, r) - \mathbf{rank}(y, l) - (P_{cum}[r] - P_{cum}[l])$$

として求められる．この計算は O(s)bits の作業領域で定数時間で実現できる．これは極大部分文字列が SA 中では連続した領域に出現するため，式 (5) をその領域で表引きしたものと考えられる．

次に，素性値として idf や binary を用いた場合だが，これは補助データ構造 [14] を利用することで同様に定数時間，O(s)bits の作業領域量で求められるが，スペースの関係上詳細は省略する．実用的には，tf と同様に，SA を一定長のブロックに分割し，その範囲内の答えを前もって表で求めて保持しておき，範囲外のものは逐次計算することで効率的に求められる．

さらに，素性値として部分文字列の長さを利用する場合は，勾配に d または $\log(d)$ を乗算すること得られる．

本章では，前章で述べた拡張接尾辞配列 (ESA) を用いて，どのように L_1 正則化付 Logistic 回帰 (L_1-LR) で有効な部分文字列を効率的に列挙するかを述べる．

次にこの高速化について考える．実装では，素性を上位 1 個だけではなく上位 k 個とりだしてきて，それらをまとめて最適化問題に与える．冗長な素性が加わっても最適化が遅くなるだけで，最適解に達することに注意されたい．次に毎回の更新で $P(y_i|x_i; \boldsymbol{w})$ の値が変わるのは H が加わった素性が含まれる文書のみであり，それらのみを更新するようにする．

最後に，形態素解析から求められた単語境界と抽出される部分文字列の境界は一致させる制約をどのように実現するかを述べる．これは，入力 T の代わりに，単語境界が存在する場所に特殊文字 # を挿入した配列を考え，これに対する拡張接尾辞配列を同じように構築する．この時，候補となる極大部分文字列を列挙する際には # が先頭にある接尾辞だけを対象にすればよい．

そして，対応する極大部分文字列が # が末尾に無い場合は，親との枝の途中にある場合は，その部分を代表とする．計算量的には，入力長が最大でも 2 倍になるだけであり線形時間は保障される．また，極大部分文字列抽出の際も # が先頭に接尾辞配列を考慮するので，全体の計算量は変わらない．

既に分かち書きされている言語の場合でも，同様の方法で，文字 N-gram ではなく単語 N-gram を抽出ように制約することができる．

なお，学習結果を用いて、推定する場合，文字列カーネルの推定同様 [3]，学習結果で得られた部分文字列集合を Trie で管理し，各節点で重みを管理する．そして，テストデータを順に操作しながら，重みを足し合わせることでテスト文書長の線形時間で推定可能である．全ての訓練データを保持しなければならない，文字列カーネルの場合に比べ最終的に得られる部分文字列数は遥かに少ないためキャッシュ効率も良く高速に推定することが可能である．

6 実験

提案手法の有効性を確かめるために，Movie, TechTC-300, の文書分類タスクを各手法で行なった結果を比較した．Movie は映画のレビューから抽出された文に対し，それが好意的意見か否かを判定するいわゆる感情極性分類である[4]．TechTC-300 は Open Directory Project で与えられたカテゴリのうち様々な難易度の 300 からなる二値分類タスクかなり，本実験ではそれらのうち SVM による精度が 7 割前後のタスク 2 つを採用した[5]．

各データセット，サイズ，単語種類数，極大部分文字列種類数をまとめたものを表 1 に示す

評価は 10 分割交差検定を用いて正解精度を利用した．表中の単語種類数，極大部分文字列種類数はその中の一つの訓練例の場合の例である．ハイパーパラメータは，

[4] http://www.cs.cornell.edu/people/pabo/ movie-review-data, Polarity dataset v2.0

[5] a: 10341-14271, b: 10539-194915

表 1: 各データセットの詳細

データ	訓練例数	総データ長 (BYTE)	単語種類数	極大部分文字列数
MOVIE	2000	7786004	38187	1685037
TC300A	200	1953894	16655	378673
TC300B	200	1424566	14430	236220

表 2: 各手法の結果

データ	提案手法	BOW+L_1	SLR	SVM
MOVIE	86.5%	83.0%	81.6%	87.2%
TC300A	80.0%	66.7%	80.0%	73.1%
TC300B	86.7%	86.7%	73.3%	71.9%

テストデータに対する正解率が最も高いものを選んでいる．評価には，正解率を用いた．

比較対象として提案手法の他に，分かち書きされた単語による BOW 表現を L_1-LR で学習したもの (BOW+L_1)，最近提案された [4] 部分文字列を素性として利用した Logistic 回帰モデルによる文書分類手法（SLR)，及び同じ BOW 表現による SVM による学習である．カーネルは多項式カーネルの次数で最もよいものを選んだ．

また素性値として，bin, tf, idf, len と，それらの全ての組み合わせを試し，BOW では tf, 提案手法では idf が最も精度が高かったため[6]，以下ではそれらの素性を用いた時の結果を示す．

表 2 に精度結果を示す．提案手法はいずれのデータセットにおいても最も精度が高い，またはそれに近い結果であった．これに対し，例えば，SVM は高い精度を出すものがある一方，TC300B など難しいデータセットがあることがみられる．これは部分文字列のノイズが大きいものと思われる．L_1 正則化により有効でない部分文字列を除いて学習可能な提案手法と (BOW+L_1) が高い精度を達成できたと思われる．また，SLR の精度はいずれの場合も提案手法と同じか低かった．SLR では部分文字列素性を Greedy に探索し選択するために，全ての部分文字列集合を候補として扱えない．そのため有効な部分文字列を抽出できなかったためとおもわれる．それに対し，提案手法は全ての部分文字列候補から選択するために，有効な部分文字列を抽出できたと思われる．

最後に提案手法のスケーラビリティを示すために，様々なサイズの入力テキストが与えられた時，その極大部分文字列を列挙するまでの時間を図 2 に示す（SA, LCP 構築を含む）．実行環境は CPU は 3GHz Xeon，主記憶は 32GB，g++のバージョンは 4.03 で-O3 オプションを用いた．入力文は日本語 Wikipedia である．入力サイズに対し，提案手法が線形時間で処理できていることが分かる．

7 まとめ

本稿では，全ての部分文字列を素性として利用した文書分類モデルを提案し，極大部分文字列を扱えば訓練データ長に比例する時間で有効な部分文字列を漏れなく抽出できることを示した．実験結果から提案手法は非常に少ない素性数で高い精度を達成できることを示せた．

今後は，提案手法の更なる性能調査及び，教師無し学習（クラスタリング）などにも同様の手法が使えるかを考えていく予定である．

[6]提案手法では，idf, len, tf-idf, idf-len, tf-len, tf-idf-len に差は殆どみられなかった

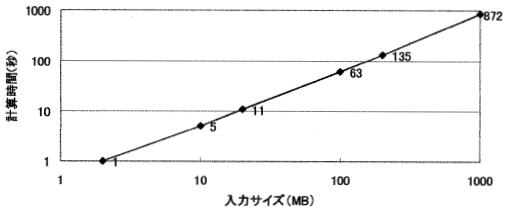

図 2: 入力サイズ，極大部分文字列の列挙時間

参考文献

[1] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[2] S. V. N. Vishwanathan and Alexander J. Smola. Fast kernels for string and tree matching. In *NIPS 15*, pages 569–576. MIT Press, 2003.

[3] Choon Hui Teo and S. V. N. Vishwanathan. Fast and space efficient string kernels using suffix arrays. In *ICML*, pages 929–936, 2006.

[4] G. Ifrim, G. Bakir, and G. Weikum. Fast logistic regression for text categorization with variable-length n-grams. In *SIGKDD*, 2008.

[5] J. Gao, G. Andrew, M. Johnson, and K. Toutanova. A comparative study of parameter estimation methods for statistical natural language processing. In *Proc. of ACL*, pages 824–831, 2007.

[6] G. Andrew and J. Gao. Scalable training of l1-regularized log-linear models. In *Proc. of ICML*, 2007.

[7] J. Yu, S. V. N. Vishwanathan, S. Guenter, and Nicol Schraudolph. A quasi-newton approach to nonsmooth convex optimization. In *ICML*, 2008.

[8] S. Perkins, K. Lacker, and J. Theiler. Grafting: fast, incremental feature selection by gradient descent in function space. *JMLR*, 3:1333–1356, 2003.

[9] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *J. Discrete Algs*, 2:53–86, 2004.

[10] G. Nong, S. Zhang, and W. H. Chan. Two efficient aglrotihms for linear suffix array constraction, 2008.

[11] D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.

[12] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *CPM*, pages 181–192, 2001.

[13] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1), 2007.

[14] K. Sadakane. Succinct data structures for flexible text retrieval systems. *Journal of Discrete Algorithms*, 5(1):12–22, 2007.