F-037

Sparse Substring Pattern Set Discovery using Linear Programming Boosting

Kazuaki Kashihara [1] Kohei Hatano[1] Hideo Bannai[1] Masayuki Takeda[1]

[1] Department of Informatics, Kyushu University

Abstract: In this paper, we consider finding a small set of substring patterns which classifies the given documents well. We formulate the problem as 1 norm soft margin optimization problem where each dimension corresponds to a substring pattern. Then we solve this problem by using LPBoost and an optimal substring discovery algorithm. Since the problem is a linear program, the resulting solution is likely to be sparse, which is useful for feature selection. We evaluate the proposed method for real data such as movie reviews.

1 Introduction

Text classification is an important problem in broad areas such as natural language processing, bioinformatics, information retrieval, recommendation tasks. Machine Learning has been applied to text classification tasks in various ways: SVMs and string kernels (n-gram kernels, subsequence kernels [14], mismatch kernels [13]) Boosting (e.g., Boostexter [20]).

In some applications regarding texts, not only classification accuracy but also what makes classification accurate is important. In other words, one might want to discover some knowledge from an accurate text classifier as well. For example, in classification task of biosequences, say, DNA or RNA, biologists want to know patterns in the data which make each sequence positive other than an accurate classifier. Simply put, one may want an accurate classifier associated with a set of patterns in the text. In particular, for the purpose of feature selection, it is desirable that such a set of patterns is small.

In this paper, we formulate the problem of finding a small set of patterns which induces an accurate classifier as 1-norm soft margin optimization over patterns. Roughly speaking, this problem is finding a linear combination of classifiers associated with patterns (or a hyperplane whose each component corresponds to a pattern) which maximizes the margin w.r.t. the given labeled texts as well as minimizing misclassification.

Our formulation has two advantages. The first advantage is accuracy of the resulting classifier. The large margin theory guarantees that linear classifier with large margin is likely to have low generalization error with high probability [19]. So, by choosing the class of patterns appropriately, solving the problem would provide us an enough accurate classifier. The second advantage is that the resulting solution is often sparse since the 1-norm soft margin optimization is a linear program. In other words, many of patterns have zero weights in the obtained linear combination. This would help us to choose a small subset of patterns from the resulting classifier.

We solve the 1-norm soft margin optimization over patterns by combining LPBoost [4] and our pattern discovery algorithm. LPBoost is a boosting algorithm which provably solves the 1-norm soft margin optimization. Given a weak learning algorithm which outputs a "weak hypothesis", LPBoost iteratively calls the weak learning algorithm w.r.t. different distributions over training texts and get different weak hypotheses. Then it combines weak hypotheses as a linear combination of them as the final classifier. In this work, we use our pattern discovery algorithm as the weak learning algorithm for LPBoost.

The pattern class we consider in this paper is that of all the possible substrings over some alphabet Σ. For substring patterns, we derive an efficient pattern discovery algorithm. A naive algorithm enumerates all the possible substrings appearing in the input texts and takes $O(N^2)$ time, where N is the length of total texts. On the other hand, ours runs in time $O(N)$. Our approach can be further extended by employing pattern discovery algorithms for other rich classes such as subsequence patterns [6] or VLDC patterns [8], which is our future work (See Shinohara's survey [21] for pattern discovery algorithms).

In our preliminary experiments, we apply our method for classification of movie reviews. In particular, for our data Movie-A, there are about 6×10^{13} possible substrings patterns. Our method outputs a classifier associated with a small set of substrings whose size is only about 800. Among such 800 patterns, we find interesting pattern candidates which explain positive and negative reviews.

Let us review some related researches. The bag of words model (BOW) has been popular in information retrieval and natural language processing. In this model, each text is regarded as a set of words appearing in the text, or equivalently, a weight vector where each component associates with a word and the value of each component is determined by the statistics of the word (say, frequency of the word in the text). The BOW model is often effective in classification of natural documents. However, we need to determine a possible set of words in advance, which is a nontrivial task. SVMs with string kernels (e.g., [23, 22]) often provide us a state-of-the-art classification for texts. However, the solutions of kernelized SVMs do not have explicit forms of patterns.

Among related researches, the work of Okanohara and Tsujii [16] would be most related to ours. They consider a similar problem over substring patterns, they deal with logistic regression with 1-norm regularization. As we will show later, in our experiments, our method gains higher accuracy than they reported. Other related researches include the work of Saigo et al [18]. They consider 1-norm soft margin optimization over graph patterns and use LPBoost. Our framework is close to theirs, but we use different techniques for pattern discovery of substrings.

2 Preliminaries

2.1 1-norm soft margin optimization

Let $\mathcal{X}$ be the set of instances. We are given a set S of labeled instances $S = ((x_1,y_1),\ldots,(x_m,y_m))$, where each instance x belongs to $\mathcal{X}$ and each label y_i is -1 or $+1$, and a set $\mathcal{H}$ of n hypotheses, i.e., a set of functions from $\mathcal{X}$ to $[-1,+1]$. The final classifier is a linear combination of hypotheses in $\mathcal{H}$, $\sum_{h\in\mathcal{H}}\alpha_h h + b$, where b is a constant called bias. Given an instance x, the prediction is $\mathrm{sign}(\sum_{h\in\mathcal{H}}\alpha_h h(x)+b)$, where $\mathrm{sign}(a)$ is $+1$ if $a>0$ and -1, otherwise. Let $\mathcal{P}^k$ be the probability simplex, i.e., $\mathcal{P}^k = \{\boldsymbol{p}\in[0,1]^k, \sum_{i=1}^k p_i = 1\}$. For a weighting $\alpha\in\mathcal{P}^n$ over hypotheses in $\mathcal{H}$ and a bias b, its margin w.r.t. a labeled instance (x,y) is defined as $y(\sum_{h\in\mathcal{H}}\alpha_h h(x)+b)$. If the margin of α w.r.t. a labeled instance is positive, the prediction is correct, that is, $y=\mathrm{sign}(\sum_{h\in\mathcal{H}}\alpha_h h(x)+b)$.

The edge of a hypothesis $h\in\mathcal{H}$ for a distribution $\boldsymbol{d}\in\mathcal{P}^m$ over S is defined as

$$\mathbf{Edge}_{\boldsymbol{d}}(h) = \sum_{i=1}^{m} y_i d_i h(x_i).$$

The edge of h can be viewed as accuracy w.r.t. the distribution $\boldsymbol{d}$. In fact, if the output of h is binary-valued ($+1$ or -1), $\mathbf{Edge}_{\boldsymbol{d}}(h) = 1-2\mathrm{Error}_{\boldsymbol{d}}(h)$, where $\mathrm{Error}_{\boldsymbol{d}}(h) is$ $\sum_i \boldsymbol{d}_i I(h(x_i)=y_i)$, where $I(\cdot)$ is the indicator function such that $I(true)=1$ and $I(false)=0$.

The 1 norm soft margin optimization problem is formulated as follows (see, e.g., [4, 24]):

$$\max_{\rho,\alpha,\xi,b} \rho - \frac{1}{\nu}\sum_{i=1}^{m}\xi_i \quad (1)$$

$$\text{sub.to}$$

$$y_i\left(\sum_j \alpha_j h_j(x_i)+b\right) \geq \rho-\xi_i \; (i=1,\ldots,m),$$

$$\alpha\in\mathcal{P}^n,\ \xi\geq\mathbf{0}.$$

That is, the problem is to find a weighting α over hypotheses and a bias b which maximize the margin among given labeled instances as well as minimizing the sum of quantities (losses) by which the weighting misclassifies. Here, the parameter ν takes values in $\{1,\ldots,m\}$ and it is fixed in advance. This parameter controls the tradeoff between maximization of the margin and minimization of losses.

By using Lagrangian duality (see, e.g., [3]), we can derive the dual problem as follows.

$$\min_{\gamma,\boldsymbol{d}} \gamma \quad (2)$$

$$\text{sub.to}$$

$$\mathbf{Edge}_{\boldsymbol{d}}(h_j) = \sum_i d_i y_i h_j(x_i) \leq \gamma \; (j=1,\ldots,n),$$

$$\boldsymbol{d}\leq\frac{1}{\nu}\mathbf{1},\ \boldsymbol{d}\in\mathcal{P}^m,$$

$$\boldsymbol{d}\cdot\boldsymbol{y}=0.$$

The dual problem is to find a distribution over instances for which the edges of hypotheses are minimized. In other words, the problem is to find the most difficult distribution for the hypotheses in $\mathcal{H}$.

It is well known that if the primal and dual problems are linear programs, they are equivalent to each other, i.e., if one solves one problem, one have also solved the other and vice versa. More precisely, let $(\rho^*,\alpha^*,\xi^*,b^*)$ be an optimizer of the primal problem (1) and let $(\gamma^*,\boldsymbol{d}^*)$ be an optimizer of the dual problem (2), respectively. Then, by the duality of the linear program, $\rho^* - \frac{1}{\nu}\sum_{i=1}^m \xi_i^* = \gamma^*$. KKT conditions (see, e.g., [3]) implies that an optimal solution has the following property.

- If $y_i\left(\sum_j \alpha_j^* h_j(x_i) + b^*\right) > \rho^*$, then $d_i^* = 0$.
- If $0 < d_i^* < 1/\nu$, then $y_i(\sum_j \alpha_j^* h_j(x_i) + b^*) = \rho^*$.
- If $\xi_i^* > 0$, then $d_i^* = 1/\nu$.

That is, only such a labeled instance (x_i, y_i) that have margin no larger than ρ^* can have a positive weight $\boldsymbol{d}_i^* > 0$. Further, note that the number of inseparable examples (for which $\xi_i^* > 0$) is at most ν. This property shows the sparsity of a dual solution. The primal solution has sparsity as well:

- If $\mathbf{Edge}_{\boldsymbol{d}^*}(h_j) < \gamma^*$, $\alpha_j^* = 0$.

Similarly, only such a hypothesis h_j that $\mathbf{Edge}_{\boldsymbol{d}^*}(h_j) = \gamma^*$ can have a positive coefficient $\alpha_j^* > 0$.

2.2 LPBoost

We review LPBoost [4] for solving the problem (2). Roughly speaking, LPBoost iteratively solves restricted dual problems to obtain a final solution.

The details of LPBoost is given in Algorithm 1. Given the initial distribution $\boldsymbol{d}_1$, LPBoost works in iterations. At each iteration t, LPBoost chooses a hypothesis h_t maximizing the edge w.r.t. $\boldsymbol{d}_t$, and add a new constraint $\mathbf{Edge}_{\boldsymbol{d}}$ $(h_t) \le \gamma$. problem and solve the linear program and get $\boldsymbol{d}_{t+1}$ and γ_{t+1}.

In fact, given a precision parameter $\varepsilon > 0$, LPBoost outputs an ε-approximation.

Theorem 1 (Demiriz et al. [4]). *LPBoost outputs a solution whose objective is an ε-approximation of an optimum.*

2.3 Strings

Let Σ be a finite *alphabet* of size σ. An element of Σ^* is called a *string*. Strings x, y and z are said to be a *prefix*, *substring*, and *suffix* of the string $u = xyz$. The length of any string u is denoted by $|u|$. Let ε denote the empty string, that is, $|\varepsilon| = 0$. Let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. The i-th character of a string u is denoted by $u[i]$ for $1 \le i \le |u|$, and the substring of u that begins at position i and ends at position j is denoted by $u[i:j]$ for $1 \le i \le j \le |u|$. For a set of strings S, let $\|S\| = \sum_{s \in S} |s|$.

Algorithm 1 LPBoost(S,ε)

```
1. Let $\boldsymbol{d}_1$ be the distribution over $S$ such that $\boldsymbol{d}_1 \cdot \boldsymbol{y} = 0$
   and $\boldsymbol{d}_1$ is uniform w.r.t. positive or negative instances only. Let $\gamma_1 = -1$.
2. For $t = 1, \ldots,$
   (a) Let $h_t = \arg\max_{h \in \mathcal{H}} \mathbf{Edge}_{\boldsymbol{d}_t}(h)$.
   (b) If $\mathbf{Edge}_{\boldsymbol{d}_t}(h_t) \le \gamma_t + \varepsilon$, let $T = t - 1$ and break.
   (c) Otherwise, solve the soft margin optimization problem (2) w.r.t. the restricted hypothesis set
       $\{h_1, \ldots, h_t\}$. That is,
           $(\gamma_{t+1}, \boldsymbol{d}_{t+1}) = \arg\min_{\gamma, \boldsymbol{d} \in \mathcal{P}^m} \gamma$
           sub. to
           $\gamma_{\boldsymbol{d}}(h_j) \le \gamma \ (j = 1, \ldots, t)$
           $\boldsymbol{d} \le \frac{1}{\nu}\mathbf{1}, \boldsymbol{d} \cdot \boldsymbol{y} = 0.$
3. Output $f(x) = \sum_{t=1}^T \alpha_t h_t(x)$, where each $\alpha_t$ $(t = 1, \ldots, T)$ is a Lagrange dual
   of the soft margin optimization problem (2).
```

2.4 Our problem

We consider the 1 norm soft margin optimization problem for string data sets, where each hypothesis corresponds to a string pattern. That is, we are given a set of labeled documents (strings), and each substring $p \in \Sigma^*$ corresponds to a hypothesis $h_p \in \mathcal{H}$, and $h_p(\vec{x})$ for $x \in \Sigma^*$ is defined as follows:

$$h_p(\vec{x}) = \begin{cases} 1 & p \text{ is a substring of } \vec{x} \\ -1 & p \text{ is not a substring of } \vec{x} \end{cases}.$$

Thus, our "weak" learner will solve the following problem. Given a set of labeled strings $S = ((x_1, y_m), \ldots, (x_m, y_m)) \subset \Sigma^* \times \{-1, +1\}$, and a distribution $\boldsymbol{d} \in \mathcal{P}^m$ over S, find a string $p \in \Sigma^*$ such that

$$p = \arg\max_{q \in \Sigma^*} \mathbf{Edge}_{\boldsymbol{d}}(q) = \sum_{i=1}^m y_i d_i h_q(x_i) \tag{3}$$

To solve this problem optimally and efficiently, we make use of the suffix array data structure [15] as well as other related data structures described in the next section.

3 Algorithms

3.1 Data Structures

Below, we describe the data structures used in our algorithm.

The suffix tree of a string T is a compacted trie of all suffixes of T. For any node v in the suffix tree, let $path(v)$ denote the string corresponding to the path from the root to node v. We assume that the string ends with a unique character '\$' not appearing elsewhere in the string, thus ensuring that the tree contains $|T|$ leaves, each corresponding to a suffix of T. The suffix tree and generalized suffix tree are very useful data structures for algorithms that consider the substrings of a given string or set of strings. Each node v in the suffix tree corresponds to a substring of the input strings, and the leaves represent occurrences of the substring $path(v)$ in the string.

The generalized suffix tree for a set of strings $(T_1, \ldots, T_m)$, can be defined as the suffix tree for the string $T = T_1\$_1 \cdots T_m\$_m$, where each $\$_i$ $(1 \leq i \leq m)$ is a unique character not appearing elsewhere in the strings, and each edge is terminated with the first appearance of any $\$_i$. We assume that the leaves of the generalized suffix tree are labeled with the document index i.

It is well known that suffix trees can be constructed in linear time [25]. In practice, it is more efficient to use a data structure called suffix arrays which require less memory. The suffix array of string T is a permutation of all suffixes of T so that the suffixes are lexicographically sorted. More precisely, the suffix array of T is an array $SA[1, \ldots, |T|]$ containing a permutation of $\{1, \ldots |T|\}$, such that $T[SA[i] : |T|] \preceq T[SA[i+1] : |T|]$, for all $1 \leq i < s$, where $\preceq$ denotes the lexicographic ordering on strings. It is well known that the suffix array for a given string can be built in time linear of its length [9, 11, 12]. Another important array often used together with the suffix array is the height array. Let $LCP[i] = lcp(T[SA[i] : |T|], T[SA[i+1 : |T|]])$ be the *height array* $LCP[1, |T|]$ of T, where $lcp(T_{SA[i]}, T_{SA[i+1]})$ is the length of the longest common prefix between $T[SA[i: |T|]]$ and $T[SA[i+1 : |T|]]$. The height array can also be constructed in linear time [10]. Also, by using the suffix array and height arrays we can simulate a bottom-up post-order traversal on the suffix tree [10]. Most other algorithms on suffix trees can be efficiently implemented using the suffix and height arrays [1].

Figure 1 shows an example of a suffix array and suffix tree for the string BANANA.

T=BANANA\$

i	SA	LCP	suffix
1	7	-1	\$
2	6	0	A\$
3	4	1	ANA\$
4	2	3	ANANA\$
5	1	0	BANANA\$
6	5	0	NA\$
7	3	2	NANA\$

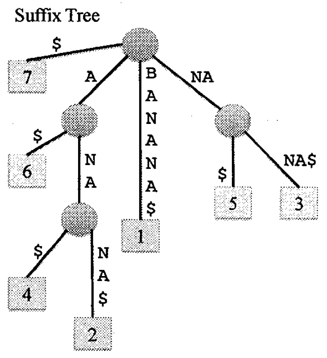


Figure 1: Suffix array (left) and suffix tree (right) for string T = BANANA\$. The column SA shows the suffix array, the column LCP shows the height array, the column 'suffix' shows the suffixes starting at position i.

3.2 Finding the Optimal Pattern

We briefly describe how we can find the substring $p \in \Sigma^*$ to maximize Equation (3) in linear time.

First, we note that it is sufficient to consider strings which correspond to nodes in the generalized suffix tree of the input strings. This is because for any string corresponding to a path that ends in the middle of an edge of the suffix tree, the string which corresponds to the path extended to the next node will occur in the same set of documents and, hence, its edge score would be the same. Figure 2 shows an example.

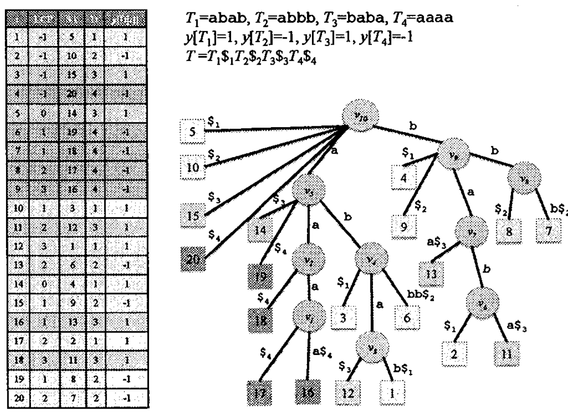
Also, notice that for any substring $p \in \Sigma^*$, we have

$$\begin{aligned}
\mathbf{Edge}_{\boldsymbol{d}}(p) &= \sum_{i=1}^{m} y_i d_i h_p(x_i) \\
&= \sum_{\{i:h_p(x_i)=1\}} y_i d_i - \sum_{\{i:h_p(x_i)=-1\}} y_i d_i \\
&= \sum_{\{i:h_p(x_i)=1\}} y_i d_i - \left(\sum_{i=1}^{m} y_i d_i - \sum_{\{i:h_p(x_i)=1\}} y_i d_i \right) \\
&= 2 \cdot \sum_{\{i:h_p(x_i)=1\}} y_i d_i - \sum_{i=1}^{m} y_i d_i.
\end{aligned}$$

Since $\sum_{i=1}^{m} y_i d_i$ can be easily computed, we need only to compute $\sum_{\{i:h_p(x_i)=1\}} y_i d_i$ for each p to compute its edge score.

This value can be computed for each string $path(v)$ corresponding to a node v in the generalized suffix tree, basically using the linear time algorithm for solving a generalized version of the color set size problem [7, 2]. When each document is assigned arbitrary numeric weights, the algorithm computes for each node v of the generalized suffix tree, the sum of weights of the documents that contain $path(v)$ as a substring. For our problem, we need only to assign the weight $y_i d_i$ to each document.

The main algorithm and optimal pattern discovery algorithms are summarized in Algorithm 2 and Algorithm 3. It is easy to see that FindOptimalSubstringPattern(...) runs in linear time: The algorithm of [2] runs in linear time. Also, since the number of nodes in a generalized suffix tree is linear in the total length of the strings, line 3 can also be computed in linear time.


T_1=abab, T_2=abbb, T_3=baba, T_4=aaaa
$y[T_1]$=1, $y[T_2]$=-1, $y[T_3]$=1, $y[T_4]$=-1
$T=T_1\$_1T_2\$_2T_3\$_3T_4\$_4$

i	LCP	SA	D	y[D]
1	-1	5	1	1
2	-1	10	2	-1
3	-1	15	3	1
4	-1	20	4	-1
5	0	14	3	1
6	1	19	4	-1
7	1	18	4	-1
8	2	17	4	-1
9	3	16	4	-1
10	1	3	1	1
11	2	12	3	1
12	3	1	1	1
13	2	6	2	-1
14	0	4	1	1
15	1	9	2	-1
16	1	13	3	1
17	2	2	1	1
18	3	11	3	1
19	1	8	2	-1
20	2	7	2	-1

Figure 2: Finding the substring that gives the maximum edge on four documents T_1, T_2, T_3, and T_4, with labels $\boldsymbol{y} = (1,-1,1,-1)$ and weights $\boldsymbol{d} = (0.3,0.1,0.2,0.4)$. The generalized suffix tree is depicted on the right, and corresponding suffix arrays and height arrays are depicted on the left. D holds the document index assigned to each leaf. For example, $\mathbf{Edge}_{\boldsymbol{d}}(path(v_1)) = 0.3*1*(-1)+0.1*(-1)*(-1)+0.2*1*(-1)+0.4*(-1)*1 = -0.8$. The optimal patterns are $path(v_3)$ = 'aba' and $path(v_6)$ = 'bab' giving an edge of 1.

Algorithm 2 Compute 1 norm soft margin optimal problem for string

```
1: Input: Data $S = ((T_1,y_1),\ldots,(T_m,y_m))$, parameter $\varepsilon$.
2: Construct suffix array SA and LCP array for string
   $T = T_1\$_1 \cdots T_m\$_m$.
3: Run Algorithm 1 (LPBoost($S$,$\varepsilon$)) using FindOptimal-
   SubstringPattern(SA, LCP, $\boldsymbol{y}$, $\boldsymbol{d}$) for line 2(a).
```

4 Experiments

We conducted sentiment classification experiments for a data set called MOVIE-A, provided by Bo Pang and Lillian Lee [17][1]. The data consists of reviews of various movies, with 1000 positive reviews, and 1000 negative reviews. Table 1 shows simple statistics of the data.

We examined the performance of our approach using 10 cross validations. More precisely, at each trial, we split the whole data randomly into training data and test data with probability 0.8 and 0.2, respectively. Then we train our method for the training data and measure the accuracy of the obtained classifier over the test data. We average the accuracy over 10 trials. The parameter ν for our method is set as $\nu = 0.1 \times m = 0.0001$, which, roughly speaking, means that we estimate the level of noise in the data as 10%.

Table 2 shows the results of our method, as well as several other methods. "SVM + Ngram" denotes the support vector machine using an ngram kernel, and "normalized SVM +Ngram" denotes a version which uses normalization (normalized SVM + Ngram). The score shown for these methods are for ngrams of length $n = 7$, which gave the best score. The score for "OT [16]" is the score taken directly from their paper.

Table 3 shows substrings with top 10 largest weights in the final weighting α. It also shows number of documents in which the pattern occurs. Our method found some interesting patterns, such as *est movie*, *or best*, *s very e*, and *s perfect*. Table 4 shows some of the context of the occurrence of these patterns.

Table 1: Detail of the data set

Corpus	# of docs	total length
MOVIE-A	2000	7786004

Table 2: Percentage of correct classifications in classification task.

Method	MOVIE-A
LPSSD	91.25%
SVM+Ngram	85.75%
normalized SVM+Ngram	89.25%
OT [16]	86.50%

[1]`http://www.cs.cornell.edu/People/pabo/movie-review-data/`, polarity dataset v2.0

Algorithm 3 FindOptimalSubstringPattern(SA, LCP, $\boldsymbol{y}$, $\boldsymbol{d}$)

```
1: $wtot := \sum y_i d_i$;
2: Calculate $w_v = \sum_{\{i: h_{path(v)}(x_i)=1\}} y_i d_i$ for each node $v$ of the generalized suffix tree, using SA, LCP and algorithm of [2];
3: $vmax := \arg\max_v \mathbf{Edge}_{\boldsymbol{d}}(path(v)) = \arg\max_v (2w_v - wtot)$;
4: return $path(vmax)$;
```

Table 3: Top 10 substrings with largest weight (α).

pattern	α	#occ in pos	#occ in neg
iase	0.01102	13	1
ronicle	0.00778	21	4
s very e	0.00776	15	2
ents r	0.00659	9	1
or best	0.00642	21	5
e of your s	0.00633	8	0
finest	0.00615	44	5
ennes	0.00575	28	8
un m	0.00567	13	1
s insid	0.00564	14	5

Table 4: Context of some substrings in top 100 largest weightings (α).

Pattern	est movie	s perfect
Context	b*est movie*	i*s perfect*
	funn*est movie*	thi*s perfect*ly
	great*est movie*	seem*s perfect*ly
Pattern	or best	s very e
Context	act*or best*	i*s very e*ffective
	aword f*or best*	i*s very e*ntertain
	nominated f*or best*	i*s very e*njoyable
Pattern	fun	o entertain
Context	*fun*	s*o entertain*ing
	*fun*ny	t*o entertain*
	*fun*niest	t*o entertain*ing
	*fun*nest	t*o entertain*ment
Pattern	much like t	s a fine
Context	*much like t*he	i*s a fine*
	*much like t*heir	deliver*s a fine*
	*much like t*is	doe*s a fine*
	*much like t*itanic	contribut*s a fine*

5 Conclusion and Future Work

We considered 1-norm soft margin optimization over substring patterns. We solve this problem by using a combination of LPBoost and an optimal substring pattern discovery algorithm. In our preliminary experiments on a data set concerning movie reviews, our method actually found some interesting pattern candidates. Also, the experimental results showed that our method achieves higher accuracy than other previous methods.

There is much room for improvements and future work. First, our method might become more scalable by employing faster solvers for 1-norm soft margin optimization, e.g., Sparse LPBoost [5]. Second, extending the pattern class to more richer ones such as VLDC patterns [8] would be interesting. Finally, applying our method to DNA or RNA data would be promising. An advantage of our method is that, unlike the bag of words models, we do not need to enumerate all possible candidates of patterns explicitly.

References

[1] Abouelhoda, M.I., Kurtz, S., Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays. Journal of Discrete Algorithms 2(1), 53–86 (2004)

[2] Bannai, H., Hyyrö, H., Shinohara, A., Takeda, M., Nakai, K., Miyano, S.: An $O(N^2)$ algorithm for discovering optimal Boolean pattern pairs. IEEE/ACM Transactions on Computational Biology and Bioinformatics 1(4), 159–170 (2004)

[3] Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)

[4] Demiriz, A., Bennett, K.P., Shawe-Taylor, J.: Linear programming boosting via column generation. Mach. Learn. 46(1-3), 225–254 (2002)

[5] Hatano, K., Takimoto, E.: Linear programming boosting by column and row generation. In: Proceedings of the 12th International Conference on Dicovery Science (DS 2009). pp. 401–408 (2009)

[6] Hirao, M., Hoshino, H., Shinohara, A., Takeda, M., Arikawa, S.: A practical algorithm to find the best subsequence patterns. Theoretical Computer Science 292(2), 465–479 (2003)

[7] Hui, L.: Color set size problem with applications to string matching. In: Proceedings of the Third Annual Symposium on Combinatorial Pattern Matching

(CPM 92). LNCS, vol. 644, pp. 230–243. Springer-Verlag (1992)

[8] Inenaga, S., Bannai, H., Shinohara, A., Takeda, M., Arikawa, S.: S.: Discovering best variable-length-don't-care patterns. In: In: Proceedings of the 5th International Conference on Discovery Science. Volume 2534 of LNAI., Springer-Verlag. pp. 86–97. Springer-Verlag (2002)

[9] Kärkkäinen, J., Sanders, P.: Simple linear work suffix array construction. In: Proc. ICALP'03. LNCS, vol. 2719, pp. 943–955 (2003)

[10] Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: CPM '01: Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching. pp. 181–192. Springer-Verlag, London, UK (2001)

[11] Kim, D.K., Sim, J.S., Park, H., Park, K.: Linear-time construction of suffix arrays. In: Proc. CPM'03. LNCS, vol. 2676, pp. 186–199 (2003)

[12] Ko, P., Aluru, S.: Space efficient linear time construction of suffix arrays. In: Proc. CPM'03. LNCS, vol. 2676, pp. 200–210 (2003)

[13] Leslie, C.S., Eskin, E., Weston, J., Noble, W.S.: Mismatch string kernels for svm protein classification. In: Advances in Neural Information Processing Systems 15 (NIPS '02). pp. 1417–1424 (2002)

[14] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.J.C.H.: Text classification using string kernels. Journal of Machine Learning Research 2, 419–444 (2002)

[15] Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Computing 22(5), 935–948 (1993)

[16] Okanohara, D., Tsujii, J.: Text categorization with all substring features. In: Proc. 9th SIAM International Conference on Data Mining (SDM). pp. 838–846 (2009)

[17] Pang, B., Lee, L.: A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In: Proceedings of the ACL (2004)

[18] Saigo, H., Nowozin, S., Kadowaki, T., Kudo, T., Tsuda, K.: gboost: a mathematical programming approach to graph classification and regression. Machine Learning 75(1), 69–89 (2009)

[19] Schapire, R.E., Freund, Y., Bartlett, P., Lee, W.S.: Boosting the margin: a new explanation for the effectiveness of voting methods. The Annals of Statistics 26(5), 1651–1686 (1998)

[20] Schapire, R.E., Singer, Y.: Boostexter: A boosting-based system for text categorization. Machine Learning 39, 135–168 (2000)

[21] Shinohara, A.: String pattern discovery. In: Proceedings of the 15th International Conference on Algorithmic Learning Theory (ALT'04). pp. 1–13 (2004)

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

[23] Vishwanathan, S.V.N., Smola, A.J.: Fast kernels for string and tree matching. In: NIPS. pp. 569–576 (2002)

[24] Warmuth, M.K., Glocer, K.A., Vishwanathan, S.V.: Entropy regularized lpboost. In: ALT '08: Proceedings of the 19th international conference on Algorithmic Learning Theory. pp. 256–271. Springer-Verlag, Berlin, Heidelberg (2008)

[25] Weiner, P.: Linear pattern-matching algorithms. In: Proc. of 14th IEEE Ann. Symp. on Switching and Automata Theory. pp. 1–11 (1973)