String Kernels Based on Variable-Length-Don't-Care Patterns

Kazuyuki Narisawa[1], Hideo Bannai[1], Kohei Hatano[1],
Shunsuke Inenaga[2], and Masayuki Takeda[1]

[1]Department of Informatics, Kyushu University
[2]Graduate School of Information Science and Electrical Engineering,
Kyushu University
744 Motooka, Nishiku, Fukuoka, 819–0395 Japan.
`{k-nari,bannai,hatano,takeda}@i.kyushu-u.ac.jp`
`inenaga@c.csce.kyushu-u.ac.jp`

Abstract. We propose a new string kernel based on *variable-length-don't-care patterns* (*VLDC patterns*). A VLDC pattern is an element of $(\Sigma \cup \{\star\})^*$, where Σ is an alphabet and $\star$ is the variable-length-don't-care symbol that matches any string in Σ^*. The number of VLDC patterns matching a given string s of length n is $O(2^{2n})$. We present an $O(n^5)$ algorithm for computing the kernel value. We also propose variations of the kernel which modify the relative weights of each pattern. We evaluate our kernels using a support vector machine to classify spam data.

1 Introduction

Text classification techniques in machine learning have a wide range of applications such as natural language processing and bioinformatics. These techniques are based on the similarity of words, sequences or other string units with a dictionary of natural language, biological actions and periodicities. It is a challenging task to efficiently compute these similarity measures, and the kernel method is one of the classical approaches for evaluating string similarity.

Several string kernels have been developed for various types of data. Mismatch kernels [1] are used for biological data such as DNA and protein sequences. This kernel allows mismatches between sequences. Rational kernels [2, 3] can separate regular languages, using weighted transducers or rational relations.

The string subsequence kernels (SSKs) and N-gram kernels (NGKs) [4] are popular string kernels. They are easy to compute, and can be applied to a variety of data since they do not make too many assumptions about the underlying text structure. SSKs map strings to a feature space where each dimension corresponds to a subsequence of length n. The value of the dimension depends on a subsequence gap weight and how the subsequence occurs in each string. NGKs map strings to a feature space where each dimension corresponds to a substring of length n, or n-gram. The value of the dimension depends on how many of each n-gram the string contains.

These kernels, however, have limitations. That is, these kernels are not suitable when the label of the texts depend not only on some relevant pattern (substring or subsequence) appearing in the texts, but on the order of such relevant patterns. For example, assume that labels of texts are positive if the word "*aab*" appears, followed by another word "*aba*". For such tests, NGKs cannot capture the order of strings in the texts. SSKs takes into account the orders of subsequences, but cannot capture the order of strings, either. In the case of SSKs, dissimilar strings may be judged similar due to the dimensions of subsequence. For example, strings "*xxxabcxxx*" and "*yayybyycy*" are clearly different. However, they may be judged similar because they have the same subsequence dimension "*abc*". Although the SSK resolves this issue by the gap weight, preferring that each character of the subsequence occur *closer* together, it cannot handle the order of two (near-substring) subsequences as in the first example.

In this paper, we propose a new string kernels based on *variable-length-don't-care* (*VLDC*) patterns [5–7]. A VLDC pattern is an element of $\Pi = (\Sigma \cup \{\star\})^*$, where Σ is alphabet and $\star$ is a wildcard that matches any string. In the running example, the VLDC pattern $ab \star bb \star ba$ matches $abaabbaba$ by replacing the $\star$'s with aa and a, respectively. The VLDC pattern gives the best of both worlds and more, since it contains both the set of substrings and subsequences as a subset. Unlike NGKs and SSKs, our kernel can handle the situations where the order of relevant strings influence the labels. Our kernel is not limited to exact matching substrings as in NGKs, and can also handle the order of two substrings.

The number of VLDC patterns that matches a given string s of length n is $O(2^{2n})$. We define several versions of kernels based on the VLDC pattern, and propose algorithms that compute kernels, for example, for a given a pair of strings in $O(max\{n^4, \ell 2^\ell |\Sigma|^\ell\})$ time and $O(n^4)$ space, where ℓ is the length of VLDC patterns and n is the maximum length of given strings. Unfortunately, the time and space complexity of computing VLDC kernels are still quite high to apply for large data, but our kernels can be applied for data of small size in reasonable computation time.

We show the running times of VLDC kernel computation in a preliminary experiment, and compare with that of SSKs, using random text data. Our main experiment is a comparison of the performance in text classification, using spam data. The experiment shows that VLDC kernels outperform both SSKs and NGKs.

2 Preliminaries

2.1 VLDC Patterns

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 $1 \leq i \leq |u|$, and the substring of u that begins at position i and ends at position j is denoted by $u[i, j]$ for $1 \leq i \leq j \leq |u|$.

Let $\star$ denote a special symbol called a *variable length don't care* (*VLDC*) or a *wildcard*, which matches any string in Σ^*. Let $\Pi = (\Sigma \cup \{\star\})^*$. Any element p of Π is called a *variable-length-don't-care pattern* (or a *VLDC pattern* in short). The length of any VLDC pattern p, denoted $|p|$, is the number of characters plus the number of $\star$'s contained in p.

Definition 1. *For any string $x \in \Sigma^*$ and any VLDC pattern $p \in \Pi$, p is said to* match *x iff x is obtained by replacing the $\star$'s in p with some strings in Σ^*. We write $p \preceq x$ when p matches x.*

Since $\star$ matches the empty string ε, for any string $x \in \Sigma^*$ there are an infinite number of VLDC patterns matching x. To limit the number of VLDC patterns matching x, consider the subset $\Pi' \subset \Pi$ of VLDC patterns which start and end with $\star$ and contain no consecutive $\star$'s. For instance, $\star a \star b \star \in \Pi'$ but $\star a \star \star b \star \notin \Pi'$ and $a \star b \star \notin \Pi'$.

Definition 2. *For any string $x \in \Sigma^*$, we define $P(x)$ by*

$$P(x) = \{p \in \Pi' \mid p \preceq x\}.$$

For any string x of length n, the size of $P(x)$ is $O(2^{n-1}\sigma^n)$. Note that $P(x)$ still contains all interesting VLDC patterns matching x, and contains only those. Using the running example, $\star a \star b \star \in \Pi'$ matches a string x if and only if $\star a \star \star b \star \notin \Pi'$ matches x. Also, we have that $\star a \star b \star \in \Pi'$ matches x if $a \star b \star \notin \Pi'$ matches x (the opposite is not true).

For each VLDC pattern p, we consider the multiset of the "maximal run" of characters of Σ in p, as follows.

Definition 3. *For any VLDC pattern $p \in \Pi'$, we define the multiset $Con(p)$ of strings as*

$$Con(p) = \begin{cases} \{w_1, \ldots, w_n\} & \text{if } p = \star w_1 \star \cdots \star w_n \star \text{ and } w_i \in \Sigma^+ \ (1 \le i \le n), \\ \emptyset & \text{if } p = \star. \end{cases}$$

2.2 Support Vector Machines and Kernels

In this subsection, we review the outline of Support Vector Machines(SVMs) and kernels. SVMs are a machine learning algorithm based on the idea of a kernel mapping, and were introduced by Boser et al [8]. This methods learn the hyperplane in order to separate two sets of points so as to maximize the margin distance, using several statistic properties. One of its properties is the mapping to high dimensional feature space, that shape the performance of SVM. So called kernels, denoted by

$$K(s,t) = \phi(s) \cdot \phi(t),$$

are used in this method. The function $K(s,t)$ calculates the inner product between instance s and t in a high dimensional feature space defined by a mapping ϕ.

3 VLDC Kernels

For a given string, we first define the VLDC kernel which considers the hyperspace where each dimension represents a VLDC pattern and its value is 1 if the pattern matches the string, and 0 otherwise. Thus, the kernel can then be defined as the number of VLDC patterns that match both strings.

Definition 4 (VLDC Kernel). *For any strings $s, t \in \Sigma^*$, we define the* VLDC kernel *$K(s,t)$ by*

$$K(s,t) = \sum_{p \in \Pi'} \delta(p,s,t),$$

where

$$\delta(p,s,t) = \begin{cases} 1 & \textit{if } p \in P(s) \cap P(t), \\ 0 & \textit{otherwise.} \end{cases}$$

In many situations, it is more natural to prefer patterns where the occurrence of each character is closer together, in order to better capture the characteristics of the text. In the SSKs, this is achieved through gap weights. For the VLDC kernel, we can achieve this by preferring longer contiguous constant parts in the VLDC pattern, as follows:

Definition 5 (Weighted VLDC Kernel). *For any strings $s, t \in \Sigma^*$, we define the* weighted VLDC kernel *$K'(s,t)$ by*

$$K'(s,t) = \sum_{p \in \Pi'} \delta'(p,s,t),$$

where

$$\delta'(p,s,t) = \begin{cases} \sum_{x \in Con(p)} \lambda^{|x|} & \textit{if } p \in P(s) \cap P(t) \textit{ and } Con(p) \neq \emptyset, \\ 0 & \textit{otherwise,} \end{cases}$$

for some $\lambda > 1$.

For the weighted VLDC kernel, we also consider a version where the length of the VLDC pattern is limited, keeping the calculation tractable.

Definition 6 (Length Restricted Weighted VLDC Kernel). *For any strings $s, t \in \Sigma^*$, we define the* length restricted weighted VLDC kernel *$K''_\ell(s,t)$ by*

$$K''_\ell(s,t) = \sum_{p \in \Pi'} \delta''(p,s,t,\ell),$$

where

$$\delta''(p,s,t,\ell) = \begin{cases} \sum_{x \in Con(p)} \lambda^{|x|} & \textit{if } p \in P(s) \cap P(t),\ Con(p) \neq \emptyset \textit{ and } |p| \leq \ell, \\ 0 & \textit{otherwise,} \end{cases}$$

for some $\lambda > 1$ and $\ell \geq 1$.

We can also restrict the number of wildcards (VLDCs) used in the pattern, as follows.

Definition 7 (Wildcard Restricted Weighted VLDC Kernel). *For any strings $s, t \in \Sigma^*$, we define the* wildcard restricted weighted VLDC kernel *$K'''_r(s,t)$ by*

$$K'''_r(s,t) = \sum_{p \in \Pi'} \delta'''(p,s,t,r),$$

where

$$\delta'''(p,s,t,r) = \begin{cases} \sum_{x \in Con(p)} \lambda^{|x|} & \text{if } p \in P(s) \cap P(t),\ 0 < |Con(p)| \leq r+1, \\ 0 & \text{otherwise,} \end{cases}$$

for some $\lambda > 1$ and $r \geq 1$.

4 Algorithms to Compute VLDC Kernels

In this section we present our algorithms to compute string kernels based on VLDC patterns introduced in the previous section.

We use the following data structure in our algorithms.

Definition 8 (WDAWG). *The* Wildcard DAWG *(*WDAWG*) of a string x, denoted $WDAWG(x)$, is the smallest automaton that accepts all VLDC patterns $p \in \Pi$ such that $p \preceq x$.*

$WDAWG(x)$ with $x = abbab$ is shown in Fig. 1.

By definition, every VLDC pattern $q \in P(x)$ is accepted by $WDAWG(x)$. This implies that there is always a path spelling out q from the initial state of $WDAWG(x)$.

Theorem 1 ([9, 10]). *For any string x of length n, $WDAWG(x)$ requires $O(n^2)$ space and can be constructed in linear time in the output size.*

Using WDAWGs we obtain the following results:

Theorem 2. *For any strings s and t, $K(s,t)$ can be computed in $O(n^5)$ time and space, where $n = \max\{|s|,|t|\}$.*

Proof. It follows from Theorem 1 that $WDAWG(s)$ and $WDAWG(t)$ can be constructed in $O(n^2)$ time and space. Then we can easily construct a DFA which accepts every element of $P(s) \cap P(t)$ by combining $WDAWG(s)$ and $WDAWG(t)$. The size of such a DFA will be $O(n^4)$ in the worst case. Furthermore, since $K(s,t) = \sum_{p \in \Pi} \delta(p,s,t) = |P(s) \cap P(t)|$, $K(s,t)$ can be computed by a simple linear-time depth-first traversal on the combined DFA, to count the number of paths from the initial state to all accepting states which start and end with $\star$ and contain no consecutive $\star$'s. However, since the number of such paths can be very large, that is, up to $O(2^n \sigma^n)$, treating these numbers at each state can require up to $O(n)$ time and space. Therefore, $K(s,t)$ can be calculated in a total of $O(n^5)$ time and space in the worst case. $\square$

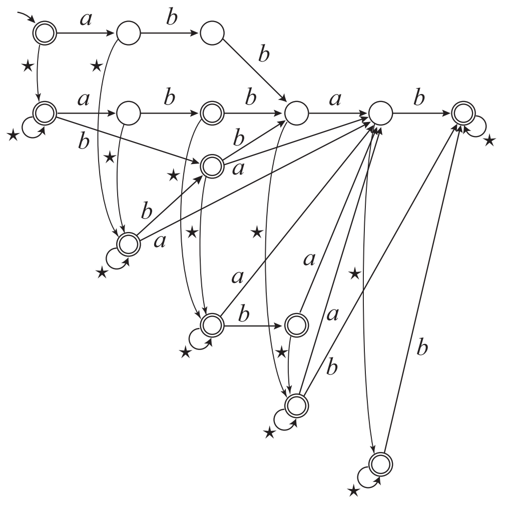

Fig. 1. *WDAWG*(x) with $x = abbab$.

Theorem 3. *For any strings s and t, $K'(s,t)$ can be computed in $O(n2^{2n})$ time and $O(n^4)$ space, where $n = \max\{|s|,|t|\}$.*

Proof. For each VLDC pattern $p \in P(s) \cap P(t)$ with $Con(p) \neq \emptyset$, we need to compute $\delta'(p,s,t) = \sum_{x \in Con(p)} \lambda^{|x|}$. Hence we execute a depth-first traversal on the DFA for all the paths from the initial state to all accepting states which start and end with $\star$ and contain no consecutive $\star$'s. There are $O(2^{2n})$ such paths. At each state corresponding to VLDC pattern p, the value of $\delta'(p,s,t)$ can be computed in $O(n)$ time utilizing information of the parent state in the traversal. Overall, the total time cost is $O(n2^n\sigma^n)$. The value of $K'(s,t)$ grows up to $O(2^{2n}\lambda^n)$ which requires $O(n)$ bits. Thus the space requirement is bounded by $O(n^4)$, the size of the combined DFA. $\square$

Due to the above theorem, computing the weighted VLDC kernel $K'(s,t)$ is not feasible for large n (long strings). However, the next theorem states that the length restricted weighted VLDC kernel $K''_\ell(s,t)$ is computable in cases where ℓ is small.

Theorem 4. *For any strings s, t and integer ℓ, $K''_\ell(s,t)$ can be computed in $O(\max\{n^4, \ell 2^\ell \sigma^\ell\})$ time and $O(n^4)$ space, where $n = \max\{|s|,|t|\}$.*

Proof. Since we have only to consider the VLDC patterns of length at most ℓ, the total number of paths we traverse in the DFA is $O(2^\ell\sigma^\ell)$. At each state p we need $O(\ell)$ time to compute $\delta''(p,s,t,\ell)$. Hence the time cost is $O(\max\{n^4, \ell 2^\ell\sigma^\ell\})$. The space requirement is $O(n^4)$ since $\ell \leq n$. $\square$

The wildcard restricted VLDC kernel $K'''_r(s,t)$ is also computable when r is small, since:

Theorem 5. *For any strings s, t and integer r, $K'''_r(s,t)$ can be computed in $O(n^{r+1}r)$ time and $O(n^4)$ space, where $n = \max\{|s|, |t|\}$.*

Proof. The number of VLDC patterns which have at most r wildcard symbols $\star$ and match both s and t is

$$O(\sum_{i=2}^{r} \times_{n-1}C_{i-2} \times n^2(i-1)) = O(n^r r).$$

Since at each state p we need $O(n)$ time to compute $\delta'''(p,s,t,r)$, the total time cost is $O(n^{r+1}r)$. The space requirement is $O(n^4)$ since $r \leq n$. □

5 Computational Experiments

In this section, we compare our VLDC kernels with SSK and NGK by experiments.

5.1 Time Comparison

Firstly, we show our experiments for comparison of computational times with randomly generated strings. The parameters of random strings are the alphabet size such as 4, 25 and 52 and string length such as 100, 200, $\cdots$, 500. We compare the length restricted VLDC kernel K'' with SSK. The lengths of the VLDC patterns for K'' and the subsequence for SSK are set to 5, 10 and 15. We utilized the SSK algorithm of [4]. We used a CentOS Linux desktop computer with two 3GHz dual core Xeon processors and 16GB memory.

Table 1 shows the running times (sec) for computing the length restricted VLDC kernel K'' for each VLDC pattern length. We have not computed kernel values for the cells with "-", since it will take too much time. Since our algorithm traverses all possible paths in the DFA where each state has at most σ transitions, the running time grows exponentially with respect to the alphabet size σ (see Theorem 4). Table 2 shows the running times (sec) for computing the SSK for each subsequence length. The running time of the SSK algorithm does not depend on the alphabet size.

5.2 Performance Comparison

We evaluated the performances of our VLDC kernels K and K'', compared to other kernels SSK and NGK. The classification algorithm we used was a free software SVM^{light}[1].

The values of our VLDC kernels may be huge, and thus we used the normalized value $\overline{K}$ for K, as follows:

$$\overline{K}(s,t) = \frac{K(s,t)}{\sqrt{K(s,s)K(t,t)}},$$

[1] `http://svmlight.joachims.org/`

Table 1. Computational times (sec) of the VLDC Kernel K'' values for random strings, with alphabet sizes 4, 26 and 52 and string lengths 100, 200, $\cdots$, 500. The the VLDC pattern length parameter ℓ was set to 5, 10 and 15. We did not compute the kernel value for each cell marked with "-", as it will apparently take too much time.

alphabet size	VLDC pattern length	string length				
		100	200	300	400	500
4	5	0.1125	0.7650	1.7250	2.9150	4.7250
	10	0.1156	0.7356	1.4467	2.7500	4.5633
	15	0.3500	1.3500	1.900	3.3100	5.1300
26	5	0.1550	0.9525	2.1350	3.8725	6.4475
	10	6.6400	34.2989	41.5500	49.6400	58.1067
	15	7468.7900	-	-	-	-
52	5	0.2175	1.445	3.5825	6.8050	11.2750
	10	24.2500	307.26	859.6867	1530.8200	2098.3830
	15	-	-	-	-	-

Table 2. Computational times (sec) of the SSK value for random strings, with alphabet sizes 4, 26 and 52 and string lengths 100, 200, $\cdots$, 500. The subsequence length parameter in SSK was set to 5, 10 and 15.

alphabet size	subsequence length	string length				
		100	200	300	400	500
4	5	0.0004	0.0040	0.0132	0.0244	0.0384
	10	0.0012	0.0120	0.0256	0.0504	0.0736
	15	0.0036	0.0184	0.0452	0.0780	0.1216
26	5	0.0000	0.0040	0.0116	0.0212	0.0356
	10	0.0020	0.0108	0.0272	0.0500	0.0792
	15	0.0020	0.0180	0.0404	0.0712	0.1180
52	5	0.0004	0.0040	0.0116	0.0228	0.0368
	10	0.0012	0.0108	0.0264	0.0488	0.0760
	15	0.0020	0.0172	0.0388	0.0720	0.1144

where $s, t \in \Sigma^*$. The normalized value $\overline{K''}$ for K'' is defined similarly.

In our experiments, we used spam data collected from the forum of Yahoo! Japan Finance[2], which were also used in [11]. We used 4 data sets of forum ID "4936", "4974", "6830", and "8473". The website contains a lot of spam messages, and the administrators watch over the forums and delete spam messages. We classified the deleted messages as spam and other remaining messages as non-spam. In order for our algorithms to run in a reasonable amount of time, we only used messages of length at most 200 in each data set. Table 3 shows the number of instances (messages) in each data set.

In Table 4, we show the accuracy (%) of classification with our VLDC kernels K, K'', SSK and NGK. The length restricted VLDC kernel K'', SSK and NGK

[2] `http://quote.yahoo.co.jp`

Table 3. The number of instances in each data set.

ID	spam	nonspam
4296	24	48
4974	4	14
6830	28	102
8473	22	84

require the following parameters; K'': VLDC pattern length ℓ and weight λ, SSK: subsequence length and gap weight, NGK: substring length.

Shorter patterns tend to give good performances for all kernels except for K which has no length parameter. The weight parameter λ of K'' was the best with $\lambda = 2.0$. Observe that, for all the forums, our length restricted VLDC kernel K'' shows the best performance.

6 Conclusions and Future Work

In this paper we proposed four types of string kernels based on VLDC patterns. Firstly, we introduced the VLDC kernel based on the all common VLDC patterns for a given pair of strings. This kernel is computable in $O(n^5)$ time and space, where n is the input string length. The needlessness of parameters for pattern length or weights is a merit of this kernel. Secondly, the weighted VLDC kernel was introduced, in which the consecutive constant characters are weighted according to the length of the consecutiveness. This kernel requires $O(n2^{2n})$ time and $O(n^4)$ space. The third kernel, the length restricted weighted VLDC kernel, has a parameter ℓ for the length of VLDC patterns, and thus is computable in $O(\max\{n^4, \ell 2^\ell \sigma^\ell\})$ time and $O(n^4)$ space. Lastly, we also proposed the wildcard restricted weighted VLDC kernel where the number of $\star$'s in each VLDC pattern is restricted by a parameter r. This kernel can be computed in $O(n^{r+1}r)$ time and $O(n^4)$ space. We evaluated performances of our VLDC kernels for the computation time and text classification ability by experiments. Computing our VLDC kernels took longer time than SSKs and NGK. VLDC kernels, however, outperformed SSK and NGK in the classification accuracy in the spam detecting experiments.

Our VLDC kernels are shown to have high potential to classify text data. However, their computational complexities may be too large to apply to large text datasets. We are currently investigating ways to lower this complexity, while still retaining the high classification accuracy.

Rational Kernels [2, 3] are known to be able to classify regular languages. We would also like to investigate the expressiveness of our VLDC kernels in this direction, and determine the class of languages that they can classify.

Table 4. The performances of VLDC kernels K, K'', SSK and NGK. Each cell shows the accuracy(%) by SVMlight. In the results of the length restricted VLDC kernel K'', "length" means the VLDC pattern length ℓ and "weight" means consecutive string weight $\lambda > 1$. In the results of SSK, "length" means subsequence length, and "weight" means the gap weight $0 < \delta < 1$. In the results of NGK, "length" means substring length, and NGK needs no string weight parameters.

			data set ID			
Kernel	Length	Weight	4296	4974	6830	8473
VLDC K			66.67	**77.78**	80.00	79.25
VLDC K''	5	1.1	68.06	**77.78**	91.54	95.29
		1.5	68.06	**77.78**	91.54	95.29
		2.0	**76.39**	72.23	**93.08**	**96.23**
	10	1.1	66.67	72.23	89.23	93.40
		1.5	66.67	72.23	89.23	93.40
		2.0	66.67	72.23	89.23	93.40
SSK	5	0.1	63.89	**77.78**	70.77	79.25
		0.3	66.67	72.23	76.93	86.79
		0.5	66.67	72.23	83.08	89.62
		0.7	69.45	72.23	67.69	86.80
		0.9	69.44	66.67	62.31	67.92
	10	0.1	66.67	50.00	79.24	51.89
		0.3	50.00	72.23	74.62	78.31
		0.5	66.67	72.23	76.92	81.14
		0.7	66.67	72.23	82.31	83.97
		0.9	66.67	72.23	78.46	79.25
	15	0.1	66.67	**77.78**	21.54	20.75
		0.3	66.67	50.00	78.46	79.25
		0.5	66.67	72.23	76.92	79.25
		0.7	65.28	72.23	83.08	79.25
		0.9	66.67	77.78	82.31	79.25
NGK	5		70.84	72.23	90.00	92.46
	6		70.84	72.23	90.00	89.63
	7		69.45	72.23	89.23	88.68
	8		69.45	72.23	90.00	87.74
	9		69.45	72.23	88.46	85.85
	10		69.45	72.23	86.92	84.91

References

1. Leslie, C.S., Eskin, E., Weston, J., Noble, W.S.: Mismatch string kernels for svm protein classification. In: Advance in Neural Information Processing Systems 15. (2002) 1417–1424
2. Cortes, C., Haffner, P., Mohri, M.: Rational kernels: Theory and algorithms. Journal of Machine Learning Research **5** (2004) 1035–1062
3. Cortes, C., Kontorovich, L., Mohri, M.: Learning languages with rational kernels. In: Proceedings of the 20th Annual Conference on Learning Theory. (2007) 349–364
4. Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. Journal of Machine Learning Research **2** (2002) 419–444
5. Inenaga, S., Bannai, H., Shinohara, A., Takeda, M., Arikawa, S.: Discovering best variable-length-don't-care patterns. In: Proceeding of the 5th International Conference Discovery Science(DS). Lecture Notes in Computer Science (LNCS) (2002) 86–97
6. Rahman, M.S., Iliopoulos, C.S., Lee, I., Mohamed, M., Smyth, W.F.: Finding patterns with variable length gaps or don't cares. In: Proc. 12th Annual International Computing and Combinatorics Conference (COCOON'06). (2006) 146–155
7. Navarro, G., Raffinot, M.: Fast and simple character classes and bounded gaps pattern matching. Journal of Computational Biology **10**(6) (2003) 903–923
8. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: COLT '92: Proceedings of the fifth annual workshop on Computational learning theory, ACM (1992) 144–152
9. Inenaga, S., Takeda, M., Shinohara, A., Hoshino, H., Arikawa, S.: The minimum DAWG for all suffixes of a string and its applications. In: Proceeding of 13th Annual Symposium on Combinatorial Pattern Matching(CPM). Volume 2373 of Lecture Notes in Computer Science (LNCS). (2002) 153–167
10. Inenaga, S., Shinohara, A., Takeda, M., Bannai, H., Arikawa, S.: Space-economical construction of index structures for all suffixes of a string. In: Proceeding of 27th International Symposium on Mathematical Foundations of Computer Science(MFCS). Volume 2420 of Lecture Notes in Computer Science (LNCS). (2002) 341–352
11. Narisawa, K., Bannai, H., Hatano, K., Takeda, M.: Unsupervised spam detection based on string alienness measures. In: Proceeding of the 10th International Conference Discovery Science(DS). Lecture Notes in Computer Science (LNCS) (2007) 159–172