STRING KERNELS WITH FEATURE SELECTION FOR SVM PROTEIN CLASSIFICATION

WEN-YUN YANG[1] and BAO-LIANG LU[1,2,*]

[1] *Department of Computer Science and Engineering, Shanghai Jiao Tong University*
[2] *Laboratory for Computational Biology, Shanghai Center for System Biomedicine*
Shanghai 200240, China
E-mail: {ywy, bllu}@sjtu.edu.cn

We introduce a general framework for string kernels. This framework can produce various types of kernels, including a number of existing kernels, to be used with support vector machines (SVMs). In this framework, we can select the informative subsequences to reduce the dimensionality of the feature space. We can model the mutations in biological sequences. Finally, we combine contributions of subsequences in a weighted fashion to get the target kernel. In practical computation, we develop a novel tree structure, coupled with a traversal algorithm to speed up the computation. The experimental results on a benchmark SCOP data set show that the kernels produced by our framework outperform the existing spectrum kernels, in both efficiency and ROC50 scores.

Keywords: kernel methods, SVMs, homology detection, feature selection

1. Introduction

Kernel methods and support vector machines (SVMs) have been proved to be highly successful in machine learning and pattern classification fields. In computational biology community, SVMs have also been widely used to yield valuable insights into massive biological data sets. However, since biological data, such as DNA, RNA, and protein sequences, are naturally represented as strings, one needs to convert string format of biological data into a numerical vector, which is the standard input format for SVMs. However, this additional conversion could brings additional computational cost and even unexpected results. Fortunately, this conversion can be avoided by using kernel methods. The key advantage of kernel methods is that they depend only on the inner products of the samples. As a result, we can calculate the inner products directly from the sequences instead of calculating the numerical vectors. In other words, the $n \times n$ matrix of inner products between each two samples is the so-called *kernel* of SVMs. We define the kernels of SVMs directly upon strings, which are also called "string kernels".[1]

The pioneering work on convolution kernels and dynamic alignment kernels for discrete objects, such as strings and trees, was conducted by Haussler[2] and

*To whom correspondence should be addressed.

Watkins,[3] respectively. Thereafter, a number of string kernels have been extensively studied. In general, those kernels take the same idea as the convolution kernels. They all define some kinds of "sub-structures" and employ recursive calculation over all those "sub-structures" to get the kernels. For example, Leslie et al. proposed spectrum string kernels,[1] mismatch string kernels,[4] and a series of inexact matching string kernels,[5] all of which are based on the "sub-structures" called "k-mers"(k-length subsequences). The only difference among those kernels relies on the specific definition for each mapping function. Moreover, Vishwanathan and Smola[6] proposed another type of fast string kernels based on weighted sum for inner products, each of which corresponds to one of the exact matching subsequences. Those above two kinds of string kernels were both applied to a protein classification problem, called remote homology detection. Besides, string kernels have also been successfully applied to natural language processing (NLP) tasks.[7-9]

We introduce a framework to reconstruct string kernels to be used with SVMs. This framework is rather general that the string kernels aforementioned can be regarded as specific instances of it. We also develop a tree data structure and an algorithm for the computation of these string kernels.

2. A string kernel framework

2.1. *Notations*

We begin by introducing some notations. Let $\mathcal{A}$ be the *alphabet* and each element in $\mathcal{A}$ is called *character*. Then we denote the whole *string space* as $P(\mathcal{A}) = \bigcup_k \mathcal{A}^k$, where $\mathcal{A}^k$ denotes the *k-spectrum set* containing all the k-length strings produced by character concatenation from $\mathcal{A}$. At the next step, we make use of *feature groups* to take the biologically mutation effect into account. Each feature group is a subset of the string space, containing certain number of relatively similar strings. Formally, we use $\mathcal{T} = \{T_i \subseteq P(\mathcal{A}) | 1 \leq i \leq m\}$ to denote the set of all the feature groups and $P(\mathcal{T}) = \bigcup_i T_i$ to denote all the strings contained in these feature groups. For each feature group T_i, we use $|T_i|$ to denote its size, and t_{ij} for $j = 1$ to $|T_i|$ to index its elements.

In the following section, "*none of two feature groups are identical*" means that $T_i \neq T_j$ if $i \neq j$ for all i and j. "*All the feature groups cover the set $\mathcal{S}$*" means $\bigcup_i T_i = \mathcal{S}$.

2.2. *Framework definition*

We propose a string kernel framework as follows. First, we define the sub-kernel between strings x and y for each feature group T_i,

$$\begin{aligned} k_{T_i}(x,y) = \mathrm{num}_{T_i}(x) \cdot \mathrm{num}_{T_i}(y) &= \textstyle\sum_{j=1}^{|T_i|} \mathrm{num}_{t_{ij}}(x) \cdot \sum_{k=1}^{|T_i|} \mathrm{num}_{t_{ik}}(y) \\ &= \textstyle\sum_{j=1}^{|T_i|} \sum_{k=1}^{|T_i|} \mathrm{num}_{t_{ij}}(x) \cdot \mathrm{num}_{t_{ik}}(y) \qquad (1) \end{aligned}$$

where $\text{num}_{T_i}(x) = \sum_{j=1}^{|T_i|} \text{num}_{t_{ij}}(x)$ counts the total numbers of occurrences of T_i's members in x. Then we combine all the sub-kernels in a weighted fashion to obtain the target kernel, formally,

$$k(x,y) = \sum_{i=1}^{m} w_{T_i} k_{T_i}(x,y) \tag{2}$$

where each w_{T_i} is the weight used to measure the significance for the corresponding feature group T_i. Following this construction framework, we can derive various kinds of string kernels. Several typical string kernel instances are given below as examples:

- Setting $w_{T_i} = 1$ and $|T_i| = 1$ for all $i = 1$ to m. None of two feature groups are identical and all the feature groups cover the k-spectrum set. It yields the *k-spectrum string kernel.*[1]
- Setting $|T_i| = 1$ for all $i = 1$ to m. None of two feature groups are identical and all the feature groups cover the string space $P(\mathcal{A})$. It yields the family of kernels proposed by Vishwanathan and Smola.[6]
- All *the kernels using inexact matching* proposed by Leslie and Kuang[5] can be regarded as specific cases of $|T_i| > 1$.
- If we can customize the members for each feature group T_i, then we will achieves a new family of string kernels which has never been studied.

2.3. *Relations with existing string kernels*

Roughly speaking, existing string kernels can be divided into two categories, kernels using exact matching and using inexact matching. Kernels using exact matching[1,6–8] only take the perfect matching subsequences into account and design optimal algorithms for the computation. However, the kernels using inexact matching can model mismatches, gaps, substitutions and other wildcards. Such kernels are more suitable for biological data. Conceptually, it is clear that the kernels using exact matching are specific instances of the our string kernel framework. Since we can assign only one feature to each feature group then produce those kernels. However practically, we note that the kernels using exact matching have been computed using various optimal algorithms.[6–8]

On the other hand, all the kernels using inexact matching[5] can be constructed equally by feature re-mapping as follows,

$$k(x,y) = \sum_{s} \left(\sum_{\alpha=R^{-1}(s)} \sum_{\beta=R^{-1}(s)} \text{num}_\alpha(x) \cdot \text{num}_\beta(y) \right) \tag{3}$$

where $R^{-1}(s) = \{s' : R(s', s)\}$ defines the set of substrings that have specific relations with substring s, for example, at most m mismatches and at most g gaps. s is used to enumerate the k-spectrum set $\mathcal{A}^k$. Comparing this definition with Equations (1) and (2), we could immediately find that the kernels using inexact matching can be constructed by $|\mathcal{A}^k|$ feature groups, each of which corresponds to one k-length substring s, containing the set $R^{-1}(s)$. Conceptually, the only difference among all these kernels depends on the specific relation R.

3. Efficient computation

Instead of calculating and storing the feature vectors explicitly, we develop an algorithm based on a novel tree data structure to efficiently compute the kernel matrix, which can be used with the SVM classifier.

3.1. *Tree data structure with leaf links*

This tree data structure shown in Fig. 1 is similar to a suffix tree or mismatch tree used before.[4] The different part is that we add leaf links to generalize the algorithm. The calculation of the kernel matrix can be summarized as follows: firstly we construct the tree based on given feature groups. Note that the tree structure is determined only by the given feature groups. Then we use an essentially sliding window to perform lexical traverse of all the substrings occurring in the data set. As a result, in each leaf we store the number of the leaf substring occurring in each sample string. Finally we calculate the kernel matrix in one traversal for all the leaves of the tree.

3.2. *Leaf traversal algorithm*

The leaves of this tree represent all the substrings occurring in the feature groups, so the number of these leaves is $|P(\mathcal{T})|$. Accordingly, all the leaves are indexed by s_i for $i = 1$ to $|P(\mathcal{T})|$. The tree is organized like a trie: the concatenation of the edge labels from root to leaf interprets the string of the leaf. Unlike the standard tree structure, we add links between two leaves if they are contained in the same feature group T_i (probably not only one). Formally we define the whole set of links as,

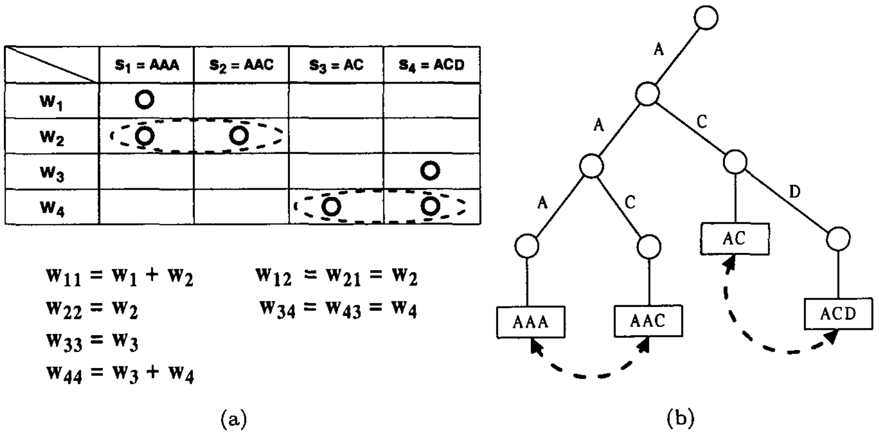
$$L = \{l_{ij} | \exists k, s_i \in T_k \wedge s_j \in T_k\}. \tag{4}$$

Then we define the set of leaves, with links to leaf s_i as $L[s_i] = \{j | l_{ij} \in L\}$. For each linked leaf pair, we can define the weight of that link as

$$w(l_{ij}) = \sum_{k: s_i \in T_k \wedge s_j \in T_k} w_{T_k}. \tag{5}$$

In the following part, we use w_{ij} as a shorthand for $w(l_{ij})$. The kernel matrix calculation within the traversal of all the leaves is summarized in Algorithm 3.1.

The correctness of this algorithm follows from the analysis of how many times the term $num_{s_i}(x) \cdot num_{s_j}(y)$ is added up to the kernel value $k(x, y)$. It can be observed from Equations (1) and (2).



	s_1 = AAA	s_2 = AAC	s_3 = AC	s_4 = ACD
w_1	O			
w_2	O	O		
w_3				O
w_4			O	O

$w_{11} = w_1 + w_2$ $\quad w_{12} = w_{21} = w_2$

$w_{22} = w_2$ $\quad w_{34} = w_{43} = w_4$

$w_{33} = w_3$

$w_{44} = w_3 + w_4$

Fig. 1. An example of the tree structure and leaf links: (a) 4 feature groups with weights from w_1 to w_4, respectively; (b) The tree constructed for the given feature groups. Here, a total of 6 links are connected. Note that for clarity, we omit the self links for each leaf node and only draw the leaf links between leaves.

Algorithm 3.1 The calculation of the kernel value $k(x, y)$

```
1: $k(x,y) \leftarrow 0$
2: for all leaf $s_i$ do
3:     for all $j \in L[s_i]$ do
4:         $k(x,y) \leftarrow k(x,y) + w_{ij} \cdot num_{s_i}(x) \cdot num_{s_j}(y)$
5:     end for
6: end for
```

4. Selecting feature groups and weights

The feature group aforementioned is a new concept for string kernels. Immediate extension can also be made for other kinds of machine learning methods. Actually we extend the notion of "feature" to "feature group" to let string kernels be more suitable to biological data. Meanwhile, it makes the construction procedure more flexible to produce various kinds of string kernels. In this section, we will develop several new approaches to demonstrate the effectiveness of the proposed framework.

Existing string kernel methods usually use the whole set of k-length subsequences as the feature set, and treat them equally in the kernel constructions. Unluckily, it leads not only to the loss of discriminative ability of significant subsequences, but also to the increase of computational cost. Apart from those, we start from learning the distribution of subsequences. Then we extracts statistically significant subsequences or groups of subsequences, which are then combined in a weighted fashion to reconstruct the string kernels.

To simplify this discussion, we restrict ourselves to two-class classification problems. Without loss of generalization, we explain our methods by using the following

BW criterion, which is based on the ratio of between-class scatter to within-class scatter. However, we also note that there are many types of statistical metrics that can be used in our proposed method.

$$BW(s) = \frac{|m^+(s) - m^-(s)|^2}{\sigma^+(s) + \sigma^-(s)} \tag{6}$$

where $m^+(s)$ and $\sigma^+(s)$ denote the mean composition and standard variance for subsequence s in the positive class, respectively, and $m^-(s)$ and $\sigma^-(s)$ are for the negative class. Usually, the numerator is called *between-class scatter* and the divisor is called *within-class scatter*.

To measure the statistical significance of a feature group, we also extend the definition of $BW(s)$ in Equation (6) to $BW(T_i)$, just by naturally defining the number of occurrences of feature group T_i as the sum of those of its members.

By using our framework, we propose two kinds of new string kernels in the following sections. Essentially, one is the reduced version of k-spectrum string kernel, and the other is the reduced version of (k, m)-mismatch string kernel.

4.1. *Reduction of spectrum string kernel*

We reconstruct the spectrum string kernels in two respects, the number of feature groups and the weights. Corresponding to the spectrum string kernel definition in Section 2, the number of feature groups is denoted by $|\mathcal{A}^k|$ and the weights are denoted by w_{T_i} for $i = 1$ to $|\mathcal{A}^k|$. For sake of computational efficiency and performance, we try to reduce feature groups $|\mathcal{A}^k|$ using two thresholds, minimum occurrence O_{min} and minimum score BW_{min}. Since we assume that the subsequences with low occurrences are either non-informative for discrimination or not influential in global performance. Similarly, the subsequences with low BW scores are also regarded with low discriminative ability.

For a proof of concept, we simply use the power of BW score, $w_{T_i} = [BW(T_i)]^\lambda$ to weight each of the feature groups, where the exponent λ is a parameter used to control the scale of weights.

4.2. *Statistically selecting feature groups*

How to choose the most discriminative feature groups and weights is at least as hard as the feature selection problem, which has 2^n subsets to be tested. This is clear since we can regard the feature selection as a specific case of feature group selection. Hence, we do not have an optimal solution for it. As an alternative approach, we propose a heuristic method to construct feature groups, each of which contains multiple members.

This method can be summarized as two steps: selecting the base subsequences s and then using a greedy expansion. The greedy expansion is an iterative process. At each iteration, the subsequence s' that lets $R(s', s)$ hold and maximize the $BW(T_i)$ score among the candidate subsequences, is selected into the feature group. This

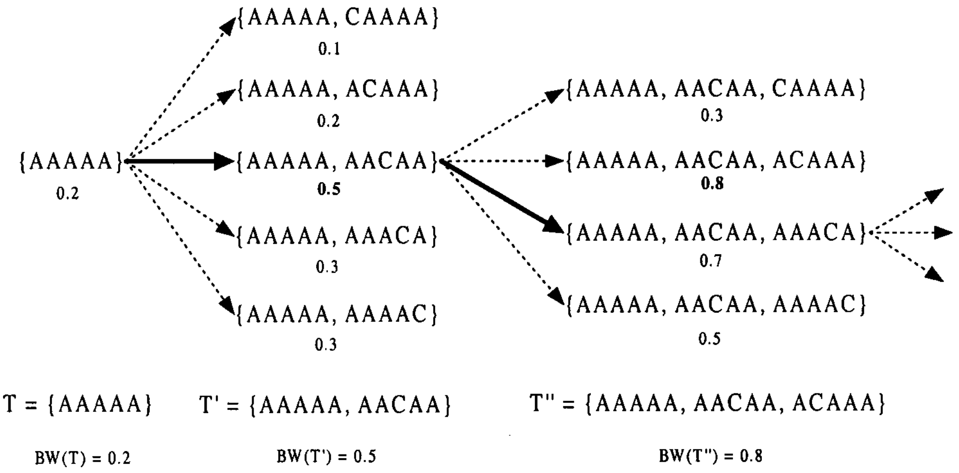


Fig. 2. An example of the greedy expansion in (5, 1) mismatch set.

process ends when no such s' is found. We give a simple example in Fig. 2. In this figure, for simplicity, we assume that the alphabet contains two letters, 'A' and 'C'. At the first iteration, AACAA is selected into the feature group, since it increases BW score more than other candidates. Then ACAAA is selected. Finally this greedy expansion terminates when there are no any features that let the BW score increase.

5. Experiment

We report the experiments on a benchmark SCOP data set (SCOP version 1.37) designed by Jaakkola et al.,[10] which is widely used to evaluate the methods for remote homology detection of protein sequences.[1,4–6] The data set[a] consists of 33 families, each of which has four sets of protein sequences, namely positive training and test sets, and negative training and test sets. The target family serves as the positive test set. The positive training set is chosen from the remaining families in the same superfamily. The negative training and test sets are chosen from the folds outside the fold of the target family.

We use ROC$_{50}$ score[11] to evaluate the performance of homology detection. The ROC$_{50}$ score is the area under the receiver operating characteristic curve (the plot of true positives as a function of false positives) up to the first 50 false positives. A score of one indicates perfect separation of positives from negatives, whereas a score of zero indicates that none of the top 50 sequences selected by the algorithm is positives. This ROC$_{50}$ score is the most standard way to evaluate the performance of remote homology detection methods in computational biology.[1,6,11]

[a]Data is available at `www.cse.ucsc.edu/research/compbio/discriminative`

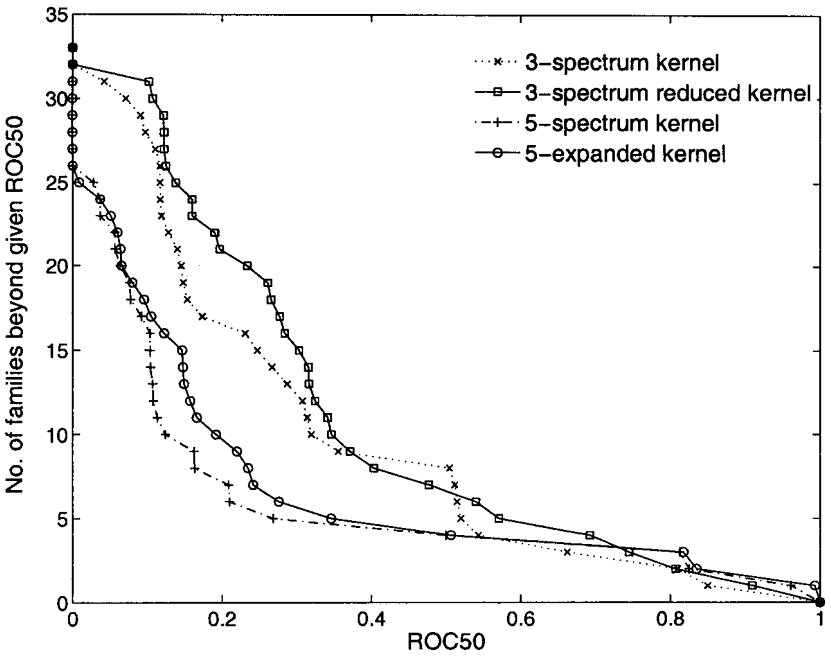


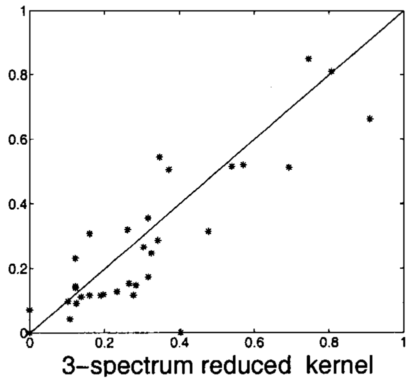
Fig. 3. Comparison of four kinds of kernels.

Table 1. The numbers of used subsequences in four kernels.

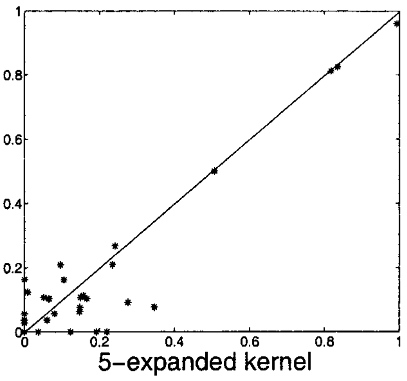
3-spectrum	3-spectrum reduced (mean/ ± SD)	5-spectrum	5-expanded (mean/ ± SD)
8000	2706/ ± 865	3.2×10^6	44926/ ± 20508

We give a performance overview in Fig. 3 for the four kinds of kernels. Table 1 shows the number of used subsequence for each kernel. The 3-spectrum and 5-spectrum kernels are the existing methods developed by Leslie et al.[1] We reduce the 3-spectrum kernel according to reduction techniques of spectrum kernels (see Section 4). The experimental result shows that better performance could be obtained even with much fewer 3-length subsequences, about 33.4% of the 3-spectrum set. This result strongly suggests that only a small portion of k-spectrum features could hold the discriminative information for remote homology. We would like to note that it is possible to further reduce the number of subsequences with comparative performance, providing that a more powerful feature selection technique is used.

We compare the kernels based on greedy expansion called 5-expanded kernel (see Fig. 2) with the existing 5-spectrum kernel. Our 5-expanded kernel can also be



(a)



(b)

Fig. 4. Family-by-family comparison of spectrum string kernels and their reduced versions. Here, the coordinates of each point are the ROC_{50} scores for one SCOP family, corresponding to the two labeld kernels, respectively

regarded as a reduced version of $(5,1)$-mismatch string kernel, since we reduce the 5-spectrum set and the members of each $R^{-1}(s)$. From the experimental result, we can observe that this kind of greedy expansion leads to a slight improvement upon 5-spectrum kernel. But our method uses only about 1.4% of 5-spectrum set, which is a significant feature reduction.

We should note that the $(5,1)$-mismatch kernel proposed by Leslie et al.[4] performs comparably with 3-spectrum kernel. On one hand, it means that our reduction of each $R^{-1}(s)$ leads to the performance decline compared with $(5,1)$-mismatch kernel. On the other hand, we obtain computational efficiency by reducing the feature number as a compensation.

We give in Fig. 4 a family-by-family comparison between the existing spectrum string kernels and our methods. It is clear that our methods perform slightly better than the existing spectrum kernels, especially for relatively hard-to-recognize families. This result suggests that carefully selected subsequences benefit hard detection tasks. However, for easy-to-recognize families, it seems always relatively easy to recognize no matter which kinds of features are used.

We select O_{min} from $\{5,10,20,50\}$, BW_{min} from $\{0.5,0.8,1\}$, and λ from $\{1,2,4,8\}$, respectively. Then the best results are reported. The 3-reduced kernel is obtained by using $O_{min}=20$, $BW_{min}=0.5$, and $\lambda=2$. The 5-expanded kernel is constructed by using greedy expansion (see Fig. 2) with parameters $O_{min}=5$, $BW_{min}=0.8$, and $\lambda=1$.

6. Discussion and future work

In this research work, we have proposed a general framework for string kernels, coupled with a general algorithm to naturally combine string kernels with feature selection techniques. This framework is applicable to almost all the kernel-based methods in biological sequence analysis. We make experiments on a benchmark SCOP data set for protein homology detection. The experimental results demonstrate that a large number of features can be reduced without any performance reduction, but conversely with improvement. We believe that this kind of string kernels, in conjunction with SVMs, will offer a more flexible and extendable approach to other protein classification problems.

For the further research, we plan to apply these string kernels to the prediction of protein subcellular locations and other biological problems. Meanwhile, we are still interested in developing new approaches to combining of feature selection and string kernels. We hope eventually this method could facilitate protein classification problems with both effectiveness and efficiency.

Acknowledgments

The authors thank James Kwok and Bo Yuan for their valuable comments and suggestions. They also thank National Institute of Information and Communications Technology, Japan, for their support with computational resources. This research is partially supported by the National Natural Science Foundation of China via the grant NSFC 60473040.

References

1. C. Leslie, E. Eleazar and W. S. Noble, The spectrum kernel: a string kernel for SVM protein classification, in *Proceedings of the Pacific Symposium on Biocomputing*, 2002.
2. D. Haussler, *Convolution kernels on discrete structures*, tech. rep., UC Santa Cruz (1999).
3. C. Watkins, *Dynamic alignment kernels*, tech. rep., UL Royal Holloway (1999).
4. C. Leslie, E. Eskin, J. Weston and W. S. Noble, Mismatch string kernels for svm protein classification, in *Advances in Neural Information Processing Systems 15*, (MIT Press, Cambridge, MA, 2003) pp. 1417–1424.
5. C. Leslie and R. Kuang, *Journal of Machine Learning Research* **5**, 1435 (2004).
6. S. Vishwanathan and A. J. Smola, Fast kernels for string and tree matching, in *Advances in Neural Information Processing Systems 15*, (MIT Press, Cambridge, MA, 2003) pp. 569–576.
7. H. Lodhi, J. Shawe-Taylor, N. Cristianini and C. Watkins, Text classification using string kernels, in *Advances in Neural Information Processing Systems 13*, (MIT Press, Cambridge, MA, 2001) pp. 563–569.
8. M. Collins and N. Duffy, Convolution kernels for natural language, in *Advances in Neural Information Processing Systems 14*, (MIT Press, Cambridge, MA, 2002) pp. 625–632.
9. J. Suzuki and H. Isozaki, Sequence and tree kernels with statistical feature mining, in *Advances in Neural Information Processing Systems 18*, (MIT Press, Cambridge, MA, 2006) pp. 1321–1328.
10. T. Jaakkola, M. Diekhans and D. Haussler, *Journal of Computational Biology* **7**, 95 (2000).
11. M. Gribskov and N. L. Robinson, *Computeres and Chemistry* **20**, 25 (1996).