Submitted 1/04; Revised 7/04; Published 11/04

Fast String Kernels using Inexact Matching for Protein Sequences

Christina Leslie CLESLIE@CS.COLUMBIA.EDU
Center for Computational Learning Systems
Columbia University
New York, NY 10115, USA

Rui Kuang RKUANG@CS.COLUMBIA.EDU
Department of Computer Science
Columbia University
New York, NY 10027, USA

Editor: Kristin Bennett

Abstract

We describe several families of k-mer based string kernels related to the recently presented mismatch kernel and designed for use with support vector machines (SVMs) for classification of protein sequence data. These new kernels – restricted gappy kernels, substitution kernels, and wildcard kernels – are based on feature spaces indexed by k-length subsequences ("k-mers") from the string alphabet Σ. However, for all kernels we define here, the kernel value $K(x,y)$ can be computed in $O(c_K(|x|+|y|))$ time, where the constant c_K depends on the parameters of the kernel but is independent of the size $|\Sigma|$ of the alphabet. Thus the computation of these kernels is linear in the length of the sequences, like the mismatch kernel, but we improve upon the parameter-dependent constant $c_K = k^{m+1}|\Sigma|^m$ of the (k,m)-mismatch kernel. We compute the kernels efficiently using a trie data structure and relate our new kernels to the recently described transducer formalism. In protein classification experiments on two benchmark SCOP data sets, we show that our new faster kernels achieve SVM classification performance comparable to the mismatch kernel and the Fisher kernel derived from profile hidden Markov models, and we investigate the dependence of the kernels on parameter choice.

Keywords: kernel methods, string kernels, computational biology

1. Introduction

The original work on string kernels – kernel functions defined on the set of sequences from an alphabet Σ rather than on a vector space (Cristianini and Shawe-Taylor, 2000) – came from the field of computational biology and was motivated by algorithms for aligning DNA and protein sequences. Pairwise alignment algorithms, in particular the Smith-Waterman algorithm for optimal local alignment and the Needleman-Wunsch algorithm for optimal global alignment (Waterman et al., 1991), model the evolutionary process of mutations – insertions, deletions, and residue substitutions relative to an ancestral sequence – and give natural sequence similarity scores related to evolutionary distance. However, standard pairwise alignment scores do not represent valid kernels (Vert et al., 2004), and the first string kernels to be defined – dynamic alignment kernels based on pair hidden Markov models by Watkins (1999) and convolution kernels introduced by Haussler (1999) – had to develop new technical machinery to translate ideas from alignment algorithms into a kernel framework. More recently, there has also been interest in the development of string kernels for use with support vector machine classifiers (SVMs) and other kernel methods in fields outside computational biology, such as text processing and speech recognition. For example, the gappy n-gram kernel developed by Lodhi et al. (2002) implemented a dynamic alignment kernel for text classification. A practical disadvantage of all these string kernels is their computational expense. In general, these kernels rely on dynamic programming algorithms for which the computation of each kernel value $K(x,y)$ is quadratic in the length of the input sequences x and y, that is, $O(|x||y|)$ with constant factor that depends on the parameters of the kernel.

The recently presented k-spectrum (gap-free k-gram) kernel and the (k,m) mismatch kernel provide an alternative model for string kernels for biological sequences and were designed, in particular, for the application of SVM protein classification. These kernels use counts of common occurrences of short k-length subsequences, called k-mers, rather than notions of pairwise sequence alignment, as the basis for sequence comparison. The k-mer idea still captures a biologically-motivated model of sequence similarity, in that sequences that diverge through evolution are still likely to contain short subsequences that match or almost match. Leslie et al. (2002a) introduced a linear time $(O(k(|x|+|y|))$ implementation of the k-spectrum kernel, using exact matches of k-mer patterns only, based on a trie data structure. Later, the (k,m)-mismatch kernel (Leslie et al., 2002b) extended the k-mer based kernel framework and achieved improved performance on the protein classification task by incorporating the biologically important notion of character mismatches (residue substitutions). Using a mismatch tree data structure, the complexity of the kernel calculation was shown to be $O(c_K(|x|+|y|))$, with $c_K = k^{m+1}|\Sigma|^m$ for k-grams with up to m mismatches from alphabet Σ. A different extension of the k-mer framework was presented by Vishwanathan and Smola (2002), who computed the weighted sum of exact-matching k-spectrum kernels for different k by using suffix trees and suffix links, allowing elimination of the constant factor in the spectrum kernel for a compute time of $O(|x|+|y|)$.

In this paper, we extend the k-mer based kernel framework in new ways by presenting several novel families of string kernels for use with SVMs for classification of protein sequence data. These kernels – restricted gappy kernels, substitution kernels, and wildcard kernels – are again based on feature spaces indexed by k-length subsequences from the string alphabet Σ (or the alphabet augmented by a wildcard character) and use biologically-inspired models of inexact matching. Thus the new kernels are closely related both to the (k,m)-mismatch kernel and the gappy k-gram string kernels used in text classification. However, for all kernels we define here, the kernel value $K(x,y)$ can be computed in $O(c_K(|x|+|y|))$ time, where the constant c_K depends on the parameters of the kernel but is independent of the size $|\Sigma|$ of the alphabet. Our efficient computation uses a recursive function based on a trie data structure and is linear in the length of the sequences, like the mismatch kernel, but we improve upon the parameter-dependent constant; a similar trie-based sequence search strategy has been used, for example, in work of Sagot (1998) for motif discovery. The restricted gappy kernels we present here can be seen as a fast approximation of the gappy k-gram kernel of Lodhi et al. (2002), where by using our k-mer based computation, we avoid dynamic programming and the resultant quadratic compute time; we note that the gappy k-gram kernel can also be seen as a special case of a dynamic alignment kernel (Watkins, 1999), giving a link between this work and some of the kernels we define. Cortes et al. (2002) have recently presented a transducer formalism for defining rational string kernels, and all the k-mer based kernels can be naturally described in this framework. We relate our new kernels to the transducer formalism and give transducers corresponding to our newer kernels. We note that Cortes et al. (2002) also describe the original convolution kernels of Haussler (1999) within their framework, suggesting that the transducer formalism is a natural unifying framework for describing many string kernels in the literature. However, the complexity for kernel computation using the standard rational kernel algorithm is $O(c_T|x||y|)$, where c_T is a constant that depends on the transducer, leading again to quadratic rather than linear dependence on sequence length.

Finally, we report protein classification experiments on two benchmark data sets based on the SCOP database (Murzin et al., 1995), where we show that our new faster kernels achieve SVM classification performance comparable to the mismatch kernel and the Fisher kernel derived from profile hidden Markov models. We also use kernel alignment scores (Cristianini et al., 2001) to investigate how different the various inexact matching models are from each other, and to what extent they depend on kernel parameters. Moreover, we show that by using linear combinations of different kernels, we can improve performance over the best individual kernel.

The current paper is an extended version of the original paper presenting these inexact string matching kernels (Leslie and Kuang, 2003). We have added the second set of SCOP experiments to allow more investigation of the dependence of SVM performance on kernel parameter choices. We have also included the kernel alignment results to explore differences between kernels and parameter choices and the advantage of combining these kernels.

2. Definitions of Feature Maps and String Kernels

Below, we review the definition of mismatch kernels (Leslie et al., 2002b) and present three new families of string kernels: restricted gappy kernels, substitution kernels, and wildcard kernels.

In each case, the kernel is defined via an explicit feature map from the space of all finite sequences from an alphabet Σ to a vector space indexed by the set of k-length subsequences from Σ or, in the case of wildcard kernels, Σ augmented by a wildcard character. For protein sequences, Σ is the alphabet of $|\Sigma| = 20$ amino acids. We refer to a k-length contiguous subsequence occurring in an input sequence as an instance k-mer (also called a k-gram in the literature). The mismatch kernel feature map obtains inexact matching of instance k-mers from the input sequence to k-mer features by allowing a restricted number of mismatches; the new kernels achieve inexact matching by allowing a restricted number of gaps, by enforcing a probabilistic threshold on character substitutions, or by permitting a restricted number of matches to wildcard characters. These models for inexact matching have all been used in the computational biology literature in other contexts, in particular for sequence pattern discovery in DNA and protein sequences (Sagot, 1998) and probabilistic models for sequence evolution (Henikoff and Henikoff, 1992, Schwartz and Dayhoff, 1978, Altschul et al., 1990).

2.1 Spectrum and Mismatch Kernels

In previous work, we defined the (k,m)-mismatch kernel via a feature map $\Phi^{\text{Mismatch}}_{(k,m)}$ to the $|\Sigma|^k$-dimensional vector space indexed by the set of k-mers from Σ. For a fixed k-mer $\alpha = a_1a_2\dots a_k$, with each a_i a character in Σ, the (k,m)-neighborhood generated by α is the set of all k-length sequences β from Σ that differ from α by at most m mismatches. We denote this set by $N_{(k,m)}(\alpha)$. For a k-mer α, the feature map is defined as

$$\Phi^{\text{Mismatch}}_{(k,m)}(\alpha) = (\phi_\beta(\alpha))_{\beta\in\Sigma^k}$$

where $\phi_\beta(\alpha) = 1$ if β belongs to $N_{(k,m)}(\alpha)$, and $\phi_\beta(\alpha) = 0$ otherwise. For a sequence x of any length, we extend the map additively by summing the feature vectors for all the k-mers in x:

$$\Phi^{\text{Mismatch}}_{(k,m)}(x) = \sum_{k\text{-mers } \alpha \text{ in } x} \Phi^{\text{Mismatch}}_{(k,m)}(\alpha).$$

Each instance of a k-mer contributes to all coordinates in its mismatch neighborhood, and the β-coordinate of $\Phi^{\text{Mismatch}}_{(k,m)}(x)$ is just a count of all instances of the k-mer β occurring with up to m mismatches in x. The (k,m)-mismatch kernel $K_{(k,m)}$ is then given by the inner product of feature vectors:

$$K^{\text{Mismatch}}_{(k,m)}(x,y) = \langle \Phi^{\text{Mismatch}}_{(k,m)}(x), \Phi^{\text{Mismatch}}_{(k,m)}(y) \rangle.$$

For $m = 0$, we obtain the k-spectrum (Leslie et al., 2002a) or k-gram kernel (Lodhi et al., 2002).

2.2 Restricted Gappy Kernels

For the (g,k)-gappy string kernel, $g \geq k$, we use the same $|\Sigma|^k$-dimensional feature space, indexed by the set of k-mers from Σ, but we define our feature map based on gappy matches of g-mers to k-mer features. For a fixed g-mer $\alpha = a_1 a_2 \ldots a_g$(each $a_i \in \Sigma$), let $G_{(g,k)}(\alpha)$ be the set of all the k-length subsequences occurring in α (with up to $g-k$ gaps). Then we define the gappy feature map on α as

$$\Phi^{\text{Gap}}_{(g,k)}(\alpha) = (\phi_\beta(\alpha))_{\beta \in \Sigma^k},$$

where $\phi_\beta(\alpha) = 1$ if β belongs to $G_{(g,k)}(\alpha)$, and $\phi_\beta(\alpha) = 0$ otherwise. In other words, each instance g-mer contributes to the set of k-mer features that occur (in at least one way) as subsequences with up to $g-k$ gaps in the g-mer. Now we extend the feature map to arbitrary finite sequences x by summing the feature vectors for all the g-mers in x:

$$\Phi^{\text{Gap}}_{(g,k)}(x) = \sum_{\text{g-mers } \alpha \in x} \Phi^{\text{Gap}}_{g,k}(\alpha).$$

The kernel $K^{\text{Gap}}_{(g,k)}(x,y)$ is defined as before by taking the inner product of feature vectors for x and y.

Alternatively, given an instance g-mer, we may wish to count the number of occurrences of each k-length subsequence and weight each occurrence by the number of gaps. Following (Lodhi et al., 2002), we can define for g-mer α and k-mer feature $\beta = b_1 b_2 \ldots b_k$ the weighting

$$\phi^\lambda_\beta(\alpha) = \frac{1}{\lambda^k} \sum_{\substack{1 \leq i_1 < i_2 < \ldots < i_k \leq g \\ a_{i_j} = b_j \text{ for } j = 1 \ldots k}} \lambda^{i_k - i_1 + 1},$$

where the multiplicative factor satisfies $0 < \lambda \leq 1$. We can then obtain a weighted version of the gappy kernel $K^{\text{Weighted Gap}}_{(g,k,\lambda)}$ from the feature map:

$$\Phi^{\text{Weighted Gap}}_{(g,k,\lambda)}(x) = \sum_{\text{g-mers } \alpha \in x} (\phi^\lambda_\beta(\alpha))_{\beta \in \Sigma^k}.$$

Here, the weighting $\dfrac{\lambda^{i_k - i_1 + 1}}{\lambda^k}$ penalizes a gappy occurrence of a k-mer by a factor λ raised to the number of internal gaps. This feature map is related to the gappy k-gram kernel defined by Lodhi et al. (2002) but enforces the following restriction: here, only those k-character subsequences that occur with at most $g-k$ gaps, rather than all gappy occurrences, contribute to the corresponding k-mer feature. When restricted to input sequences of length g, our feature map coincides with that of the usual gappy k-gram kernel. Note, however, that for our kernel, a gappy k-mer instance (occurring with at most $g-k$ gaps) is counted in all (overlapping) g-mers that contain it, whereas in Lodhi et al. (2002), a gappy k-mer instance is only counted once. If we wish to approximate the gappy k-gram kernel, we can define a small variation of our restricted gappy kernel where one only counts a gappy k-mer instance if its first character occurs in the first position of a g-mer window. That is, the modified feature map is defined on each g-mer α by coordinate functions

$$\widetilde{\phi}^{\lambda}_{\beta}(\alpha) = \frac{1}{\lambda^k} \sum_{\substack{1=i_1<i_2<\ldots<i_k\le g \\ a_{i_j}=b_j \text{ for } j=1\ldots k}} \lambda^{i_k-i_1+1},$$

$0<\lambda\le 1$, and is extended to longer sequences by adding feature vectors for g-mers. This modified feature map now gives a “truncation” of the usual gappy k-gram kernel.

In Section 3, we show that our restricted gappy kernel has $O(c(g,k)(|x|+|y|))$ computation time, where constant $c(g,k)$ depends on size of g and k, while the original gappy k-gram kernel has complexity $O(k(|x||y|))$. Note in particular that we do not compute the standard gappy k-gram kernel on every pair of g-grams from x and y, which would necessarily be quadratic in sequence length since there are $O(|x||y|)$ such pairs. We will see that for reasonable choices of g and k, we obtain much faster computation time, while in experimental results reported in Section 5, we still obtain good classification performance.

2.3 Substitution Kernels

The substitution kernel is again similar to the mismatch kernel, except that we replace the combinatorial definition of a mismatch neighborhood with a similarity neighborhood based on a probabilistic model of character substitutions. In computational biology, it is standard to compute pairwise alignment scores for protein sequences using a substitution matrix (Henikoff and Henikoff, 1992, Schwartz and Dayhoff, 1978, Altschul et al., 1990) that gives pairwise scores $s(a,b)$ derived from estimated evolutionary substitution probabilities. In one scoring system (Schwartz and Dayhoff, 1978), the scores $s(a,b)$ are based on estimates of conditional substitution probabilities $P(a|b)=p(a,b)/q(b)$, where $p(a,b)$ is the probability that a and b co-occur in an alignment of closely related proteins, $q(a)$ is the background frequency of amino acid a, and $P(a|b)$ represents the probability of a mutation into a during fixed evolutionary time interval given that the ancestor amino acid was b. We define the mutation neighborhood $M_{(k,\sigma)}(\alpha)$ of a k-mer $\alpha=a_1a_2\ldots a_k$ as follows:

$$M_{(k,\sigma)}(\alpha) = \{\beta = b_1b_2\ldots b_k \in \Sigma^k : -\sum_{i}^{k} \log P(a_i|b_i) < \sigma\}.$$

Mathematically, we can define $\sigma=\sigma(N)$ such that $\max_{\alpha\in\Sigma^k}|M_{(k,\sigma)}(\alpha)|<N$, so we have theoretical control over the maximum size of the mutation neighborhoods. In practice, choosing σ to allow an appropriate amount of mutation while restricting neighborhood size may require experimentation and cross-validation.

Now we define the substitution feature map analogously to the mismatch feature map:

$$\Phi^{\text{Sub}}_{(k,\sigma)}(x) = \sum_{k\text{-mers } \alpha \text{ in } x} (\phi_\beta(\alpha))_{\beta\in\Sigma^k},$$

where $\phi_\beta(\alpha) = 1$ if β belongs to the mutation neighborhood $M_{(k,\sigma)}(\alpha)$, and $\phi_\beta(\alpha) = 0$ otherwise.

2.4 Wildcard Kernels

Finally, we can augment the alphabet Σ with a wildcard character denoted by $*$, and we map to a feature space indexed by the set $\mathcal{W}$ of k-length subsequences from $\Sigma \cup \{*\}$ having at most m occurrences of the character $*$. The feature space has dimension $\sum_{i=0}^{m} \binom{k}{i} |\Sigma|^{k-i}$.

A k-mer α matches a subsequence β in $\mathcal{W}$ if all non-wildcard entries of β are equal to the corresponding entries of α (wildcards match all characters). The wildcard feature map is given by

$$\Phi^{\text{Wildcard}}_{(k,m,\lambda)}(x) = \sum_{k\text{-mers } \alpha \text{ in } x} (\phi_\beta(\alpha))_{\beta \in \mathcal{W}},$$

where $\phi_\beta(\alpha) = \lambda^j$ if α matches pattern β containing j wildcard characters, $\phi_\beta(\alpha) = 0$ if α does not match β, and $0 < \lambda \leq 1$.

Other variations of the wildcard idea, including specialized weightings and use of groupings of related characters, are described by Eskin et al. (2003).

3. Efficient Computation

All the k-mer based kernels we define above can be efficiently computed using a trie (retrieval tree) data structure, similar to the mismatch tree approach previously presented (Leslie et al., 2002b). In this framework, k-mer features correspond to paths from the root to the leaf nodes of the tree, and the data structure is used to organize a traversal of all the inexact matching instance patterns in the data that contribute to each k-mer feature count. We will describe the computation of the gappy kernel in most detail, since the other kernels are easier adaptations of the mismatch kernel computation. For simplicity, we explain how to compute a single kernel value $K(x,y)$ for a pair of input sequences; computation of the full kernel matrix in one traversal of the data structure is a straightforward extension.

3.1 (g,k)-Gappy Kernel Computation

For the (g,k)-gappy kernel, we represent our feature space as a rooted tree of depth k where each internal node has $|\Sigma|$ branches and each branch is labeled with a symbol from Σ. In this depth k trie, each leaf node represents a fixed k-mer in feature space by concatenating the branch symbols along the path from root to leaf and each internal node represents the prefix for those for the set of k-mer features in the subtree below it.

Using a depth-first traversal of this tree, we maintain at each node that we visit a set of pointers to all g-mer instances in the input sequences that contain a subsequence (with gaps) that matches the current prefix pattern; we also store, for each g-mer instance, an index pointing to the last position we have seen so far in the g-mer. At the root, we store pointers to all g-mer instances, and for each instance, the stored index is 0, indicating that we have not yet seen any characters in the g-mer. As we pass from a parent node to a child node along a branch labeled with symbol a, we process each of parent's instances by scanning ahead to find the next occurrence of symbol a in each g-mer. If such a character exists, we pass the g-mer to the child node along with its updated index; otherwise, we drop the instance and do not pass it to the child. Thus at each node of depth d, we have effectively performed a greedy gapped alignment of g-mers from the input sequences to the current d-length prefix, allowing insertion of up to $g-k$ gaps into the prefix sequence to obtain each alignment. When we encounter a node with an empty list of pointers (no valid occurrences of the current prefix), we do not need to search below it in the tree; in fact, unless there is a valid g-mer instance from each of x and y, we do not have to process the subtree. When we reach a leaf node, we sum the contributions of all instances occurring in each source sequence to obtain feature values for x and y corresponding to the current k-mer, and we update the kernel by adding the product of these feature values. Since we are performing a depth-first traversal, we can accomplish the algorithm with a recursive function and do not have to store the full trie in memory. Figure 1 shows expansion down a path during the recursive traversal.

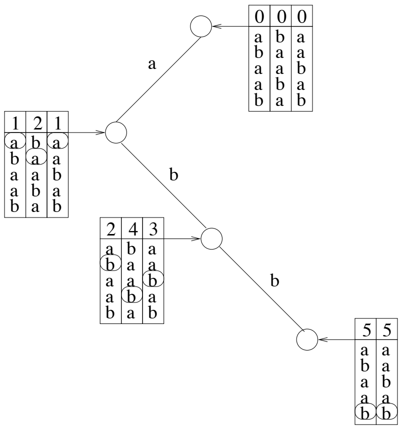
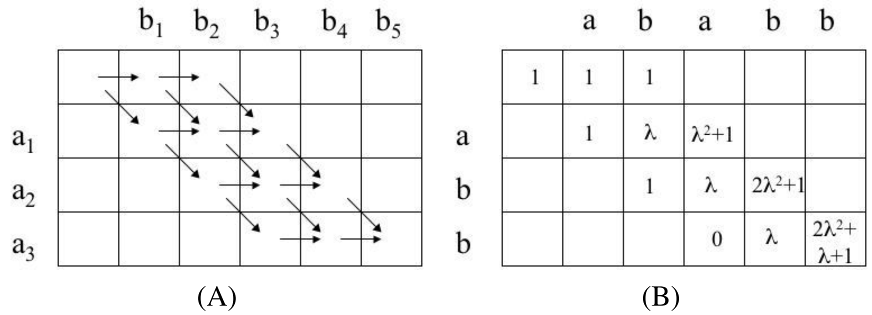


Figure 1: **Trie traversal for gappy kernel.** Expansion along a path from root to leaf during traveral of the trie for the $(5,3)$-gappy kernel, showing only the instance 5-mers for a single sequence $x = abaabab$. Each node stores its valid 5-mer instances and the index to the last match for each instance. Instances at the leaf node contribute to the kernel for 3-mer feature abb.

The computation at the leaf node depends on which version of the gappy kernel one uses. For the unweighted feature map, we obtain the feature values of x and y corresponding to the current k-mer by counting the g-mer instances at the leaf coming from x and from y, respectively; the product of these counts gives the contribution to the kernel for this k-mer feature. For the λ-weighted gappy feature map, we need a count of all alignments of each valid g-mer instance against the k-mer feature allowing up to $g-k$ gaps. This can be computed with a simple dynamic programming routine (similar to the Needleman-Wunsch algorithm), where we sum over a restricted set of paths, as shown in Figure 2. The complexity is $O(k(g-k))$, since we fill a restricted trellis of $(k+1)(g-k+1)$ squares. Note that when we align a subsequence $b_{i_1}b_{i_2}\ldots b_{i_k}$ against a k-mer $a_1a_2\ldots a_k$, we only penalize interior gaps corresponding to non-consecutive indices in $1 \leq i_1 < i_2 \ldots < i_k \leq g$. Therefore, the multiplicative gap cost is 1 in the zeroth and last rows of the trellis and λ in the other rows.

In order to determine the worst case complexity of the kernel computation, we estimate the traversal time – which can be bounded by the total number of g-mer instances that are processed



(A) (B)

Figure 2: **Dynamic programming at the leaf node.** The trellis in (A) shows the restricted paths for aligning a g-mer against a k-mer, with insertion of up to $g-k$ gaps in the k-mer, for $g=5$ and $k=3$. The basic recursion for summing path weights is $S(i,j)=m(a_i,b_j)S(i-1,j-1)+g(i)S(i,j-1)$, where $m(a,b)=1$ if a and b match, 0 if they are different, and the gap penalty $g(i)=1$ for $i=0,k$ and $g(i)=\lambda$ for other rows. That is, except for the top and bottom rows, every time we move to the right, we introduce an additional internal gap and incur a multiplicative penalty of λ; when we move diagonally, we see whether the corresponding characters match or not. Trellis (B) shows the example of aligning *ababb* against 3-mer *abb*. An explanation of how to interpret trellis diagrams for dynamic programming can be found in Durbin et al. (1998).

at the leaf nodes multiplied by the maximum number of times an instance is operated on as it is passed from root to leaf – plus the processing time at the leaf nodes need to compute the kernel update. Each g-mer instance in the input data can contribute to $\binom{g}{k}$ k-mer features, which we can write as $O(g^{g-k})$ if $g-k$ is smaller than k and $O(g^k)$ otherwise. For simplicity, we assume the more typical former case, and we let the reader make simple adjustments in the latter case. Therefore, at most $O(g^{g-k}(|x|+|y|)$ g-mer instances are processed at leaf nodes in the traversal. Since we iterate through at most g positions of each g-mer instance as we pass from root to leaf, the traversal time is $O(g^{g-k+1}(|x|+|y|))$. The total processing time at leaf nodes is $O(g^{g-k}(|x|+|y|))$ for the unweighted gappy kernel and $O(k(g-k)g^{g-k}(|x|+|y|))$ for the weighted gappy kernel. Therefore, in both cases, we have total complexity of the form $O(c(g,k)(|x|+|y|))$, with $c(g,k)=O((g-k)g^{g-k+1})$ for the more expensive kernel. Further discussion of the complexity argument and pseudocode for the algorithm can be found in Shawe-Taylor and Cristianini (2004).

Note that with the definition of the gappy feature maps given above, a gappy k-character subsequence occuring with $c \leq g-k$ gaps is counted in each of the $g-(k+c)+1$ g-length windows that contain it. To obtain feature maps that count a gappy k-character subsequence only once, we can make minor variations to the algorithm by requiring that the first character of a gappy k-mer occurs in the first position of the g-length window in order to contribute to the corresponding k-mer feature.

3.2 (k,σ)-Substitution Kernel Computation

For the substitution kernel, computation is very similar to the mismatch kernel algorithm. We use a depth k trie to represent the feature space. We store, at each depth d node that we visit, a set of pointers to all k-mer instances α in the input data whose d-length prefixes have current mutation score $-\sum_{i=1}^{d}\log P(a_i|b_i) < \sigma$ of the current prefix pattern $b_1b_2\ldots b_d$, and we store the current mutation score for each k-mer instance. As we pass from a parent node at depth d to a child node at depth $d+1$ along a branch labeled with symbol b, we process each k-mer α by adding $-\log P(a_{d+1}|b)$ to the mutation score and pass it to the child if and only if the score is still less than σ. As before, we update the kernel at the leaf node by computing the contribution of the corresponding k-mer feature.

The number of leaf nodes visited in the traversal is $O(N_\sigma(|x|+|y|))$, where the constant is the maximum mutation neighborhood size, $N_\sigma = \max_{\alpha\in\Sigma^k}|M_{(k,\sigma)}|$. We can choose σ sufficiently small to get any desired bound on N_σ, but it is difficult to estimate how to set the parameters to obtain good SVM classification performance except by empirical results. Total complexity for the kernel value computation is $O(kN_\sigma(|x|+|y|))$.

3.3 (k,m)-Wildcard Kernel Computation

Computation of the wildcard kernel is again very similar to the mismatch kernel algorithm. We use a depth k trie with branches labeled by characters in $\Sigma\cup\{*\}$, and we prune (do not traverse) subtrees corresponding to prefix patterns with greater than m wildcard characters. At each node of depth d, we maintain pointers to all k-mers instances in the input sequences whose d-length prefixes match the current d-length prefix pattern (with wildcards) represented by the path down from the root.

Each k-mer instance in the data matches at most $\sum_{i=0}^{m}\binom{k}{i} = O(k^m)$ k-length patterns having up to m wildcards. Thus the number of leaf nodes visited is in the traversal is $O(k^m(|x|+|y|))$, and total complexity for the kernel value computation is $O(k^{m+1}(|x|+|y|))$.

3.4 Comparison with Mismatch Kernel Complexity

For the (k,m) mismatch kernel, the size of the mismatch neighborhood of an instance k-mer is $O(k^m|\Sigma|^m)$, so total kernel value computation is $O(k^{m+1}|\Sigma|^m(|x|+|y|))$. All the other kernels presented here have running time $O(c_K(|x|+|y|))$, where constant c_K depends on the parameters of the kernel but not on the size of the alphabet Σ. Therefore, we have improved constant term for larger alphabets (such as the alphabet of 20 amino acids). In Section 5, we show that these new, faster kernels have performance comparable to the mismatch kernel in protein classification experiments.

4. Transducer Representation

Cortes et al. (2002) recently showed that many known string kernels can be associated with and constructed from weighted finite state transducers with input alphabet Σ. We briefly outline their transducer formalism and give transducers for some of our newly defined kernels. For simplicity, we only describe transducers over the probability semiring $\mathbb{R}_+ = [0,\infty)$, with regular addition and multiplication.

Following the development in Cortes et al. (2002), a weighted finite state transducer over $\mathbb{R}_+$ is defined by a finite input alphabet Σ, a finite output alphabet Δ, a finite set of states Q, a set of input states $I \subset Q$, a set of output states $F \subset Q$, a finite set of transitions $E \subset Q \times (\Sigma \cup \{\varepsilon\}) \times (\Delta \cup \{\varepsilon\}) \times \mathbb{R}_+ \times Q$, an initial weight function $\lambda : I \to \mathbb{R}_+$, and a final weight function $\rho : F \to \mathbb{R}_+$. Here, the symbol ε represents the empty string. The transducer can be represented by a weighted directed graph with nodes indexed by Q and each transition $e \in E$ corresponding to a directed edge from its origin state $p[e]$ to its destination state $n[e]$ and labeled by the input symbol $i[e]$ it accepts, the output symbol $o[e]$ it emits, and the weight $w[e]$ it assigns. We write the label as $i[e] : o[e]/w[e]$ (abbreviated as $i[e] : o[e]$ if the weight is 1).

For a path $\pi = e_1 e_2 \dots e_k$ of consecutive transitions (directed path in graph), the weight for the path is $w[\pi] = w[e_1]w[e_2]\dots w[e_k]$, and we denote $p[\pi] = p[e_1]$ and $n[\pi] = n[e_k]$. We write $\Sigma^* = \cup_{k \geq 0} \Sigma^k$ for the set of all strings over Σ. For an input string $x \in \Sigma^*$ and output string $z \in \Delta^*$, we denote by $P(I,x,z,F)$ the set of paths from initial states I to final states F that accept string x and emit string z. A transducer T is called regulated if for any pair of input and output strings (x,z), the output weight $[[T]](x,z)$ that T assigns to the pair is well-defined. The output weight is given by:

$$[[T]](x,z) = \sum_{\pi \in P(I,x,z,F)} \lambda(p[\pi])w[\pi]\rho(n[\pi])$$

A key observation from Cortes et al. (2002) is that there is a general method for defining a string kernel from a weighted transducer T. Let $\Psi : \mathbb{R}_+ \to \mathbb{R}$ be a semiring morphism (for us, it will simply be inclusion), and denote by T^{-1} the transducer obtained from T by transposing the input and output labels of each transition. Then if the composed transducer $S = T \circ T^{-1}$ is regulated, one obtains a rational string kernel for alphabet Σ via

$$K(x,y) = \Psi([[S]](x,y)) = \sum_z \Psi([[T]](x,z))\Psi([[T]](y,z))$$

where the sum is over all strings $z \in \Delta^*$ (where Δ is the output alphabet for T) or equivalently, over all output strings that can be emitted by T. Therefore, we can think of T as defining a feature map indexed by all possible output strings $z \in \Delta^*$ for T.

Using this construction, Cortes et al. showed that the k-gram counter transducer T_k corresponds to the k-gram or k-spectrum kernel, and the gappy k-gram counter transducer $T_{k,\lambda}$ gives the unrestricted gappy k-gram kernel from Lodhi et al. (2002). Figure 3(a) shows the diagram of the 3-gram transducer T_3, and Figure 3(b) gives the gappy 3-gram transducer $T_{3,\lambda}$. Our (g,k,λ)-gappy kernel $K^{\text{Weighted Gap}}_{(g,k,\lambda)}$ can be obtained from the composed transducer $T = T_{k,\lambda} \circ T_g$ using the $T \circ T^{-1}$ construction. (In all our examples, we use $\lambda(s) = 1$ for every initial state s and $\rho(t) = 1$ for every final state t.)

For the (k,m)-wildcard kernel, we set the output alphabet to be $\Delta = \Sigma \cup \{*\}$ and define a transducer with $m+1$ final states, as indicated in the figure. The $m+1$ final states correspond to destinations of paths that emit k-grams with 0, 1, …, m wildcard characters, respectively. The $(3,1)$-wildcard transducer is shown in Figure 4.

The (k,σ)-substitution kernel does not appear to fall exactly into this framework, though if we threshold individual substitution probabilities independently rather than threshold the product probability over all positions in the k-mer, we can define a transducer that generates a similar kernel. Starting with the k-gram transducer, we add additional transitions (between "consecutive" states of the k-gram) of the form $a : b$ for those pairs of symbols with $-\log P(a|b) < \sigma_o$. Now there will be a (unique) path in the transducer that accepts k-mer $\alpha = a_1 a_2 \dots a_k$ and emits $\beta = b_1 b_2 \dots b_k$ if and only if every substitution satisfies $-\log P(a_i|b_i) < \sigma_o$.

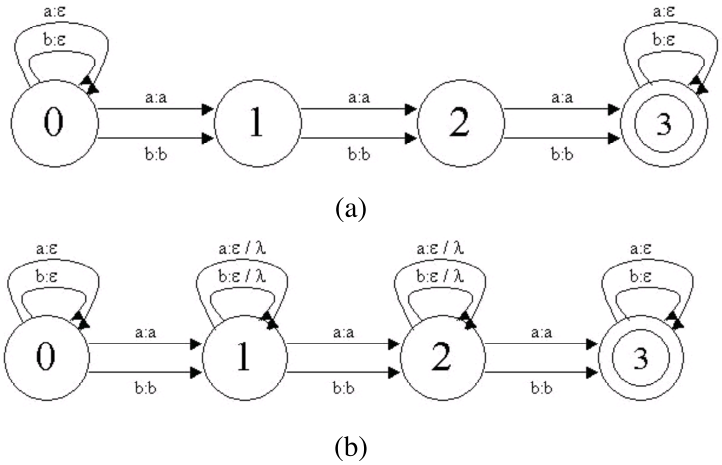


Figure 3: **The k-gram and gappy k-gram transducers.** The diagrams show the 3-gram transducer (a) and the gappy 3-gram transducer (b) for a two-letter alphabet. Here, the edge label $a : \varepsilon : \lambda$, for example, means "accept symbol a, output the empty symbol ε, multiply the weight by λ".

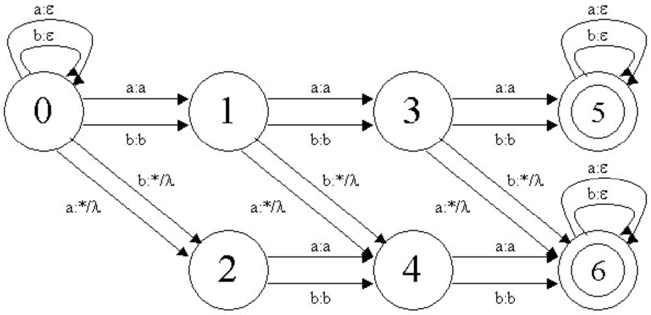


Figure 4: **The (k,m)-wildcard transducer.** The diagram shows the $(3,1)$-wildcard transducer for a two-letter alphabet.

5. Experiments

We tested all the new string kernels with SVM classifiers on two benchmark data sets (Jaakkola et al., 1999, Weston et al., 2003), both designed for the remote protein homology detection problem, in order to compare to results with the mismatch kernel reported previously (Leslie et al., 2002b) and other recent kernel representations for protein sequence data. We also present results to explore how parameter choices for the new kernels affect SVM classification performance. The benchmarks are based on different versions of the SCOP database (Murzin et al., 1995), an expert-curated database of protein domains with known 3D structure, organized hierarchically into folds, superfamilies, and families. Protein domain sequences belonging to different families but the same superfamily are considered to be remote homologs in SCOP. In these experiments, remote homology is simulated by holding out all members of a target SCOP family from a given superfamily as a test set, while examples chosen from the remaining families in the same superfamily form the positive training set. The negative test and training examples are chosen from disjoint sets of folds outside the target family's fold, so that negative test and negative training sets are unrelated to each other and to the positive examples. More details of the experimental set-up can be found in Jaakkola et al. (1999).

While in principle, we can define and test inexact matching string kernels for a wide range of parameters, in practice, only a small parameter range is biologically motivated for use in the remote protein homology detection problem. For the exact matching k-spectrum kernel (Leslie et al., 2002a), the only interesting parameter choices are $k = 3$ and $k = 4$, since exact occurrences of k-mers of length $k \geq 5$ in remotely homologous proteins are so rare that the spectrum kernel would mostly be 0 off the diagonal. By incorporating inexact matching such as mismatches or gaps, we can use slightly longer subsequence instances and allow a few mismatches or gaps; however, using very long subsequences, or allowing a great amount of mutation of subsequence instances in our inexact matching scheme, would not capture biologically realistic sequence similarity. For example, for the gappy kernel, we expect that length $(g,k) = (6,4)$ – 4-mers allowing up to 2 gaps – would be a useful parameter choice, while allowing many more gaps and hence a longer g-length window would be less useful. We test a range of parameter choices that seem reasonable in the experiments below.

In the first and larger SCOP benchmark data set, based on SCOP version 1.37, we compare to the Fisher kernel of Jaakkola et al. (1999) in addition to our previous mismatch kernel. In the Fisher kernel method, the feature vectors are derived from profile HMMs trained on the positive training examples. The feature vector for sequence x is the gradient of the log likelihood function $\log P(x|\theta)$ defined by the model and evaluated at the maximum likelihood estimate for model parameters: $\Phi(x) = \nabla_\theta \log P(x|\theta)|_{\theta=\theta_0}$. The Fisher kernel was the best performing method on this data set prior to the mismatch-SVM approach, whose performance is as good as Fisher-SVM and better than all other standard methods tried (Leslie et al., 2002b). We note that in this data set, additional positive training sequences were pulled in from the non-redundant protein sequence database using an iterative training method for the profile HMMs. The presence of these additional "domain homologs" makes the learning task easier for all methods.

We also include a second set of experiments to further investigate the dependence of SVM performance on parameter choices for the new kernels. This second data set is based on SCOP version 1.59 and contains only sequences from the SCOP database – no domain homologs are added. The experiments are similar to those described by Liao and Noble (2002) but use a more recent version of SCOP. In this data set, the positive training sets are quite small, and the learning task is more difficult in this setting. In particular, there is not enough positive training data to train profile HMMs in these experiments, so we do not report Fisher kernel results (which are not competitive in this setting).

There is another successful feature representation for protein classification, the SVM-pairwise method presented in Liao and Noble (2002). Here one uses an empirical kernel map based on pairwise Smith-Waterman (Waterman et al., 1991) alignment scores

$$\Phi(x) = (d(x_1,x),\dots,d(x_m,x)),$$

where x_i, $i = 1 \dots m$, are the training sequences and $d(x_i, x)$ is the E-value for the alignment score between x and x_i. We have previously shown (Leslie et al., 2004) that the mismatch kernel used with an SVM classifier is competitive with SVM-pairwise on the SCOP benchmark presented in Liao and Noble (2002), so we do not repeat the SVM-pairwise experiments for the very similar benchmark here.

In both experiments, we normalized the kernel by $k(x,y) \leftarrow \frac{k(x,y)}{\sqrt{k(x,x)}\sqrt{k(y,y)}}$. All methods are evaluated using the receiver operating characteristic (ROC) score, which is the area under the receiver operating curve, which plots the rate of true positives as a function of the rate of false positives as the threshold for the classifier varies (Gribskov and Robinson, 1996). Perfect ranking of all positives above all negatives gives an ROC score of 1, while a random classifier has an expected score close to 0.5. We also use the ROC-50 score, which is the normalized area under the receiver operating curve up to the first 50 false positives; this score focuses on the top of the ranking of the test examples produced by the classifier and is more informative when there are very few positive examples in the test set. Use of ROC-50 scores (or other ROC-N scores) is the most standard way of evaluating performance of homology detection methods in bioinformatics (Gribskov and Robinson, 1996).

Finally, we use kernel alignment scores (Cristianini et al., 2001) on the second SCOP data set to investigate the empirical differences between the different inexact matching models for protein sequence data, and we investigate methods for combining kernels to improve SVM performance.

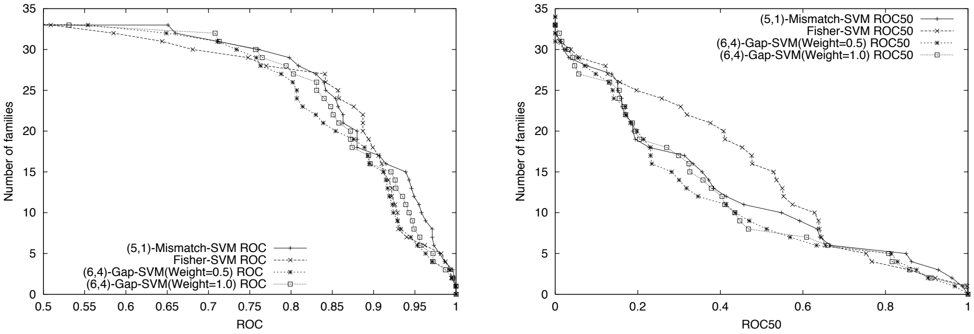
5.1 SCOP Experiments with Domain Homologs: Comparison with Fisher and Mismatch Kernels

We first present experimental results for the new kernels on the larger of the two SCOP data sets, the Fisher-SCOP benchmark introduced by Jaakkola et al. (1999) that contains domain homologs for additional positive training data, and compare SVM classification performance to both the mismatch kernel and the Fisher kernel.

We tested the (g,k)-gappy kernel with parameter choices $(g,k) = (6,4), (7,4), (8,5), (8,6)$, and $(9,6)$. Among them $(g,k) = (6,4)$ yielded the best results, though other choices of parameters had quite similar performance (data not shown). We also tested the alternative weighted gappy kernel, where the contribution of an instance g-mer to a k-mer feature is a weighted sum of all the possible matches of the k-mer to subsequences in the g-mer with multiplicative gap penalty λ $(0 < \lambda \leq 1)$. We used gap penalty $\lambda = 1.0$ and $\lambda = 0.5$ with the $(6,4)$ weighted gappy kernel. We found that $\lambda = 0.5$ weighting slightly weakened performance (results not shown). In Figure 5, we see that unweighted and weighted ($\lambda = 1.0$) gappy kernels have comparable results to $(5,1)$-mismatch kernel and Fisher kernel.

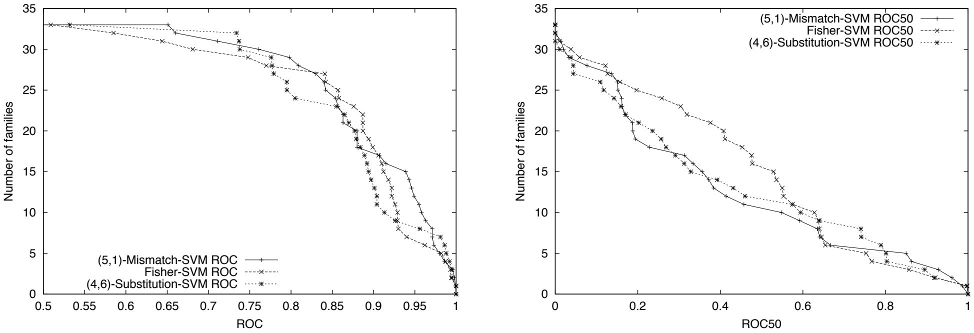
We tested the substitution kernels with $(k,\sigma) = (4, 6.0)$. Here, $\sigma = 6.0$ was chosen so that the members of a mutation neighborhood of a particular 4-mer would typically have only one position with a substitution, and such substitutions would have fairly high probability. Therefore, the mutation neighborhoods were much smaller than, for example, $(4,1)$-mismatch neighborhoods. The results are shown in Figure 6. Again, the substitution kernel has comparable performance with mismatch-SVM and Fisher-SVM, though results are perhaps slightly weaker for more difficult test families.

In order to compare with the $(5,1)$-mismatch kernel, we tested wildcard kernels with parameters $(k,m,\lambda) = (5,1,1.0)$ and $(k,m,\lambda) = (5,1,0.5)$. Results are shown in Figure 7. The wildcard kernel with $\lambda = 1.0$ seems to perform as well or almost as well as the $(5,1)$-mismatch kernel and Fisher



(a) (b)

Figure 5: **Comparison of of Mismatch-SVM, Fisher-SVM and Gappy-SVM.** The graph plots the total number of families for which a given method exceeds an ROC score threshold (a) or ROC-50 score threshold (b).



(a) (b)

Figure 6: **Comparison of mismatch-SVM, Fisher-SVM and substitution-SVM.** The graph plots the total number of families for which a given method exceeds an ROC score threshold (a) or ROC-50 score threshold (b).

kernel, while enforcing a penalty on wildcard characters of $\lambda = 0.5$ seems to weaken performance somewhat.

If we compare results for the best-performing parameter choices that we tried from each kernel family – the $(5,1)$-mismatch kernel, the $(5,1,1.0)$-wildcard kernel, the $(6,4)$-gappy kernel with $\lambda = 1.0$, and the $(4,6.0)$-substitution kernel – then a signed ranked test with Bonferroni correction for multiple comparisons (Henikoff and Henikoff, 1992, Salzberg, 1997) and a p-value cut-off of 0.05 finds no significant differences between the four kernels, either on the basis of ROC or ROC-50 scores.

5.2 SCOP Experiments without Domain Homologs: Dependence on Parameters

In the second set of SCOP experiments, we take advantage of the smaller data set from Weston et al. (2003) to generate kernels corresponding to a wider range of parameter values, so that we

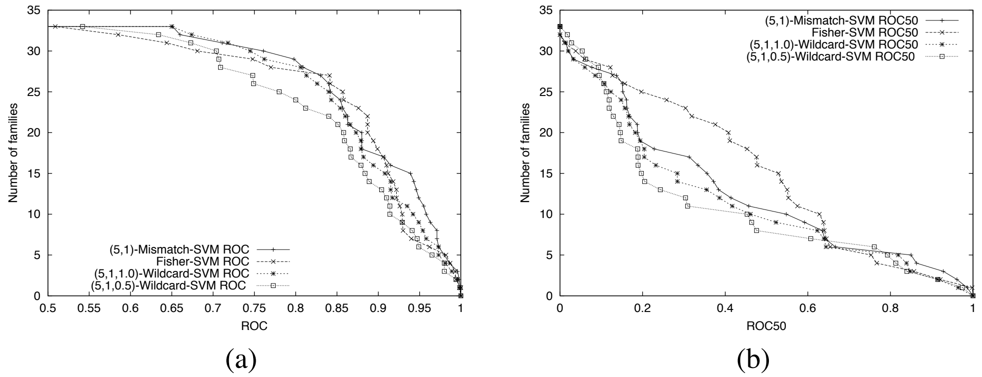


Figure 7: **Comparison of mismatch-SVM, Fisher-SVM and wildcard-SVM.** The graph plots the total number of families for which a given method exceeds an ROC score threshold (a) or ROC-50 score threshold (b).

can explore how parameter choices affect SVM classification performance. We also use kernel alignment and kernel-target alignment scores (Cristianini et al., 2001) to investigate differences between different kernel models. Note that this data set contains no domain homologs, and thus the small amount of positive training data makes the experiments more difficult.

In experiments with the gappy kernel, we chose parameter values $(g,k) = (6,4), (7,5)$ and $(8,6)$ and set the gap penalty to $\lambda = 1.0$, the preferred choice from the previous experiments. The choice $(g,k) = (6,4)$ still produced the best classification results, which were slightly but not significantly weaker than those of $(5,1)$-mismatch kernel. The results are shown in Figure 8. Performance deteriorates as larger values of the g parameter are chosen with the number of gaps held fixed.

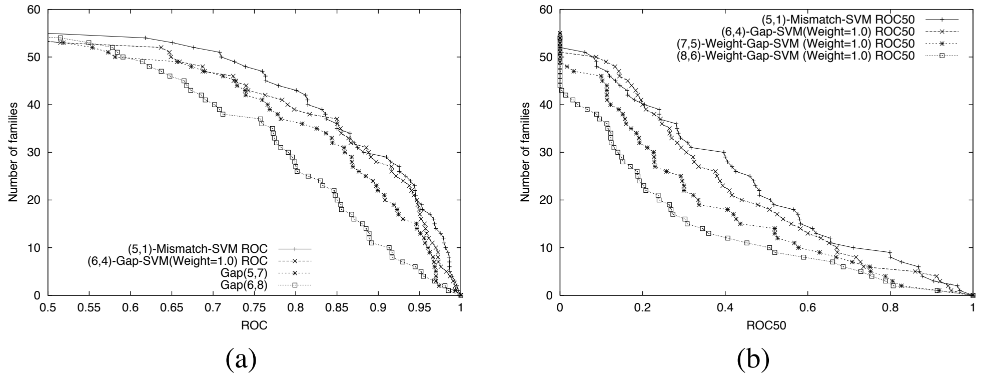
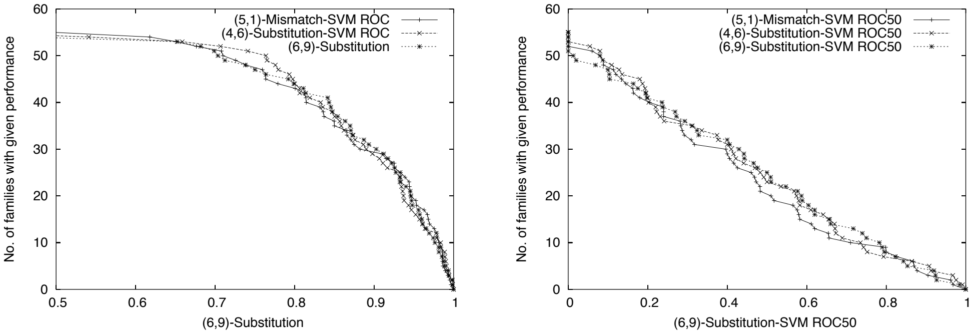


Figure 8: **Dependence on parameters for the gappy kernel.** The graph plots the total number of families for which a given method exceeds an ROC (a) or ROC-50 (b) score threshold.

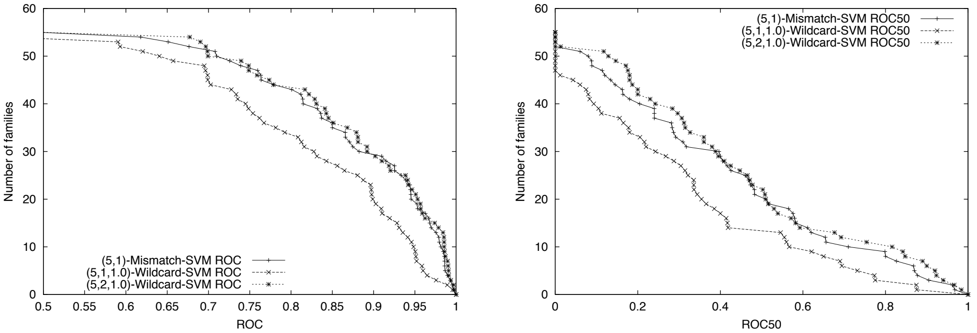
The substitution kernel was tested with parameter choices $(k,\sigma) = (4,6.0), (5,7.5)$ and $(6,9.0)$. All of these three kernels gave slightly stronger performance than the $(5,1)$-mismatch kernel, and results for the different parameter choices were remarkably similar, as shown in Figure 9. Thus, more so than for other inexact matching models, the substitution kernel performance seems stable as we vary k while σ is adjusted additively; however, as we see below, the Gram matrices produced by these different choices of kernels are in fact quite different.



(a) (b)

Figure 9: **Dependence on parameters for substitution kernel.** The graph plots the total number of families for which a given method exceeds an ROC (a) or ROC-50 (b) score threshold. Results for three parameter choices give almost identical results.

We tested the wildcard kernel with $(k,m,\lambda) = (5,1,1.0)$ and $(5,2,1.0)$. We observed a significant improvement in performance when we allowed up to 2 wildcards instead of 1 with $k = 5$. The performance of $(5,2,1.0)$-wildcard kernel gave the best results among all kernel families and parameters that we tried, though several other kernel choices gave very similar performance. The results are shown in Figure 10. Intuitively, it is clear that allowing 1 mismatch is closer to permitting 2 wildcards than to permitting a single wildcard: two k-mers that are identical except in *two* positions have intersecting $(k,1)$-mismatch neighborhoods and hence their $(k,1)$-mismatch feature vectors have non-zero inner product; similarly, such a pair of k-mers have non-orthogonal $(k,2)$-wildcard feature vectors but orthogonal $(k,1)$-wildcard feature vectors.



(a) (b)

Figure 10: **Dependence on parameters for the wildcard kernel.** The graph plots the total number of families for which a given method exceeds an ROC (a) or ROC-50 (b) score threshold. In the graph, the curve of $(5,2,1.0)$-wildcard kernel clearly outperforms the $(5,1,1.0)$-wildcard kernel.

Kernel	Kernel Alignment	ROC	ROC-50
(5,1)-mismatch	0.0982	0.875	0.416
(6,4)-gappy	0.1428	0.851	0.387
(7,5)-gappy	0.0269	0.825	0.315
(8,6)-gappy	0.0090	0.782	0.242
(4,6.0)-substitution	0.1643	0.876	0.441
(5,7.5)-substitution	0.0369	0.865	0.428
(6,9.0)-substitution	0.0170	0.871	0.442
(5,1,1.0)-wildcard	0.0310	0.816	0.304
(5,2,1.0)-wildcard	0.1565	0.881	0.447

Table 1: Mean ROC and ROC-50 scores over 54 target families.

Kernel	$(5,1)$-mismatch	$(6,4)$-gappy	$(4,6)$-subst	$(6,9)$-subst	$(5,1)$-wildcard	$(5,2)$-wildcard
$(5,1)$-mismatch	1.000	0.923	0.812	0.947	0.968	0.864
$(6,4)$-gappy		1.000	0.915	0.742	0.775	0.955
$(4,6)$-subst			1.000	0.591	0.622	0.942
$(6,9)$-subst				1.000	0.991	0.626
$(5,1)$-wildcard					1.000	0.669
$(5,2)$-wildcard						1.000

Table 2: Pairwise kernel alignment scores over the full SCOP data set.

In Table 1, we summarize the mean ROC and ROC-50 scores across the 54 target families for all the string kernels families and parameter values chosen. The table also shows mean training set *kernel-target alignment* scores across the experiments. Kernel alignment was introduced by Cristianini et al. (2001) as a measure of similarity between pairs of kernels or between a kernel and a target function. The *empirical kernel alignment* score between two kernels is defined as the value $\frac{\langle K_1, K_2\rangle}{\sqrt{\langle K_1, K_1\rangle\langle K2, K2\rangle}}$, where K_1 and K_2 are the Gram matrices for the kernels on the sample data, and $\langle\cdot,\cdot\rangle$ is the euclidean inner product when the Gram matrices are viewed as vectors (Hilbert-Schmidt inner product). Thus the alignment score is simply the cosine of the angle between the two vectors representing Gram matrices. The *empirical kernel-target alignment* is the kernel alignment for a Gram matrix and the target $\mathbf{yy}^t$, where $\mathbf{y}$ is the column vector of labels.

Table 1 shows that for the gappy and wildcard kernels, high kernel-target alignment scores do seem to correlate with good SVM classification performance. However, for the substitution kernels, the kernel-target alignment is low for larger values of k while performance remains strong. In Table 2, we show the pairwise kernel alignment scores between normalized kernels on the full SCOP data set of 7329 sequences. In some cases, the alignment scores between kernels of the same family with different parameters can be quite low, for example the $(5,1,1.0)$-wildcard kernel and $(5,2,1.0)$-wildcard kernel. Surprisingly, the $(6,9)$-substitution kernel Gram matrix is very similar to the $(5,1,1.0)$-wildcard kernel Gram matrix when compared by alignment score, even though their SVM performance is somewhat different, showing that the score gives only a rough measure of kernel similarity. The $(6,4)$-gappy kernel, $(4,6)$-substitution kernel and $(5,2,1.0)$-wildcard kernel are a group of well aligned Gram matrices. $(5,1)$-mismatch kernel seems to be in between the two previous groups in terms of kernel alignment. Clearly, all the models of inexact matching are fairly similar, but there do appear to be several significantly different Gram matrices in the set below that all successfully represent the data for the purposes of SVM learning.

Kernel	ROC	ROC-50
$K_{eq}(\alpha_i = 1)$	0.907	0.520
$K_{opt}(\sigma = 0.01)$	0.901	0.502

Table 3: Mean ROC and ROC-50 scores of linearly combined kernels. Here $K_{opt} = \sum_{i=1}^{N} \alpha_i K_i$, where N is the number of kernels, α is the optimal vector for the best alignment with target yy', and the the regularization parameter depends on σ as described in the text.

Since different kernels capture somewhat different notions of sequence similarity, we consider whether a convex combination of kernels $K(\alpha) = \sum_{i=1}^{N} \alpha_i K_i$, with $\alpha_i \geq 0$ for $i = 1 \ldots N$, can outperform individual kernels. We consider two schemes for choosing such a linear combination. In the first approach, we simply assign equal weights $\alpha_i = 1/N$ for all i to obtain a new kernel K_{eq}. For a second approach, we follow Kandola et al. (2002), who proposed a general method for learning the α_i by solving a optimization problem to maximize the kernel alignment between Gram matrix of $K(\alpha)$ and target yy',

$$A(S, K(\alpha), yy') = \frac{y' K(\alpha) y}{|y| ||K(\alpha)||},$$

yielding a new kernel K_{opt}. Here, one introduces a regularization parameter λ to constrain $||\alpha||$ and prevent over-alignment; the optimization then amounts to a quadratic programming problem that can be solved through standard methods. We now pick 6 kernels with relatively good performance and low pairwise kernel alignment as components for the new kernel – $(5,1)$-mismatch, $(6,4)$-gappy, $(4,6.0)$-substitution, $(5,7.5)$-substitution, $(6,9.0)$-substitution and $(5,2,1.0)$-wildcard – and repeat the second set of SCOP experiments with these two linear combination kernels. For K_{opt}, we use a regularization parameter of the form $\lambda = \frac{\sigma}{N^2} \sum_{ij} \langle K_i, K_j \rangle$, where $\langle \cdot, \cdot \rangle$ is the Hilbert-Schmidt inner product between matrices. We found that performance varied slightly but significantly as we varied $\sigma = .001, .01, .1, 1, 10, 100, 1000$ (results not shown); since the experiments do not contain a cross-validation set, we simply report the performance of the best parameter choice ($\sigma = .01$) with the caveat that this result may be somewhat optimistic. We report the mean ROC and ROC-50 scores across 54 experiments for the simple case K_{eq}, and the optimal alignment case K_{opt} in Table 3. We found that K_{opt} with the best regularization parameter choice does achieve significant improvement over the best individual kernel (indeed, almost all regularization parameters that we tried displayed some advantage over the best individual kernel); however, the simple weighting used in K_{eq} slightly outperformed K_{opt} in these experiments. Interestingly, for most of 54 experiments, K_{opt} ($\sigma = 0.01$) had non-zero weights only for the two best performing kernels, the $(4,6.0)$-substitution and $(5,2,1.0)$-wildcard kernels, with the weight for the latter about an order of magnitude smaller than that of the former. These results suggest that some of the kernels are complementary to each other and that combining them can help improve performance, though it appears that optimal alignment does not outperform a simple uniform weighting scheme for combining kernels.

6. Discussion

We have presented a number of different k-mer based string kernels that capture a notion of inexact matching – through use of gaps, probabilistic substitutions, and wildcards – but maintain fast computation time. Using a recursive function based on a trie data structure, we show that for all our new kernels, the time to compute a kernel value $K(x,y)$ is $O(c_K(|x|+|y|))$, where the constant c_K depends on the parameters of the kernel but not on the size of the alphabet Σ. Thus we improve on the constant factor involved in computation of the previously presented mismatch kernel, in which $|\Sigma|$ as well as k and m control the size of the mismatch neighborhood and hence the constant c_K.

We also show how many of our kernels can be obtained through the recently presented transducer formalism of rational $T \circ T^{-1}$ kernels and give the transducer T for several examples. This connection gives an intuitive understanding of the kernel definitions and could inspire new string kernels.

Finally, we present results on two benchmark SCOP data sets for the remote protein homology detection problem and show that many of the new, faster kernels achieve performance comparable to the mismatch kernel. We also investigate how kernel performance depends on parameter choice for the different inexact matching models. Intuitively, it is clear that the only biological reasonable choices involve short k-mer features, since as we allow k to grow, we cannot permit sufficient inexact matching without also introducing noise. However, within these constraints, our results demonstrate the somewhat different behavior of the various kernel families.

We note that Vishwanathan and Smola (2002) used counting statistics and a suffix tree construction to eliminate the constant factor of k in computation time for the exact-matching spectrum kernel (Leslie et al., 2002a). It may be possible to extend this technique to the fast inexact-matching kernels presented here.

A promising direction for applied work in this area is combining string kernel representations with semi-supervised approaches for leveraging the abundant unlabeled protein sequence data (sequences whose 3D structure is unknown) available in sequence databases. One recent approach is presented by Weston et al. (2003), where string kernels are used as a base kernel representation, and unlabeled sequence data together with a dissimilarity measure between sequence examples are used to build *cluster kernels* that modify the base kernel for a richer representation. In more recent work (Kuang et al., 2004), we define k-mer based string kernels for probabilistic sequence profiles (Gribskov et al., 1987), which also give a richer representation of sequences by estimating position specific residue emission probabilities from unlabeled data. These profile-based string kernels provide another promising semi-supervised approach for kernel representation of protein sequence data.

Acknowledgments

We would like to thank Eleazar Eskin, Risi Kondor and William Stafford Noble for helpful discussions and Corinna Cortes, Patrick Haffner and Mehryar Mohri for explaining their transducer formalism to us. This work is supported by an Award in Informatics from the PhRMA Foundation, NIH grant LM07276-02, and NSF grant ITR-0312706.

References

S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

C. Cortes, P. Haffner, and M. Mohri. Rational kernels. *Neural Information Processing Systems 16*, 2002.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge, 2000.

N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola. On kernel-target alignment. In *Neural Information Processing Systems*, volume 15, 2001.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis*. Cambridge UP, 1998.

E. Eskin, W. S. Noble, Y. Singer, and S. Snir. A unified approach for sequence prediction using sparse sequence models. Technical report, Hebrew University, 2003.

M. Gribskov, A. D. McLachlan, and D. Eisenberg. Profile analysis: Detection of distantly related proteins. *PNAS*, pages 4355–4358, 1987.

M. Gribskov and N. L. Robinson. Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching. *Computers and Chemistry*, 20(1):25–33, 1996.

D. Haussler. Convolution kernels on discrete structure. Technical report, UC Santa Cruz, 1999.

S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *PNAS*, 89: 10915–10919, 1992.

T. Jaakkola, M. Diekhans, and D. Haussler. Using the Fisher kernel method to detect remote protein homologies. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158. AAAI Press, 1999.

J. Kandola, J. Shawe-Taylor, and N. Cristianini. Optimizing kernel alignment over combinations of kernels. NeoroCOLTTechnicalReport NC-TR-2002-121, http://www.neurocolt.org, 2002.

R. Kuang, E. Ie, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C. Leslie. Profile-based string kernels for remote homology detection and motif extraction. In *Computational Systems Bioinformatics*, 2004.

C. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 2004.

C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. *Proceedings of the Pacific Biocomputing Symposium*, 2002a.

C. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for SVM protein classification. *Neural Information Processing Systems 16*, 2002b.

C. Leslie and R. Kuang. Fast kernels for inexact string matching. *Proceedings of COLT/Kernel Workshop*, 2003.

C. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. *Proceedings of the Sixth Annual International Conference on Research in Computational Molecular Biology*, 2002.

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

A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.

M. Sagot. Spelling approximate or repeated motifs using a suffix tree. *Lecture Notes in Computer Science*, 1380:111–127, 1998.

S. L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1:371–328, 1997.

R. M. Schwartz and M. O. Dayhoff. Matrices for detecing distant relationships. In *Atlas of Protein Sequence and Structure*, pages 353–358, Silver Spring, MD, 1978. National Biomedical Research Foundation.

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

J.-P. Vert, H. Saigo, and T. Akutsu. *Kernel Methods in Computational Biology*, chapter Local alignment kernels for biological sequences. MIT Press, 2004.

S. V. N. Vishwanathan and A. Smola. Fast kernels for string and tree matching. *Neural Information Processing Systems 16*, 2002.

M. S. Waterman, J. Joyce, and M. Eggert. *Computer alignment of sequences*, chapter Phylogenetic Analysis of DNA Sequences. Oxford, 1991.

C. Watkins. Dynamic alignment kernels. Technical report, UL Royal Holloway, 1999.

J. Weston, C. Leslie, D. Zhou, A. Elisseeff, and W. S. Noble. Cluster kernels for semi-supervised protein classification. *Neural Information Processing Systems 17*, 2003.