Fast Kernels for Inexact String Matching

Christina Leslie and Rui Kuang

Columbia University, New York NY 10027, USA
{rkuang,cleslie}@cs.columbia.edu

Abstract. We introduce several new families of string kernels designed in particular for use with support vector machines (SVMs) for classification of protein sequence data. These kernels – restricted gappy kernels, substitution kernels, and wildcard kernels – are based on feature spaces indexed by k-length subsequences from the string alphabet Σ (or the alphabet augmented by a wildcard character), and hence they are related to the recently presented (k, m)-mismatch kernel and 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. 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 mismatch kernel. We compute the kernels efficiently using a recursive function based on a trie data structure and relate our new kernels to the recently described transducer formalism. Finally, we report protein classification experiments on a benchmark SCOP dataset, 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.

Keywords: Kernel methods, string kernels, computational biology.

1 Introduction

Recently, there has been considerable interest in the development of string kernels for use with support vector machine classifiers and other kernel methods in applications like text categorization, speech recognition, and protein sequence classification. Previous work includes convolution kernels defined by Haussler [5], dynamic alignment kernels based on pair hidden Markov models by Watkins [15], and the gappy n-gram kernel developed for text classification by Lodhi *et al.* [11]. A practical disadvantage of these string kernels is their computational expense. Most of the 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 gave a linear time $(O(k(|x| + |y|))$ implementation of a kernel based on a trie data structure for use in SVM protein classification. Vishwanathan *et al.* [13] extended this work to compute the weighted sum of 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 $O(|x| + |y|)$. Finally, the (k, m)-mismatch kernel [9] achieved improved performance on the protein classification task by incorporating the biologically important notion of character mismatches. 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.

In this paper, we introduce several new families of string kernels designed for use with SVMs for classification of protein sequence data. These kernels – restricted gappy kernels, substitution kernels, and wildcard kernels – are based on feature spaces indexed by k-length subsequences from the string alphabet Σ (or the alphabet augmented by a wildcard character), and hence they are closely related to the (k, m)-mismatch kernel and 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. 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. Therefore, we provide a number of different models for incorporating a notion of inexact matching while maintaining fast computation. We describe how to compute these kernels efficiently using a recursive function based on a trie data structure. We also relate our new kernels to the recently described transducer formalism [2] and give transducers corresponding to some of our kernels.

Finally, we report protein classification experiments on a benchmark SCOP dataset, 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.

2 Definitions of Feature Maps and String Kernels

Below, we review the definition of mismatch kernels [9] and introduce three new families: restricted gappy kernels, substitution kernels, and wildcard kernels.

In each case, the kernel is defined via an explicit feature map 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.

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 $\alpha = a_1 a_2 \ldots a_k$, 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 [8] or k-gram kernel [11].

2.2 Restricted Gappy Kernels

For the (g, k)-gappy string kernel, 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 [11], we can define for g-mer α and k-mer feature $\beta = b_1 b_2 \ldots b_k$ the weighting

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

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:

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

This feature map is related to the gappy k-gram kernel defined in [11] 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 [11], 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

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

$0 < \lambda \leq 1$, 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 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 [6, 12, 1] that gives pairwise scores $s(a, b)$ derived from estimated evolutionary substitution probabilities. In one scoring system [12], 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 $\alpha = a_1 a_2 \ldots a_k$ as follows:

$$M_{(k,\sigma)}(\alpha) = \{\beta = b_1 b_2 \ldots b_k \in \Sigma^k : -\sum_i^k \log P(a_i|b_i) < \sigma\}$$

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 in [3].

3 Efficient Computation

All the kernels we define above can be efficiently computed using a trie data structure, similar to the mismatch tree approach previously presented [9]. 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.

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 $a_1 a_2 \ldots a_k$, 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.

Each g-mer instance in the input data can contribute to $\binom{g}{k} = O(g^{g-k})$ k-mer features (assuming that $g-k$ is smaller than k). Therefore, we visit at most $O(g^{g-k}(|x|+|y|)$ 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 $O(g^{g-k+1}(|x|+|y|))$.

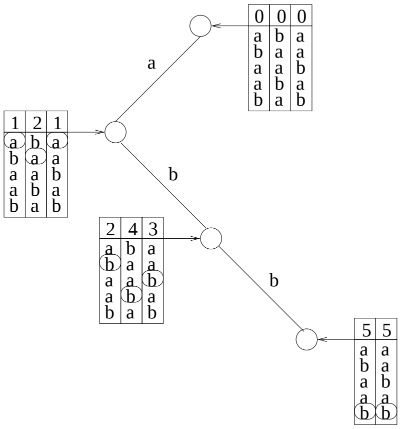


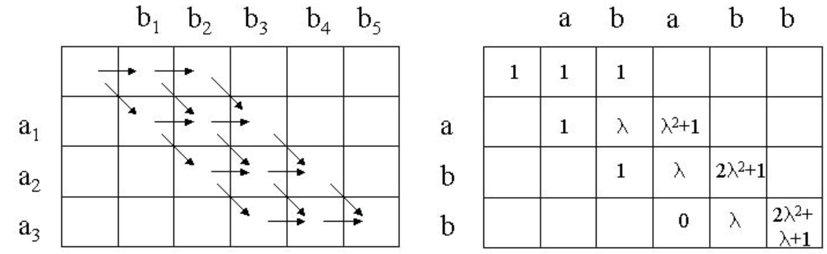
Fig. 1. Trie traversal for gappy kernel. Expansion along a path from root to leaf during traveral of the trie for the $(5, 3)$-gappy kernel, showing only the instance 5-mers for a single sequence $x = abaabab$. Each node stores its valid 5-mer instances and the index to the last match for each instance.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

The 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.

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 $b_1 b_2 \ldots b_d$, 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.



(A)

(B)

Fig. 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. Trellis (B) shows the example of aligning $ababb$ against 3-mer abb.

The number of leaf nodes visited is in the traversal is $O(N_\sigma(|x| + |y|))$, where $N_\sigma = \max_{\alpha \in \Sigma^k} |M_{(k,\sigma)}|$. We can choose σ sufficiently small to get any desired bound on N_σ. 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.* [2] 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 [2], 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 $E \subset Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{R}_+ \times Q$, 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 [2] 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 [11]. Figure 3 shows diagrams of the 3-gram transducer T_3 and 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.)

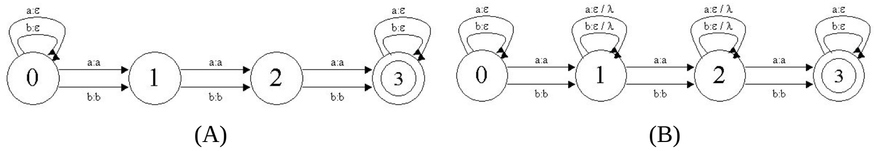


Fig. 3. The k-gram and gappy k-gram transducers. The diagrams show the 3-gram transducer (A) and the gappy 3-gram transducer (B) for a two-letter alphabet.

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.

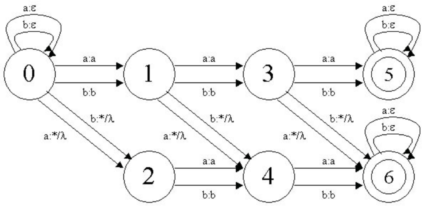


Fig. 4. The (k, m)-wildcard transducer. The diagram shows the $(3, 1)$-wildcard transducer for a two-letter alphabet.

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 $\alpha = a_1 a_2 \ldots a_k$ and emits $\beta = b_1 b_2 \ldots b_k$ if and only if every substitution satisfies $-\log P(a_i|b_i) < \sigma_o$.

5 Experiments

We tested all the new string kernels with SVM classifiers on a benchmark SCOP dataset from Jaakkola *et al.* [7], which is designed for the remote protein homology detection problem, in order to compare to results with the mismatch kernel reported in [9]. 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 [7].

We compare the SVM classification performance of the three new string kernels with both the mismatch kernel and the Fisher kernel of Jaakkola *et al.* [7]. 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 dataset prior to the mismatch-SVM approach, whose performance is as good as Fisher-SVM and better than all other standard methods tried [9].

We note that there is another successful feature representation for protein classification, the SVM-pairwise method presented in [10]. Here one uses an empirical kernel map based on pairwise Smith-Waterman [14] alignment scores

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

where x_i, $i = 1 \ldots m$, are the training sequences and $d(x_i, x)$ is the E-value for the alignment score between x and x_i. In the longer version of [9], we will show that the mismatch kernel used with an SVM classifier is competitive with SVM-pairwise on the smaller SCOP benchmark presented in [10]. For this reason, and because the SVM-pairwise feature map is expensive to compute on the larger SCOP dataset from [7] (each feature vector is $O(|x|^2 m)$, where m is the number of training sequences), we compare the new kernels only to the mismatch kernel and the Fisher kernel.

All methods are evaluated using the receiver operating characteristic (ROC) score, which is the area under the graph of the rate of true positives as a function of the rate of false positives as the threshold for the classifier varies [4]. 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.

5.1 Restricted Gappy Kernels

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.

5.2 Substitution Kernels

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

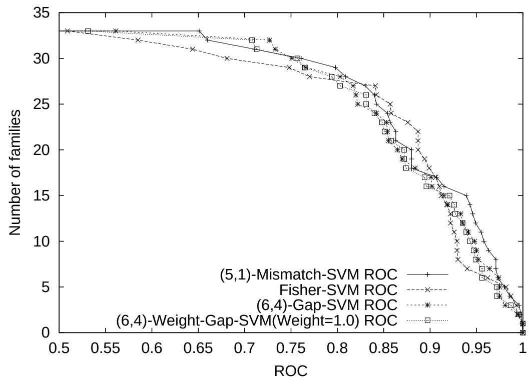


Fig. 5. Comparison of of Mismatch-SVM, Fisher-SVM and Gappy-SVM. The graph plots the total number of families for which a given method exceeds an ROC score threshold. The $(6, 4)$-Gap-SVM uses the unweighted gappy string kernel, in which an instance g-mer contributes a value of 1 to a k-mer feature if the k-mer occurs in it as a subsequence. The $(6, 4)$-Weight-Gap-SVM uses the weighted version of the gappy string kernel, which counts the total number alignments of a k-mer against a g-mer with multiplicative gap penalty of λ.

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.

5.3 Wildcard Kernels

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

6 Discussion

We have presented a number of different 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 the mismatch kernel computation, in which $|\Sigma|$ as well as k and m control the size of the mismatch neighborhood and hence the constant c_K.

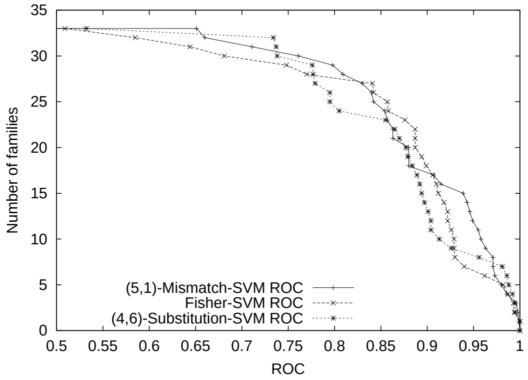


Fig. 6. Comparison of mismatch-SVM, Fisher-SVM and substitution-SVM. The graph plots the total number of families for which a given method exceeds an ROC score threshold.

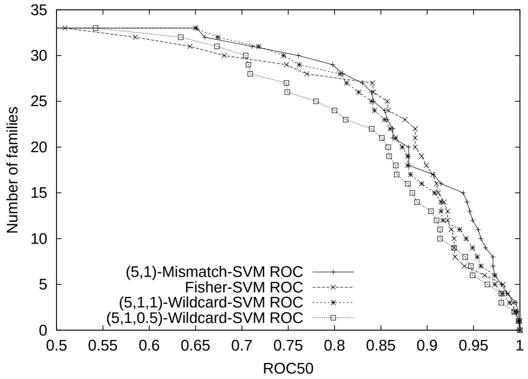


Fig. 7. Comparison of mismatch-SVM, Fisher-SVM and wildcard-SVM. The graph plots the total number of families for which a given method exceeds an ROC score threshold.

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 a benchmark SCOP dataset for the remote protein homology detection problem and show that many of the new, faster kernels achieve performance comparable to the mismatch kernel. Therefore, these new kernels seem promising for applications in computational biology and other domains involving learning from 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. CL is supported by an Award in Informatics from the PhRMA Foundation and by NIH grant LM07276-02.

References

1. 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.
2. C. Cortes, P. Haffner, and M. Mohri. Rational kernels. *Neural Information Processing Systems*, 2002.
3. 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.
4. 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.
5. D. Haussler. Convolution kernels on discrete structure. Technical report, UC Santa Cruz, 1999.
6. S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *PNAS*, 89:10915–10919, 1992.
7. 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.
8. C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. *Proceedings of the Pacific Biocomputing Symposium*, 2002.
9. C. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for SVM protein classification. *Neural Information Processing Systems 15*, 2002.
10. 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. To appear.
11. Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.
12. 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.
13. S. V. N. Vishwanathan and A. Smola. Fast kernels for string and tree matching. *Neural Information Processing Systems 15*, 2002.
14. M. S. Waterman, J. Joyce, and M. Eggert. *Computer alignment of sequences*, chapter Phylogenetic Analysis of DNA Sequences. Oxford, 1991.
15. C. Watkins. Dynamic alignment kernels. Technical report, UL Royal Holloway, 1999.