Efficient evaluation of large sequence kernels

Pavel P. Kuksa
Machine Learning Department
NEC Laboratories America, Inc
Princeton, NJ 08540
pkuksa@nec-labs.com

Vladimir Pavlovic
Department of Computer Science
Rutgers University
Piscataway, NJ 08854
vladimir@cs.rutgers.edu

ABSTRACT

Classification of sequences drawn from a finite alphabet using a family of string kernels with inexact matching (e.g., spectrum or mismatch) has shown great success in machine learning. However, selection of optimal mismatch kernels for a particular task is severely limited by inability to compute such kernels for long substrings with potentially many mismatches. In this work we introduce a new method that allows us to exactly evaluate kernels for large k, m and arbitrary alphabet size. The task can be accomplished by first solving the more tractable problem for small alphabets, and then trivially generalizing to any alphabet using a small liner system of equations. This makes it possible to explore a larger set of kernels with a wide range of kernel parameters, opening a possibility to better model selection and improved performance of the string kernels. To investigate the utility of large (k,m) string kernels, we consider several sequence classification problems, including protein remote homology detection, fold prediction, and music classification. Our results show that increased k-mer lengths with larger substitutions can improve classification performance.

Keywords

sequence classification, string kernels

1. INTRODUCTION

Analysis of large scale sequential data has become an important task in machine learning and data mining, inspired by applications such as biological sequence analysis, text and audio mining. Classification of string data, sequences of discrete symbols, has attracted particular interest and has led to a number of new algorithms [1, 7, 12, 19]. These algorithms often exhibit state-of-the-art performance on tasks such as protein superfamily and fold prediction, music genre classification and document topic elucidation. In particular, string kernel methods with inexact matching (e.g., mismatch) have been successful in solving challenging problems in bioinformatics, text mining, image categorization, music classification, etc (e.g. [15, 17, 16, 12, 7, 4]).

A family of state-of-the-art approaches to scoring similarity between pairs of sequences relies on fixed length, substring spectral representations and the notion of mismatch kernels, c.f. [7, 12]. There, a sequence is represented as the spectra (counts) of all short substrings (k-mers) contained within a sequence. The similarity score is established by exact or approximate matches of k-mers. Initial work, e.g., [12, 18], has demonstrated that this similarity can be computed using trie-based approaches in $O(k^{m+1}|\Sigma|^m(|X|+|Y|))$ time, for strings X and Y with symbols from alphabet Σ and up to m mismatches between k-mers.

More recently, [9] introduced linear time algorithms with *alphabet-independent* complexity $O(c_{k,m}(|X|+|Y|)$ applicable to computation of a large class of existing string kernels. In particular, for mismatch kernels, their computation for strings X and Y is based on cumulative pairwise comparison of all substrings α and β contained in X and Y, with the level of similarity of each pair of substrings (α, β) taken as the number of identical substrings their mutational neighborhoods $N_{k,m}(\alpha)$ and $N_{k,m}(\beta)$ give rise to ($N_{k,m}(\alpha)$ is the set of all k-mers that differ from α by at most m mismatches). This result however requires that the number of identical substrings in (k, m)-mutational neighborhoods of k-mers a and b (the intersection size) be known in advance, for every possible pair of m and the Hamming distance d between k-mers (k and $|\Sigma|$ are free variables). Obtaining the closed form expression for the intersection size for arbitrary k, m is challenging, with no clear systematic way of enumerating the intersection of two mutational neighborhoods. Closed form solutions obtained in [9] were only provided for cases when m is small ($m \leq 3$).

In this work we introduce a systematic and efficient procedure for obtaining intersection sizes (Sec. 3) that can be used for large k and m and arbitrary alphabet size $|\Sigma|$. This will allow us to effectively explore a much larger class of (k, m) kernels in the process of model selection. We will investigate performance impact of using large (k,m) and illustrate it on a number of sequence classification problems including detecting homology of remotely related proteins, multi-class fold prediction and classification of music samples (Sec. 4.1). We will show importance of kernel parameter selection in Sec. 4.2 and show that using large (k, m) kernels could further improve performance of the string kernel method.

2. BACKGROUND AND RELATED WORK

The key idea of basic string kernel methods is to apply a mapping $\Phi(\cdot)$ to map sequences of variable length into a fixed-dimensional vector space. In this space a standard kernel classifier, such as a support vector machine (SVM) [21], can then be applied. As SVMs require only inner products between examples in the feature space, rather than the feature vectors themselves, one can define a *string kernel* which computes the inner product in the feature space without explicitly computing the feature vectors:

$$K(X,Y) = \langle \Phi(X), \Phi(Y) \rangle, \quad (1)$$

where $X, Y \in D$, D is the set of all sequences composed of elements which take on a finite set of possible values from the alphabet Σ.

Sequence matching is frequently based on co-occurrence of exact sub-patterns (k-mers, features), as in spectrum kernels [11] or substring kernels [22]. Inexact comparison in this framework is typically achieved using different families of mismatch [12] or profile [7] kernels. Both spectrum-k and mismatch(k,m) kernel directly extract string features from the observed sequence, X. On the other hand, the profile kernel, proposed by Kuang *et al.* in [7], builds a profile [3] P_X and uses a similar $|\Sigma|^k$-dimensional representation, now derived from P_X. Constructing the profile for each sequence may not be practical in some application domains, since the size of the profile is dependent on the size of the alphabet set. While for bio-sequences $|\Sigma| = 4$ or 20, for music or text data $|\Sigma|$ can potentially be very large, on the order of tens of thousands of symbols.

Some of the most efficient available trie-based algorithms [12, 18] for mismatch kernels have a strong dependency on the size of alphabet set Σ and the number of allowed mismatches. Both need to be restricted in practice to control the complexity of matching algorithms. Under the trie-based framework, the list of k-mers extracted from given input strings is traversed in a depth-first search with branches corresponding to all possible $\sigma \in \Sigma$. Each leaf node at depth k corresponds to a particular k-mer feature (either exact or inexact instance of the observed exact string features) and contains a list of matching features from each string. The kernel matrix is updated at leaf nodes with corresponding matching feature counts. The complexity of the trie-based algorithm for mismatch kernel computation for two strings X and Y is $O(k^{m+1}|\Sigma|^m(|X|+|Y|))$ [12]. The algorithm complexity depends on the size of Σ, with possible substitutions explicitly drawn from Σ during the trie traversal. Consequently, to control the complexity of the algorithm we need to restrict the number of allowed mismatches (m), as well as the alphabet size ($|\Sigma|$). Typically, (k, m) are set to no more than $(5, 2)$ for alphabets of size $|\Sigma| = 20$.

Recent work in [9] introduced linear time algorithms with *alphabet-independent* complexity for Hamming-distance based matching. This enables efficient computation of a wide class of existing string kernels for datasets with large $|\Sigma|$. The authors show that it is possible to compute an inexact (k, m) kernel as

$$K(X,Y|m,k) = \sum_{a \in X} \sum_{b \in Y} \mathcal{I}(a,b) = \sum_{i=0}^{\min(2m,k)} M_i \mathcal{I}_i, \quad (2)$$

where $\mathcal{I}(a,b)$ is the number of common substrings in the intersection of the mutation neighborhoods of a and b, $\mathcal{I}_i$ is the size of the intersection of k-mer mutational neighborhood for Hamming distance i, and M_i is the number of observed k-mer pairs in X and Y having Hamming distance i. The algorithms introduced in [9], however, require as an input the intersection sizes $\mathcal{I}_i$ (i.e. number of identical substrings in the mutational neighborhoods of k-mers a and b) for all possible Hamming distances between a and b. No systematic way of obtaining these intersection sizes has been proposed in [9]. Closed-form solutions have been suggested for cases of small m ($m \leq 3$). For larger values of m ($m > 3$), it becomes substantially more difficult to combinatorially enumerate the intersection of the neighborhoods as the neighborhood grow exponentially with m. In the next section we address these issues and propose a systematic way of computing intersection sizes that can be used for larger k and m.

3. FINDING INTERSECTION SIZES FOR LARGE K AND M

For large values of k and m finding intersection sizes needed for kernel computation can be problematic. This is because while for smaller values of m combinatorial closed form solution can be found easily, for larger values of m finding it becomes more difficult due to an increase in the number of combinatorial possibilities as the mutational neighborhood increases (exponentially) in size. On the other hand, direct computation of the intersection by trie traversal algorithm is computationally difficult for large k and m as the complexity of traversal is $O(k^{m+1}|\Sigma|^m)$, i.e. is exponential in both k and m. The above mentioned issues do not allow for efficient kernel evaluation for large k and m. We will further discuss in what follows approaches to computing intersection sizes and propose an efficient method that can effectively compute intersection sizes for large k and m and allows to explore more complex kernels (with large k and m) that as we will show in Section 4.2 can further improve performance of the string kernel method.

The number of k-mers at the Hamming distance of at most m from both k-mers a and b, $\mathcal{I}(a,b)$, can be found in a weighted form

$$\mathcal{I}(a,b) = \sum_{i=0}^{m} w_i (|\Sigma| - 1)^i. \quad (3)$$

Coefficients w_i depend only on the Hamming distance $d(a,b)$ between k-mers a and b for fixed k, m, and $|\Sigma|$. The intersection size $\mathcal{I}$ and coefficients w_i can be precomputed for a given setting $(k, m, |\Sigma|)$ in a number of different ways. In the following, we discuss possible approaches for intersection size computation.

Brute-force. Directly computes intersection sizes for $0 \leq d \leq 2m$ by performing $2m$ trie traversals for k-mer pairs a and b over alphabet Σ at distances $d = 1 \ldots 2m$. This does not compute w_i explicitly. Brute-force trie traversal for computing intersection sizes is applicable only for moderate values of $|\Sigma|$, k, m.

Analytic (closed-form) solution. The set of coefficients w_i for every possible distance $d(a,b)$ and a fixed m can be found

in a closed form with k and $|\Sigma|$ as variables (see [9]). For example, when $m = 2$ the intersection sizes can be found in closed form as

$$\underset{(m=2)}{\mathcal{I}(a,b)} = \begin{cases} |N_{k,m}|, d(a,b) = 0 \\ 1 + k(|\Sigma| - 1) + (k-1)(|\Sigma| - 1)^2, d(a,b) = 1 \\ 1 + 2(k-1)(|\Sigma| - 1) + (|\Sigma| - 1)^2, d(a,b) = 2 \\ 6(|\Sigma| - 1), d(a,b) = 3 \\ \binom{4}{2}, d(a,b) = 4 \end{cases}$$

Once the closed form solution is found for every (m, d) pair, the intersection sizes for given k, $|\Sigma|$ can be found in constant time. However, it is difficult to obtain closed form solution for larger k, m. Currently, analytic solutions are known for $m < 4$, with no known solutions for $m \geq 4$.

Finding coefficients by solving linear systems of equations. It is possible to efficiently compute the intersection sizes by reducing $(k, m, |\Sigma|)$ intersection size problem to a set of less complex intersection size computations. We discuss this approach below.

3.1 Reduction-based computation of intersection size coefficients

For every Hamming distance $0 \leq d(a,b) \leq 2m$, the corresponding set of coefficients $w_i, i = 0, 1, \ldots, m$ can be found by solving a linear system $Aw = \mathcal{I}$ of $m + 1$ equations with each equation corresponding to a particular alphabet size $|\Sigma| \in \{2, 3, \ldots, m + 2\}$. The left-hand side matrix A is an $(m+1,m+1)$ matrix with elements $a_{ij} = (i + 1)^{j-1}$, $i = 1, \ldots, m + 1$, $j = 1, \ldots, m + 1$.

$$A = \begin{pmatrix} 1^0 & 1^1 & 1^2 & \ldots & 1^m \\ 2^0 & 2^1 & 2^2 & \ldots & 2^m \\ & & & \ldots & \\ (m+1)^0 & (m+1)^1 & (m+1)^2 & \ldots & (m+1)^m \end{pmatrix}$$

The right-hand side $\mathcal{I} = (I_0, I_1, \ldots, I_m)^T$ is a vector of intersection sizes for a particular setting of k, m, d, $|\Sigma| = 2, 3, \ldots, m + 2$. Here, $I_i, i = 0 \ldots m$ is the intersection size for a pair of k-mers over alphabet size $i + 2$. Note that I_i need only be computed for small alphabet sizes, up to $m + 2$. Hence, this vector can feasibly be computed using a trie traversal for a pair of k-mers at Hamming distance d even for moderately large k as the size of the trie is only $(m + 2)^k$ as opposed to $|\Sigma|^k$. This allows now to evaluate kernels for large k and m as the traversal is performed over much smaller tries, e.g., even in case of relatively small protein alphabet with $|\Sigma| = 20$, for $m = 6$ and $k = 13$, the size of the trie is $20^{13}/(6 + 2)^{13} = 149011$ times smaller. Coefficients w obtained by solving $Aw = \mathcal{I}$ do not depend on the alphabet size $|\Sigma|$. In other words, once found for a particular combination of values (k, m), these coefficients can be used to determine intersection sizes for any given finite alphabet $|\Sigma|$ using Eq. 3.

We summarize the intersection size computation in Algorithm 1. The algorithm receives as the input the problem parameters k, m, and $|\Sigma|$ and returns both the vector of $(\min(k, 2m) + 1)$ intersection sizes $\mathcal{I}_{k,m,|\Sigma|}$ (lookup table) for a given $(k, m, |\Sigma|)$-problem and the weight matrix $W_{k,m}$ that can be used to obtain intersection sizes for any alphabet size $|\Sigma|$. The overall complexity of the algorithm is $O((2m+1)k^{m+1}(m+2)^m)$. Compared to $O((2m+1)k^{m+1}|\Sigma|^m)$ complexity of computing intersection sizes directly using trie traversals, proposed algorithm has lower complexity by a factor of $\left(\frac{|\Sigma|}{m+2}\right)^m$ (as we show empirically in Sec. 4.3 this agrees with observed running time improvements).

Algorithm 1: Intersection size computation

Input: kernel parameters k and m, alphabet size $|\Sigma|$

1: **for** $d = 0$ to $\min(k, 2m)$ **do**
2: {for each Hamming distance d}
3: **for** $i = 2$ to $m + 2$ **do**
4: {for each alphabet size $i = 2 \ldots m + 2$}
5: Create a pair of k-mers a and b with Hamming distance $h(a, b) = d$
6: Obtain intersection size I_i by computing (k, m)-mismatch kernel $K_{k,m}(a, b)$ using trie traversal over i^k tree.
7: **end for**
8: Set $(m + 1) \times (m + 1)$ matrix A with $A_{ij} = (i + 1)^{(j-1)}$, $i = 1, \ldots, m + 1$, $j = 1, \ldots, m + 1$
9: Solve $Aw = I$ to obtain weight vector w
10: Set $(d + 1)$th row of $W_{k,m}$ to w
11: **end for**
12: Compute vector of intersection sizes $\mathcal{I}_{k,m,|\Sigma|} = W_{k,m}((|\Sigma| - 1)^0 \ldots (\Sigma - 1)^m)^T$

Output: Weight matrix $W_{k,m}$ of size $(\min(k, 2m) + 1, m + 1)$, vector of intersection sizes (lookup table) $\mathcal{I}_{k,m,|\Sigma|}$

Some examples of weight matrices $W_{k,m}$ for various settings of k and m are shown in the Appendix. Intersection sizes $\mathcal{I}$ can be obtained for a particular alphabet size $|\Sigma|$ by multiplying weight matrix $W_{k,m}$ by a vector of $(|\Sigma| - 1)$ powers as in Eq. 3, i.e.

$$\mathcal{I}_{k,m,|\Sigma|} = W_{k,m} \begin{pmatrix} |\Sigma|^0 & |\Sigma|^1 & \ldots & |\Sigma|^{m-1} \end{pmatrix}^T.$$

We note that weight matrices $W_{k,m}$ (steps 1-11 of Algorithm 1) can be pre-computed, and then once computed used to solve (k, m)-mismatch problems for a given alphabet Σ (i.e. weight matrices $W_{k,m}$ are *alphabet-size independent*).

4. EXPERIMENTS

We evaluate the utility of large (k, m) computations as a proxy for model selection, by allowing a significantly wider range of kernel parameters to be investigated during the selection process. In these evaluations we follow the experimental settings considered in [10] and [9].

We use three standard benchmark datasets: the SCOP dataset (7329 sequences, 54 experiments) [23] for remote protein homology detection, the Ding-Dubchak dataset[1] (27 folds, 694 seqs) [2, 4] for multi-class protein fold recognition, and music genre data[2] (10 genres, 1000 seqs) [13] for multi-class genre prediction. For remote protein homology experiments, we follow standard experimental setup used in previous studies [23] and evaluate average classification performance on

[1] `http://ranger.uta.edu/~chqding/bioinfo.html`
[2] `http://opihi.cs.uvic.ca/sound/genres`

54 remote homology experiments, each simulating the remote homology detection problem by training on a subset of families under the target superfamily and testing the superfamily classifier on the remaining (held out) families according to SCOP hierarchy (Figure 1). For music genre classification, we use vector quantization (VQ) to represent original sequences of 13-dim. MFCC vectors as strings over $|\Sigma| = 2048$ alphabet, and we use 5-fold cross-validation error to evaluate classification performance. For multi-class fold prediction, we use standard data splits as described in [2, 4]. Data and source code are available at the supplementary website [20].

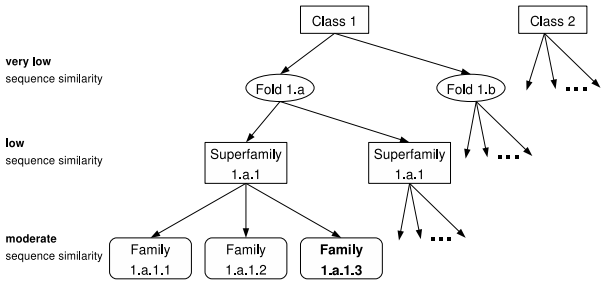

Figure 1: SCOP hierarchy. Protein domains organized into classes, folds, superfamilies, and families. Protein sequences from same superfamily but different families are considered remote homologs.

4.1 Large (k,m) performance evaluation

In this section, we investigate the impact of kernel parameters on the classification performance on the three sequence classification tasks. For each task, we evaluate classification performance over a large range of settings for k and m. Such large range evaluation is the first of its kind, made possible by our efficient kernel evaluation algorithm.

Evaluation measures. The methods are evaluated using 0-1 and top-q balanced error rates as well as F1 scores and precision and recall rates. Under the top-q error cost function, a classification is considered correct if the rank of the correct label, obtained by sorting all prediction confidences in non-increasing order, is at most q. On the other hand, under the balanced error cost function, the penalty of mis-classifying one sequence is inversely proportional to the number of sequences in the target class (i.e. mis-classifying a sequence from a class with a small number of examples results in a higher penalty compared to that of mis-classifying a sequence from a large, well represented class). We evaluate remote protein homology performance using standard Receiver Operating Characteristic (ROC) and ROC50 scores. The ROC50 score is the (normalized) area under the ROC curve computed for up to 50 false positives. With a small number of positive test sequences and a large number of negative test sequences, the ROC50 score is typically more indicative of the prediction accuracy of a homology detection method than the ROC score.

Results of mismatch kernel classification for the remote homology detection problem are shown in Table 1. We observe that larger values of k and m perform better compared to typically used values of k=5-6, m=1-2. For instance,

Table 1: Remote homology. Classification performance (mean ROC50) of the mismatch kernel method in the supervised setting

Kernel	Mean ROC	Mean ROC50
mismatch(5,1)	87.75	41.92
mismatch(5,2)	90.67	49.09
mismatch(6,2)	90.74	49.66
mismatch(6,3)	90.98	49.36
mismatch(7,3)	91.31	52.00
mismatch(7,4)	90.84	49.29
mismatch(9,3)	90.27	47.00
mismatch(9,4)	91.45	53.51
mismatch(10,4)	91.02	50.49
mismatch(10,5)	**91.60**	**53.78**
mismatch(13,6)	90.98	50.11

Table 2: Statistical significance of the differences between (k,m) kernels (p-values according to signed rank test). Kernels with large (k, m) perform better than the traditionally used (k, m) kernels

	(5,1)	(5,2)	(6,2)	(6,3)	(7,4)	(9,4)
(5,2)	3.4e-04					
(6,2)	3.7e-06	3.8e-01				
(6,3)	1.4e-03	7.2e-01	7.7e-01			
(7,4)	2.2e-03	7.0e-01	9.8e-01	8.7e-01		
(9,4)	1.6e-06	3.2e-03	7.3e-04	1.4e-02	2.9e-02	
(10,5)	3.1e-06	1.8e-03	1.5e-03	2.5e-03	9.4e-03	5.6e-01

(k=10,m=5)-mismatch kernel achieves significantly higher average ROC50 score of 53.78 compared to ROC50 of 41.92 and 49.09 for the (k=5,m=1)- and (k=5,m=2)- mismatch kernels (with p-values 3.1e-6 and 1.8e-3, respectively). Table 2 shows for each of the (k, m) kernel methods p-values of the Wilcoxon signed-rank test on the ROC50 scores against other (k,m) kernels. The utility of such large mismatch kernels was not possible to investigate prior to this study. As can also be seen from comparison with other state-of-the-art methods (including PSI-BLAST, Smith-Waterman similarity-based SVM [14], subsequence [15] and recently proposed spatial sample kernels (SSSK) [8] and sequence learner (SEQL) [5] approaches) in Table 3, large (k, m) mismatch kernels display state-of-the-art performance.

Table 3: Remote homology prediction. Comparison with state-of-the-art methods

method	ROC	ROC50
PSI-BLAST [7]	74.29	29.25
SVM-Fisher [6, 7]	75.66	31.90
SVM-Pairwise (Smith-Waterman) [14]	89.30	43.40
Subsequence kernel [15]	87.23	40.37
SSSK [8]	91.48	51.18
SEQL [5]	**92.20**	52.37
Mismatch(k=10,m=5)	91.60	**53.78**

On the multi-class remote fold prediction problem (Table 4), larger values of k tend to increase precision and result in higher F1 scores. For instance, top-5 F1 score of 84.00 for the $(k = 11, m = 6)$-mismatch kernel is higher than that of $(k = 5, m = 1)$ or $(k = 5, m = 2)$ mismatch kernels

Table 4: Multi-class remote fold prediction (Ding&Dubchak dataset). Classification performance of the mismatch kernel method in the supervised setting

Kernel	Error	Top-5 Error	Balanced Error	Top-5 Balanced Error	Recall	Top-5 Recall	Precision	Top-5 Precision	F1	Top-5 F1
mismatch(5,1)	51.17	22.19	53.22	28.65	46.78	71.35	90.52	95.50	61.69	81.68
mismatch(5,2)	**42.56**	19.32	**45.51**	22.60	**54.49**	77.40	67.18	84.68	60.17	80.88
mismatch(5,3)	47.78	23.24	49.69	24.94	50.31	75.06	57.81	77.09	53.80	76.06
mismatch(6,2)	46.74	19.06	50.85	24.55	49.15	75.45	89.64	95.24	63.49	**84.19**
mismatch(6,3)	43.34	19.84	46.74	**22.08**	53.26	**77.92**	65.40	83.77	58.71	80.74
mismatch(7,3)	43.86	18.80	47.21	22.68	52.79	77.32	86.46	93.85	**65.55**	**84.78**
mismatch(7,4)	43.60	20.37	47.15	22.68	52.85	77.32	63.92	81.98	57.86	79.58
mismatch(9,2)	63.71	34.46	63.65	43.94	36.35	56.06	**93.01**	96.88	52.27	71.02
mismatch(9,3)	57.96	30.29	61.51	40.76	38.49	59.24	91.51	96.01	54.19	73.27
mismatch(9,4)	54.57	21.15	57.42	29.00	42.58	71.00	91.56	96.44	58.13	81.79
mismatch(10,3)	63.97	32.64	63.84	43.05	36.16	56.95	**93.01**	**97.02**	52.07	71.77
mismatch(10,4)	57.96	28.72	61.53	39.21	38.47	60.79	91.36	96.00	54.14	74.44
mismatch(10,5)	51.70	19.58	55.65	26.90	44.35	73.10	91.76	96.55	59.80	83.20
mismatch(10,6)	**42.82**	19.06	47.28	**22.43**	52.72	**77.57**	76.32	89.43	62.36	83.08
mismatch(11,4)	62.92	30.81	64.08	42.36	35.92	57.64	92.28	96.90	51.72	72.28
mismatch(11,5)	57.70	27.68	61.37	38.53	38.63	61.47	91.32	96.01	54.29	74.95
mismatch(11,6)	48.56	**18.28**	53.33	25.28	46.67	74.72	90.80	95.91	61.65	**84.00**
mismatch(13,4)	65.80	34.20	65.40	44.97	34.60	55.03	**92.73**	96.91	50.40	70.19
mismatch(13,5)	64.75	32.11	64.68	44.05	35.32	55.95	**92.84**	**97.05**	51.18	70.98

with F1 scores of 81.68 and 80.88, respectively. For longer k increasing the maximum number of allowed mismatches m tends to increase recall rates, while keeping precision rates almost the same. For example, recall rates change from 36.16 to 52.72 for the $k = 10$ case as m changes from 3 to 6. We also note that using $m > k/2$ compared to $m \leq k/2$ can result in significant drops in precision rates, e.g. $(k = 10, m = 6)$-mismatch kernel has much lower precision rate of 76.32 compared to 91.76 of the $(k = 10, m = 5)$-mismatch kernel.

For the music genre classification task (Table 5), parameter combinations with moderately long k and larger values of m tend to perform better than kernels with small m. As can be seen from results, larger values of m are important for achieving good classification accuracy and outperform setting with small values of m. However, increasing (k,m) does not result in high performance gains, possibly because of significant reduction in recall for these large alphabet sequences.

Table 5: Multi-class music genre recognition. Classification performance of the mismatch method

Kernel	Error	Top-2 Error	F1	Top-2 F1
mismatch(5,1)	34.8±3.49	18.3	65.36	81.95
mismatch(5,2)	32.6±2.63	18.0	67.51	82.21
mismatch(6,3)	32.4±0.96	19.0	67.79	81.22
mismatch(7,4)	**31.1±2.07**	18.0	**68.96**	82.16
mismatch(9,3)	31.4±1.25	18.0	68.59	82.33
mismatch(9,4)	32.2±1.82	17.8	67.83	82.36
mismatch(10,3)	32.3±1.3	18.0	67.65	82.12
mismatch(10,4)	31.7±1.52	19.1	68.29	81.04

4.2 Class-specific kernel/parameter selection

In what follows, we discuss approaches for kernel parameter selection. In particular, we focus on a question of selecting mismatch kernel parameters k and m that would be appropriate for a task at hand. We note that reduced running time requirements of our algorithm open the possibility to consider various parameter selection strategies (validation, uniform, MKL) with a larger set of (k,m) kernels. The results presented here demonstrate that such large (k,m) kernels could lead to significant performance improvements.

4.2.1 Parameter selection by validation

One approach to the parameter selection is to select k and m that perform best on average according to the validation set. We demonstrate this approach on remote protein homology detection task within SCOP hierarchy. In SCOP, a manually curated protein data set, sequences are grouped into a tree hierarchy containing classes, folds, superfamilies, and families, from root to leaf (Fig. 1). As validation data is not available on the standard benchmark dataset, we split the original training set into two disjoint subsets, a now smaller training subset and a validation subset. Validation subset is obtained by holding out a single family (positive validation examples) and 20% of the negative training sequences. After training on a now smaller training subset, the performance of a particular combination of values k and m is measured on a validation set. We then select the best performing combination (k, m) (according to the validation set) as kernel parameters for the mismatch kernel.

We show results in Table 6 for various choices of k and m (results are average ROC and ROC50 scores for 54 experiments). One can observe that according to performance on the validation set larger values of k and m are preferred. For instance, on the validation set, mismatch-(k=9, m=4) and (k=10,m=5) achieve top two ROC50 scores of

Table 6: Remote homology. Classification performance (mean ROC and ROC50) of the mismatch kernel method in the supervised setting (with validation set)

Kernel	ROC (Test)	ROC50 (Test)	ROC (Validation)	ROC50 (Validation)
mismatch(5,1)	85.75	39.47	88.28	73.21
mismatch(5,2)	89.62	48.10	90.36	77.17
mismatch(6,2)	89.26	48.72	90.61	77.89
mismatch(6,3)	89.95	48.38	90.32	77.05
mismatch(7,3)	90.08	51.64	91.17	79.21
mismatch(7,4)	89.79	47.74	90.04	76.40
mismatch(9,3)	87.14	44.27	89.55	76.47
mismatch(9,4)	89.97	**52.59**	91.90	**81.57**
mismatch(10,4)	88.77	48.47	90.90	79.15
mismatch(10,5)	90.40	**52.96**	**92.34**	**81.30**
mismatch(11,5)	89.68	50.14	91.52	80.25
mismatch(11,6)	**90.58**	52.72	**92.38**	81.23
mismatch(13,5)	86.57	42.56	87.79	70.12

81.57 and 81.30 compared to 77.17 or 79.21 of traditionally used mismatch-(k=5,m=2) and (k=7,m=3). We also observe that best performing kernels according to the validation set also perform best on the test set. The top two kernels with (k=9,m=4) and (k=10,m=5) achieve highest average ROC50 scores of 52.59 and 52.96. Similar trends are observed with respect to ROC.

4.2.2 Uniform multiple kernel combination

Another approach for setting kernel parameters is to consider the kernel equivalent to a combination of multiple kernels, with each individual kernel corresponding to a particular choice of k and m. This approach does not select a particular value for k or m, it rather uses a set of (k, m) values in a uniform combination. This may be advantageous compared to selecting particular values for k and m as there might not be a single choice of parameter values which would work best in all cases, e.g., for all superfamilies. By not fixing the values of k and m, the classifier (SVM) can learn importance/usefulness of a particular choice of k and m.

We present selection of results on remote homology detection in Table 7 using a uniform mixture of mismatch kernels with k = 1...13 and m = 1...6. As evident from the results, the performance of the uniform mixture (column 1) is very similar to the performance of the best single kernel as found by the validation (Sec. 4.2.1, Table 6).

4.2.3 Multiple kernel learning

Using a weighted combination of multiple kernels may potentially give better performance compared to a uniform kernel combination. Here we show results for using multiple kernel learning (MKL) [24] on remote homology detection. We again use the same large pool of mismatch kernels k=5 . . . 13, m=1 . . . 6 and learn a multiple kernel combination for each of the 54 remote homology problems.

The results for multiple kernel learning on remote homology detection are shown in Table 7 (only a selection of the tasks is shown). While the resulting average ROC50 (column 2 in Table 7) are slightly lower compared to uniform combination (column 1), the results are similar and seem to suggest that for different superfamilies, a different setting of parameters k and m may be needed. This is also clear from the best-case results for the selection of parameter values based on the test set performance (last column in Table 7). We observe that best performing k and m differ across different superfamilies, suggesting the need for a kernel combination.

4.2.4 Comparison of kernel selection approaches

According to the SCOP hierarchy, (1) a single kernel can be selected for the entire hierarchy, (2) kernels can be selected at a superfamily level, i.e. on a per-superfamily basis, (3) kernels can be selected at a family level, i.e. on a per-family basis. Selection itself can be based on train, validation, or test performance. In Table 8 we summarize performance of various kernel selection strategies including validation-based kernel selection, best-case (test-based) selection, uniform kernel combination and multiple kernel learning. While multiple kernel learning selects kernel mixture at a family level, uniform kernel combination is the same for all superfamilies/families, i.e. it is similar to selecting a single best kernel based on average performance across all superfamilies/families. For superfamily-level selection, a single kernel is selected for a given superfamily based on the average performance across families belonging to the superfamily.

Table 8: Remote homology. Comparison of kernel selection approaches.

Method	Mean ROC50
Validation-based selection (single kernel)	52.59
Validation-based selection (per family)	47.43
Validation-based selection (per superfamily)	49.89
Test-based selection (single kernel)	53.78
Test-based selection (per family)	60.32
Test-based selection (per superfamily)	56.16
Uniform multiple kernel combination	52.11
Multiple kernel learning (per family)	50.78

As we noted before, results for best-case selection (i.e. for selecting best kernel per superfamily/family based on test set performance) suggest the need for per-superfamily/per-family parameter selection to improve the classification performance. The reduced running time requirements of our algorithm allows to consider various parameter selection strategies (validation, uniform, MKL) with a larger set of (k,m) kernels. Our results show (Tables 7, 8) that per-class selec-

Table 7: Remote homology detection. Kernel selection. Note that best choice of parameter values (k,m) is class-specific (i.e. depends on protein superfamily/family). Kernels with larger values k,m achieve higher prediction accuracy compared to settings with smaller k,and m.

Experiment	Uniform kernel	MKL top (k,m)	Test ROC50	Best k, m (based on validation set)	Test ROC50	Best k, m (based on test set)	Test ROC50
2.56.1.2	13.75	(13,6)	**37.75**	11, 6)	17.35	(7, 3)	18.11
1.27.1.2	**58.57**	(13,6)	51.71	(11, 6)	51.02	(7, 3)	58.60
1.4.1.2	**67.20**	(7,3)	57.6	(5, 3)	62.04	(7, 3)	73.47
3.2.1.5	**57.71**	(13,6)	52.86	(11, 6)	52.48	(13, 4)	73.47
2.28.1.3	**30.40**	(13,6)	27.20	(6, 2)	26.12	(5, 3)	30.61
7.3.6.1	89.11	(5,2)	**97.56**	(5, 3)	80.95	(11, 6)	93.20
2.1.1.2	80.62	(7,3)	**90.50**	(11, 4)	81.76	(11, 5)	88.65
2.38.4.5	**46.25**	(13,6)	33.0	(9, 2)	0.00	(5, 3)	56.89
1.36.1.2	26.00	(7,3)	**36.33**	(5, 3)	1.36	(10, 5)	38.78
1.41.1.2	96.00	(7,3)	**97.67**	(5, 1)	73.47	(11, 6)	98.98
Mean ROC50:	51.47		50.78		47.43		60.32

Table 9: Running time for kernel computation on music and protein sequence data

$(k, m, \lvert\Sigma\rvert)$	Running time (s)
(9,2,2048)	235
(9,3,2048)	373
(9,4,2048)	416
(10,2,20)	82
(10,3,20)	196
(10,5,20)	610

tion with large (k,m) could lead to significant improvements (average ROC50 53.78 -> 60.32). However, for practical application (e.g., parameter selection using validation approach) more data is necessary.

4.3 Running time

We measure the running time for full mismatch kernel matrix computations for SCOP and music genre datasets (i.e. we compute 7329×7329 and 1000×1000 kernel matrices). As can be seen from the running times in Table 9 our method allows efficient evaluation for large k, m kernels (running times are given for protein (7329 seqs) and music (1000 seqs) datasets). We note that the running times include time required for intersection size computations (Algorithm 1).

In Table 10, we report running time improvements of our proposed method for intersection size computation (Algorithm 1) over brute-force (trie-based) computation of intersection sizes. As can be seen from the table, ratios of running times T_{trie}/T_{linear} as a function of kernel parameters k,m, and alphabet size $|\Sigma|$ are as expected from theoretical analysis. For example, the obtained speed-up for computing intersection sizes for k=10,m=5,$|\Sigma|$=20 is 208x, while the expected running time ratio $\left(\frac{|\Sigma|}{m+2}\right)^m$ is $(20/(5+2))^5$=190. We also note that because of high $O(k^{m+1}|\Sigma|^m)$ computational complexity of finding intersection sizes using explicit trie traversal, for $m > 3$ and large alphabet $|\Sigma|$ running time is excessively long (for these cases only expected running time ratios are shown in Table 10).

Table 10: Intersection size computation. Running time ratio T_{trie}/T_{linear} (speed-up) of the trie-based method and proposed algorithm as a function of kernel parameters k, m, and alphabet size $|\Sigma|$. Observed speed-ups are on the order of $\left(\frac{|\Sigma|}{m+2}\right)^m$ as expected from theoretical analysis

(k,m)	$\lvert\Sigma\rvert = 20$	$\lvert\Sigma\rvert = 100$
(5,2)	6	254
(5,3)	27	5931
(6,2)	9	393
(6,3)	45	10118
(7,2)	13	571
(7,3)	57	12848
(7,4)	120	n/a / $8 \times 10^{4\dagger}$
(9,4)	132	n/a / $8 \times 10^{4\dagger}$
(9,5)	197	n/a / $6 \times 10^{5\dagger}$
(10,5)	208	n/a / $6 \times 10^{5\dagger}$

† expected running time ratio only due to excessive running time for trie-based method

5. CONCLUSIONS

In this work we proposed a new systematic method that allows evaluation of inexact string family kernels for long substrings k with large number of mismatches m. The method finds the intersection set sizes by explicitly computing them for small alphabet size $|\Sigma|$ and then generalizing this to arbitrary large alphabets. We show that this enables one to explore a larger set of kernels corresponding to a large range of kernel parameter values which as we demonstrate experimentally can further improve performance of the string kernels. We illustrate impact of the kernel parameters on the classification performance on a number of sequence classification tasks and evaluate a number of kernel/parameter selection strategies. We demonstrate improved performance on protein remote homology detection, multi-class fold prediction, and classification of music samples.

6. REFERENCES

[1] J. Cheng and P. Baldi. A machine learning information retrieval approach to protein fold recognition. *Bioinformatics*, 22(12):1456–1463, June 2006.

[2] C. H. Ding and I. Dubchak. Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics*, 17(4):349–358, 2001.

[3] M. Gribskov, A. McLachlan, and D. Eisenberg. Profile analysis: detection of distantly related proteins. *Proceedings of the National Academy of Sciences*,

84:4355–4358, 1987.
[4] E. Ie, J. Weston, W. S. Noble, and C. Leslie. Multi-class protein fold recognition using adaptive codes. In *ICML '05*, pages 329–336, New York, NY, USA, 2005. ACM.
[5] G. Ifrim and C. Wiuf. Bounded coordinate-descent for biological sequence classification in high dimensional predictor space. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 708–716, New York, NY, USA, 2011. ACM.
[6] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2):95–114, 2000.
[7] R. Kuang, E. Ie, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C. S. Leslie. Profile-based string kernels for remote homology detection and motif extraction. In *CSB*, pages 152–160, 2004.
[8] P. Kuksa, P.-H. Huang, and V. Pavlovic. Fast protein homology and fold detection with sparse spatial sample kernels. In *ICPR 2008*, 2008.
[9] P. Kuksa, P.-H. Huang, and V. Pavlovic. Scalable algorithms for string kernels with inexact matching. In *NIPS*, 2008.
[10] P. P. Kuksa and V. Pavlovic. Spatial representation for efficient sequence classification. In *ICPR*, 2010.
[11] C. S. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Pacific Symposium on Biocomputing*, pages 566–575, 2002.
[12] C. S. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for SVM protein classification. In *NIPS*, pages 1417–1424, 2002.
[13] T. Li, M. Ogihara, and Q. Li. A comparative study on content-based music genre classification. In *SIGIR '03*, pages 282–289, New York, NY, USA, 2003. ACM.
[14] L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *RECOMB*, pages 225–232, 2002.
[15] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, 2002.
[16] Z. Lu and H. Ip. Image categorization with spatial mismatch kernels. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 397 –404, june 2009.
[17] C. Saunders, D. R. Hardoon, J. Shawe-Taylor, and G. Widmer. Using string kernels to identify famous performers from their playing style. *Intell. Data Anal.*, 12:425–440, December 2008.
[18] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.
[19] S. Sonnenburg, G. Rätsch, and B. Schölkopf. Large scale genomic sequence SVM classifiers. In *ICML '05*, pages 848–855, New York, NY, USA, 2005.
[20] Supplementary data and code. `http://seqam.rutgers.edu/projects/bioinfo/largekmkernels`.
[21] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, September 1998.
[22] S. V. N. Vishwanathan and A. Smola. Fast kernels for string and tree matching. *Advances in Neural Information Processing Systems*, 15, 2002.
[23] J. Weston, C. Leslie, E. Ie, D. Zhou, A. Elisseeff, and W. S. Noble. Semi-supervised protein classification using cluster kernels. *Bioinformatics*, 21(15):3241–3247, 2005.
[24] Z. Xu, R. Jin, I. King, and M. R. Lyu. An extended level method for efficient multiple kernel learning. In *NIPS*, pages 1825–1832, 2008.

7. APPENDIX: EXAMPLES OF WEIGHT MATRICES

$$W_{6,3} = \begin{pmatrix} 1 & 6 & 15 & 20 \\ 1 & 6 & 15 & 10 \\ 1 & 6 & 21 & 4 \\ 1 & 3 & 21 & 1 \\ 2 & 12 & 12 & 0 \\ -10 & 30 & 0 & 0 \\ 20 & 0 & 0 & 0 \end{pmatrix}, \quad W_{7,3} = \begin{pmatrix} 1 & 7 & 21 & 35 \\ 1 & 7 & 21 & 15 \\ 1 & 7 & 31 & 5 \\ 1 & 3 & 27 & 1 \\ 2 & 18 & 12 & 0 \\ -10 & 30 & 0 & 0 \\ 20 & 0 & 0 & 0 \end{pmatrix},$$

$$W_{7,4} = \begin{pmatrix} 1 & 7 & 21 & 35 & 35 \\ 1 & 7 & 21 & 35 & 20 \\ 1 & 7 & 21 & 45 & 10 \\ 1 & 7 & 15 & 49 & 4 \\ 1 & 10 & 24 & 40 & 1 \\ 0 & -10 & 60 & 20 & 0 \\ 20 & -40 & 90 & 0 & 0 \\ -70 & 140 & 0 & 0 & 0 \end{pmatrix},$$

$$W_{9,4} = \begin{pmatrix} 1 & 9 & 36 & 84 & 126 \\ 1 & 9 & 36 & 84 & 56 \\ 1 & 9 & 36 & 119 & 21 \\ 1 & 9 & 21 & 109 & 6 \\ 1 & 14 & 66 & 64 & 1 \\ 0 & -30 & 120 & 20 & 0 \\ 20 & 0 & 90 & 0 & 0 \\ -70 & 140 & 0 & 0 & 0 \\ 70 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$W_{10,5} = \begin{pmatrix} 1 & 10 & 45 & 120 & 210 & 252 \\ 1 & 10 & 45 & 120 & 210 & 126 \\ 1 & 10 & 45 & 120 & 280 & 56 \\ 1 & 10 & 45 & 85 & 280 & 21 \\ 1 & 10 & 60 & 160 & 205 & 6 \\ 1 & 5 & -40 & 310 & 105 & 1 \\ 2 & 80 & -90 & 360 & 30 & 0 \\ -28 & -70 & 210 & 210 & 0 & 0 \\ 182 & -420 & 560 & 0 & 0 & 0 \\ -378 & 630 & 0 & 0 & 0 & 0 \\ 252 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$W_{11,6} = \begin{pmatrix} 1 & 11 & 55 & 165 & 330 & 462 & 462 \\ 1 & 11 & 55 & 165 & 330 & 462 & 252 \\ 1 & 11 & 55 & 165 & 330 & 588 & 126 \\ 1 & 11 & 55 & 165 & 260 & 616 & 56 \\ 1 & 11 & 55 & 200 & 365 & 511 & 21 \\ 1 & 11 & 40 & 20 & 665 & 331 & 6 \\ 1 & 16 & 215 & -230 & 915 & 156 & 1 \\ 0 & -98 & 140 & 70 & 840 & 42 & 0 \\ 42 & 322 & -910 & 1120 & 420 & 0 & 0 \\ -378 & 882 & -1260 & 1680 & 0 & 0 & 0 \\ 1302 & -3528 & 3150 & 0 & 0 & 0 & 0 \\ -1848 & 2772 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$