Large Scale Genomic Sequence SVM Classifiers

S. Sonnenburg[1], G. Rätsch[2], and B. Schölkopf[3]

[1] Fraunhofer Institute FIRST, Kekuléstr. 7, 12489 Berlin, Germany
[2] Friedrich Miescher Laboratory of the Max Planck Society, Spemannstr. 35, Tübingen, Germany
[3] Max Planck Institute for Biological Cybernetics, Spemannstr. 38, 72076, Tübingen, Germany

Abstract. In genomic sequence analysis tasks like splice site recognition or promoter identification, large amounts of training sequences are available, and indeed needed to achieve sufficiently high classification performances. In this work we study two recently proposed and successfully used kernels, namely the *Spectrum kernel* and the *Weighted Degree kernel* (WD). In particular, we suggest several extensions using Suffix Trees and modifications of an SMO-like SVM training algorithm in order to accelerate the training of the SVMs and their evaluation on test sequences. Our simulations show that for the spectrum kernel and WD kernel, large scale SVM training can be accelerated by factors of 20 and 5 times, respectively, while using much less memory (e.g. no kernel caching). The evaluation on new sequences is often several thousand times faster using the new techniques (depending on the number of Support Vectors). Our method allows us to train on sets as large as one million sequences.

1 Introduction

Support Vector Machines (SVMs) (cf. [3, 22, 4, 23]) have been successfully used to solve biological sequence analysis tasks (cf. [24, 17] and references therein). They employ a so-called kernel function $\mathrm{k}(\mathbf{s}_i, \mathbf{s}_j)$ which intuitively computes the similarity between two sequences $\mathbf{s}_i$ and $\mathbf{s}_j$. The result of SVM learning is a $\boldsymbol{\alpha}$-weighted linear combination of N kernel elements and the bias b:

$$f(\mathbf{s}) = \operatorname{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \,\mathrm{k}(\mathbf{s}_i, \mathbf{s}) + b\right).$$

When applying SVMs on non-vectorial data types such as sequences, we face the following dilemma: on the one hand, we often need huge datasets in order to achieve state of the art performances. On the other hand, we have to use nontrivial kernels in order to deal with the data in an appropriate way. These two goals can be in conflict; indeed, for SVMs, huge training sets are easiest to deal with using linear kernels, in which case one can work directly in the primal problem. The content of the paper is to strike the best possible balance in this conflict, for the case of sequence data. Five types of kernels have been proposed in order to deal with the discrete nature of biological sequences: (a) polynomial-like kernels (including the locality improved kernel; e.g. [29]), (b) kernels derived from probabilistic models (including the Fisher and TOP kernels; cf. [7, 26]), (c) alignment based kernels (e.g. SVM-Pairwise [14] and Local Alignment kernels [27]), (d) the spectrum and mismatch kernel considering all appearing K-mers in a sequence (independent of their position; cf. [12, 13, 28]) and (e) kernels such as the Weighted Degree kernel proposed in [20] which incorporate positional information when comparing two sequences (see also [15, 28] for related approaches).

In this work we aim at accelerating and improving two representatives of the latter two families of kernels, namely the Spectrum kernel and the Weighted Degree kernel. Both kernels have already been extensively studied, however, we report several novel ways to more efficiently compute those kernels using suffix trees. While the idea of using trees to optimize kernel computation has been proposed before [12, 28], we show in Section 3.2 that the newly proposed algorithms for instance for K-mer kernels with m mismatches can be computed $\mathcal{O}(|\Sigma|^m k^m)$ times faster during testing, where $|\Sigma|$ is the size of the alphabet. In Section 3.3 we show that the same idea can be applied to the Weighted Degree kernel leading to significant speedups. Moreover, we show in Section 4 how the trees can be exploited to drastically reduce training times of SVMs while using significantly less memory.

The rest of the paper is structured as follows: In Section 2 we briefly review the Spectrum, Mismatch and Weighted Degree Kernel. In Section 3 we propose and discuss several improvements and extensions of these kernels and describe a simple extension of SMO-like algorithms (such as SVM^{light}; cf. [9]) in Section 4. We conclude the paper with simulation experiments on up to one million training sequences in a splice site recognition task, illustrating the efficiency of the new algorithms (Section 5).

2 String Kernels for Sequence Analysis

2.1 The Spectrum Kernel

The spectrum kernel [12] implements the n-gram or bag-of-words kernel [8] as originally defined for text classification in the context of biological sequence analysis. The idea is to count how often a K-mer (a contiguous string of length K) is contained in the sequences $\mathbf{s}$ and $\mathbf{s}'$. Summing up the product of these counts for every possible K-mer (note that there are exponentially many) gives rise to the kernel value which formally is defined as follows: Let Σ be an alphabet and $\boldsymbol{u} \in \Sigma^K$ a K-mer and $\#\boldsymbol{u}(\mathbf{s})$ the number of occurrences of $\boldsymbol{u}$ in $\mathbf{s}$. Then the spectrum kernel is defined as the inner product of $\mathrm{k}(\mathbf{s}, \mathbf{s}') = \Phi(\mathbf{s}) \cdot \Phi(\mathbf{s}')$, where $\Phi(\mathbf{s}) = (\#\boldsymbol{u}(\mathbf{s}))_{\boldsymbol{u} \in \Sigma^K}$. Note that spectrum-like kernels cannot extract any positional information from the sequence which goes beyond the K-mer length. It is well suited for describing the content of a sequence but is less well suited for instance for analyzing signals where motifs may appear in a certain order. Note that spectrum-like kernels are capable of dealing with sequences with varying length.

The Spectrum kernel can be efficiently computed in $\mathcal{O}(K(|\mathbf{s}| + |\mathbf{s}'|))$ using suffix trees [12], where $|\mathbf{s}|$ denotes the length of sequence $\mathbf{s}$. An easier way to compute the kernel for two sequences $\mathbf{s}$ and $\mathbf{s}'$ is to separately extract and sort the N K-mers in each sequence, which can be done in a pre-processing step. Note that for instance DNA K-mers of length $K \leq 16$ can be efficiently represented as a 32-bit integer value. Then one iterates over all K-mers of sequences $\mathbf{s}$ and $\mathbf{s}'$ simultaneously and counts which K-mers appear in both sequences and sums up the product of their counts. The computational complexity of the kernel computation is $\mathcal{O}(\log(|\Sigma|)K(|\mathbf{s}| + |\mathbf{s}'|))$.

2.2 The Weighted Degree Kernel

The so-called *weighted degree* kernel efficiently computes similarities between sequences while taking positional information of k-mers into account. The main idea of the WD kernel is to count the (exact) co-occurrences of k-mers at corresponding positions in the two sequences to be compared. The *WD kernel of order* K compares two sequences $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ of length L by summing all contributions of k-mer matches of lengths $k \in \{1, \ldots, K\}$, weighted by coefficients β_k:

$$\mathrm{k}(\mathbf{s}_i, \mathbf{s}_j) = \sum_{k=1}^{K} \beta_k \sum_{l=1}^{L-k+1} \mathbf{I}(\boldsymbol{u}_{k,l}(\mathbf{s}_i) = \boldsymbol{u}_{k,l}(\mathbf{s}_j)). \tag{1}$$

Here, $\boldsymbol{u}_{k,l}(\mathbf{s})$ is the oligomer of length k starting at position l of the sequence $\mathbf{s}$ and $I(.)$ is the indicator function which evaluates to 1 when its argument is true and to 0 otherwise. It is not a-priory clear how to choose the weighting coefficients. For the task of splice site recognition [20] proposed to use $\beta_k = 2(K-k+1)/(K(K+1))$. Matching sub-strings are thus rewarded with a score depending on the length of the sub-string. Note that although in our case $\beta_{k+1} < \beta_k$, longer matches nevertheless contribute more strongly than shorter ones: this is due to the fact that each long match also implies several short matches, adding to the value of (1). Exploiting this knowledge allows for reformulation of the kernel using "block-weights" as will be discussed in Section 3.3.

In another approach the weighting coefficients can also be automatically determined in SVM-training using Multiple Kernel Learning (MKL) [25] where for a linear combination of kernels one learns a SVM solution $(\boldsymbol{\alpha}, b)$ and a kernel weighting $\boldsymbol{\beta}$ simultaneously. As the WD kernel can be written as a linear combination of subkernels [25] $\mathrm{k}(\mathbf{s}_i, \mathbf{s}_j) = \sum_{k=1}^{K} \beta_k \, \mathrm{k}_k(\mathbf{s}_i, \mathbf{s}_j)$ where each subkernel counts matches of length k $\mathrm{k}_k(\mathbf{s}_i, \mathbf{s}_j) = \sum_{l=1}^{L-k+1} \mathbf{I}(\boldsymbol{u}_{k,l}(\mathbf{s}_i) = \boldsymbol{u}_{k,l}(\mathbf{s}_j))$, one can apply MKL to also determine the weighting $\boldsymbol{\beta}$.

Note that the WD kernel can be understood as a Spectrum kernel where each position is treated independently from the others. Moreover, it does not only consider oligomers of length exactly K, but also all shorter matches. Hence, the feature space for each position has $\sum_{k=1}^{K} |\Sigma|^k = \frac{|\Sigma|^{K+1}-1}{|\Sigma|-1} - 1$ and additionally duplicated L times (i.e. leading to $\mathcal{O}(L|\Sigma|^K)$ dimensions). However, the computational complexity of the WD kernel is in the worst case $\mathcal{O}(KL)$ as can be directly seen from (1).

3 Faster String Kernels and Extensions

3.1 Efficient Storage of Sparse Weights

All considered kernels correspond to a feature space that can be huge. For instance in the case of the WD kernel on DNA sequences of length 100 with $K = 20$, the corresponding feature space is 10^{14} dimensional. However, most

dimensions in the feature space are not used since only a few of the many different k-mers actually appear in the sequences. In this section we briefly discuss three methods to efficiently deal with sparse vectors $\boldsymbol{v}$. We assume that the elements of the vector $\boldsymbol{v}$ are indexed by some index set $\mathcal{U}$ (for sequences, e.g. $\mathcal{U} = \Sigma^K$) and that we only need three operations: `clear`, `add` and `lookup`. The first operation sets the vector $\boldsymbol{v}$ to zero, the `add` operation increases the weight of a dimension for an element $\boldsymbol{u} \in \mathcal{U}$ by some amount α, i.e. $v_{\boldsymbol{u}} = v_{\boldsymbol{u}} + \alpha$ and `lookup` requests the value $v_{\boldsymbol{u}}$. The latter two operations need to be performed as quickly as possible (whereas the performance of the `lookup` operation is of higher importance).

Explicit Map If the dimensionality of the feature space is small enough, then one might consider keeping the whole vector $\boldsymbol{v}$ in memory and to perform direct operations on its elements. Then each read or write operation is $\mathcal{O}(1)$.[4] This approach has expensive memory requirements ($\mathcal{O}(|\Sigma|^K)$), but is very fast and best suited for instance for the Spectrum kernel on DNA sequences with $K \leq 14$ and on protein sequences with $K \leq 6$.

Sorted Arrays More memory efficient but computationally more expensive are sorted arrays of index-value pairs $(\boldsymbol{u}, v_{\boldsymbol{u}})$. Assuming the L indices are given and sorted in advance, one can efficiently change or look up a single $v_{\boldsymbol{u}}$ for a corresponding $\boldsymbol{u}$ by employing a binary search procedure ($\mathcal{O}(\log(L))$). When given L' look up indexes at once, one may sort them in advance and then simultaneously traverse the two arrays in order to determine which elements appear in the first array (i.e. $\mathcal{O}(L + L')$ operations – omitting the sorting of the second array – instead of $\mathcal{O}(\log(L)L')$). This method is well suited for cases where L and L' are of comparable size, as for instance for computations of single Spectrum kernel elements (as proposed in [13]).

Suffix Trees If the number of non-zero elements in the vector v becomes very large, then the Sorted Arrays method become infeasible. If furthermore the dimensionality of the index set is too large to use the Explicit Mapping, then we need suffix trees in order to introduce a structure over the non-zero weights that allows fast insertion and look up of elements. The idea is to use a tree with at most $|\Sigma|$ siblings of depth K. The leaves store a single value: the element $v_{\boldsymbol{u}}$, where $\boldsymbol{u} \in \Sigma^K$ is a K-mer and the path to the leaf corresponds to $\boldsymbol{u}$. To `add` or `lookup` an element one only needs K operations to reach a leaf of the tree (and to create neccessary nodes on the way in an `add` operation). Note that the computational complexity of the operations is independent of the number of K-mers/elements stored in the tree. On the other hand, a tree has a considerably larger storage overhead compared with for instance Sorted Arrays, as each node needs to store pointers to its parent and siblings.

3.2 Spectrum Kernel with Mismatches

When considering long K-mers, the probability that exactly the same K-mer appears in another sequence drops to zero very fast. Therefore, it can be advantageous (depending on the problem at hand) to consider not only exact matches but also matches with a few mismatching positions. [13] proposed to use the following kernel:

$$\mathrm{k}(\mathbf{s}, \mathbf{s}') = \langle \Phi_m(\mathbf{s}), \Phi_m(\mathbf{s}') \rangle$$

where $\Phi_m(\mathbf{s}) = \sum_{\boldsymbol{u} \in \mathbf{s}} \Phi_m(\boldsymbol{u})$, $\Phi_m(\boldsymbol{u}) = (\phi_\sigma(\boldsymbol{u}))_{\sigma \in \Sigma^K}$, where $\phi_\sigma(\boldsymbol{u}) = 1$ if σ mismatches with $\boldsymbol{u}$ in at most m positions and zero otherwise. This kernel is equivalent to

$$\mathrm{k}_{2m}(\mathbf{s}, \mathbf{s}') = \sum_{\boldsymbol{u} \in \mathbf{s}} \sum_{\boldsymbol{u}' \in \mathbf{s}'} \Delta_{2m}(\boldsymbol{u}, \boldsymbol{u}'), \tag{2}$$

where $\Delta_{2m}(\boldsymbol{u}, \boldsymbol{u}') = 1$ if $\boldsymbol{u}$ mismatches $\boldsymbol{u}'$ in at most $2m$ positions. Note that if $m = 1$ then one already considers matches of k-mers which mismatch in *two* positions.[5] [13] proposed a suffix tree based algorithm that computes a single kernel element in $\mathcal{O}(K^{m+1}|\Sigma|^m(|\mathbf{s}| + |\mathbf{s}'|))$. While we cannot improve the single kernel computation, we will show that it is possible to compute N dot products between $\mathbf{s}$ with N sequences $\mathbf{s}_1, \ldots, \mathbf{s}_N$ of length L in $\mathcal{O}(KNL)$ after a preparation of a tree which needs $\mathcal{O}(K^{2m+1}|\Sigma|^{2m}|\mathbf{s}|)$ operations. The idea is to add for each $\boldsymbol{u} \in \mathbf{s}$ all

[4] More precisely, it is $\log K$, but for small enough K (which we have to assume anyway) the computational effort is exactly one memory access.

[5] By using the formulation (2) one may of course also consider the case with at most one mismatch (i.e. $m = \frac{1}{2}$). While this kernel is empirically positive definite, it is theoretically not clear whether it always has this property.

$\binom{K}{2m}(|\Sigma|-1)^{2m}$ oligomers of length K to the tree which mismatch with $\boldsymbol{u}$ in at most $2m$ positions. After the tree construction, a single `lookup` operation only takes K operations (finding the right leaf) and therefore it only takes $\mathcal{O}(KNL)$ to perform NL `lookup` operations. Note, however, that the resulting tree may become huge for larger m, i.e. only at the expense of increased memory usage we achieve a considerable speedup.

Additionally note that one can drastically speedup the computation of a linear combination of kernels (for instance in testing), i.e.

$$g(\mathbf{s}) = \sum_{i \in I} \alpha_i \, \mathrm{k}(\mathbf{s}_i, \mathbf{s}),$$

where I is some index set (for instance the set of support vectors). One simply follows the above recipe for each $\boldsymbol{u} \in \mathbf{s}_i$ ($i \in I$) and adds the corresponding α_i to the value at the leaf addressed by $\boldsymbol{u}$. Then the evaluation of $g(\mathbf{s})$ only needs $\mathcal{O}(KL)$ operations per test example, while the generation of the tree needs $\mathcal{O}(|I|K^{2m+1}|\Sigma|^{2m}|\mathbf{s}|)$ operations.

3.3 Faster WD Kernel Computations

Identification of Blocks In the weighting scheme (1) higher-order matches seem to get lower weights, which appears counter-intuitive. Note, however, that a k-mer contains two $(k-1)$-mers, three $(k-2)$-mers etc. Hence, a block of

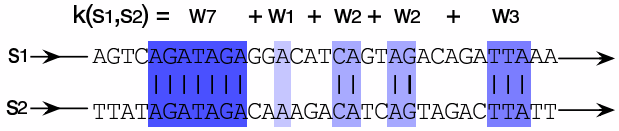

Fig. 1. Given two sequences $\mathbf{s}_1$ and $\mathbf{s}_2$ of equal length, the kernel consists of a weighted sum to which each match in the sequences makes a contribution w_B depending on its length B, where longer matches contribute more significantly.

length k contains $k-b+1$ blocks of length b. We can make use of this finding and reformulate the kernel. Instead of counting all matches of length $1, 2, \ldots, K$ one moves along the sequence only weighting the longest matching block (and not the smaller ones contained within, c.f. Figure 1) using different weights $\boldsymbol{w}$ which can be computed from the original weights as follows: For matches of length B with $B \leq K$ the "block weights" w_B are given by

$$\begin{aligned} w_B &= \sum_{b=1}^{B} m(b) \frac{2(K-b+1)}{K(K+1)} = \sum_{b=1}^{B} (B+1-b) \frac{2(K-b+1)}{K(K+1)} \\ &= \frac{B(-B^2+3K\cdot B+3K+1)}{3K(K+1)} \end{aligned}$$

where $m(b)$ is the number of times blocks of length b fit within blocks of length B. When the length of the matching block is larger than the maximal degree, i.e. $B > K$, the block weights are given by:

$$w_B = \sum_{b=1}^{B} m(b) \frac{2(K-b+1)}{K(K+1)} = \frac{3B-K+1}{3}$$

To compute the kernel one determines the longest matches between the sequences $\mathbf{s}$ and $\mathbf{s}'$ and adds up their corresponding weights. This requires only L steps reducing the computational complexity to $\mathcal{O}(L)$. For illustration, Figure 2 displays the weighting w_B for different block lengths B at fixed K: longer matching blocks get increased weights; while the first few weights up to $b = K$ increase polynomially higher order weights increase only linearly. Note that in real world data sets very few higher order matches exists and thus the speedup to (1) can be less than K.

3.4 Suffix Trees

While we cannot hope to further improve a single kernel evaluation (which is already $\mathcal{O}(L)$), it turns out to be possible to drastically speedup the computation of a linear combination of kernels, i.e. $g(\mathbf{s}) = \sum_{i \in I} \alpha_i \, \mathrm{k}(\mathbf{s}_i, \mathbf{s})$, where I is

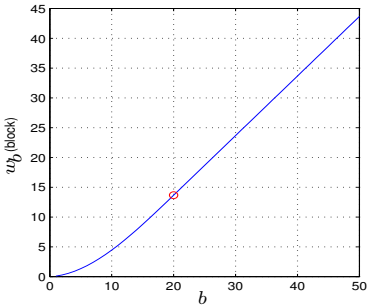

Fig. 2. Illustration of the block weights where the maximum match-length was set to $K = 20$ and the sequence length to $N = 50$. The circle marks the switch from polynomial to linear growth in terms of the weights.

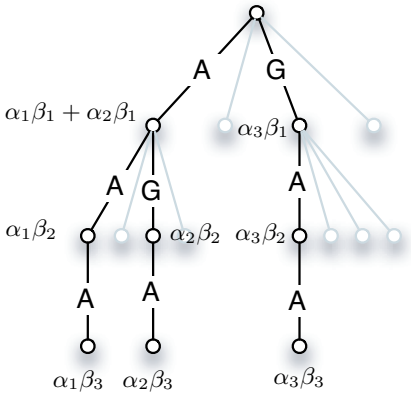

Fig. 3. Three sequences AAA, AGA, GAA beeing added to the tree. The plot displays the resulting weights at the nodes.

the index set. The idea is to create a suffix tree for each position $l = 1, \ldots, L$ of the sequence as done before for the Spectrum kernel. The main difference is that the WD kernel not only considers K-mers but also k-mers with $k \leq K$. We therefore propose to attach weights not only to the leaves of the tree but also to internal nodes, allowing an efficient storage for $k < K$. Now we may add all k-mers ($k = 1, \ldots, K$) of $\mathbf{s}_i$ ($i \in I$) starting at position l to the tree associated with position l (using weight $\alpha_i\beta_k$; operations per position: $\mathcal{O}(K|I|)$). Then the `lookup` algorithm for sub-sequences $\boldsymbol{u}$ starting at position l of $\mathbf{s}$ traverses down the tree associated with position l (following the path defined by $\boldsymbol{u}$) and adds all weights along the way (stopping when no children exists), see Figure 3. Note that we now can compute g in $\mathcal{O}(LK)$ operations (compared to $\mathcal{O}(|I|LK)$ in the original formulation).

3.5 WD Kernel with Improved Positional Invariance

The WD kernel works well for problems where the position of motifs are approximately constant in the sequence or when sufficiently many training examples are available. However, if for instance the sequence was shifted by only one position (cf. Figure 1), then potentially existing matches would not be found anymore. We therefore extend the WD kernel in order to find sequence motifs which are less precisely localized. Our proposed kernel lies in between the completely position dependent WD kernel and kernels like the spectrum kernel which does not use positional information.

The kernel with shifts is defined as

$$\mathrm{k}(\mathbf{s}_i, \mathbf{s}_j) = \sum_{k=1}^{K} \beta_k \sum_{l=1}^{L-k+1} \sum_{\substack{s=0 \\ s+l \leq L}}^{S(l)} \delta_s \, \mu_{k,l,s,\mathbf{s}_i,\mathbf{s}_j}, \tag{3}$$

$$\mu_{k,l,s,\mathbf{s}_i,\mathbf{s}_j} = \mathbf{I}(\boldsymbol{u}_{k,l+s}(\mathbf{s}_i) = \boldsymbol{u}_{k,l}(\mathbf{s}_j)) + \mathbf{I}(\boldsymbol{u}_{k,l}(\mathbf{s}_i) = \boldsymbol{u}_{k,l+s}(\mathbf{s}_j)),$$

where β_j is as before, δ_s is the weight assigned to shifts (in either direction) of extent s, and $S(l)$ determines the shift range at position l. Depending on the problem at hand, one may use $\delta_s = 1/(2(s+1))$ and $S(l) = \gamma|p-l|$, where p is the position of the signal of interest: the further away a motif is from the site, the less precisely it needs to be localized in order to contribute to the kernel value. We successfully applied this kernel for detection of alternative splice events [21]. See [15] for a different approach of improving the positional invariance of predictions applied to the problem of prokaryotic translation initiation sites recognition.

From a mathematical point of view, it is important to ask the question whether this kernel is positive definite. Suppose T is a shift operator, and Φ is the map associated with the zero-shift kernel k. Then the kernel $\tilde{\mathrm{k}}(\mathbf{s},\mathbf{s}') := \langle \Phi(\mathbf{s}) + \Phi(T\mathbf{s}), \Phi(\mathbf{s}') + \Phi(T\mathbf{s}')\rangle$ is trivially positive definite. On the other hand, we have $\tilde{\mathrm{k}}(\mathbf{s},\mathbf{s}') = \langle \Phi(\mathbf{s}), \Phi(\mathbf{s}')\rangle + \langle \Phi(T\mathbf{s}), \Phi(T\mathbf{s}')\rangle + \langle \Phi(T\mathbf{s}), \Phi(\mathbf{s}')\rangle + \langle \Phi(\mathbf{s}), \Phi(T\mathbf{s}')\rangle = \mathrm{k}(\mathbf{s},\mathbf{s}') + \mathrm{k}(T\mathbf{s},T\mathbf{s}') + \mathrm{k}(T\mathbf{s},\mathbf{s}') + \mathrm{k}(\mathbf{s},T\mathbf{s}')$. Assuming that we may discard edge effects, $\mathrm{k}(T\mathbf{s},T\mathbf{s}')$ is identical to $\mathrm{k}(\mathbf{s},\mathbf{s}')$; we then know that $2\,\mathrm{k}(\mathbf{s},\mathbf{s}') + \mathrm{k}(T\mathbf{s},\mathbf{s}') + \mathrm{k}(\mathbf{s},T\mathbf{s}')$ is positive definite. Our kernel (3), however, is a linear combination, with positive coefficients, of kernels of this type, albeit multiplied with different constants δ_s. The above arguments show that if δ_0 is at least twice as large as the sum of the remaining δ_s, the kernel will be positive definite. In our experience, δ_0 does not in all cases satisfy this condition. Nevertheless, we have always found the kernel to be positive definite on the given training data, i.e., leading to positive definite matrices, and thus posing no difficulties for the SVM optimizer.[6]

Note that the proposed WD kernel with shifts can be implemented using the previously used suffix trees leading to faster computations of linear combinations of kernel elements.

3.6 WD Kernel with Mismatches

Finally, we briefly discuss the extension of the WD kernel to consider mismatching k-mers. We propose to use the following kernel

$$\mathrm{k}(\mathbf{s}_i,\mathbf{s}_j) = \sum_{k=1}^{K}\sum_{m=0}^{M}\beta_{k,m}\sum_{l=1}^{L-k+1}\mathbf{I}(\boldsymbol{u}_{k,l}(\mathbf{s}_i) \neq_m \boldsymbol{u}_{k,l}(\mathbf{s}_j)), \tag{4}$$

where $\boldsymbol{u} \neq_m \boldsymbol{u}'$ evaluates to true if and only if there are exactly m mismatches between $\boldsymbol{u}$ and $\boldsymbol{u}'$. When considering $\mathrm{k}(\boldsymbol{u},\boldsymbol{u}')$ as a function of $\boldsymbol{u}'$, then one would wish that full matches are fully counted while mismatching $\boldsymbol{u}'$ sequences should be less influential, in particular for a large number of mismatches. If we choose $\beta_{k,m} = \beta_k/\left(\binom{k}{m}(|\Sigma|-1)^m\right)$ for $k > m$ and zero otherwise, then an m-mismatch gets the full weight divided by the number of possible m-mismatching k-mers, which seems a reasonable choice. Note that this kernel can be implemented such that its computation only needs $\mathcal{O}(LK)$ operations (instead of $\mathcal{O}(MLK)$). This kernel has been successfully used in a siRNA efficacy prediction task [19].

As discussed in Sections 3.2 and 3.3, it is possible to adapt the ideas developed for the Spectrum kernel in order to generate a tree in $\mathcal{O}((|\Sigma|-1)^m\binom{K}{m})$ operations per position that has the property that a single `lookup` operation ($\mathcal{O}(K)$) is necessary in order to compute the kernel between some fixed $\boldsymbol{u}$ and another $\boldsymbol{u}'$. We therefore omit details of the algorithm.

4 Speeding up SVM Training

It is not feasible to use standard optimization tools (e.g. MINOS, CPLEX, LOQO) for solving the SVM training problems on data sets containing more than a few thousand examples. So-called decomposition techniques overcome

[6] One may construct weighting schemes that enforce positive definiteness by a simple generalization of the $\Phi(\mathbf{s}) + \Phi(T\mathbf{s})$ construction. Suppose we have a kernel k with a feature map ϕ, and a transformation group T_n whose elements are indexed by $n \in \mathbb{Z}$, satisfying $T_nT_m = T_{n+m}$, and acting on $\mathcal{X}$, the domain where the data live. Then we can define a positive definite transformation kernel of order d as $\tilde{k}(x,x') :=< \tilde{\Phi}(x), \tilde{\Phi}(x') >$, where $\tilde{\Phi}(x) := \sum_{n=0}^{d} g_n\Phi(T_nx)$. Here, the $g_n \in \mathbb{R}$ are arbitrary weighting coefficients. A positive g_n ensures more invariance of order n, a negative g_n ensures less invariance. This leads to a kernel $\tilde{k}(x,x') = \sum_{n=0}^{d}\sum_{m=0}^{d} g_ng_mk(T_nx,T_mx)$. As above, we disregard edge effects, assuming $k(T_nx,T_mx') = k(T_{n-m}x,x') = k(x,T_{m-n}x')$. A short calculation of all terms contributing to a given $0 \leq \delta := |n-m| \leq d$ yields $\tilde{k}(x,x') = (\sum_{n=0}^{d} g_n^2)k(x,x') + \sum_{\delta=1}^{d}\sum_{n=0}^{d-\delta} g_ng_{n+\delta}[k(T_\delta x,x') + k(x,T_\delta x')]$. Under the above assumptions, all such kernels (with arbitrary $g_n \in \mathbb{R}$) are positive definite. In our experiments, we found that empirically, the kernels we used were always positive definite on the given data, independent of the above considerations.

this limitation by exploiting the special structure of the SVM problem. The key idea of decomposition is to freeze all but a small number of optimization variables (*working set*) and to solve a sequence of constant-size problems (subproblems of the SVM dual quadratic optimization problem [3]).

The general idea of the Sequential Minimal Optimization (SMO) algorithm has been proposed by [18] and is implemented in many SVM software packages. While [18] used $Q = 2$ as a working set size, other implementations such as SVM^{light} [9] typically uses larger values (e.g. $Q = 40$). The SVM optimization algorithm internally needs the output $\hat{f}_j = \sum_i \alpha_i y_i \, \mathrm{k}(\mathbf{s}_i, \mathbf{s}_j)$ for all training examples in order to select the next variables for optimization [9]. In order to update $\hat{f}_j$ one needs to compute full rows j of the kernel matrix for every changed α_j. One typically uses kernel-caching to reduce the computational effort of this operation, which is, however, in case of large scale simulations not efficient enough.[7] Fortunately, for the considered string kernels we can efficiently compute linear combinations of kernel elements. Using the techniques described in Sections 3.2 and 3.3 we generate for instance suffix trees such that the computation of $g(\mathbf{s}) = \sum_{q=1}^{Q} (\alpha_{i_q} - \alpha_{i_q}^{old}) y_{i_q} \, \mathrm{k}(\mathbf{s}_{i_q}, \mathbf{s})$ becomes more efficient as shown in Algorithm 1. When

Algorithm 1 Outline of the Linadd SMO-like algorithm that exploits the fast computations of linear combinations of kernels (e.g. by suffix trees).

{INITIALIZATION}
$f_i = 0, \alpha_i = 0$ for $i = 1, \ldots, N$

{LOOP UNTIL CONVERGENCE}
for $t = 1, 2, \ldots$ **do**
Check optimality conditions and stop if optimal; select Q variables $i_1, \ldots, i_Q$ based on $\mathbf{f}$ and $\boldsymbol{\alpha}$

$$\boldsymbol{\alpha}^{old} = \boldsymbol{\alpha}$$

solve SVM dual w.r.t. the selected variables and update $\boldsymbol{\alpha}$ generate data structures to prepare efficient computation of

$$g(\mathbf{s}) = \sum_{q=1}^{Q} (\alpha_{i_q} - \alpha_{i_q}^{old}) y_{i_q} \, \mathrm{k}(\mathbf{s}_{i_q}, \mathbf{s})$$

and update

$$f_i = f_i + g(\mathbf{s}_i) \text{ for all } i = 1, \ldots, N$$

end for

using the WD kernel this leads to a speedup of a factor of Q, in case of the Spectrum kernel with mismatches it can be considerably higher. Note that creating the suffix tree(s) on Q examples can be expensive, however, it is a fixed cost (given that Q is fixed) per iteration. If the number of examples is large enough, then the speedup of the evaluation when using trees will eventually lead to an advantage.

Finally note that most time is spent in evaluating $g(\mathbf{s})$ for all training examples. When using suffix trees, one may perform parallel `lookup` operations using several shared memory CPUs, speeding up computations. Moreover, this situation is almost ideal to distribute this part of the computations to many CPUs (little communication while large chunks of computations can be done independently).

5 Results and Discussion

5.1 Speed Comparison

Experimental Setup To demonstrate the effect of the several proposed algorithmic optimizations, namely the WD block formulation and the Linadd-SMO SVM training Algorithm 1 extension for the WD, the Spectrum and the Mismatch-WD kernel, we applied each of the algorithms to a real world splice site data, comparing it to the original

[7] For instance when using a million examples one can only fit 125 rows into 1 GB. Moreover, caching 125 rows is insufficient when for instance having many thousands of active variables.

WD formulation and the case where the weighting coefficients where learning using Multiple Kernel Learning. The splice data set contains 1,026,036 acceptor splice site sequences each 201 base pairs long. See Appendix A for more details on data generation. We trained SVMs using SVM^{light} [9] on 500, 1000, 5000, 10000, 30000, 50000, 100000, 200000, 500000 and 10^6 randomly sub-sampled examples and measured the time needed in SVM training. We set the degree parameter to $K = 20$ for the WD kernel and to $K = 8$ for the spectrum kernel fixing the SVMs regularization parameter to $C = 10$. SVM^{light}'s subproblem size (parameter `qpsize`) and convergence criterion (parameter `epsilon`) were initially set to $Q = 41$ and $\epsilon = 10^{-5}$, respectively, while a kernel cache of 1GB was used for all kernels except the precomputed kernel and algorithms using the Linadd-SMO extension for which the kernel-cache was disabled. Later on we measure how changing ϵ and the quadratic subproblem size Q influences SVM training time and accuracy. Experiments were performed on a PC powered by a 2.4GHz AMD Opteron(tm) Processor running Linux. We measured the training time for each of the algorithms and data set sizes.

WD Kernel Algorithm Comparison The obtained training times for the Weighted Degree Kernel are displayed in Table 1 and in Figure 4. These SVMs were trained using the different kernel algorithms: First the kernel matrix was precomputed using the standard WD kernel implementation (Pre). The training time including the time needed to pre-compute the full kernel matrix as presented is in all cases larger than the times obtained using the original WD kernel demonstrating the effectiveness of SVM^{light}'s kernel cache. The block-formulation of the WD kernel, although theoretically K times faster only leads to a further 70% speedup which is due to the very few higher order matches between two DNA sequences in the training set. Note that starting from 10,000 (30,000) examples Linadd-SMO optimization becomes more efficient than the original (blockwise) WD kernel algorithm as at the same time the kernel cache cannot hold all kernel elements.[8] In the case of one million of examples the Linadd formulation outperforms the original WD kernel by a factor of 5. Finally training with the original WD kernel with a sample size of 1,000,000 takes about 36 hours – which is even slower than the single mismatch WD kernel. Using Linadd, Multiple Kernel Learning of the weighting coefficients on a million of examples is still feasible as it takes less than 14 days. Note that the Linadd-SMO optimization using the original WD kernel is not significantly slower than the block-formulation using Linadd-SMO.

N	Pre	WD	Block	Linadd	LinB	MKL	WDMis	WDLinMis	Spec	LinSpec
500	0	1	**0**	3	3	4	**1**	14	1	**0**
1000	1	1	**1**	5	5	7	**3**	23	2	**1**
5000	28	19	**11**	24	23	82	**43**	102	36	**5**
10000	108	58	**33**	45	49	178	**131**	208	123	**17**
30000	965	317	177	**159**	**174**	1655	**703**	779	3462	**142**
50000	-	794	485	**355**	**312**	3525	1827	**1575**	9025	**423**
100000	-	2507	1576	**761**	**741**	15818	5464	**3932**	-	**2283**
200000	-	8863	5226	**2024**	**2031**	58261	18524	**10317**	-	**11673**
500000	-	40632	23946	**9119**	**9071**	299782	-	42361	-	-
1000000	-	131379	84737	**26107**	**26085**	1188204	-	129247	-	-

Table 1. (left) Speed Comparison of the original Weighted Degree Kernel algorithm (WD) in SVM^{light} training, compared to a precomputed version (Pre), its blockwise formulation (Block) the SMO Linadd extension used in conjunction with the original WD kernel (Linadd), its block formulation (LinB) and when determining the WD kernel weight by Multiple Kernel Learning (MKL). The first column shows the sample size N of the data set used in SVM training while the following columns display the time (measured in seconds) needed in the training phase. (right) Speed Comparison analoguous to (left) of a single mismatch Weighted Degree Kernel and the Spectrum Kernel, with (WDLinMis and LinSpec) and without (WDMis and Spec) the SMO Linadd extension.

WD Mismatch and Spectrum Kernel Comparison For the single mismatch WD and the spectrum kernel the SVM training times are listed in Table 1 and diagrammed in Figure 5.

[8] When double precision 8-byte floating point numbers are used, caching all kernel elements is possible when training with up to 11585 examples.

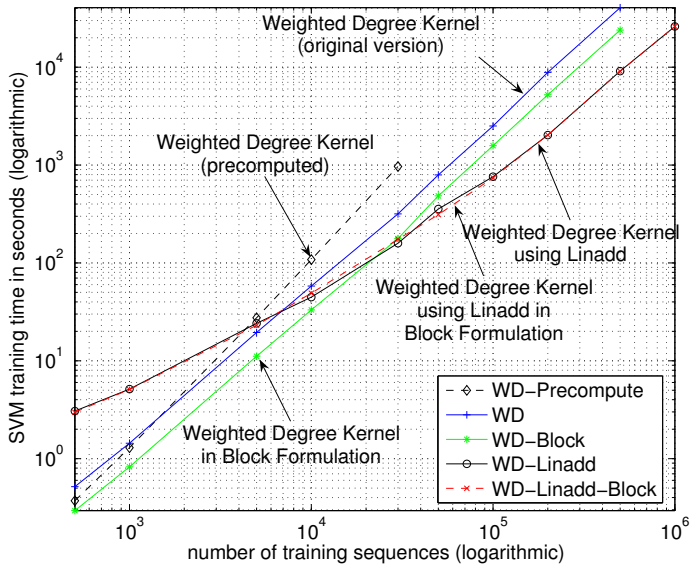

Fig. 4. Comparison of the running time of the different weighted degree kernel algorithms. Note that as this is a log-log plot small appearing distances are large for larger N and that each slope corresponds to a different exponent. The empirically determined complexity of the precomputed WD kernel is N^2, of the original WD kernel it is $N^{1.71}$, of the block formulation $N^{1.62}$ and for the Linadd-SMO variants $N^{1.55}$ (for $N > 10^5$).

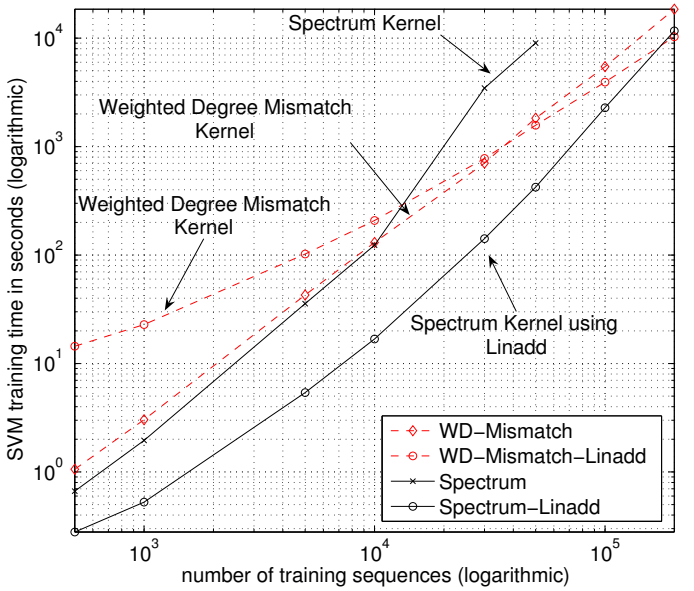

Fig. 5. In analogy to Figure 4: Comparison of the single mismatch WD and spectrum kernel with and without Linadd-SMO optimization. Empirically determined complexity of the mismatch WD is $N^{1.64}$. Complexity estimates for the mismatch WD kernel using linadd are $N^{1.3}$ and for spectrum kernel using linadd $N^{2.32}$. Note however that more data points are needed to give reliable estimates as the curves seem linear only for $N > 10000$ and $N > 200000$ respectively.

N	qpsize													
	11	21	31	41	51	71	81	91	101	111	121	131	141	151
10000	53	41	**39**	**39**	40	43	45	45	47	54	47	60	58	48
30000	230	175	**158**	180	**157**	**164**	183	168	194	211	181	226	222	205
50000	468	358	**309**	**313**	**303**	**319**	**316**	349	329	395	386	333	350	391
100000	1331	890	862	891	**819**	858	1049	**821**	867	874	861	**811**	826	830
200000	3700	2653	2456	2301	2321	**2234**	2349	**2284**	**2207**	2382	2366	2551	2418	2439
500000	15309	10587	9309	8989	9653	9581	9439	8879	9145	**8745**	8866	8903	9012	**8769**
1000000	46741	32686	30235	28022	28229	27458	26580	29197	26672	25902	28548	27068	**24970**	28956

Table 2. Influence on training time when varying the size of the quadratic program (qpsize) in SVM^{light}, when using the LinAdd+Block formulation of the WD kernel. While training times do not vary dramatically one still observes the tendency that with larger sample size a larger `qpsize` becomes optimal. The `qpsize` $= 41$ column displays the same result as column LinB in Tab.1, leaving aside a certain variance in running time.

SVM ϵ	0.000001	0.000010	0.000100	0.001000	0.010000	0.100000	0.500000
AUC	99.7322%	99.7322%	99.7322%	99.7322%	99.7322%	99.7317%	99.7206%
Training Time	2343	2168	2029	1719	1544	1233	1013

Table 3. Running time and Area Under the Curve (AUC) of the SVM classifier when trained on 200,000 and evaluated on the remaining 800,000 examples when varying the SVM^{light} `epsilon` parameter. While with increased ϵ training time decreases, the AUC seems to not change much.

Using the SMO optimization we gain speedups of 80% with respect to the mismatch WD kernel and even by a factor of 21 to the linear spectrum kernel.

Varying SVM^{lights}'s parameters `qpsize` *and* `epsilon` As discussed in Section 4 using the LinAdd algorithm computing the output for all training examples w.r.t. to some working set can be speed up by a factor of Q (i.e. the size of the quadratic subproblems, termed `qpsize` in SVM^{light}). However there is a trade-off in choosing Q as solving larger quadratic subproblems is expensive (quadratic to cubic effort). Table 2 discusses this issue. For example the gain in speed between choosing $Q = 11$ and $Q = 141$ for 1 million of examples is 87%, while it is much less for a choice in the mid-range, e.g. for $Q = 71$ about 7%. Sticking with a mid-range Q seems to be a good idea for this task. Still one observes the tendency that for larger training set sizes choosing larger quadratic subproblems is optimal. However large variance can be observed as SVM training time to a large extend depends on which Q variables are selected in each optimization step.

The story is similiar when varying ϵ (see Tab.3). Larger values of ϵ correspond to shorter running time, but this time at the cost of achieving a slightly lower area under the Receiver Operator Characteristic Curve [16, 2] (AUC). Surprisingly the choice of ϵ has very little effect, which can be explained by the fact that the influence due to ϵ depends on the scale in which the data lives, as in SVM^{light} the stopping criteria is the maximum violation of a misclassified example w.r.t. the regularization value C. Throughout the benchmarks the kernel value was not normalized to one and thus reaches its maximum values of $w_B \approx 195$ for $B = 201, K = 20$ on the diagonal. One therefore should either normalize k (e.g. by $\mathrm{k}'(\mathbf{s}, \mathbf{s}') = \frac{\mathrm{k}(\mathbf{s},\mathbf{s}')}{\sqrt{\mathrm{k}(\mathbf{s},\mathbf{s})\,\mathrm{k}(\mathbf{s}',\mathbf{s})}}$ which we omitted to enable a comparison with MKL), choose larger values of ϵ or even adaptively adjust the stopping criterion based on data scaling as has been done for nu-SVMs [11].

N	500	1000	5000	10000	30000	50000	100000	200000	500000	1000000
AUC	96.91%	97.82%	98.96%	99.28%	99.58%	99.65%	99.73%	99.80%	99.84%	99.87%
Relative AUC Improvement	-	29.45%	52.29%	30.77%	41.67%	16.67%	22.86%	25.93%	20.00%	18.75%
Test Error	6.03%	6.03%	3.38%	2.40%	1.57%	1.31%	1.07%	0.92%	0.83%	0.71%

Table 4. The achieved AUC and test error for the WD-SVM trained on 500 to 1,000,000 examples. Test Error (AUC) are steadily decreasing (increasing). After reaching 30,000 examples the relative improvement (i.e. the improvement relative to the previous result) remains at a level of $\approx 20\%$

Classification Performance In Table 4 the Test Error and AUC achieved on the splice site classification task for several sample sizes are shown[9]. With one million examples the method achieves 0.71% test error and 99.87% AUC. This is a relative improvement upon training on a 500 sample of 96%. Going from 500 to 5000 or from 5000 to $50,000$ examples leads to an improvement of 66% and the accuracy is still improved by 54% when increasing the sample size from $50,000$ to $500,000$.[10]

Conclusion We developed an efficient SMO-like SVM training algorithm, particularly well suited for string kernels like the Weighted Degree and Spectrum kernel and formulated linear time algorithms for kernel computation and SVM classifier prediction. Using the Spectrum, Weighted Degree and Mismatch Weighted Degree kernel in a large scale splice site recognition experiment with up to one million of sequences we demonstrated significant speedups while at the same time shrinking memory requirements (as kernel caching is not required). We show that SVM training is up to 20 times faster using the Linadd-SMO algorithm in combination with the spectrum and up to 5 times faster in combination with the WD kernel. For the WD kernel we developed a blockwise-formulation and extend it allowing for mismatches or making it invariant with respect to small positional sequence shifts, demonstrating its effectiveness on the splice site recognition task.

Acknowledgments

The authors gratefully acknowledge partial support from the PASCAL Network of Excellence (EU #506778), DFG grants JA 379 / 13-2 and MU 987/2-1.

References

1. M.S. Boguski and T.M. Lowe C.M. Tolstoshev. dbEST–database for "expressed sequence tags". *Nat Genet.*, 4(4):332–3, 1993.
2. A.P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.
3. C. Cortes and V.N. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.
4. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.
5. D.L. Wheeler et al. Database resources of the national center for biotechnology. *Nucl. Acids Res*, 31:38–33, 2003.
6. Harris, T.W. et al. Wormbase: a multi-species resource for nematode biology and genomics. *Nucl. Acids Res.*, 32, 2004. Database issue:D411-7.
7. T. Jaakkola, M. Diekhans, and D. Haussler. Using the Fisher kernel method to detect remote homologies. In T. Lengauer, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, H.-W. Mewes, and R. Zimmer, editors, *Intelligent Systems in Molecular Biology*, pages 149–158, 1999.
8. T. Joachims. Text categorization with support vector machines: Learning with many relevant features. Technical Report 23, LS VIII, University of Dortmund, 1997.
9. T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.
10. W.J. Kent. Blat–the blast-like alignment tool. *Genome Res.*, 12(4):656–64, 2002.
11. W. Kienzle and B. Schö:lkopf. A minimal primal dual method for support vector learning. In *Proceedings of the International Conference on Machine Learning, ICML*, 2005. (submitted).
12. C. Leslie, E. Eskin, and W.S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, Kaua'i, Hawaii, 2002.
13. C. Leslie, Rui Kuang, and E. Eskin. Inexact matching string kernels for protein classification. In *Kernel Methods in Computational Biology*, MIT Press series on Computational Molecular Biology, pages 95–112. MIT Press, 2003.
14. L. Liao and W.S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *Proceedings of the Sixth Annual International Conference on Research in Computational Molecular Biology*, pages 225–232, April 2002.
15. P. Meinicke, M. Tech, B. Morgenstern, and R. Merkl. Oligo kernels for datamining on biological sequences: A case study on prokaryotic translation initiation sites. *BMC Bioinformatics*, 5(169), 2004.

[9] For all sample sizes, testing was done on the same 26,036 examples

[10] Note that the degree of the WD kernel and the SVM-C were fixed to $K = 20$ and $C = 10$ throughout the experiments.

16. C.E. Metz. Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, VIII(4), October 1978.
17. K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001.
18. J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.
19. G. Rätsch and J.Q. Candela. Predicting siRNA efficacy. In *European Conference on Computational Biology, ECCB*, 2005. (submitted).
20. G. Rätsch and S. Sonnenburg. *Accurate Splice Site Prediction for Caenorhabditis Elegans*, pages 277–298. MIT Press series on Computational Molecular Biology. MIT Press, 2004.
21. G. Rätsch, S. Sonnenburg, and B. Schölkopf. Rase: Recognition of alternatively spliced exons in c. elegans. In *ISMB 2005*, 2005. (accepted).
22. B. Schölkopf. *Support vector learning*. Oldenbourg Verlag, Munich, 1997.
23. B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.
24. B. Schölkopf, K. Tsuda, and J.P. Vert, editors. *Kernel Methods in Computational Biology*. MIT Press series on Computational Molecular Biology. MIT Press, 2003.
25. S. Sonnenburg, G. Rätsch, and C. Schäfer. Learning interpretable svms for biological sequence classification. In *RECOMB 2005, LNBI 3500*, pages 389–407. Springer-Verlag Berlin Heidelberg, 2005.
26. K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K.R. Müller. A new discriminative kernel from probabilistic models. *Neural Computation*, 14(10):2397–414, 2002.
27. J.-P. Vert, H. Saigo, and T. Akutsu. Local alignment kernels for biological sequences. In *Kernel Methods in Computational Biology*, MIT Press series on Computational Molecular Biology, pages 131–154. MIT Press, 2003.
28. S.V.N. Vishwanathan and A.J. Smola. Fast kernels for string and tree matching. In *Kernel Methods in Computational Biology*, MIT Press series on Computational Molecular Biology, pages 113–130. MIT Press, 2003.
29. A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K.-R. Müller. Engineering Support Vector Machine Kernels That Recognize Translation Initiation Sites. *BioInformatics*, 16(9):799–807, September 2000.

A Splice Site Data Generation

EST and cDNA Sequences We collected all known *C. elegans* ESTs from Wormbase [6] (release WS118; 236,868 sequences), dbEST [1] (as of February 22, 2004; 231,096 sequences) and UniGene [5] (as of October 15, 2003; 91,480 sequences). Using *blat* [10] we aligned them against the genomic DNA (release WS118). The alignment was used to confirm exons and introns. We refined the alignment by correcting typical sequencing errors, for instance by removing minor insertions and deletions. If an intron did not exhibit the consensus `GT`/`AG` or `GC`/`AG` at the 5' and 3' ends, then we tried to achieve this by shifting the boundaries up to 2 nucleotides (nt). If this still did not lead to the consensus, then we split the sequence into two parts and considered each subsequence separately. For each sequence we determined the longest open reading frame (ORF) and only used the part of each sequence within the ORF. In a next step we merged alignments, if they did not disagree and shared at least one complete exon. This lead to a set of 135,239 unique EST-based sequences.

We repeated the above procedure with all known cDNAs from Wormbase (release WS118; 4,848 sequences) and UniGene (as of October 15, 2003; 1,231 sequences), which lead to 4,979 unique sequences. We removed all EST matches fully contained in the cDNA matches, leaving 109,693 EST-base sequences.

Clustering We clustered the sequences in order to obtain independent training, validation and test sets. In the beginning each of the above EST and cDNA sequences were in a separate cluster. We iteratively joined clusters, if any two sequences from distinct clusters (a) match to the genome at most 100nt apart (this includes many forms of alternative splicing) or (b) have more than 20% sequence overlap (at 90% identity, determined by using *blat*). We obtained 17,763 clusters with a total of 114,672 sequences. There are 3,857 clusters that contain at least one cDNA. Finally, we removed all clusters that showed alternative splicing.

Data Set generation From the cDNA clusters we generated windows of fixed length around true acceptor splice site sequences (59,968 of them). Each sequence is 201nt long and has the `AG` dimer at position 101-102. Negative examples were generated from any occurring `AG` within the ORF of the sequence (966,068 of them were found). Finally we obtain 1,026,036 sequences to be used throughout our experiments. Note that only 5.84% of them are positive examples.