Large Scale Genomic Sequence SVM Classifiers

Support Vector Machine, n-gram Kernel, Spectrum Kernel, Weighted Degree Kernel, Suffix Tree, Sequence Analysis

Sören Sonnenburg SOEREN.SONNENBURG@FIRST.FRAUNHOFER.DE
Fraunhofer Institute FIRST, Kekuléstr. 7, 12489 Berlin, Germany

Gunnar Rätsch GUNNAR.RAETSCH@TUEBINGEN.MPG.DE
Friedrich Miescher Laboratory of the Max Planck Society, Spemannstr. 37, 72076, Tübingen, Germany

Bernhard Schölkopf BERNHARD SCHOELKOPF@TUEBINGEN.MPG.DE
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 4 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. Cortes & Vapnik, 1995; Schölkopf, 1997; Cristianini & Shawe-Taylor, 2000; Schölkopf & Smola, 2002) have been successfully used to solve biological sequence analysis tasks (cf. Schölkopf et al., 2003; Müller et al., 2001 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:

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

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. Zien et al., 2000), (b) kernels derived from probabilistic models (including the Fisher and TOP kernels; cf. Jaakkola et al., 1999; Tsuda et al., 2002), (c) alignment based kernels (e.g. SVM-Pairwise (Liao & Noble, 2002) and Local Alignment kernels (Vert et al., 2003)), (d) the spectrum and mismatch kernel considering all appearing K-mers in a sequence (independent of their position; cf. Leslie et al., 2002; Leslie et al., 2003; Vishwanathan & Smola, 2003) and (e) kernels such as the Weighted Degree kernel proposed in Rätsch and Sonnenburg (2004) which incorporate positional information when comparing two sequences (see also Meinicke et al., 2004; Vishwanathan & Smola, 2003 for related

Appearing in *Proceedings of the 22^{st} International Conference on Machine Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

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 (Leslie et al., 2002; Vishwanathan & Smola, 2003), 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. Joachims, 1999) 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 (Leslie et al., 2002) implements the n-gram or bag-of-words kernel (Joachims, 1997) 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 (Leslie et al., 2002), 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:

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

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. For the weighting coefficients Rätsch and Sonnenburg (2004) proposed to use $\beta_k = 2(K-k+1)/(K(K+1))$. Matching substrings 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.

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. An appropriate choice of the data representation is crucial for fast algorithms. If the data can be efficientiently represented for the general class of kernels which can be written as the inner product of some sparse feature space, one achieves significant speedups in SVM *training* when using the in Section 4 proposed "Linadd" algorithm.

In this section we briefly discuss three methods to efficiently deal with sparse vectors $\boldsymbol{v}$. We suppose 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 $\mathcal{O}(1)$.[1] 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 (Leslie et al., 2003)).

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. Leslie et al. (2003) 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

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

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.[2] Leslie et al. (2003) 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

[1] 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.

[2] 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 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:

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

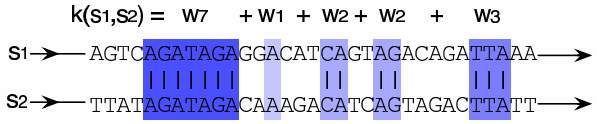


Figure 1. Given two sequences $\mathbf{s}_1$ and $\mathbf{s}_2$ of equal length, the kernel consists of a weighted sum to which each match in the sequences makes a contribution w_B depending on its length B, where longer matches contribute more significantly.

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 quadratically higher order weights increase only linearly.

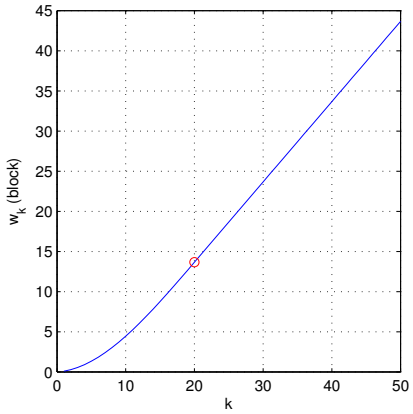


Figure 2. How the block weights in the Weighting Degree Kernel are chosen. In the figure the maximum match-length was set to $K = 20$ and the sequence length to $N = 50$. The circle marks the the switch from polynomial to linear growth in terms of the weights.

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 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

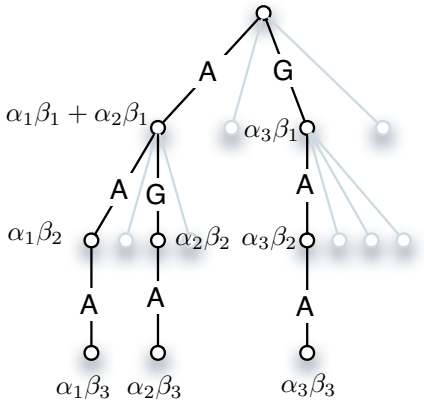


Figure 3. Three sequences AAA, AGA, GAA beeing added to the tree. The plot displays the resulting weights at the nodes.

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).

3.4. WD Kernel with Mismatches

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

$$\mathrm{k}(\mathbf{s}_i, \mathbf{s}_j) = \sum_{k=1}^{K} \sum_{m=0}^{M} \beta_{k,m} \sum_{l=1}^{L-k+1} \mathbf{I}(\boldsymbol{u}_{k,l}(\mathbf{s}_i) \neq_m \boldsymbol{u}_{k,l}(\mathbf{s}_j)), \tag{3}$$

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 (Rätsch & Candela, 2005).

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.

For a WD kernel formulation with improved positional invariance see (Rätsch et al., 2005).

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 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 (Cortes & Vapnik, 1995)).

The general idea of the Sequential Minimal Optimization (SMO) algorithm has been proposed by Platt (1999) and is implemented in many SVM software packages. While Platt (1999) used $Q = 2$ as a working set size, other implementations such as SVM^{light} (Joachims, 1999) 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 (Joachims, 1999). 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.[3] 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 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

[3] 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.

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

$f_i = 0, \alpha_i = 0$ for $i = 1, \ldots, N$
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 $g(\mathbf{s}) = \sum_{q=1}^{Q} (\alpha_{i_q} - \alpha_{i_q}^{old}) y_{i_q} \mathrm{k}(\mathbf{s}_{i_q}, \mathbf{s})$
  update $f_i = f_i + g(\mathbf{s}_i)$ for all $i = 1, \ldots, N$
end for

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 set containing 1,026,036 acceptor splice site sequences each 201 base pairs long. We trained SVMs using SVM^{light} (Joachims, 1999) on 500, 1000, 5000, 10000, 30000, 50000, 100000, 200000, 500000 and 10^6 randomly subsampled 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 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. 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.[4] In the case of 0.5 million of examples the Linadd formulation outperforms the original WD kernel by a factor of 4. Finally training with a sample size of 1,000,000 takes 7 hours and 15 minutes. Note that the Linadd-SMO optimization using the original WD kernel is not significantly slower than the block-formulation using Linadd-SMO.

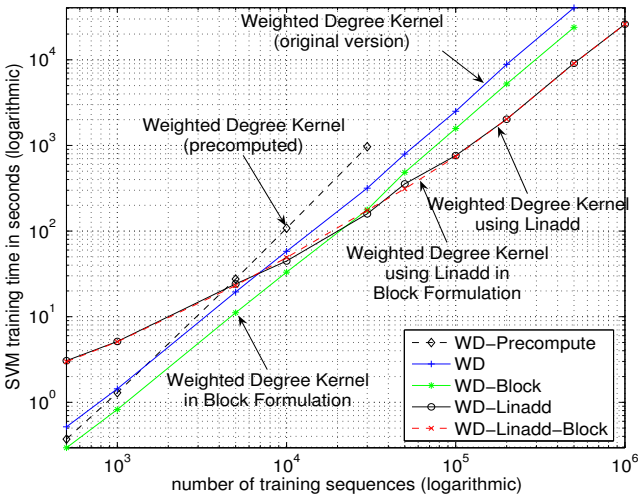


Figure 4. Comparison of the running time of the different weighted degree kernel algorithms. Note that as this is a log-log plot small appearing distances are large for larger N and that each slope corresponds to a different exponent. The empirically determined complexity of the precomputed WD kernel 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 $N > 10^5$).

WD Mismatch and Spectrum Kernel Comparison For the single mismatch WD and the spectrum kernel the SVM training times are listed in Table 3 and diagrammed in Figure 5.

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

Table 1. 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) and its block formulation (LinB). 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.

N	Pre	WD	Block	Linadd	LinB
500	0	1	**0**	3	3
1000	1	1	**1**	5	5
5000	28	19	**11**	24	23
10000	108	58	**33**	45	49
30000	965	317	177	**159**	**174**
50000	-	794	485	**355**	**312**
100000	-	2507	1576	**761**	**741**
200000	-	8863	5226	**2024**	**2031**
500000	-	40632	23946	**9119**	**9071**
1000000	-	-	-	**26107**	**26085**

Using the SMO optimization we gain speedups of 180% with respect to the mismatch WD kernel and even by a factor of 21 to the linear spectrum kernel.

Classification Performance In Table 2 the Test Error and AUC achieved on the splice site classification task for several sample sizes are shown. With one million examples the WD kernel method achieves 0.71% test error and 99.87% AUC. This is a relative improvement upon training on a 500 sample of 96%. Regardless of the amount of training data, when doubling the sample size, the relative improvement of the area under the curve (AUC) is approximately 20% (though slightly higher for small sample sizes). As can be seen the position dependent WD kernel is well suited for that task in good agreement to (Rätsch & Sonnenburg, 2004). As the spectrum kernel is position independent, it is not suited for that task (results not shown). [5]

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).

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

Table 2. The achieved AUC, relative AUC improvement (i.e. the improvement relative to the previous result) 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 remains at a level of $\approx 20\%$

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

Table 3. Speed Comparison (analoguous to Table 1) of a single mismatch Weighted Degree Kernel and the Spectrum Kernel, with (LinMis and LinSpec) and without (Mismatch and Spec) the SMO Linadd extension.

N	MismWD	LinMis	Spec	LinSpec
500	**1**	14	1	**0**
1000	**3**	23	2	**1**
5000	**43**	102	36	**5**
10000	**131**	208	123	**17**
30000	**703**	779	3462	**142**
50000	1827	**1575**	9025	**423**
100000	5464	**3932**	-	**2283**
200000	18524	**10317**	-	**11673**

We show that SVM training is up to 20 times faster using the Linadd-SMO algorithm in combination with the spectrum and up to 4 times faster in combination with the WD kernel. For the WD kernel we developed a blockwise-formulation, extend it allowing for mismatches and demonstrate 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

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning*, *20*, 273–297.

Cristianini, N., & Shawe-Taylor, J. (2000). *An intro-*

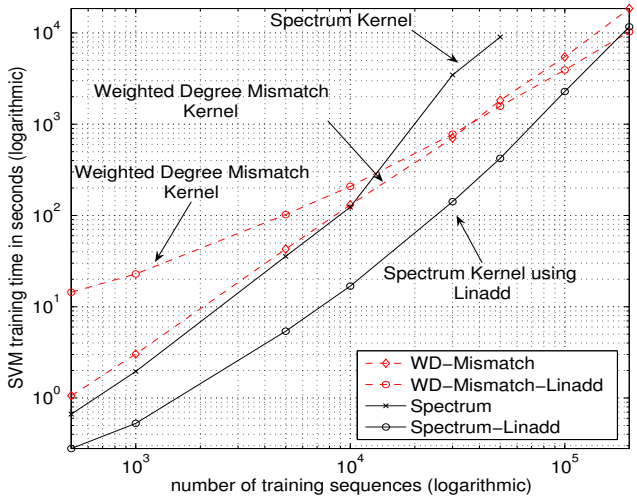


Figure 5. In analogy to Figure 4: Comparison of the single mismatch WD and spectrum kernel with and without Linadd-SMO optimization. Empirically determined complexity of the mismatch WD is $N^{1.64}$. Complexity estimates for the mismatch WD kernel using linadd are $N^{1.3}$ and for spectrum kernel using 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.

duction to support vector machines. Cambridge, UK: Cambridge University Press.

Jaakkola, T., Diekhans, M., & Haussler, D. (1999). Using the Fisher kernel method to detect remote homologies. *Intelligent Systems in Molecular Biology* (pp. 149–158).

Joachims, T. (1997). *Text categorization with support vector machines: Learning with many relevant features* (Technical Report 23). LS VIII, University of Dortmund.

Joachims, T. (1999). Making large–scale SVM learning practical. *Advances in Kernel Methods — Support Vector Learning* (pp. 169–184). Cambridge, MA: MIT Press.

Leslie, C., Eskin, E., & Noble, W. (2002). The spectrum kernel: A string kernel for SVM protein classification. *Proceedings of the Pacific Symposium on Biocomputing*. Kaua'i, Hawaii.

Leslie, C., Kuang, R., & Eskin, E. (2003). Inexact matching string kernels for protein classification. *Kernel Methods in Computational Biology* (pp. 95–112). MIT Press.

Liao, L., & Noble, W. (2002). Combining pairwise sequence similarity and support vector machines for remote protein homology detection. *Proceedings of the Sixth Annual International Conference on Research in Computational Molecular Biology* (pp. 225–232).

Meinicke, P., Tech, M., Morgenstern, B., & Merkl, R. (2004). Oligo kernels for datamining on biological sequences: A case study on prokaryotic translation initiation sites. *BMC Bioinformatics*, *5*.

Müller, K.-R., Mika, S., Rätsch, G., Tsuda, K., & Schölkopf, B. (2001). An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, *12*, 181–201.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods — Support Vector Learning* (pp. 185–208). Cambridge, MA: MIT Press.

Rätsch, G., & Candela, J. (2005). Predicting siRNA efficacy. *European Conference on Computational Biology, ECCB*. (submitted).

Rätsch, G., & Sonnenburg, S. (2004). *Accurate splice site prediction for caenorhabditis elegans*, 277–298. MIT Press series on Computational Molecular Biology. MIT Press.

Rätsch, G., Sonnenburg, S., & Schölkopf, B. (2005). Rase: Recognition of alternatively spliced exons in c. elegans. *ISMB 2005*. (accepted).

Schölkopf, B. (1997). *Support vector learning*. Munich: Oldenbourg Verlag.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels*. Cambridge, MA: MIT Press.

Schölkopf, B., Tsuda, K., & Vert, J. (Eds.). (2003). *Kernel methods in computational biology*. MIT Press series on Computational Molecular Biology. MIT Press.

Tsuda, K., Kawanabe, M., Rätsch, G., Sonnenburg, S., & Müller, K. (2002). A new discriminative kernel from probabilistic models. *Neural Computation*, *14*, 2397–414.

Vert, J.-P., Saigo, H., & Akutsu, T. (2003). Local alignment kernels for biological sequences. *Kernel Methods in Computational Biology* (pp. 131–154). MIT Press.

Vishwanathan, S., & Smola, A. (2003). Fast kernels for string and tree matching. *Kernel Methods in Computational Biology* (pp. 113–130). MIT Press.

Zien, A., Rätsch, G., Mika, S., Schölkopf, B., Lengauer, T., & Müller, K.-R. (2000). Engineering Support Vector Machine Kernels That Recognize Translation Initiation Sites. *BioInformatics*, *16*, 799–807.