Learning Interpretable SVMs for Biological Sequence Classification

S. Sonnenburg[1], G. Rätsch[2], and C. Schäfer[1]

[1] Fraunhofer Institute FIRST, Kekuléstr. 7, 12489 Berlin, Germany
[2] Friedrich Miescher Lab, Max Planck Society, Spemannstr. 39, Tübingen, Germany

Abstract. We propose novel algorithms for solving the so-called Support Vector Multiple Kernel Learning problem and show how they can be used to understand the resulting support vector decision function. While classical kernel-based algorithms (such as SVMs) are based on a single kernel, in Multiple Kernel Learning a quadratically-constraint quadratic program is solved in order to find a sparse convex combination of a set of support vector kernels. We show how this problem can be cast into a semi-infinite linear optimization problem which can in turn be solved efficiently using a boosting-like iterative method in combination with standard SVM optimization algorithms. The proposed method is able to deal with thousands of examples while combining hundreds of kernels within reasonable time.

In the second part we show how this technique can be used to understand the obtained decision function in order to extract biologically relevant knowledge about the sequence analysis problem at hand. We consider the problem of splice site identification and combine string kernels at different sequence positions and with various substring (oligomer) lengths. The proposed algorithm computes a *sparse* weighting over the length and the substring, highlighting which substrings are important for discrimination. Finally, we propose a bootstrap scheme in order to reliably identify a few statistically significant positions, which can then be used for further analysis such as consensus finding.

Keywords: Support Vector Machine, Multiple Kernel Learning, String Kernel, Weighted Degree Kernel, Interpretation of SVM results, Splice Site Prediction

1 Introduction

Kernel based methods such as Support Vector Machines (SVMs) have been proven powerful for sequence analysis problems frequently appearing in computational biology (e.g. [27, 11, 26, 15]). 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:

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

One of the problems with kernel methods compared to probabilistic methods (such as position weight matrices or interpolated Markov models [6]) is that the resulting decision function (1) is hard to interpret and, hence, difficult to use in order to extract relevant biological knowledge from it (see also [14, 26]). We approach this problem by considering the use of convex combinations of M kernels, i.e.

$$\mathrm{k}(\mathbf{s}_i, \mathbf{s}_j) = \sum_{j=1}^{M} \beta_j k_j(\mathbf{s}_i, \mathbf{s}_j)$$

with $\beta_j \geq 0$ and $\sum_j \beta_j = 1$, where each kernel k_j uses only a distinct set of features of the sequence. For appropriately designed sub-kernels, the optimized combination coefficients can then be used to understand which features of the sequence are of importance for discrimination. This is an important property missing in current kernel based algorithms.

Sequence analysis problems usually come with large number of examples and potentially many kernels to combine. Unfortunately, algorithms proposed for Multiple Kernel Learning (MKL) so far are not capable of solving the optimization problem for realistic problem sizes (e.g. $\geq 10,000$ examples) within reasonable time. Even recently

proposed SMO-like algorithms for this problem, such as the one proposed in [1], are not efficient enough since they suffer from the inability to keep all kernel matrices ($K_j \in \mathbb{R}^{N\times N}, j = 1, \ldots, M$) in memory.[3] We consider the reformulation of the MKL problem into a semi-infinite linear problem (SILP), which can be iteratively approximated quite efficiently. In each iteration one only needs to solve the classical SVM problem (with one of the efficient and publicly available SVM implementations) and then performs an adequate update of the kernel convex combination weights $\boldsymbol{\beta}$. Separating the SVM optimization from the optimization of the kernel coefficients can thus lead to significant improvements for large scale problems with general kernels. We will, however, show how one can take advantage of the special structure of string kernels (in particular the one below).

We illustrate the usefulness of the proposed algorithm in combination with a recently proposed string kernel on DNA sequences — the so-called *weighted degree* (WD) kernel [23]. Its main idea is to count the (exact) co-occurrence of k-mers at position l in the sequence between two compared DNA sequences. The kernel can be written as a linear combination of d parts with coefficients β_k ($k = 1, \ldots, d$):

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

where L is the length of the $\mathbf{s}$'s, d is the maximal oligomer order considered and $\boldsymbol{u}_{k,l}(\mathbf{s})$ is the oligomer of length k at position l of sequence $\mathbf{s}$. One question is how the weights β_k for the various k-mers should be chosen. So far, only heuristic settings in combination with expensive cross-validation have been used. The MKL approach offers a clean and efficient way to find the optimal weights $\boldsymbol{\beta}$. One would define d kernels

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

and then optimize the convex combination of these kernels by the newly proposed algorithm. The optimal weights $\boldsymbol{\beta}$ indicate which oligomer lengths are important for the classification problem at hand. Moreover, one would expect a slight performance gain for optimized weights and since the $\boldsymbol{\beta}$'s are sparse, also an increased prediction speed.

Additionally, it is interesting to introduce an importance weighting over the position of the subsequence. Hence, we define a separate kernel for each position and each oligomer order, i.e.

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

and optimize the weightings of the combined kernel, which may be written as

$$\mathrm{k}(\mathbf{s}_i, \mathbf{s}_j) = \sum_{k=1}^{d} \sum_{l=1}^{L-k} \beta_{k,l} \mathbf{I}(\boldsymbol{u}_{k,l}(\mathbf{s}_i) = \boldsymbol{u}_{k,l}(\mathbf{s}_j)) = \sum_{k,l} \beta_{k,l} k_{k,l}(\mathbf{s}_i, \mathbf{s}_j).$$

Obviously, if one would be able to obtain an accurate classification by a sparse weighting $\beta_{k,i}$, then one could easily interpret the resulting decision function. For instance for signal detection problems (such as splice site detection), one would expect a few important positions with long oligomers near the site and some additional positions only capturing nucleotide compositions (short nucleotides).

By bootstrapping and employing a combinatorial argument, we derive a statistical test that discovers the most important kernel weights. On simulated pseudo-DNA sequences with two hidden 7-mers we elucidate which k-mers in the sequence were used for the SVM decision. Finally we apply the method to splice sites classification of *C. elegans* and show that the optimized convex kernel combination may help extracting biological knowledge from the data.

2 Methods

2.1 Support Vector Machines

We use Support Vector Machines[5] which are extensively studied in literature (e.g. [19]). Their classification function can be written as in (1) The α_i's are Lagrange multipliers and b is the usual bias which are the results of SVM training.

[3] Note that also kernel caching becomes insufficient if the number of combined kernels is large

The kernel k is the *key ingredient* for learning with SVMs. It implicitly defines the feature space and the mapping Φ via

$$\mathrm{k}(\mathbf{s}, \mathbf{s}') = (\Phi(\mathbf{s}) \cdot \Phi(\mathbf{s}')).$$

In case of the afore mentioned WD kernel, Φ maps into a *feature space* $\mathbb{R}^D$ of all possible k-mers of length up to d for each sequence position ($D \approx 4^{d+1}L$). For a given sequence $\mathbf{s}$, a dimension of $\phi(\mathbf{s})$ is 1, if it contains a certain substring at a certain position. The dot-product between two mapped examples then counts the co-occurrences of substrings at all positions.

For a given set of training examples $(\mathbf{s}_i, y_i)$ $(i = 1, \ldots, N)$, the SVM solution is obtained by solving the following optimization problem that maximizes the soft margin between both classes:

$$\begin{aligned} \min \ & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{w.r.t.} \ & \mathbf{w} \in \mathbb{R}^D, b \in \mathbb{R}, \boldsymbol{\xi} \in \mathbb{R}^N_+ \\ \text{s.t.} \ & y_i((\mathbf{w} \cdot \Phi(\mathbf{s}_i)) + b) \geq 1 - \xi_i, \quad i = 1, \ldots, N, \end{aligned} \tag{1}$$

where the parameter C determines the trade-off between the size of the margin and the margin errors ξ_i. The dual optimization problem is as follows:

$$\begin{aligned} \max \ & \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j \,\mathrm{k}(\mathbf{s}_i, \mathbf{s}_j), \\ \text{w.r.t.} \quad & \boldsymbol{\alpha} \in \mathbb{R}^N_+ \text{ with } \boldsymbol{\alpha} \leq C \text{ and } \sum_{i=1}^{N}\alpha_i y_i = 0. \end{aligned} \tag{2}$$

Note that there exist a large variety of different software packages that can efficiently solve the above optimization problem even for more than hundred thousands of examples.

2.2 The Multiple Kernel Learning Optimization Problem

Idea In the Multiple Kernel Learning (MKL) problem one is given N data points $(\tilde{\mathbf{s}}_i, y_i)$ $(y_i \in \{\pm 1\})$, where $\tilde{\mathbf{s}}_i$ is subdivided into M components $\tilde{\mathbf{s}}_i = (\mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,M})$ with $\mathbf{s}_{i,j} \in \mathbb{R}^{k_j}$ and k_j is the dimensionality of the j-th component. Then one solves the following convex optimization problem [1], which is equivalent to the linear SVM for $M = 1$:

$$\begin{aligned} \min \ & \frac{1}{2}\left(\sum_{j=1}^{M} d_j\beta_j\|\mathbf{w}_j\|_2\right)^2 + C\sum_{i=1}^{N}\xi_n \\ \text{w.r.t.} \ & \mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_M), \mathbf{w}_j \in \mathbb{R}^{k_j}, \boldsymbol{\xi} \in \mathbb{R}^N_+, \boldsymbol{\beta} \in \mathbb{R}^M_+, b \in \mathbb{R} \\ \text{s.t.} \ & y_i\left(\sum_{j=1}^{M}\beta_j\mathbf{w}_j^\top\mathbf{s}_{i,j} + b\right) \geq 1 - \xi_i, \forall i = 1, \ldots, N \\ & \sum_{j=1}^{M}\beta_j = 1, \end{aligned} \tag{3}$$

where d_j is a prior weighting of the kernels (in [1], $d_j = 1/\sum_i(\mathbf{s}_{i,j} \cdot \mathbf{s}_{i,j})$ has been chosen such that the combined kernel has trace one). For simplicity, we assume that $d_j = 1$ for the rest of the paper and that the normalization is done within the mapping ϕ (if necessary). Note that the ℓ_1-norm of $\boldsymbol{\beta}$ is constrained to one, while one is penalizing the ℓ_2-norm of $\mathbf{w}_j$ in each block j separately. The idea is that ℓ_1-norm constrained or penalized variables tend to have sparse optimal solutions, while ℓ_2-norm penalized variables do not [21]. Thus the above optimization problem offers the possibility to find sparse solutions on the block level with non-sparse solutions within the blocks.

Reformulation as a Semi-Infinite Linear Program The above optimization problem can also be formulated in terms of support vector kernels [1]. Then each block j corresponds to a separate kernel $(K_j)_{r,s} = k_j(\mathbf{s}_{r,j}, \mathbf{s}_{s,j})$ computing the dot-product in feature space of the j-th component. In [1] it has been shown that the following optimization problem is equivalent to (3):

$$\begin{aligned} \min \;& \frac{1}{2}\gamma^2 - \sum_i \alpha_i \\ \text{w.r.t.} \;& \gamma \in \mathbb{R}, \boldsymbol{\alpha} \in \mathbb{R}^N \\ \text{s.t.} \;& 0 \leq \boldsymbol{\alpha} \leq C, \sum_i \alpha_i y_i = 0 \\ & \underbrace{\sum_{r,s} \alpha_r \alpha_s y_r y_s (K_j)_{r,s}}_{=:S_j(\boldsymbol{\alpha})} \leq \gamma^2 \end{aligned} \tag{4}$$

In order to solve (4), one may solve the following saddle point problem (Lagrangian):

$$L := \frac{1}{2}\gamma^2 - \sum_i \alpha_i + \sum_{j=1}^M \beta_j (S_j(\boldsymbol{\alpha}) - \gamma^2) \tag{5}$$

minimized w.r.t. $\boldsymbol{\alpha} \in \mathbb{R}_+^N, \gamma \in \mathbb{R}$ (subject to $\boldsymbol{\alpha} \leq C$ and $\sum_i \alpha_i y_i = 0$) and maximized w.r.t. $\boldsymbol{\beta} \in \mathbb{R}_+^M$. Setting the derivative w.r.t. to γ to zero, one obtains the constraint $\sum_j \beta_j = \frac{1}{2}$ and (5) simplifies to:

$$L := \underbrace{\frac{1}{2}\sum_{j=1}^M \beta_j S_j(\boldsymbol{\alpha})}_{=:S(\boldsymbol{\alpha})} - \sum_i \alpha_i \tag{6}$$

Assume $\boldsymbol{\alpha}^*$ would be the optimal solution, then $\theta^* := S(\boldsymbol{\alpha}^*) - \sum_i \alpha_i$ is minimal and, hence, $S(\boldsymbol{\alpha}) - \sum_i \alpha_i \geq \theta^*$ for all $\boldsymbol{\alpha}$ (subject to the above constraints). Hence, finding a saddle-point of (6) is equivalent to solving the following semi-infinite linear program:

$$\begin{aligned} \max \;& \theta \\ \text{w.r.t.} \;& \theta \in \mathbb{R}, \boldsymbol{\beta} \in \mathbb{R}_+^M \text{ with } \sum_j \beta_j = 1 \\ \text{s.t.} \;& \sum_{j=1}^M \beta_j \left(\frac{1}{2} S_j(\boldsymbol{\alpha}) - \sum_i \alpha_i \right) \geq \theta \\ & \text{for all } \boldsymbol{\alpha} \text{ with } 0 \leq \boldsymbol{\alpha} \leq C \text{ and } \sum_i y_i \alpha_i = 0 \end{aligned} \tag{7}$$

Note that there are infinitely many constraints (one for every vector $\boldsymbol{\alpha}$). Typically algorithms for solving semi-infinite problems work by iteratively finding violated constraints, i.e. $\boldsymbol{\alpha}$ vectors, for intermediate solutions $(\boldsymbol{\beta}, \theta)$. Then one adds the new constraint (corresponding to the new $\boldsymbol{\alpha}$) and resolves for $\boldsymbol{\beta}$ and θ [10] (see next section for details).

Fortunately, finding the constraint that is most violated corresponds to solving the SVM optimization problem for a fixed weighting of the kernels:

$$\sum_{j=1}^M \beta_j \left(\frac{1}{2} S_j(\boldsymbol{\alpha}) - \sum_i \alpha_i \right) = \sum_{r,s} \alpha_r \alpha_s y_r y_s K_{r,s} - \sum_i \alpha_i,$$

where $K = \sum_j \beta_j K_j$. Due to the number of efficient SVM Optimizers, the problem of finding the most violated constraint can be solved efficiently, too.

Algorithm 1 The column generation algorithm (left) employs a linear programming solver and the boosting-like algorithm (right) uses exponential updates to iteratively solve the semi-infinite linear optimization problem (7). The accuracy parameter ϵ is assumed to be given to the algorithm.

$D^0 = 1$, $\theta^1 = 0$, $\beta_k^1 = \frac{1}{M}$ for $k = 1, \ldots, M$
for $t = 1, 2, \ldots$ **do**
  obtain SVM's $\boldsymbol{\alpha}^k$ with kernel

$$\mathrm{k}^t(\mathbf{s}_i, \mathbf{s}_j) := \sum_{k=1}^{M} \beta_k^t \, \mathrm{k}_k(\mathbf{s}_i, \mathbf{s}_j)$$

  for $k = 1, \ldots, M$ **do**
    $D_k^t = \frac{1}{2} \sum_{r,s} \alpha_r^t \alpha_s^t y_r y_s \, \mathrm{k}_k(\mathbf{s}_r, \mathbf{s}_s) - \sum_r \alpha_r^t$
  end for
  $D^t = \sum_{k=1}^{M} \beta_k^t D_k^t$
  $(\boldsymbol{\beta}^{t+1}, \theta^{t+1}) = \operatorname{argmax}\ \theta$
    w.r.t. $\boldsymbol{\beta} \in \mathbb{R}_+^M, \theta \in \mathbb{R}$ with $\sum_k \beta_k = 1$
    s.t. $\sum_{k=1}^{M} \beta_k D_k^r \geq \theta$ for $r = 1, \ldots, t$
  if $|1 - \frac{\theta^{t+1}}{D^t}| \leq \epsilon$ **then break**
end for

$D^0 = 1$, $\rho^1 = \tau_k^1 = 0$, $\beta_k^1 = \frac{1}{M}$ for $k = 1, \ldots, M$
for $t = 1, 2, \ldots$ **do**
  obtain SVM's $\boldsymbol{\alpha}^k$ with kernel

$$\mathrm{k}^t(\mathbf{s}_i, \mathbf{s}_j) := \sum_{k=1}^{M} \beta_k^t \, \mathrm{k}_k(\mathbf{s}_i, \mathbf{s}_j)$$

  for $k = 1, \ldots, M$ **do**
    $D_k^t = \frac{1}{2} \sum_{r,s} \alpha_r^t \alpha_s^t y_r y_s \, \mathrm{k}_k(\mathbf{s}_r, \mathbf{s}_s) - \sum_r \alpha_r^t$
  end for
  $D^t = \sum_{k=1}^{M} \beta_k^t D_k^t$
  $\gamma_t = \operatorname{argmin}_{\gamma \in [0,1]} \sum_{k=1}^{M} \beta_k^t \exp\left\{\gamma(D_k - \rho^t)\right\}$
  for $k = 1, \ldots, M$ **do**
    $\tau_k^{t+1} = \tau_k^t + \gamma_t D_k^t$
    $\beta_k^{t+1} = \beta_k^t \exp(\gamma_t D_k^t) / \left(\sum_{k'} \beta_{k'}^t \exp(\gamma_t D_{k'}^t)\right)$
  end for
  $\rho^{t+1} = \max_k \tau_k^{t+1} / \sum_{r=1}^{t} \gamma_r$
  if $|1 - \frac{\rho^{t+1}}{D^t}| \leq \epsilon$ **then break**
end for

Finally, one needs some convergence criterion. Note that the problem is solved when all constraints are satisfied while the $\boldsymbol{\beta}$'s and θ are optimal. Hence, it is a quite natural choice to use the normalized maximal constraint violation as a convergence criterion. In our case this would be:

$$\epsilon := \left| 1 - \frac{\sum_{j=1}^{M} \beta_j^k \left(\frac{1}{2} S_j(\boldsymbol{\alpha}^{k+1}) - \sum_i \alpha_i^{k+1}\right)}{\theta^k} \right|,$$

where $(\boldsymbol{\beta}^k, \theta^k)$ is the optimal solution at iteration k and $\boldsymbol{\alpha}^{k+1}$ corresponds to the newly found maximally violating constraint of the next iteration (i.e. the SVM solution for weighting $\boldsymbol{\beta}$). We usually only try to approximate the optimal solution and set ϵ to either 10^{-4} or 10^{-2} in our experiments.

Column Generation & Boosting There are several ways for solving (7). As outlined above, one may iteratively extend and solve an reduced problem, only using t constraints corresponding to $\boldsymbol{\alpha}_t$ ($t = 1, 2, \ldots$). By repeatedly adding the most violating constraints one typically converges fast to the optimal solution [2]. Note, however, that there are no known convergence rates for such algorithms [10], but it often converges to the *optimal* solution in a small number of iterations [2, 22].

We would like to consider the use of a boosting-like technique [8] which has been used to solve semi-infinite problems [24]. It has been shown that in order to solve a semi-infinite problem like (7), one needs at most $T = \mathcal{O}(\log(M)/\hat{\epsilon}^2)$ iterations (i.e. SVM optimization), where $\hat{\epsilon}$ is the unnormalized constraint violation and the constants may depend on the kernels and the number of examples N. At least for not too small values of $\hat{\epsilon}$ this technique produces reasonably fast good approximate solutions. We cannot go into detail of the derivation of the algorithm, but only state the boosting-like algorithm that has the above property and is similar to the Arc-GV algorithm [4] and to AdaBoost* [21].
The pseudo codes for both algorithms are given for completeness Algorithm 1.

An SMO-like algorithm for simultaneous optimization of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ Usually it is infeasible to use standard optimization tools (e.g. MINOS, CPLEX, LOQO) for solving the SVM training problems on datasets containing more

Algorithm 2 Outline of the SMO algorithm that optimizes $\boldsymbol{\alpha}$ and the kernel weighting $\boldsymbol{\beta}$ simultaneously. The accuracy parameter ϵ and the subproblem size Q are assumed to be given to the algorithm. For simplicity we omit the removal of inactive constraints. Also note that from one iteration to the next the LP only differs by one additional constraint. This can usually be exploited to save computing time for solving the LP.

$f_{k,i}=0$, $\hat{f}_i=0$, $\alpha_i=0$ for $k=1,\ldots,M$ and $i=1,\ldots,N$
for $t=1,2,\ldots$ **do**
  Check optimality conditions and stop if optimal
  select Q suboptimal variables $i_1,\ldots,i_Q$ based on $\hat{\mathbf{f}}$ and $\boldsymbol{\alpha}$
  $\boldsymbol{\alpha}^{old}=\boldsymbol{\alpha}$
  solve (2) with respect to the selected variables and update $\boldsymbol{\alpha}$
  create suffix trees to prepare efficient computation of $g_k(\mathbf{s})=\sum_{q=1}^{Q}(\alpha_{i_q}-\alpha_{i_q}^{old})y_{i_q}\,\mathrm{k}_k(\mathbf{s}_{i_q},\mathbf{s})$
  $f_{k,i}=f_{k,i}+g_k(\mathbf{s}_i)$ for all $k=1,\ldots,M$ and $i=1,\ldots,N$
  for $k=1,\ldots,M$ **do**
    $D_k^t=\frac{1}{2}\sum_r f_{k,r}\alpha_r^t y_r-\sum_r \alpha_r^t$
  end for
  $D^t=\sum_{k=1}^{M}\beta_k^t D_k^t$

  if $|1-\frac{\theta^t}{D^t}|\geq\epsilon$
    $(\boldsymbol{\beta}^{t+1},\theta^{t+1})=\operatorname{argmax}\ \theta$
      w.r.t. $\boldsymbol{\beta}\in\mathbb{R}_+^M,\theta\in\mathbb{R}$ with $\sum_k\beta_k=1$
      s.t. $\sum_{k=1}^{M}\beta_k D_k^r\geq\theta$ for $r=1,\ldots,t$
  else
    $\theta^{t+1}=\theta^t$
  end if
  $\hat{f}_i=\sum_k\beta_k^{t+1}f_{k,i}$ for all $i=1,\ldots,N$
end for

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 (2)).

The general idea of Sequential Minimal Optimization (SMO) algorithm has been proposed by [20] and is implemented in many SVM software packages. Here we would like to propose an extension of the SMO algorithm to optimize the kernel weights $\boldsymbol{\beta}$ and the example weights $\boldsymbol{\alpha}$ at the same time. The algorithm is motivated from an insufficiency of the column-generation algorithm described in the previous section: If the $\boldsymbol{\beta}$'s are not optimal yet, then the optimization of the $\boldsymbol{\alpha}$'s until optimality is not necessary and therefore inefficient. It would be considerably faster if for any newly obtained $\boldsymbol{\alpha}$ in the SMO iterations, we could efficiently recompute the optimal $\boldsymbol{\beta}$ and then continue optimizing the $\boldsymbol{\alpha}$'s using the new kernel weighting.

Recomputing $\boldsymbol{\beta}$ involves solving a linear program and the problem grows with each additional $\boldsymbol{\alpha}$-induced constraint. Hence, after many iterations solving the LP may become infeasible. Fortunately, there are two facts making it still possible: (1) only a small number of the added constraints are active and one may for each newly added constraint remove an old inactive one — this prevents the LP to grow arbitrarily and (2) for Simplex-based LP optimizers such as CPLEX there exists the so-called *hot-start feature* which allows one efficiently recompute the new solution, if one, for instance, only adds an additional constraint.

The SVM optimizer 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 [12]. However, if one changes the kernel weights, then the stored $\hat{f}_j$ values become invalid and need to be recomputed. In order to avoid the full re-computation one has to additionally store a $M\times N$ matrix $f_{k,j}=\sum_i\alpha_i y_i\,\mathrm{k}_k(\mathbf{s}_i,\mathbf{s}_j)$, i.e. the outputs for each kernel separately. If the $\boldsymbol{\beta}$'s change, then $\hat{f}_j$ can be quite efficiently recomputed by $\hat{f}_j=\sum_k\beta_k f_{k,j}$.

Finally, in each iteration the SMO optimizer may change a subset of the $\boldsymbol{\alpha}$'s. In order to update $\hat{f}_j$ and $f_{j,k}$ one needs to compute full rows j of each kernel for every changed α_j. Usually one uses kernel-caching to reduce the computational effort of this operation, which is, however, in our case not efficient enough. Fortunately, for the afore mentioned WD kernel there is a way to avoid this problem by using so-called suffix trees (as similarly proposed

in [17]). Due to a lack of space we cannot go into more detail here. We provide, however, the pseudo-code of the algorithm which takes the above discussion into account and provides a few more details in Algorithm 2.

Empirically we noticed that the proposed SMO-like algorithm is often 3-5 times faster than the column-generation algorithm proposed in the last section, while achieving the same accuracy. In the experiments in Section 3 we therefore only used the SMO-like algorithm.

2.3 Estimating the Reliability of a Weighting

Finally we want to assess the reliability of the learned weights scheme $\boldsymbol{\beta}$. For this purpose we generate T bootstrap samples and rerun the whole procedure resulting in T weightings $\boldsymbol{\beta}^t$.

To test the importance of a weight $\beta_{k,i}$ (and therefore the corresponding kernels for position and oligomer length) we apply the following method: define a Bernoulli variable $X^t_{k,i} \in \{0,1\},\ k = 1,\ldots,d, i = 1,\ldots,L, t = 1,\ldots,T$ by

$$X^t_{k,i} = \begin{cases} 1, & \beta^t_{k,i} > \tau := \mathbf{E}_{k,i,t} X^t_{k,i} \\ 0, & \text{else} \end{cases}.$$

The sum $Z_{k,i} = \sum_{t=1}^{T} X^t_{k,i}$ has a binomial distribution Bin(T,p_0), p_0 unknown. We estimate p_0 with $\hat{p}_0 = \#(\beta^t_{k,i} > \tau)/T \cdot M$, i.e. the empirical probability to observe $P(X^t_{k,i} = 1), \forall k, i, t$. We test whether $Z_{k,i}$ is as large as could be expected under Bin(T,$\hat{p}_0$) or larger: $\mathcal{H}_0 : p \leq c^*$ vs $\mathcal{H}_1 : p > c^*$. Here c^* is defined as $\hat{p}_0 + 2\mathbf{Std}_{k,i,t} X^t_{k,i}$ and can be interpreted as an upper bound of the confidence interval for p_0. This choice is taken to be adaptive to the noise level of the data and hence the (non)-sparsity of the weightings $\boldsymbol{\beta}^t$. The hypotheses are tested with a Maximum-Likelihood test on an α-level of $\alpha = 0.05$; that is c^{**} is the minimal value for that the following inequality hold:

$$0.05 = \alpha \geq P_{\mathcal{H}_0}(\text{reject } \mathcal{H}_0) = P_{\mathcal{H}_0}(Z_{k,i} > c^{**}) = \sum_{j=c^{**}}^{T} \binom{T}{j} \hat{p}_0 (1 - \hat{p}_0).$$

For further details on the test see [18] or [16]. This test is carried out for every $\beta^t_{k,i}$. (We assume independence between the weights in one single $\boldsymbol{\beta}$, and hence assume that the test problem is the same for every $\beta_{k,i}$). If $\mathcal{H}_0$ can be rejected, the kernel learned at position i on the k-mer is important for the detection and thus (should) contain biologically interesting knowledge about the problem at hand.

3 Results and Discussion

The main goal of this work is to provide an explanation of the SVM decision rule, for instance by identifying sequence positions that are important for discrimination. In the first part on our evaluation, however, we show that the proposed algorithm optimizing kernel weights does perform also slightly better than the standard kernel and leads to SVM classification functions that are computationally more efficient. A detailed comparison of the WD kernel approach with other state-of-the-art methods is provided in [23] and not considered in this work. In the remaining part we show how the weights can be used to obtain a deeper understanding of how the SVM classifies sequences and match it with knowledge about the underlying biological process.

3.1 Comparison with the original WD Kernel

To compare classification performance and running time with the original WD kernel, we trained SVMs on the *C. elegans* acceptor splice dataset using $100,000$ sequences in training, $100,000$ examples for validation and $60,000$ examples to test the classifiers performance (cf. Appendix A.2). In this dataset each sequence is a window centered around the true splice site containing 141 nucleotides. Using this setup we perform cross-validation over the parameters $M \in \{10, 12, 15, 17, 20\}$ and $C \in \{0.5, 2, 5, 10\}$. Accuracy was set to $\epsilon = 0.01$.

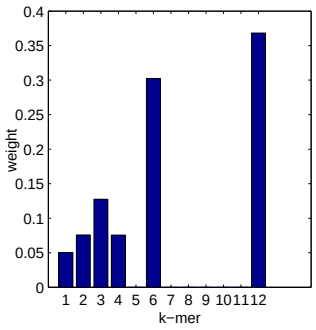


Fig. 1: Optimized WD kernel weights

We find on the validation set that for the original WD kernel $M = 20$ and $C = 0.5$ gives best classification performance (ROC Score 99.66), while for the SVM using the WD kernel that also learns the weighting give best results ($M = 12$ and $C = 12$; ROC Score also 99.66).[4] The figure on the left shows the weights the proposed WD kernel has learned, suggesting that 12-mers and 6-mers seem to be of high importance. On the test dataset the original WD kernel performs as good as on the validation dataset (ROC Score 99.66% again), while the new WD kernel achieves a 99.67% ROC Score. Astonishingly training the new WD kernel SVM (i.e. with weight optimization) was 1.5 times faster than training the the original SVM, which might be due to using a suffix tree requiring no kernel cache. Also note that the resulting classifier provided by the new algorithm is considerably faster than the one obtained by the classical SVM since many weights are zero [7].

3.2 Relation to Positional Weight Matrices

An interesting relation of the learned weightings to the relative entropy between Positional Weight Matrices can be shown with the following experiment: We train an SVM with a WD kernel that consists of 60 first-order sub-kernels on acceptor splice sites from *C. elegans* ($100,000$ sequences for training, $160,000$ sequences for validation). For the SVMs, $C = 1$ and for the WD Kernel accuracy $\epsilon = 0.01$ were chosen. The AG consensus is at position $31 - 32$ and a window of length ± 29 around the consensus was chosen. The learned weights β_k are shown in figure 2 (left). For comparison we computed the Positional Weight Matrices for the positive and the negative class separately and

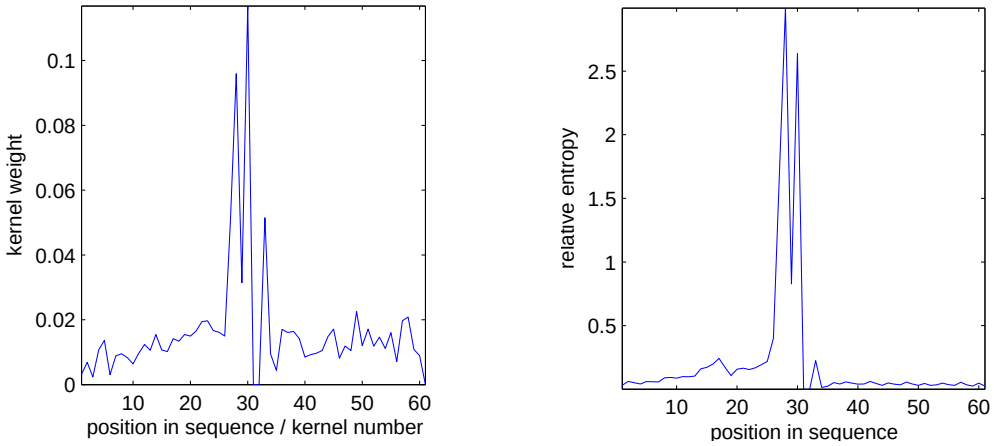


Fig. 2: (left) Value of the learned weightings of an SVM with a WD kernel of 60 first-order sub-kernels, (right) relative entropy obtained between the Positional Weight Matrices for the positive and the negative class, both trained for acceptor splice site detection.

computed the relative entropy Δ_i between the two probability estimates $p^+_{i,j}$ and $p^-_{i,j}$ at each position j by $\Delta_i = \sum_{i=1}^{4} p^+_{i,j} \log(p^+_{i,j}/p^-_{i,j})$, which is shown in Figure 2 (right). The shape of both plots is very similar, i.e. both methods consider upstream information, as well as a position directly after the splice site to be highly important. As a major difference the WD-weights in the exons remain on a high level. Note that both methods use only first order information. Nevertheless the classification accuracy is extremely high. On the separate validation set the SVM already achieves a ROC score of 99.07% and the Positional Weight Matrices a ROC score of 98.83%.

In another experiment we considered the larger region from -50 to +60nt around the splice site and used a kernel of degree 15. We defined a kernel for every position that only accounts for substrings that start at the corresponding

[4] While the model selection results are at the border of the parameter range checked, the obtained ROC scores are always $\geq 99.65\%$.

position (up to length 15). To get a smoother weighting and to reduce the computing time we only used $\lceil 111/2 \rceil = 56$ weights (combining every two positions to one weight).

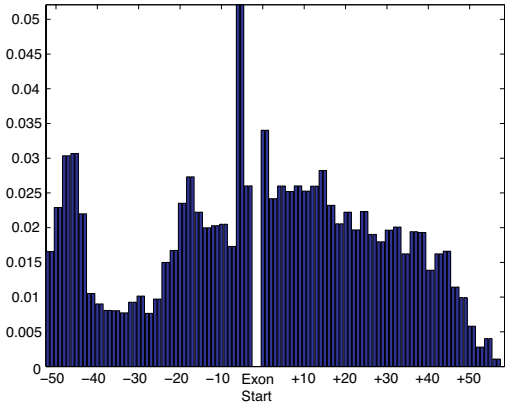


Fig. 3: Optimized WD kernel weights considering subsequences starting at different positions (one weight per two positions)

The average computed weighting on ten bootstrap runs trained on around $65,000$ examples is shown in Figure 3. Several regions of interested can be identified: a) The region -50 to -40, which corresponds to the donor splice site of the previous exon (many introns in *C. elegans* are very short, often only 50nt), b) the region -25 to -15 that corresponds to the location of the branch point, c) the intronic region closest to the splice site with greatest weight (-8 to -1; the weights for the AG dimer are zero, since it appears in splice sites and decoys) and d) the exonic region (0 to +50). Slightly surprising are the high weights in the exonic region, which we suspect only model triplet frequencies. The decay of the weights seen from $+15$ to $+45$ might be explained by the fact that not all exons are actually long enough. Furthermore, since the sequence ends in our case at +60nt, the decay after $+45$ can be explained by the shorter substrings that can be matched.

3.3 Detecting motifs in a Toy Dataset

As a prove of concept, we test the our method on a toy datasets at four different noise levels (for a detailed description of the data see Appendix A.1). For every noise level, we train on 100 bootstrap replicates ($C = 2$, $\epsilon = 0.001$) and learn $M = 350$ WD kernel parameters in each run. On the resulting 100 weightings we performed the reliability test (cf. Section 2.3). The results are shown in Figure 4 (columns correspond to different noise levels — increasing from left to right). Each figure shows a kernel weighting $\boldsymbol{\beta}$, where columns correspond to weights used at a certain sequence position and rows to the k-mer length used at that position. The plots in the first row show the weights that are detected to be important at a significance level of $\alpha = 0.05$ in bright (yellow) color. The likelihood for every weight to be detected by the test and thus to reject $\mathcal{H}_0$ is illustrated in the plots in the second row. Here bright color means it is more likely to reject $\mathcal{H}_0$.

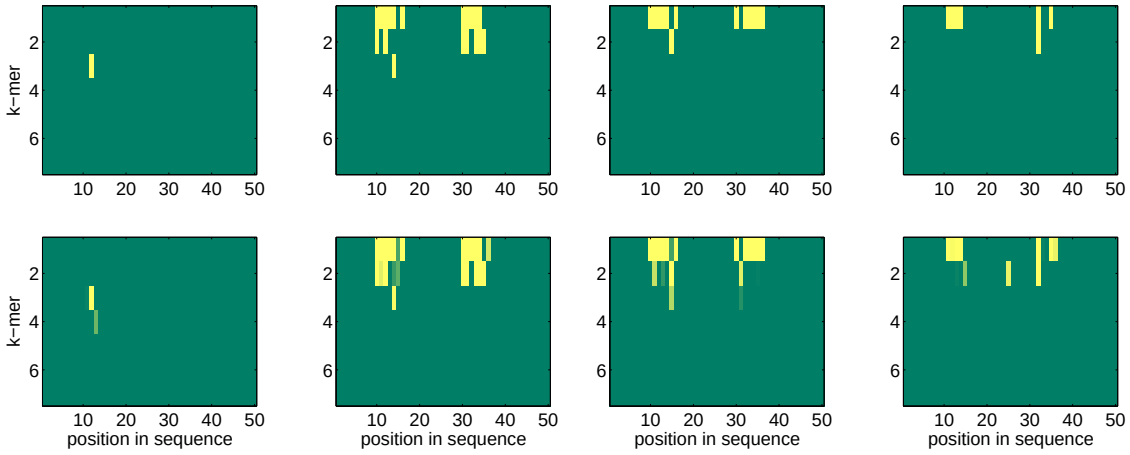


Fig. 4: In this "figure matrix", columns correspond to the noise level, i.e. different numbers of nucleotides randomly substituted in the motif of the toy dataset cf. Appendix A.1. Each sub-plot shows the kernel weighting $\boldsymbol{\beta}$, where columns correspond to weights used at a certain sequence position and rows to the oligomer length used at that position. The first row shows the kernel weights that are significant, while the second row describes the likelihood of every weight to be rejected under $\mathcal{H}_0$.

As long as the noise level does not exceed 2/7, higher order matches of length 3 and 4 seem sufficient to distinguish sequences containing motifs from the rest. However, only the 3-mer is detected with the test procedure. When more nucleotides in the motifs are replaced with noise, more weights are determined to be of importance. This becomes especially obvious in column 3 were 4 out of 7 nucleotides within each motif were randomly replaced, but still an average ROC score of 99.6% is achieved. In the last column the ROC score drops down to 83%, but only weights in the correct range $10 \dots 16$ and $30 \dots 36$ are found to be significant.

3.4 Detecting motifs in Splice data

As the results on the toy dataset are promising, we now apply our method to a more realistic problem. We consider the classification of acceptor splice sites against non-acceptor splice sites (with AG dimer) from the *C. elegans* (cf. Appendix A.2 for details on the generation of the data sets).

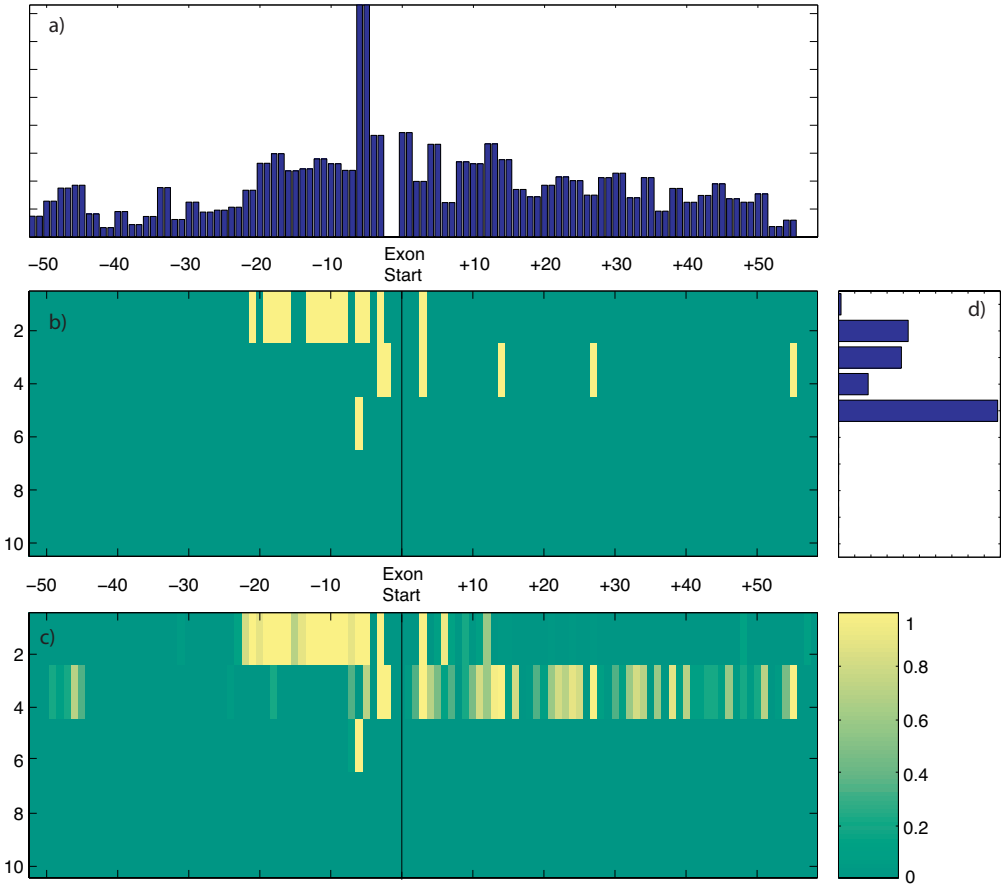


Fig. 5: Figure a) shows the average weight (over 10 runs) of the weights per position (fixed degree weighting; one weight for two positions) and d) the averaged weights per degree (uniform position weighting). Figures b) displays the position/degree combinations that were found to be significantly used (40 bootstrap runs). Figure c) shows the likelihood for rejecting $\mathcal{H}_0$. All runs only used $5,000$ training examples.

We trained our Multiple Kernel Learning algorithm ($C = 2$, $\epsilon = 10^{-4}$) on 5,000 randomly chosen sequences of length 111 with a maximal oligomer length of $d = 10$. This leads to $M = 1110$ kernels in the convex combination. Figure 5 shows the results obtained for this experiment (similarly organized as Figure 4). We can observe (cf. Figure 5 b & c) that the optimized kernel coefficients are biologically plausible: longer significant oligomers were found close to the splice site position, oligomers of length 3 and 4 are mainly used in the exonic region (modeling triplet usage) and

short oligomers near the branch site. Note, however, that one should use more of the available examples for training in order to extract more meaningful results (adapting 1110 kernel weights may have lead to overfitting). In some preliminary tests using more training data we observed that longer oligomers and also more positions in the exonic and intronic regions become important for discrimination.

Note that the weight matrix would be the outer product of the position weight vector and the oligomer-length weight vector, if position and oligomer length would be independent. This is clearly not the case: it seems very important (according to the weight for oligomer-length 5) to consider longer oligomers for discrimination (see also Figure 1) in the central region, while it is only necessary and useful to consider monomers and dimers in other parts of the sequence.

4 Summary

In this work we have developed a novel Multiple Kernel Learning algorithm for large-scale sequence analysis problems. The reformulation as a semi-infinite linear programming problem allowed us to reuse efficient SVM optimization implementations for finding the optimal convex combination of kernels. We proposed a simple, easy-to-implement but yet effective boosting-like technique to solve the resulting optimization problem, which comes with good convergence guarantees. The suggested column-generation technique, however, is in practice often faster but requires an efficient LP solver (such as CPLEX). Additionally, we showed that for kernels like the WD kernel an efficient SMO-like algorithm can be derived which in turn is even more efficient than the proposed column-generation algorithm, as it exploits special properties of string kernels.

In experiments on toy and splice-site detection problems we illustrated the usefulness of the Multiple Kernel Learning approach. The optimized kernel convex combination gives valuable hints at which positions discriminative oligomers of which length are hidden in the sequences. This solves to a certain extend one of the major problems with Support Vector Machines: now the decisions become interpretable. On the toy data set we re-discovered hidden sequence motifs even in presence of a large amount of noise. In first preliminary experiments on the acceptor splice site detection problem we discovered patterns in the optimized weightings which are biologically plausible. It is future work to perform a more extensive computational evaluation on splice sites and other signal detection problems.

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. We thank Alexander Zien and K.-R. Müller for great discussions and proof reading the manuscript.

N.B. The appendix contains details regarding the data generation and estimates of the computational complexity of the proposed SMO-like optimization technique. Additional information about this work can be found at `http://ida.first.fraunhofer.de/~sonne/mkl_splice`.

References

1. Francis R. Bach, Gert R. G. Lanckriet, and Michael I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Twenty-first international conference on Machine learning*. ACM Press, 2004.
2. K.P. Bennett, A. Demiriz, and J. Shawe-Taylor. A column generation algorithm for boosting. In P. Langley, editor, *Proceedings, 17th ICML*, pages 65–72, San Francisco, 2000. Morgan Kaufmann.
3. M.S. Boguski and T.M. Lowe C.M. Tolstoshev. dbEST–database for "expressed sequence tags". *Nat Genet.*, 4(4):332–3, 1993.
4. L. Breiman. Prediction games and arcing algorithms. Technical Report 504, Statistics Department, University of California, December 1997.
5. C. Cortes and V.N. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.
6. A.L. Delcher, D. Harmon, S. Kasif, O. White, and S.L. Salzberg. Improved microbial gene identification with GLIMMER. *Nucleic Acids Research*, 27(23):4636–4641, 1999.
7. Y. Engel, S. Mannor, and R. Meir. Sparse online greedy support vector regression. In *ECML*, pages 84–96, 2002.
8. Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT: European Conference on Computational Learning Theory*. LNCS, 1994.
9. Harris, T.W. et al. Wormbase: a multi-species resource for nematode biology and genomics. *Nucl. Acids Res.*, 32, 2004. Database issue:D411-7.
10. R. Hettich and K.O. Kortanek. Semi-infinite programming: Theory, methods and applications. *SIAM Review*, 3:380–429, September 1993.
11. T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *J Comput Biol.*, 7(1-2):95–114, February 2000.
12. 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.
13. W.J. Kent. Blat–the blast-like alignment tool. *Genome Res.*, 12(4):656–64, 2002.
14. R. Kuang, E. Ie, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C. Leslie. Profile-based string kernels for remote homology detection and motif extraction. In *Computational Systems Bioinformatics Conference 2004*, pages 146–154, 2004.
15. G.R.G. Lanckriet, T. De Bie, N. Cristianini, M.I. Jordan, and W.S. Noble. A statistical framework for genomic data fusion. *Bioinformatics*, 2004.
16. E.L. Lehmann. *Testing Statistical Hypotheses*. Springer, New York, second edition edition, 1997.
17. 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.
18. A.M. Mood, F.A. Graybill, and D.C. Boes. *Introduction to the Theory of Statistics*. McGraw-Hill, third edition edition, 1974.
19. 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.
20. 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.
21. G. Rätsch. *Robust Boosting via Convex Optimization*. PhD thesis, University of Potsdam, Computer Science Dept., August-Bebel-Str. 89, 14482 Potsdam, Germany, 2001.
22. G. Rätsch, A. Demiriz, and K. Bennett. Sparse regression ensembles in infinite and finite hypothesis spaces. *Machine Learning*, 48(1-3):193–221, 2002. Special Issue on New Methods for Model Selection and Model Combination. Also NeuroCOLT2 Technical Report NC-TR-2000-085.
23. G. Rätsch and S. Sonnenburg. *Accurate Splice Site Prediction for Caenorhabditis Elegans*, pages 277–298. MIT Press series on Computational Molecular Biology. MIT Press, 2003.
24. G. Rätsch and M.K. Warmuth. Marginal boosting. NeuroCOLT2 Technical Report 97, Royal Holloway College, London, July 2001.
25. Wheeler, D.L. et al. Database resources of the national center for biotechnology. *Nucl. Acids Res*, 31:38–33, 2003.
26. X.H. Zhang, K.A. Heller, I. Hefter, C.S. Leslie, and L.A. Chasin. Sequence information for the splicing of human pre-mrna identified by support vector machine classification. *Genome Res*, 13(12):637–50, 2003.
27. 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 Data Generation

A.1 Toy Data

We generated $11,000$ sequences of length 50, where the symbols of the alphabet $\{A, C, G, T\}$ follow a uniform distribution. We chose $1,000$ of these sequences to be positive examples and hid two motifs of length seven: at position 10 and 30 the motifs GATTACA and AGTAGTG, respectively. The remaining $10,000$ examples were used as negatives. Thus the ratio between examples of class $+1$ and class -1 is $\approx 9\%$. In the positive examples, we then randomly replaced $s \in \{0, 2, 4, 5\}$ symbols in each motif. Leading to four different data sets which where randomly permuted and split such that the first $1,000$ examples became training and the remaining $10,000$ validation examples.

A.2 Splice Site Sequences

EST and cDNA Sequences We collected all known *C. elegans* ESTs from Wormbase [9] (release WS118; 236,868 sequences), dbEST [3] (as of February 22, 2004; 231,096 sequences) and UniGene [25] (as of October 15, 2003; 91,480 sequences). Using *blat* [13] 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 splitted 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.

Splitting into Training, Validation and Test Sets Since the resulting data set is still pretty large, we only used sequences from randomly chosen 20% of clusters with cDNA and 30% of clusters without cDNA to generate true acceptor splice site sequences (15,507 of them). Each sequence is 398nt long and has the AG dimer at 200. Negative examples were generated from any occurring AG within the ORF of the sequence (246,914 of them were found).

For experiments on *C. elegans* we used a random subset of $60,000$ examples for testing, $100,000$ examples for parameter tuning and up to $100,000$ examples for training (unless stated otherwise).

B Estimating the Computational Complexity

The computational complexity of the proposed algorithm is determined by the complexity of training a SVM and by the number of iterations. For boosting type algorithms the number of iterations in order to achieve an accuracy of ϵ with M variables/kernels can be bounded by $\log(M)/\epsilon^2$ [24]. Moreover, the running time of SVM optimization is often between N^2 and N^3 for state-of-the-art SVM optimizers (such as *SVM-light* [12]). Hence, for the boosting-like algorithm one would expect a computational complexity of $\mathcal{O}(N^2 M \log(M)/\epsilon^2)$ (one needs an additional factor of M to prepare the combined kernels). In practice, however, the column-generation algorithm is considerably faster than the boosting-like algorithm [2]. Furthermore, for the algorithm, which performs the optimization w.r.t. the $\boldsymbol{\alpha}$'s and $\boldsymbol{\beta}$'s simultaneously, one would expect an additional speedup.

In order to obtain a more realistic estimate of the computational complexity, we ran the latter algorithm (simultaneous optimization) over a wide range of $N \in [10^2, 2 \cdot 10^5]$, $M \in [10, 398]$ and $\epsilon \in [3 \cdot 10^{-7}, 1]$ (cf. Figure 6). From these running times we coordinate-wisely estimated the powers of N, M and $\log(1/\epsilon)$, and obtained the following running time estimate for standard PC hardware (*Athlon 2200+*):

$$t(N, M, \epsilon) = cM^{2.22}N^{1.68}\log_2(1/\epsilon)^{2.52} + \mathcal{O}(1),$$

where $c \approx 5 \cdot 10^{-11}$.

One can observe that the measured computational complexity is considerably lower than expected theoretically (at least w.r.t. N and ϵ). In order to make our algorithm's running times comparable to the ones given in [1], we give two examples: For $N = 6212$, $M = 4$ and $\epsilon = 10^{-4}$, we need approximately 2 seconds, while [1] needs 670 seconds. Furthermore, for $N = 1605$, $M = 48$ and $\epsilon = 10^{-4}$, we need about 44 seconds, while [1] needs 618 seconds.

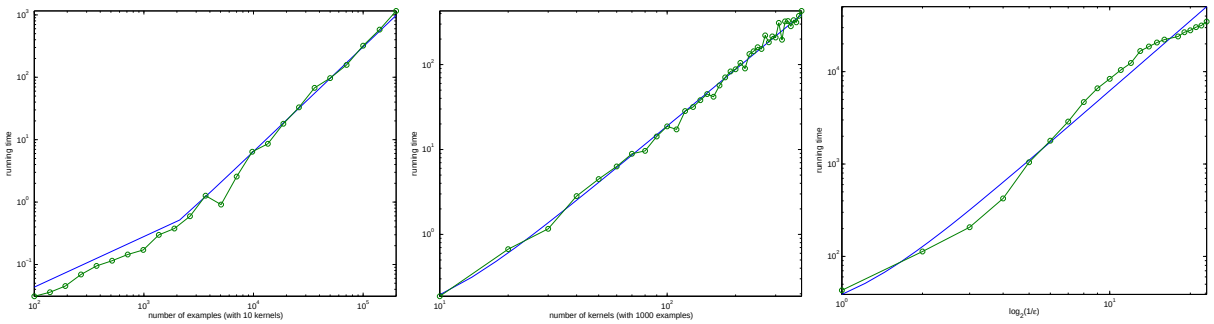


Fig. 6: Running times in seconds for various numbers of examples N (number of kernels $M = 10$, $\epsilon = 10^{-2}$), different numbers of kernels (number of examples $N = 10, 00$, $\epsilon = 10^{-2}$) and some values of ϵ (the relative gap; $N = 10,000$ and $M = 100$).