BMC Bioinformatics

BioMed Central

Proceedings

Open Access

Learning Interpretable SVMs for Biological Sequence Classification

Gunnar Rätsch*[1], Sören Sonnenburg[2] and Christin Schäfer[2]

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

Email: Gunnar Rätsch* - gunnar.raetsch@tuebingen.mpg.de; Sören Sonnenburg - soeren.sonnenburg@first.fraunhofer.de; Christin Schäfer - christin.schaefer@first.fraunhofer.de

* Corresponding author

from NIPS workshop on New Problems and Methods in Computational Biology
Whistler, Canada. 18 December 2004

Published: 20 March 2006

BMC Bioinformatics 2006, **7**(Suppl 1):S9 doi:10.1186/1471-2105-7-S1-S9

Abstract

Background: Support Vector Machines (SVMs) – using a variety of string kernels – have been successfully applied to biological sequence classification problems. While SVMs achieve high classification accuracy they lack interpretability. In many applications, it does not suffice that an algorithm just detects a biological signal in the sequence, but it should also provide means to interpret its solution in order to gain biological insight.

Results: We propose novel and efficient algorithms for solving the so-called Support Vector Multiple Kernel Learning problem. The developed techniques can be used to understand the obtained support vector decision function in order to extract biologically relevant knowledge about the sequence analysis problem at hand. We apply the proposed methods to the task of acceptor splice site prediction and to the problem of recognizing alternatively spliced exons. Our algorithms compute *sparse* weightings of substring locations, highlighting which parts of the sequence are important for discrimination.

Conclusion: The proposed method is able to deal with thousands of examples while combining hundreds of kernels within reasonable time, and reliably identifies a few statistically significant positions.

1 Background

Kernel based methods such as Support Vector Machines (SVMs) have proven to be powerful for sequence analysis problems frequently appearing in computational biology (e.g. [1-4]). They employ a so-called kernel function $k(\mathbf{s}_i, \mathbf{s}_j)$ which intuitively computes the similarity between two sequences $\mathbf{s}_i$ and $\mathbf{s}_j$. The result of SVM learning is a α-weighted linear combination of kernel elements and the bias b (see Section 4.1 for more details):

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

where the $\mathbf{s}_i$'s are N labeled training sequences ($\gamma_i \in \{\pm 1\}$). One of the problems with kernel methods compared to probabilistic methods (such as position weight matrices or interpolated Markov models [5]) 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 [3,6]). We approach this problem by considering convex combinations of M kernels, i.e.

$$k(\mathbf{s}_i, \mathbf{s}_j) = \sum_{k=1}^{M} \beta_k k_k(\mathbf{s}_i, \mathbf{s}_j) \qquad (2)$$

with $\beta_k \geq 0$ and $\sum_k \beta_k = 1$, where each kernel k_k uses only a distinct set of features of the sequence. For appropriately designed sub-kernels k_k, 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.

In this work we consider the problem of finding the optimal convex combination of kernels (i.e. determining the optimal β's in (2)). This problem is known as the *Multiple Kernel Learning* (MKL) problem [4,7,8] (see also [9,10,38] for related approaches). Sequence analysis problems usually come with large numbers of examples and one may wish to combine many kernels representing many possibly important features. Unfortunately, algorithms proposed for Multiple Kernel Learning so far are not capable of solving the optimization problem for realistic problem sizes (e.g. ≥ 10,000 examples) within reasonable time. Even recently proposed decomposition algorithms for this problem, such as the one proposed in [7], are not efficient enough since they suffer for instance from the inability to keep all kernel matrices ($K_j \in \mathbb{R}^{N \times N}$, $j = 1, \ldots, M$) in memory. (Note that kernel caching can become ineffective if the number of combined kernels is large.) 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; cf. references in [11] and also [12,39] to gain a further speedup in case of string kernels) and then performs an update of the kernel convex combination weights β. Separating the SVM optimization from the optimization of the kernel coefficients can thus lead to significant improvements for large scale problems with general kernels (cf. Section 4 for details).

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 [13]. Its main idea is to count the (exact) co-occurrence of k-mers at position l of two compared DNA sequences of equal length L (e.g. a window around some signal on the DNA). The kernel can be written as a linear combination of d parts with coefficients β_k ($k = 1, \ldots, d$):

$$\mathrm{k}(\mathbf{s}_i, \mathbf{s}_j) = \sum_{k=1}^{d} \beta_k \sum_{l=1}^{L-k} \mathbf{I}(\boldsymbol{u}_{k,l}(\mathbf{s}_i) = \boldsymbol{u}_{k,l}(\mathbf{s}_j)), \qquad (3)$$

where L is the length of the sequences s, d is the maximal oligomer length considered and $\boldsymbol{u}_{k,l}(\mathbf{s})$ is the oligomer of length k at position l of sequence s. Moreover, $\mathbf{I}(true) := 1$ and 0 otherwise.

One question is how the weights β_k for the various k-mers in (3) should be chosen. So far, only heuristic settings in combination with expensive model-selection methods have been used (e.g. [13]). The MKL approach offers a clean and efficient way to find the optimal weights β: We define d kernels

$$\mathrm{k}_k(\mathbf{s}_i, \mathbf{s}_j) = \sum_{l=1}^{L-k} \mathbf{I}(\boldsymbol{u}_{k,l}(\mathbf{s}_i) = \boldsymbol{u}_{k,l}(\mathbf{s}_j)), \qquad (4)$$

and then optimize the convex combination of these kernels (with coefficients β) using the MKL algorithm (cf. (3)). The optimal weights β indicate which oligomer lengths are important for the classification problem at hand (see results in Section 2.2).

Additionally, it is interesting to introduce an importance weighting over the positions in the subsequence. Hence, we define a separate kernel for each position and each oligomer length, 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

$$\begin{aligned} \mathrm{k}(\mathbf{s}_i, \mathbf{s}_j) = & \\ & \sum_{k=1}^{d} \sum_{l=1}^{L-k} \beta_{k,l} \mathbf{I}(\boldsymbol{u}_{k,l}(\mathbf{s}_i) = \boldsymbol{u}_{k,l}(\mathbf{s}_j)) \\ & = \sum_{k,l} \beta_{k,l} k_{k,l}(\mathbf{s}_i, \mathbf{s}_j). \end{aligned} \qquad (5)$$

The simpler case would be to only consider one kernel per position in the sequence: $\mathrm{k}(\mathbf{s}_i, \mathbf{s}_j) = \sum_{l=1}^{L} \beta_l \mathrm{k}_l(\mathbf{s}_i, \mathbf{s}_j)$ with

$$\mathrm{k}_l(\mathbf{s}_i, \mathbf{s}_j) = \sum_{k=1}^{d} \gamma_k \mathbf{I}(\boldsymbol{u}_{k,l}(\mathbf{s}_i) = \boldsymbol{u}_{k,l}(\mathbf{s}_j)), \qquad (6)$$

where γ is the default weighting as used in [13].

Obviously, if one would be able to obtain an accurate classification by a *sparse* weighting $\beta_{k,l}$, then one can quite 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 within the exon capturing the nucleotide composition (short oligomers; cf. Sections 2.4 and 2.5).

While the proposed MKL algorithms are applicable to arbitrary kernels, we particularly consider the case of string kernels and show how their properties can be

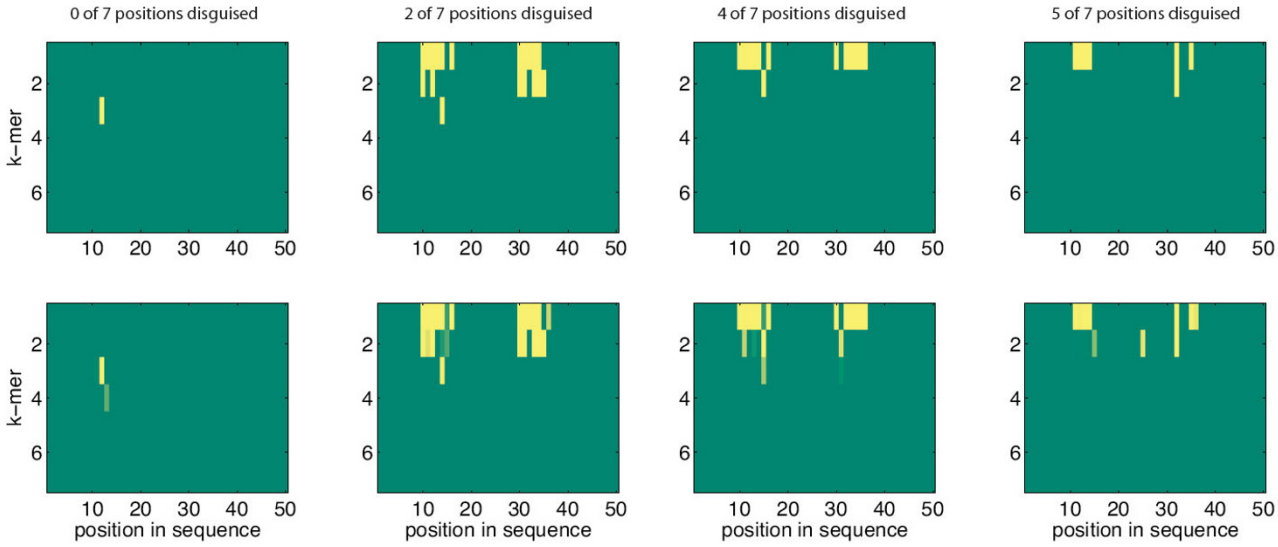

Figure 1

In this "figure matrix", columns correspond to the noise level, i.e. different numbers of nucleotides randomly substituted in the motif of the toy data set (cf. Appendix A.1). Each sub-figure shows a matrix with each element corresponding to one kernel weight: columns correspond to weights used at a certain sequence position (1–50) and rows to the oligomer length used at that position (1–7). The first row of the figure matrix shows the kernel weights that are significant, while the second row depicts the likelihood of every weight to be rejected under $\mathcal{H}_0$.

exploited in order to significantly speedup the computations. We extend previous work by [8,14,15] and employ tries [16] during training and testing. In Section 4 we develop a method that can avoid kernel caching and we therefore obtain very memory efficient and fast algorithms (which also speedup standard SVM training).

By bootstrapping and applying a combinatorial argument, we derive a statistical test that discovers the most important kernel weights. Using this test, we elucidate on simulated pseudo-DNA sequences with two hidden 7-mers which *k*-mers in the sequence were used for the SVM decision. Additionally we apply our method to the problem of splice site classification (*C. elegans* acceptor sites) and to the problem recognizing alternatively spliced exons [17].

2 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. As a first test we apply our method to a toy problem where everything is known and we can directly validate the findings of our algorithm with the underlying truth. As a next step, we show that our MKL algorithm performs as well or slightly better than the standard SVM and leads to SVM classification functions that are computationally more efficient. 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.

2.1 MKL Learning Detects Motifs in Toy Data set

As a proof of concept, we test our method on a toy data set with two hidden 7-mers (at positions 10 & 30) at four different noise levels (we used different numbers of random positions in the 7-mers that were replaced with random nucleotides; for a detailed description of the data see Appendix A. 1). We use the kernel as defined in (5) with one sub-kernel per position and oligomer length. We consider sequences of length $L = 50$ and oligomers up to length $d = 7$, leading to $M = 350$ sub-kernels. For every noise level, we train on 100 bootstrap replicates and learn the 350 WD kernel parameters in each run. On the resulting 100 weightings we performed the reliability test (cf. Section 4.3). The results are shown in Figure 1 (columns correspond to different noise levels – increasing from left to right). Each figure shows a kernel weighting β, 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 the null hypothesis $\mathcal{H}_0$ is illustrated in the plots in the second row (cf. Section 4.3 for details). Bright colors mean that it is more likely to reject $\mathcal{H}_0$.

As long as the noise level does not exceed 2/7, longer 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% (not shown), but only weights in the correct range 10 ... 16 and 30 ... 36 are found to be significant.

2.2 Optimization of WD Kernel Weights Speeds up Computations and Improves Accuracy

We compare the standard SVM with WD kernel (default weighting as in [13]) and kernel caching (SVM-light implementation [18]) and our MKL-SVM algorithm with WD kernel (optimized weighting) and using tries (cf. Section 4). We applied both algorithms on the *C. elegans* acceptor splice data set 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 data set each sequence is a window centered around a AG dimer containing 141 nucleotides (nt), together with the corresponding label +1 for true acceptor splice sites and -1 for decoys (cf. [13] and Appendix A.2 for more details). Using this setup we perform 5-fold cross-validation over the maximal oligomer length $d \in \{10,12,15,17,20\}$ (cf. (3)) and the SVM regularization constant $C \in \{0.5, 2, 5, 10\}$. A detailed comparison of the WD kernel approach with other state-of-the-art methods is provided in [13] and goes beyond the scope of this work.

On the validation set we find that for the SVM using the standard WD kernel (using the default weighting), $d = 20$ and $C = 0.5$ gives best classification performance (ROC score 99.66% on validation set), while the MKL-SVM using the WD kernel (optimized weighting) gives best results for $d = 12$ and $C = 1$ (ROC score also 99.66% on validation set). Figure 2 shows the WD kernel weights computed by the MKL-SVM approach. It suggests that 12-mers and 6-mers seem to be of high importance and 1-4-mers are also important. On the test data set the resulting SVM classifier with standard WD kernel performs as good as on the validation data set (ROC score 99.66% again), while the classifier obtained by MKL-SVMs with optimized WD kernel weights achieves a 99.67% ROC score. Astonishingly training the MKL-SVM (i.e. with weight optimization and tries) was 1.5 times faster than training the original SVM (with kernel caching). Also, the resulting classifier provided by the new algorithm is considerably faster than the one obtained by the classical SVM since many β-weights are zero (see also [19]).

It should be noted that the obtained weighting in this experiment is only partially useful for interpretation. In the case of splice site detection, it is unlikely that k-mers of length 12 play the most important role. More likely to be important are oligos of length up to six. We believe that the large weight for the longest oligo is an artifact which comes from the fact that we are combining kernels with quite different properties. (The 12th kernel leads to a kernel matrix that is most diagonally dominant, which we believe is the reason for having a large weight. This problem can be partially alleviated by including the identity matrix in the convex combination. However as ℓ_2-norm soft margin SVMs can be implemented by adding a constant to the diagonal of the kernel [20,21], this leads effectively to an additional ℓ_2-norm penalization.) In the following example we consider one weight per position. In this case the combined kernels are more similar to each-other and we expect more interpretable results.

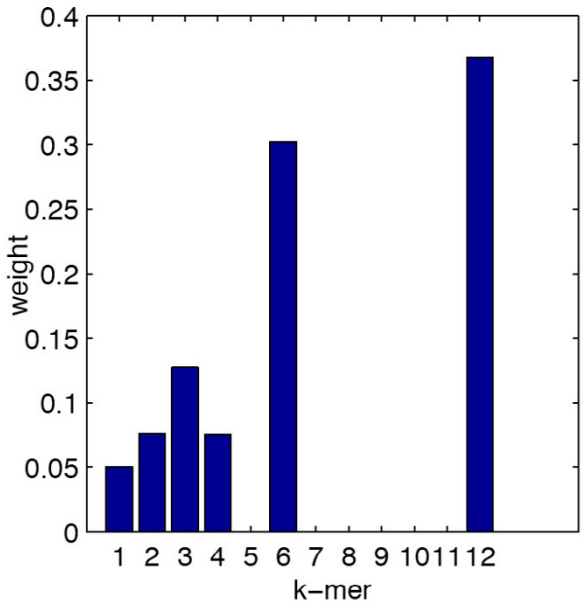

Figure 2
Optimized WD Kernel Weights.

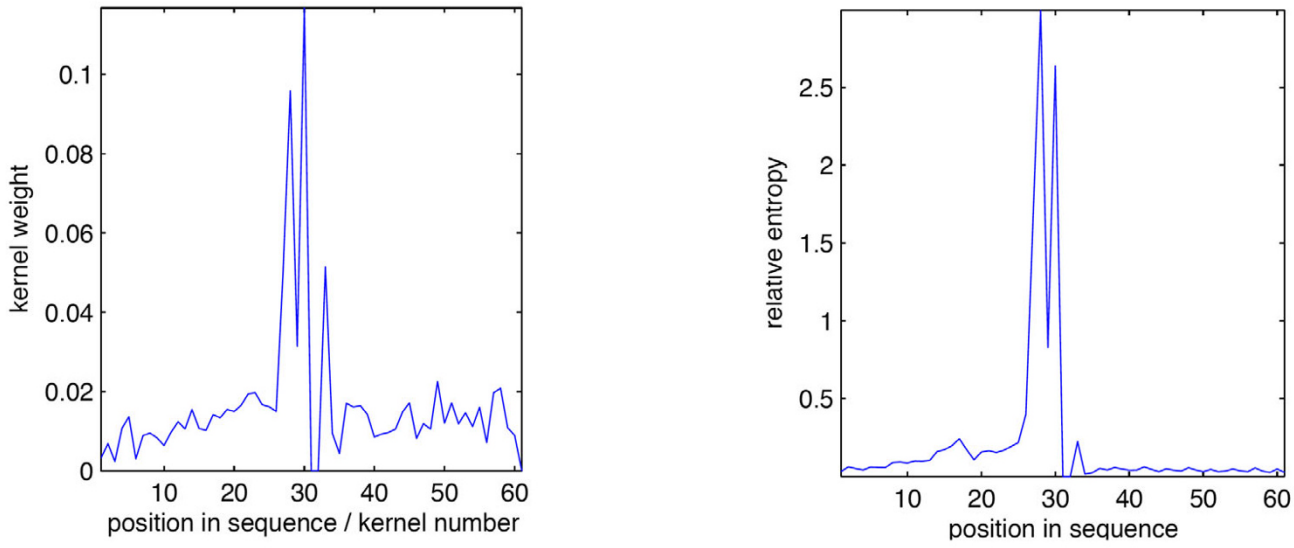

Figure 3
(left) Value of the learned weightings of an SVM with a WD kernel of 60 first-order sub-kernels, (right) relative entropy obtained between the Positional Weight Matrices for the positive and the negative class, both trained for acceptor splice site detection.

2.3 Optimal Positional Importance Weighting is Related to Positional Weight Matrices

An interesting relation of the learned weightings to the relative entropy between Positional Weight Matrices (PWMs) can be shown with the following experiment: We train an SVM with a WD kernel that consists of 60 *first-order* sub-kernels (i.e. only single nucleotide matches are considered) on acceptor splice sites from *C. elegans* (100,000 sequences for training, 160,000 sequences for validation). The characteristic acceptor splice site AG dimer is at positions 31 & 32. We extracted the sequences from a window (-30, +28) around the dimer. The learned weights β_k are shown in Figure 3 (left). For comparison we computed the PWMs (Markov chains of zero-th order) for the positive and the negative class separately (denoted by $p^+_{i,j}$ and $p^-_{i,j}$). Additionally, we computed the relative entropy Δ_i between the two probability estimates $p^+_{i,j}$ and $p^-_{i,j}$ at each position j by $\Delta_j = \sum_{i=1}^{4} p^+_{i,j} \log(p^+_{i,j} / p^-_{i,j})$, leading to Figure 3 (right). The shape of both plots is quite 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 zero-th order information. Nevertheless the classification accuracy is already quite 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%.

2.4 Positional WD Kernel Weights Helps Understanding Splice Site Classification

Note that Markov chains become intractable and less accurate for high orders, which seem on the other hand necessary for achieving high accuracies in many sequence analysis tasks. SVMs, however, are efficient and accurate even for great oligomer lengths. We therefore expect that MKL-SVMs may also in this case provide useful insights at which positions the discriminative information is hidden.

In order to illustrate this idea we perform another experiment: We considered the larger region from -50 nt to +60 nt around the splice site and used the WD kernel with d = 15. We defined a kernel for every position that only accounts for substrings that start at the corresponding position (up to length 15). To get a smoother weighting and to reduce the computing time we only used [111/2] = 56 weights (combining every two positions to one weight). Figure 4 shows the average computed weighting on ten bootstrap runs trained on about 65,000

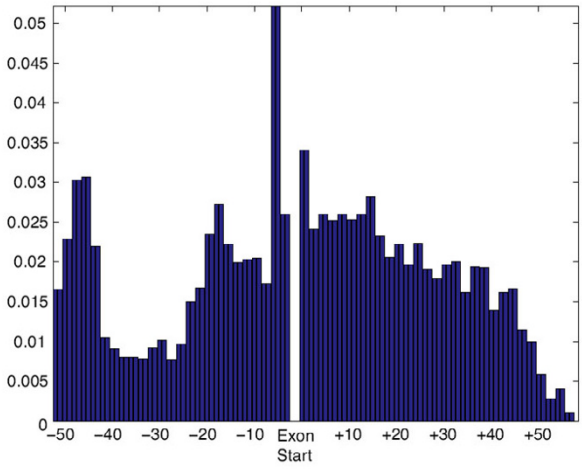

Figure 4
Optimized WD kernel weights considering subsequences starting at different positions (one weight per two positions).

examples. Several regions of interest can be identified: a) The region -50 nt to -40 nt, which corresponds to the donor splice site of the previous exon (many introns in *C. elegans* are very short, often only 50 nt), b) the region -25 nt to -15 nt that coincides with the location of the branch point, c) the intronic region closest to the splice site with greatest weight (-8 nt to -1 nt; the weights for the AG dimer are zero, since it appears in splice sites and decoys) and d) the exonic region (0 nt to +50 nt). 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 nt to +45 nt might be explained by the fact that not all exons are actually long enough. Furthermore, since the sequence ends in our case at +60 nt, the decay after +45 nt is an edge effect as longer substrings cannot be matched.

2.5 Finding Motifs for Splice Site Detection

We again consider the classification of acceptor splice sites against non-acceptor splice sites (with centered AG dimer) from the *C. elegans* (cf. Appendix A.2 for details on the generation of the data sets). We trained our Multiple Kernel Learning algorithm ($C = 2$) on 5,000 randomly chosen sequences of length 111 nt 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 1). We can observe (cf. Figure 5b&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 (cf. Figure 5a) and the oligomer-length weight vector (cf. Figure 5d), 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 2) in the central region, while it is only necessary and useful to consider monomers and dimers in other parts of the sequence.

2.6 Understanding the Recognition of Alternatively Spliced Exons

In this section we consider the problem of recognizing one major form of alternative splicing, namely the exclusion of exons from the transcript. It has been shown that alternatively spliced exons have certain properties that distinguish them from constitutively spliced exons (cf. [17] and references therein). In [17] we developed a method that only uses information that is available to the splicing machinery, i.e. the DNA sequence itself, and accurately distinguishes between alternatively and constitutively spliced exons (50% true positive rate at a 1% false positive rate; see http://www.fml.tuebingen.mpg.de/raetsch/projects/RASE for more details). Using our MKL method we have identified regions near the 5' and 3' end of the considered exons that carry most of the discriminative information. We show that these regions contain many hexamers that are significantly more often present than average in constitutively spliced exons.

In order to recognize alternatively spliced exons we consider the 5' and 3' end of the exons separately and use an extended version of the WD kernel (exhibiting an improved positional invariance, cf. [17]) on a 201 nt window centered around the exon start and end together with additional kernels capturing information about the length of the exon and the flanking introns [17].

To interpret the SVM classifiers result we employ Multiple Kernel Learning to determine the weights $\boldsymbol{\beta}^{5'}$ and $\boldsymbol{\beta}^{3'}$ for the two WD kernels around the acceptor (5') and donor (3') site. In Figure 6 the learned weighting is shown (weights for other subkernels not shown). A higher weight at a certain position in the sequence corresponds to an increased importance of substrings starting at this location. Given this weighting, we can identify five regions

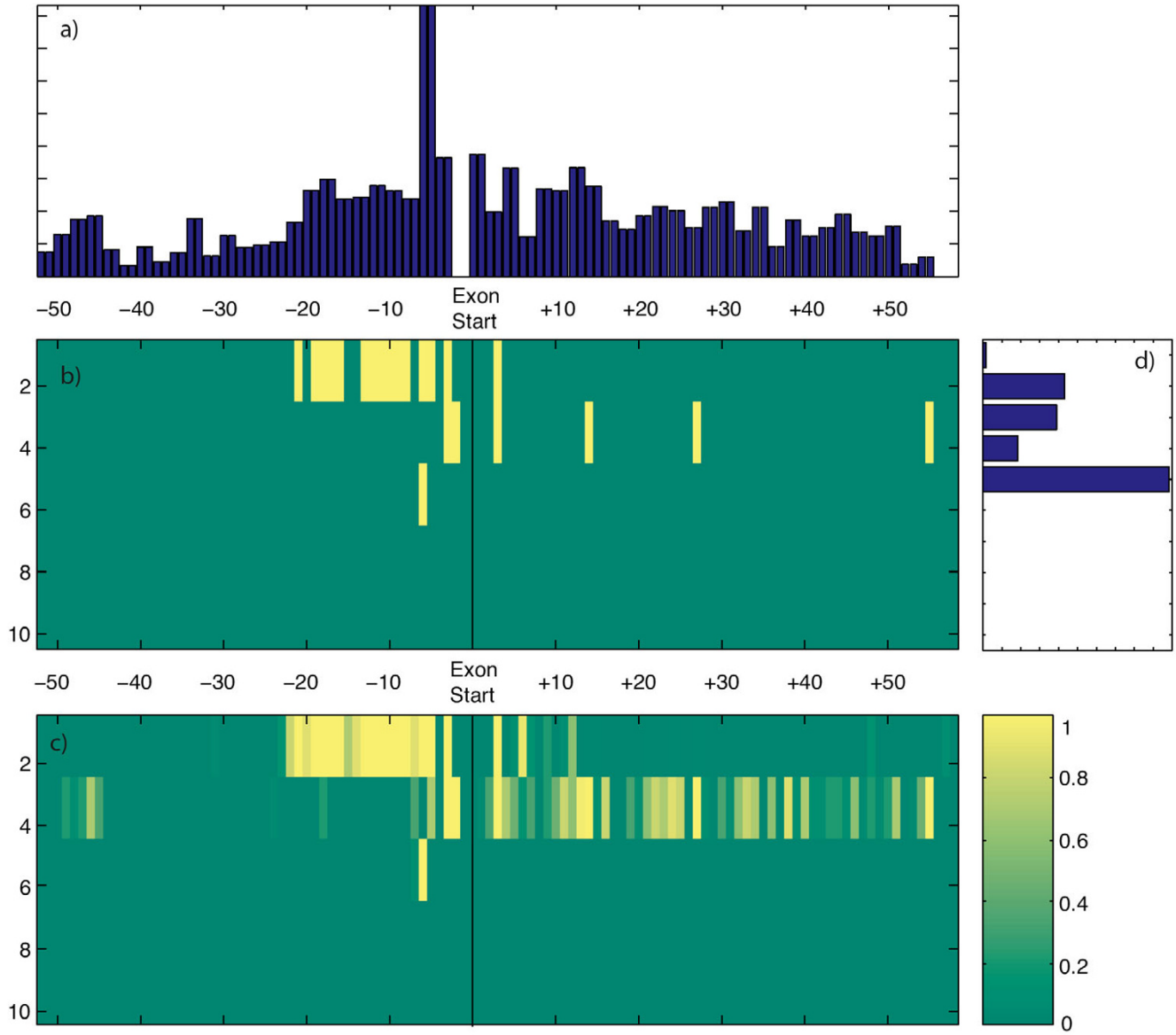

Figure 5

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

which seem particularly important for discrimination: a-b) within the upstream intron the region -70 nt to -40 nt and -30 nt to 0 nt (relative to the end of the intron), c) the exon positions +30 nt to +70 nt (relative to the beginning of the exon) and d) -90 nt to -30 nt (relative to the end of the exon). And finally e) the downstream intron positions 0 – 70 nt (relative to the beginning of the intron).

To illustrate that these regions represent distinct discriminative features for the problem at hand, we counted the occurrence of all *hexamers* in the positive and negative examples. Using the frequency p^- of occurrence of a hexamer in the negative examples as background model, we computed how likely it is to observe the frequency p^+ in the positive sequences (E-value; using the binomial distribution). In Table 1 we display for each of the five regions

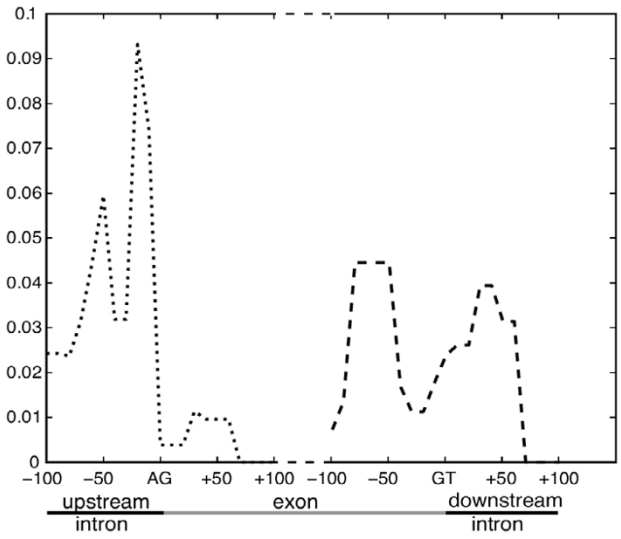

Figure 6
We use Multiple Kernel Learning to determine weights for the WD kernel. Shown is learned weighting for the WD kernel at the acceptor and at the donor site. From areas of higher weight (upstream intron: regions -70 nt to -40 nt and -30 nt to 0 nt, Exon: +30 nt to +70 nt and -30 nt to -90 nt, downstream intron 0 nt to +70 nt) overrepresented hexamers have been extracted and are shown in Table 1.

the six hexamers with highest E-value. In region a) the motif CTAACC frequently appears in various variations, while region b) is rich with C's and T's. Particularly interesting seem the motifs AGTGAG and CAGCAG which only appear significantly in the region near the exon start and exon end, respectively. The downstream intron contains many G's and T's. (Members of the CELF gene family bind for instance to GT-rich regions; A. Zahler, personal communication).) A more complete list of the over-represented hexamers are found on the supplementary web-site http://www.fml.tuebingen.mpg.de/raetsch/projects/RASE.

3 Conclusion

In this work we have developed a novel Multiple Kernel Learning algorithm applicable to large-scale sequence analysis problems that additionally assists in understanding how decisions are made. Using a novel reformulation of the MKL problem, we were able to reuse available SVM implementations that, in combination with using tries, have lead us to a very efficient MKL algorithm suitable for the analysis of large scale sequence analysis problems. In experiments on toy, splice-site detection and alternative exon recognition problems we have 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 located in the sequences. This solves to a certain extent 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 the first experiments on the acceptor splice site detection problem we discovered patterns in the optimized weightings which are biologically plausible. For the recognition of alternatively spliced exons we have identified several regions near the 5' and 3' end of the exons that display distinguished patterns. It is future work to extend our computational evaluation and to consider other signal detection problems.

4 Methods

4.1 Support Vector Machines

We use Support Vector Machines [22] which are extensively studied in the literature (e.g. [11,20,21]). Their classification function can be written as in (1). The α_i's are the Lagrange multipliers and b is the usual bias which are the results of SVM training. The kernel k is the *key ingredient* for learning with SVMs. It implicitly defines the feature space and the mapping Φ via

$$k(s,s') = \langle \Phi(s), \Phi(s') \rangle.$$

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 s, a dimension of Φ (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 $(\mathbf{s}_i, \gamma_i)$ $(i = 1, ..., N)$, the SVM solution is obtained by solving the following

optimization problem that maximizes the soft margin between both classes [23]:

$$\begin{aligned} \min \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{w.r.t.} \quad & \mathbf{w} \in \mathbb{R}^D, b \in \mathbb{R}, \boldsymbol{\xi} \in \mathbb{R}_+^N \\ \text{s.t.} \quad & \gamma_i\left(\langle \mathbf{w}, \Phi(\mathbf{s}_i)\rangle + b\right) \geq 1 - \xi_i, \quad i = 1,...,N, \end{aligned} \tag{7}$$

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:

Table 1: Shown are the top six ranked hexamers (by E-value) extracted for the upstream intron the in between exon and the following downstream exon. The first column in the upper part shows the most important hexamers in the intron for the region -70 nt to -40 nt relative to the end of the intron. The lower part states 6-mers contained -30 nt until the end of the upstream intron. Similarly the second column displays hexamers in the exon from +30 nt to +70 nt (upper half, relative to exon start) and -30 nt to - 90 nt (lower part, relative to exon end) and the last column 6-mers in the downstream intron from 0 nt to +70 nt.

Upstr. Intron		Exon		Downstr. Intron	
6-mer	E-val.	6-mer	E-val.	6-mer	E-val.
CTAACC	1.2e-17	AGTGAG	4.2e-11	TGTGTG	5.9e-31
CCCCCC	3.8e-11	TTTTTT	2.7e-09	TTGTGT	1.7e-24
TAACCC	9.8e-10	ATATAT	1.3e-08	GTGTGT	3.6e-16
CACTTT	6.2e-09	TATATA	3.6e-07	GTTGTG	4.4e-15
ATCCCC	1.6e-07	ATAGGT	4.8e-07	TGTTGT	3.3e-14
CTTTCC	2.4e-07	TAGGTT	5.0e-07	TGCATG	1.3e-13
CATTCT	1.3e-09	TTTAAA	1.8e-12		
CTCTCT	1.9e-09	AATTTT	2.2e-10		
GCATGT	4.4e-09	ATTTTA	2.9e-09		
GTTGTC	4.4e-09	CAGCAG	1.2e-08		
TCTCTA	2.2e-08	TAATTT	8.3e-08		
CTCTAT	1.1e-07	TTCCCC	2.1e-07		

$$\begin{aligned} &\max \quad \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j\gamma_i\gamma_j \mathrm{k}(\mathbf{s}_i,\mathbf{s}_j), \\ &\text{w.r.t.} \quad \boldsymbol{\alpha} \in \mathbb{R}_+^N \text{ with } \boldsymbol{\alpha} \le C \text{ and } \sum_{i=1}^{N}\alpha_i\gamma_i = 0. \end{aligned} \tag{8}$$

Note that there exist a large variety of different software packages that can efficiently solve the above optimization problem even for more than one hundred thousand of examples (cf. references in [11] and also [12] to gain further speedups when string kernels are used).

4.2 The Multiple Kernel Learning Optimization Problem

4.2.1 Idea

In the Multiple Kernel Learning (MKL) problem one is given N data points $(\tilde{\mathbf{s}}_i, \gamma_i)$ ($\gamma_i \in \{\pm 1\}$), where $\tilde{\mathbf{s}}_i$ is subdivided into M components $\tilde{\mathbf{s}}_i = (\mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,M})$ with $s(i,j) \in \mathbb{R}^{k_j}$ and k_j is the dimensionality of the j-th component. Then one solves the following convex optimization problem [7], which is equivalent to the linear SVM for $M = 1$:

$$\begin{aligned} &\min \quad \frac{1}{2}\left(\sum_{j=1}^{M} d_j\beta_j \left\|\mathbf{w}_j\right\|_2\right)^2 + C\sum_{i=1}^{N}\xi_i \\ &\text{w.r.t.} \quad \mathbf{w} = (\mathbf{w}_1,\ldots,\mathbf{w}_M), \mathbf{w}_j \in \mathbb{R}^{k_j}, \boldsymbol{\xi} \in \mathbb{R}_+^N, \boldsymbol{\beta} \in \mathbb{R}_+^M, b \in \mathbb{R} \\ &\text{s.t.} \quad \gamma_i\left(\sum_{j=1}^{M}\beta_j\left\langle \mathbf{w}_j, \mathbf{s}_{i,j}\right\rangle + b\right) \ge 1 - \xi_i, \forall_i = 1,\ldots,N \\ &\qquad \sum_{j=1}^{M}\beta_j = 1, \end{aligned} \tag{9}$$

where d_j is a prior weighting of the kernels (in [7], $d_j = 1/\sum_i \left\langle s_{i,j}, s_{i,j}\right\rangle$ 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 β 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 [24]. Thus the above optimization problem offers the possibility to find sparse solutions on the block level with non-sparse solutions within the blocks.

4.2.2 Reformulation as a Semi-Infinite Linear Program

The above optimization problem can also be formulated in terms of support vector kernels [7]. 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 [7] it has been shown that the following optimization problem is equivalent to (9):

$$\begin{aligned} \min \quad & \frac{1}{2}\gamma^2 - \sum_i \alpha_i \\ \text{w.r.t.} \quad & \gamma \in \mathbb{R}, \boldsymbol{\alpha} \in \mathbb{R}^N \\ \text{s.t.} \quad & 0 \le \boldsymbol{\alpha} \le C, \sum_i \alpha_i y_i = 0 \\ & \underbrace{\sum_{r,s} \alpha_r \alpha_s y_r y_s (\boldsymbol{K}_j)_{r,s} \le \gamma^2}_{=:S_j(\boldsymbol{\alpha})} \end{aligned} \tag{10}$$

In order to solve (10), one may solve the following saddle point problem (Lagrangian):

$$L := \frac{1}{2}\gamma^2 - \sum_i \alpha_i + \sum_{j=1}^{M} \beta_j (S_j(\boldsymbol{\alpha}) - \gamma^2) \tag{11}$$

minimized w.r.t. $\alpha \in \mathbb{R}_+^N$, $\gamma \in \mathbb{R}$ (subject to $\alpha \le C$ and $\sum_i \alpha_i y_i = 0$) and maximized w.r.t. $\beta \in \mathbb{R}_+^M$. Setting the derivative w.r.t. to γ to zero, one obtains the constraint $\sum_j \beta_j = \frac{1}{2}$ and (11) simplifies to:

$$L := \underbrace{\frac{1}{2}\sum_{j=1}^{M} \beta_j S_j(\boldsymbol{\alpha}) - \sum_i \alpha_i}_{=:S(\boldsymbol{\alpha})} \tag{12}$$

Assume α^* would be the optimal solution, then $\theta^* := S(\alpha^*) - \sum_i \alpha_i$ is minimal and, hence, $S(\alpha) - \sum_i \alpha_i \ge \theta^*$ for all α (subject to the above constraints). Hence, finding a saddle-point of (12) is equivalent to solving the following semi-infinite linear program:

$$\begin{aligned} \max \quad & \theta \\ \text{w.r.t.} \quad & \theta \in \mathbb{R}, \beta \in \mathbb{R}_+^M \text{ with } \sum_j \beta_j = 1 \\ \text{s.t.} \quad & \sum_{j=1}^{M} \beta_j \left(\frac{1}{2} S_j(\boldsymbol{\alpha}) - \sum_i \alpha_i \right) \ge \theta \\ & \text{for all } \boldsymbol{\alpha} \text{ with } 0 \le \boldsymbol{\alpha} \le C \text{ and } \sum_i y_i \alpha_i = 0 \end{aligned} \tag{13}$$

4.2.3 A Column Generation Method

Note that there are infinitely many constraints (one for every vector α). Typically algorithms for solving semi-infinite problems work by iteratively finding violated constraints, i.e. α vectors, for intermediate solutions (β, θ). Then one adds the new constraint (corresponding to the new α) and resolves for β and θ [25]. The pseudo-code is outlined in Algorithm 1. Note, however, that there are no known convergence rates for such algorithms [25], but it often converges to the *optimal* solution in a small number of iterations [26,27]. (It has been shown that solving semi-infinite problems like (13), using a method related to boosting (e.g. [28]) one needs at most $T = O\left(\log(M)/\varepsilon^2\right)$ iterations, where $\hat{\varepsilon}$ is the *unnormalized* constraint violation and the constants may depend on the kernels and the number of examples N [24,29]. At least for not too small values of $\hat{\varepsilon}$ this technique produces reasonably fast good approximate solutions. See [8] for more 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.

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

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

where (β^t, θ^t) is the optimal solution at iteration t - 1 and α^t corresponds to the newly found maximally violating constraint of the next iteration (i.e. the SVM solution for weighting β^t; cf. Algorithm 1 for details). We usually only try to approximate the optimal solution and stop the optimization as soon as $\varepsilon_t \le \varepsilon$, were ε was set to 10^{-4} or 10^{-3} in our experiments.

4.2.4 A chunking algorithm for simultaneous optimization of α and β

Usually it is infeasible 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 (8)).

The general idea of Chunking and Sequential Minimal Optimization (SMO) algorithm has been proposed by [30,31] and is implemented in many SVM software packages. Here we would like to propose an extension of the Chunking algorithm to optimize the kernel weights β and the example weights α at the same time. The algorithm is motivated from an insufficiency of the column-generation algorithm described in the previous section: If the β's are not optimal yet, then the optimization of the α's until optimality is not necessary and therefore inefficient. It would be considerably faster if for any newly obtained α in the Chunking iterations, we could efficiently recompute the optimal β and then continue optimizing the α's using the new kernel weighting.

Intermediate Recomputation of β Recomputing β involves solving a linear program and the problem grows with each additional α-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 from growing arbitrarily and (2) for Simplex-based LP optimizers such as CPLEX there exists the so-called *hot-start feature* which allows one to efficiently recompute the new solution, if one, for instance, only adds a few additional constraints. The SVM-light optimizer which we are going to modify, 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 [18]. 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 β's change, then $\hat{f}_j$ can be quite efficiently recomputed by $\hat{f}_j = \sum_k \beta_k f_{k,j}$).

Faster α Optimization using Tries Finally, in each iteration the Chunking optimizer may change a subset of the α'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 since the effect of the kernel caches degrades drastically in the case of having many kernels. Fortunately, for the WD kernel there is a way to avoid this problem by using so-called tries (cf. [16]; similarly proposed by [14] and others). While we cannot improve a single kernel evaluation (which is already $O(L)$), 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 trie for each position $l = 1, \ldots, L$ of the sequence. We propose to attach weights to internal nodes and the leaves of the trie, allowing an efficient storage of weights for k-mers ($1 \le k \le d$). Now we may add all k-mers ($k = 1, \ldots, d$) of $\mathbf{s}_i$ ($i \in I$) starting at position l to the trie associated with position l (using weight $\alpha_i \beta_k$; operations per position: $O(d|I|)$). Once created, the l-th trie can be traversed down in order to lookup which k-mers in a test sequence (starting at position l) have a non-zero contribution to $g(\mathbf{s})$:

Following the path defined by the k-mer $\boldsymbol{u}$ one adds all weights along the way and stops when no children exists (see Figure 7). Note that we now can compute g in $O(Ld)$ operations (compared to $O(|I|Ld)$ in the original formulation). Empirically we noticed that the proposed Chunking 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 2 we only used the Chunking algorithm with a chunk size of $Q = 41$.

The pseudo-code of the algorithm which takes the discussion of this section into account is displayed in Algorithm 2.

4.3 Estimating the Reliability of a Weighting

Finally we want to assess the reliability of the learned weights β. For this purpose we generate T bootstrap samples and rerun the whole procedure resulting in T weightings β^t.

To test the importance of a weight $\beta_{k,i}$ (and therefore the corresponding kernels for position and oligomer length)

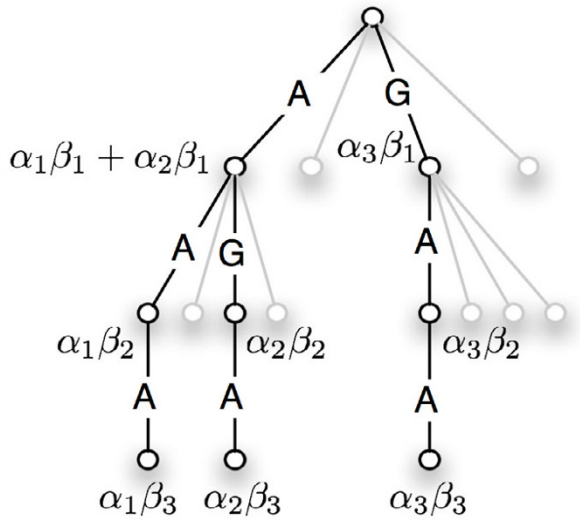

Figure 7
Three sequences AAA, AGA, GAA beeing added to the trie. The plot displays the resulting weights at the nodes.

we apply the following method: We define a Bernoulli variable $X^t_{k,i} \in \{0,1\}$, $k = 1, ..., d$, $i = 1, ..., L$, $t = 1, ..., T$ 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 binomial distribution Bin (T, p_0), p_0 unknown. We estimate p_0 with $\hat{p}_0 = \#(\beta^t_{k,i} > \tau)/TM$, 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, i.e. the null-hypothesis is $\mathcal{H}_0 : p \le c^*$ (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 β. 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:

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

For further details on the test see [32] or [33]. This test is carried out for every $\beta^t_{k,i}$. (We assume independence between the weights in one single β, 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.

Authors' contributions

GR proposed and implemented the SILP formulation of the MKL problem, prepared data sets, drafted the manuscript and helped in carrying out experiments. SS invented the Weighted Degree Kernel, analyzed several weighting schemes and reformulated it as a MKL problem, helped implementing the MKL algorithms and carried out most of the experiments. CS developed the statistical significance test and critically revised the article.

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 ≈ 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

We collected all known *C. elegans* ESTs from Wormbase [34] (release WS118; 236,868 sequences), dbEST [35] (as of February 22, 2004; 231,096 sequences) and UniGene [36] (as of October 15, 2003; 91,480 sequences). Using *blat* [37] we aligned them against the genomic DNA (release WS118). We refined the alignment by correcting typical sequencing errors, for instance by removing minor insertions and deletions. If an intron did not exhibit the GT/AG or GC/AG dimers at the 5' and 3' ends, respectively, then we tried to achieve this by shifting the boundaries up to 2 nucleotides. 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 agreeing alignments, leading to 135,239 unique EST-based sequences. We repeated the above 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-based sequences.

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 l00 nt 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.

Since the resulting data set is still too 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 398 nt long and has the AG dimer at position 200. Negative examples were generated from any occurring AG within the ORF of the sequence (246,914 of them were found). 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).

Algorithms

Algorithm 1 The column generation algorithm employs a linear programming solver to iteratively solve the semi-infinite linear optimization problem (13). The accuracy parameter ε is a parameter of the algorithm.

$D^0 = 1$, $\theta^1 = 0$, $\beta_k^t = \frac{1}{M}$ for $k = 1, ..., M$

for $t = 1,2, ...$ **do**

obtain SVM's α^t with kernel $k^t(\mathbf{s}_i, \mathbf{s}_j) := \sum_{k=1}^{M} \beta_k^t k_k(\mathbf{s}_i, \mathbf{s}_j)$

for $k = 1, ..., M$ do

$$D_k^t = \frac{1}{2}\sum_{r,s} \alpha_r^t \alpha_s^t \gamma_r \gamma_s k_k(\mathbf{s}_r, \mathbf{s}_s) - \sum_r \alpha_r^t$$

end for

$$D^t = \sum_{k=1}^{M} \beta_k^t D_k^t$$

if $\left|1 - \frac{D^t}{\theta^t}\right| \le \varepsilon$ **then break**

$(\beta^{t+1}, \theta^{t+1})$ = argmax θ

w.r.t $\beta \in \mathbb{R}_+^M$, $\theta \in \mathbb{R}$ with $\sum_k \beta_k = 1$

$$\text{s.t.} \quad \sum_{k=1}^{M} \beta_k D_k^r \ge \theta \text{ for } r = 1, ..., t$$

end for

Algorithm 2 Outline of the Chunking algorithm (extension to SVM-light) that optimizes α and the kernel weighting β 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.

$f_{k,i} = 0$, $\hat{f}_i = 0$, $\alpha_i = 0$, $\beta_k^t = \frac{1}{M}$ for $k = 1, ..., M$ and $i = 1, ..., N$

for $t = 1, 2, ...$ **do**

Check optimality conditions and stop if optimal

select **Q** suboptimal variables $i_1, ... i_Q$ based on $\hat{\mathbf{f}}$ and α

$\alpha^{old} = \alpha$

solve (8) with respect to the selected variables and update α

create trie-structures to prepare efficient computation of $g_k(\mathbf{s}) = \sum_{q=1}^{Q} (\alpha_{i_q} - \alpha_{i_q}^{old}) \gamma_{i_q} k_k(\mathbf{s}_{i_q}, \mathbf{s})$

$f_{k,i} = f_{k,i} + g_k(\mathbf{s}_i)$ for all $k = 1, ..., M$ and $i = 1, ..., N$

for $k = 1, ..., M$ **do**

$$D_k^t = \tfrac{1}{2}\sum_r f_{k,r} \alpha_r \gamma_r - \sum_r \alpha_r$$

end for

$$D^t = \sum_{k=1}^{M} \beta_k^t D_k^t$$

if $\left|1 - \frac{D^t}{\theta^t}\right| \ge \varepsilon$

$(\beta^{t+1}, \theta^{t+1})$ = argmax θ

w.r.t $\beta \in \mathbb{R}_+^M$, $\theta \in \mathbb{R}$ with $\sum_k \beta_k = 1$

s.t. $\sum_{k=1}^{M} \beta_k D_k^r \ge \theta$ for $r = 1, ..., t$

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

Acknowledgements

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, K.-R. Müller, B. Schölkopf, D. Weigel and M.K. Warmuth for great discussions and C.-S. Ong for proof reading the manuscript. G.R. would like to acknowledge a visiting appointment with *National ICT Australia* during the preparation of this work.

N.B. The appendix contains details regarding the data generation. Additional information about this work can be found at http://www.fml.tuebingen.mpg.de/raetsch/projects/mkl_splice.

References

1. Zien A, Rätsch G, Mika S, Schölkopf B, Lengauer T, Müller KR: **Engineering Support Vector Machine Kernels That Recognize Translation Initiation Sites.** *BioInformatics* 2000, **16(9):**799-807.
2. Jaakkola T, Diekhans M, Haussler D: **discriminative framework for detecting remote protein homologies.** *J Comput Biol* 2000, **7(1–2):**95-114.
3. Zhang X, Heller K, Hefter I, Leslie C, Chasin L: **Sequence information for the splicing of human pre-mRNA identified by support vector machine classification.** *Genome Res* 2003, **13(12):**637-50.
4. Lanckriet G, Bie TD, Cristianini N, Jordan M, Noble W: **A statistical framework for genomic data fusion.** *Bioinformatics* 2004, **20:**2626-2635.
5. Delcher A, Harmon D, Kasif S, White O, Salzberg S: **Improved microbial gene identification with GLIMMER.** *Nucleic Acids Research* 1999, **27(23):**4636-4641.
6. Kuang R, Ie E, Wang K, Wang K, Siddiqi M, Freund Y, Leslie C: **Profile-based string kernels for remote homology detection and motif extraction.** *Computational Systems Bioinformatics Conference 2004* 2004:146-154.
7. Bach FR, Lanckriet GRG, Jordan MI: **Multiple kernel learning, conic duality, and the SMO algorithm.** *Twenty-first international conference on Machine learning* 2004, **69:**. ACM Press
8. Sonnenburg S, Rätsch G, Schäfer C: **Learning Interpretable.** *RECOMB LNBI 3500, Springer-Verlag Berlin Heidelberg* 2005, **for Biological Sequence Classification.:**389-407.
9. Chapelle O, Vapnik V, Bousquet O, Mukherjee S: **Choosing Multiple Parameters for Support Vector Machines.** *Machine Learning* 2002, **46(1–3):**131-159.
10. Ong C, Smola A, Williamson R: **Learning the Kernel with Hyperkernels.** *Journal of Machine Learning Research* 2005, **6:**1043-1071.
11. Müller KR, Mika S, Rätsch G, Tsuda K, Schölkopf B: **An Introduction to Kernel-Based Learning Algorithms.** *IEEE Transactions on Neural Networks* 2001, **12(2):**181-201.
12. Sonnenburg S, Rätsch G, Schölkopf B: **Large Scale Genomic Sequence SVM Classifiers.** *Proceedings of the International Conference on Machine Learning, ICML* 2005.
13. Rätsch G, Sonnenburg S: *Accurate Splice Site Prediction for Caenorhabditis Elegans, MIT Press. MIT Press series on Computational Molecular Biology* 2003:277-298.
14. Leslie C, Eskin E, Noble W: **The Spectrum Kernel: A String Kernel for SVM protein Classification.** *Proceedings of the Pacific Symposium on Biocomputing, Kaua'i, Hawaii* 2002:564-575.
15. Vishwanathan S, Smola A: **Fast Kernels for String and Tree Matching.** *Kernel Methods in Computational Biology, MIT Press series on Computational Molecular Biology, MIT Press* 2003:113-130.
16. Fredkin E: **Trie Memory.** *Comm ACM* 1960, **3(9):**490-499.
17. Rätsch G, Sonnenburg S, Schölkopf B: **RASE: Recognition of Alternatively Spliced Exons in C. elegans.** *Bioinformatics* 2005, **21:**i369-i377.
18. Joachims T: **Making Large-Scale SVM Learning Practical.** In *Advances in Kernel Methods – Support Vector Learning* Edited by: *Schölkopf B, Burges C, Smola A. Cambridge, MA: MIT Press*; 1999:169-184.
19. Engel Y, Mannor S, Meir R: **Sparse Online Greedy Support Vector Regression.** *ECML* 2002:84-96.
20. Cristianini N, Shawe-Taylor J: *An Introduction to Support Vector Machines Cambridge, UK: Cambridge University Press*; 2000.
21. Schölkopf B, Smola AJ: *Learning with Kernels Cambridge, MA: MIT Press*; 2002.
22. Cortes C, Vapnik V: **Support Vector Networks.** *Machine Learning* 1995, **20:**273-297.
23. Vapnik V: *The nature of statistical learning theory New York: Springer Verlag*; 1995.
24. Rätsch G: **Robust Boosting via Convex Optimization.** In *PhD thesis University of Potsdam, Computer Science Dept., August-Bebel-Str. 89, 14482 Potsdam, Germany*; 2001.
25. Hettich R, Kortanek K: **Semi-Infinite Programming: Theory, Methods and Applications.** *SIAM Review* 1993, **3:**380-429.
26. Bennett K, Demiriz A, Shawe-Taylor J: **A Column Generation Algorithm for Boosting.** In *Proceedings, 17th ICML* Edited by: *Langley P. San Francisco: Morgan Kaufmann*; 2000:65-72.
27. Rätsch G, Demiriz A, Bennett K: **Sparse Regression Ensembles in Infinite and Finite Hypothesis Spaces.** *Machine Learning* 2002, **48(1–3):**193-221. Special Issue on New Methods for Model Selection and Model Combination. Also NeuroCOLT2 Technical Report NC-TR-2000-085.
28. Meir R, Rätsch G: **An Introduction to Boosting and Leveraging.** In *Proc. of the first Machine Learning Summer School in Canberra, LNCS* Edited by: *Mendelson S, Smola A. Springer*; 2003 in press.
29. Rätsch G, Warmuth MK: **Efficient Margin Maximization with Boosting.** *Journal of Machine Learning Research* 2005, **6(Dec):**2131-2152.
30. Vapnik V: *Estimation of Dependences Based on Empirical Data Berlin: Springer-Verlag*; 1982.
31. Platt J: **Fast Training of Support Vector Machines using Sequential Minimal Optimization.** In *Advances in Kernel Methods – Support Vector Learning* Edited by: *Schölkopf B, Burges C, Smola A. Cambridge, MA: MIT Press*; 1999:185-208.
32. Mood A, Graybill F, Boes D: *Introduction to the Theory of Statistics* third edition. *McGraw-Hill*; 1974.
33. Lehmann E: *Testing Statistical Hypotheses. Springer, New York, second edition edition* 1997.
34. Harris TW, *et al.*: **WormBase: a multi-species resource for nematode biology and genomics.** *Nucl Acids Res* 2004, **32:**. Database issue:D411-7
35. Boguski M, Tolstoshev TLC: **dbEST-Database for "Expressed Sequence Tags".** *Nat Genet* 1993, **4(4):**332-3.
36. Wheeler DL, *et al.*: **Database Resources of the National Center for Biotechnology.** *Nucl Acids Res* 2003, **31:**38-33.
37. Kent W: **BLAT-the BLAST-like alignment tool.** *Genome Res* 2002, **12(4):**656-64.
38. Bennett KP, Momma M, Embrechts MJ: **MARK: a boosting algorithm for heterogeneous kernel models.** *KDD* 2002:24-31.
39. Sonnenburg S, Rätsch G, Schäfer S, Schölkopf B: **Large Scale Multiple Kernel Learning.** *Journal of Machine Learning Research* 2006. Accepted