Learning interpretable SVMs for biological sequence classification

Sören Sonnenburg[†], Gunnar Rätsch[†,♮], Christin Schäfer[†]

Fraunhofer Institut Rechnerarchitektur und Softwaretechnik

FIRST

MAX-PLANCK-GESELLSCHAFT

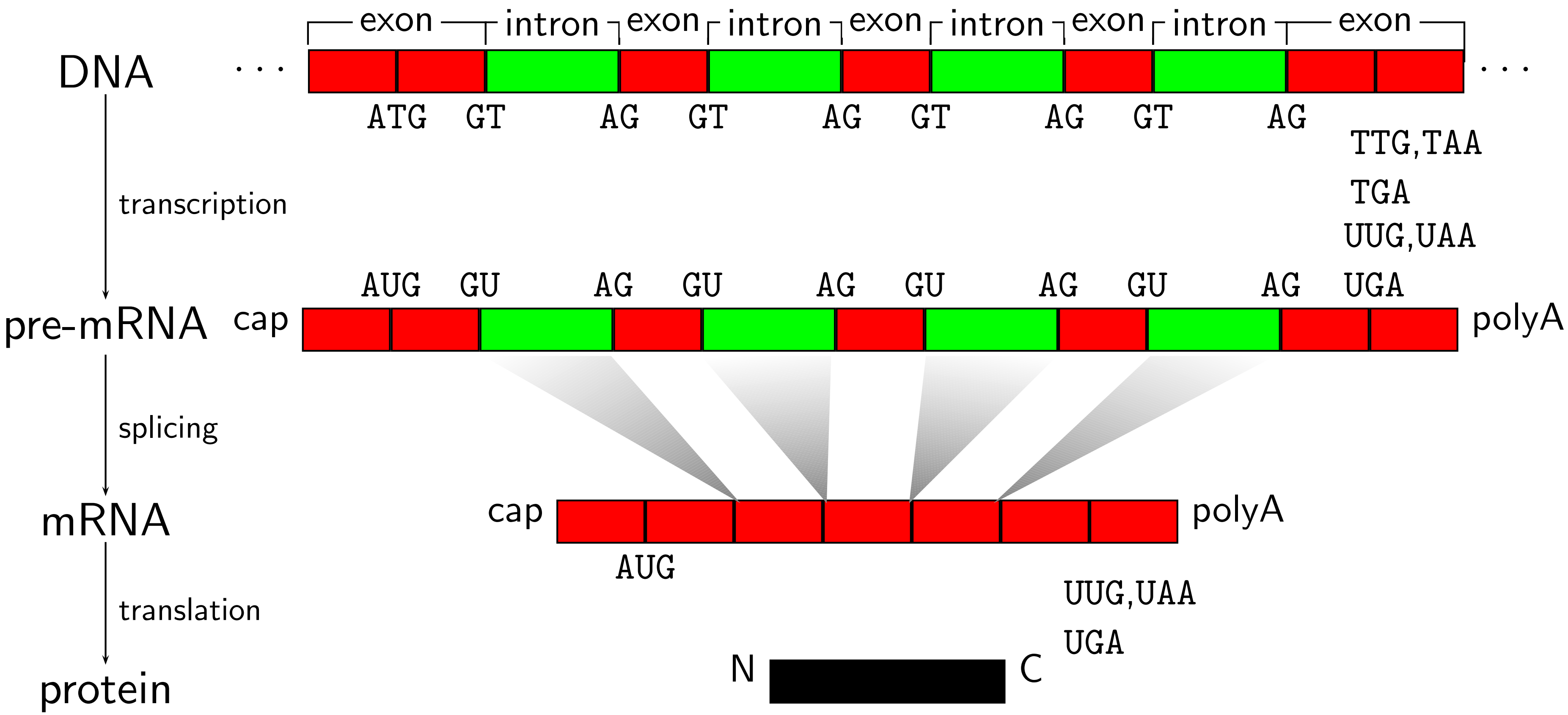
† Fraunhofer FIRST.IDA, Kekuléstr. 7, 12489 Berlin, Germany
♮ Friedrich Miescher Laboratory of the Max Planck Society,
Spemannstr. 39, 72076 Tübingen, Germany
{Soeren.Sonnenburg, Christin.Schaefer}@first.fraunhofer.de,
Gunnar.Raetsch@tuebingen.mpg.de

Roadmap:

- The Motivating Application
- Multiple Kernel Learning (MKL)
- Derivation of the MKL Optimization Problem
- Algorithms
- Significance Analysis
- Results
- Outlook and Conclusion

The Motivating Application

Splice sites are locations on DNA at boundaries of
- exons (which code for proteins)
- introns (which do not)



Biology: Detection of Splice Sites

```
                       Intron Exon
AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT
CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC
TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC
TTGTTTTAATATTCAATTTTTTACAGTAAGTTGCCAATTCAATGTTCCAC
TACCTAATTATGAAATTAAAATTCAGTGTGCTGATGGAAACGGAGAAGTC
```

- aligned sequences of fixed length (AG always at position)
- Task: distinguish splice sites from fake - splice sites

$\Rightarrow$ 2-class classifcation problem

Approach: String Kernel + SVM

- use SVM classifier $$f(\boldsymbol{x}) = \mathrm{sign}\left(\sum_{i=1}^{N} y_i \alpha_i \mathrm{k}(\boldsymbol{x}, \boldsymbol{x}_i) + b\right)$$

- find parameters $\boldsymbol{\alpha}$ by solving quadratic optimization problem:

$$\begin{array}{ll} \max\limits_{\boldsymbol{\alpha}} & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \mathrm{k}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \text{subject to} & \alpha_i \in [0, C], i = 1, \ldots, N, \sum_{i=1}^{N} \alpha_i y_i = 0. \end{array}$$

- **Solution has no local minima**

Key ingredient Kernel here "Weighted Degree Kernel"

$$\mathrm{k}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{k=1}^{d} \beta_k \sum_{l=1}^{L-k} \mathrm{I}(\boldsymbol{u}_{k,l}(\boldsymbol{x}) = \boldsymbol{u}_{k,l}(\boldsymbol{x}')),$$

Weighted Degree Kernel

$$\color{red}{\mathrm{k}(\boldsymbol{x},\boldsymbol{x}')} = \sum_{k=1}^{d} \beta_k \sum_{l=1}^{L-k} \mathrm{I}(\boldsymbol{u}_{k,l}(\boldsymbol{x}) = \boldsymbol{u}_{k,l}(\boldsymbol{x}'))$$

- L length of the sequence $\boldsymbol{x}$
- d maximal "match length" taken into account
- $\boldsymbol{u}_{k,l}(\boldsymbol{x})$ subsequence of length k at position l of sequence $\boldsymbol{x}$

Example degree $d = 3$:

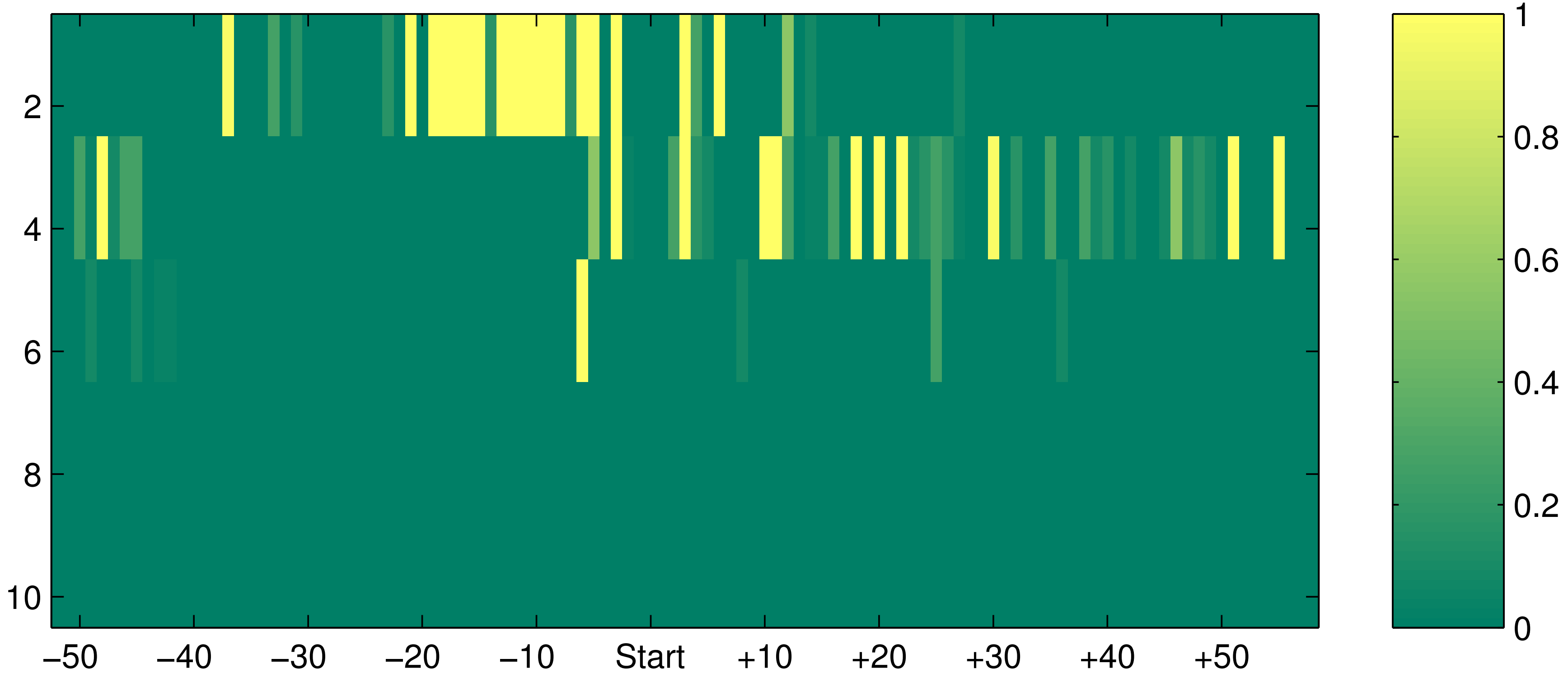
$\boldsymbol{x}$ AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
#1-mers . | . | . | | | . | . . | | . | . | . . | | | . | | . . . | . . . . | . . . | | | . . . . . . | . .
#2-mers . . . . . | | . . . . . | . . . . | | . . | . . . . | | . . . .
#3-mers . . . . . | . . . . | . . . | . . . . | . . . .
$\boldsymbol{x}'$ TACCTAATTATGAAATTAAATTTCAGTGTGCTGATGGAAACGGAGAAGTC

$$\color{red}{\mathrm{k}(\boldsymbol{x},\boldsymbol{x}')} = \beta_1 \cdot 21 + \beta_2 \cdot 8 + \beta_3 \cdot 4$$

Position dependant Weighting β

Even more weights:

$$\mathrm{k}(\boldsymbol{x},\boldsymbol{x}') = \sum_{k=1}^{d} \sum_{l=1}^{L-k} \beta_{kl} \mathrm{I}(\boldsymbol{u}_{k,l}(\boldsymbol{x}) = \boldsymbol{u}_{k,l}(\boldsymbol{x}'))$$



Success

Choosing a particular weighting

$$\beta_k = \frac{2(d-k+1)}{(d(d+1))}$$

seems to solve the task: on 500,000 training examples test AUC $99,80\%$ (test error 0.78%)

Open questions

- Why does that weighting make sense ?
- Is there a better weighting ?
  - in terms of sense
  - classification performance
- Can we learn that weighting ?
- What does that have to do with Multiple Kernel Learning ?

Reformulation: Multiple Kernel Learning

The Weighted Degree kernel is a linear combination of kernels !

$$\mathrm{k}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{k=1}^{d} \beta_k \sum_{l=1}^{L-k} \mathrm{I}(\boldsymbol{u}_{k,l}(\boldsymbol{x}) = \boldsymbol{u}_{k,l}(\boldsymbol{x}')) = \sum_{k=1}^{d} \beta_k \mathrm{k}_k(\boldsymbol{x}, \boldsymbol{x}')$$

with

$$\mathrm{k}_k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{l=1}^{L-k} \mathrm{I}(\boldsymbol{u}_{k,l}(\boldsymbol{x}) = \boldsymbol{u}_{k,l}(\boldsymbol{x}')).$$

(Can also be applied to other String Kernels)

$\Rightarrow$ need to solve the so called **Multiple Kernel Learning Problem**, i.e. determine $(\boldsymbol{\beta}, \boldsymbol{\alpha}, b)$ simultaneously.

FIRST

Motivation

- Kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \Phi(\boldsymbol{x}) \cdot \Phi(\boldsymbol{x}')$ used in standard SVM Classifier

$$f(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i=1}^{\ell} y_i \alpha_i \mathrm{k}(\boldsymbol{x}, \boldsymbol{x}_i) + b\right)$$

- Now: linear combination of kernels (again a kernel)

$$\mathrm{k}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{j=1}^{M} \beta_j \, \mathrm{k}_j(\boldsymbol{x}, \boldsymbol{x}'), \; \beta_j \geq 0$$

- **useful:** Polynomial kernels of different degree, kernels on different domain

- **but:** How to learn and constrain weights β_j ?

Constraining the weights

$\mathbf{L_2}$-vs. $\mathbf{L_1}$-Norm

- in max problem weights certain β_j would grow infinitely (min shrink to zero) $\Rightarrow$ **constraining β_j necessary**

- Dense $\|\boldsymbol{\beta}\|_2 = 1$ (Lin and Zhang 2004)

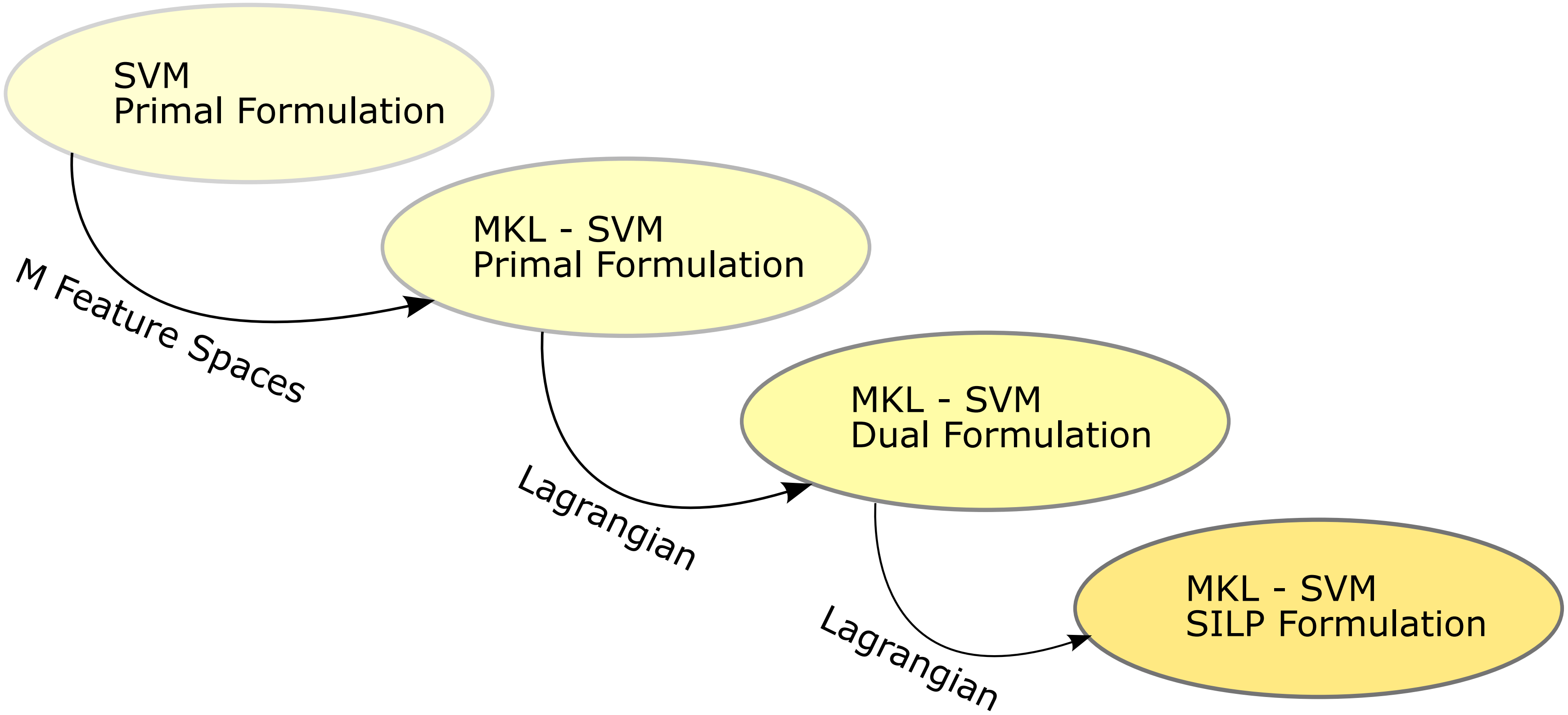
- vs. Sparse $\|\boldsymbol{\beta}\|_1 = 1$ (Bach, Lanckriet and Jordan 2004)
  - convex combination of kernels
  - sparse solution in terms of kernels
  - allows for interpretation of result

constraints on β_j:

$$\sum_{j=1}^{N} \beta_j = 1, \ \beta_j \geq 0$$

Rewriting the MKL formulation

From the Standard SVM Primal to the Semi-Infinite Linear Program:



Standard SVM Optimization Problem

This we all know

SVM Primal formulation:

$$
\begin{array}{rl}
\min & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{N}\xi_n \\
\text{w.r.t.} & \boldsymbol{w}\in\mathbb{R}^k, \boldsymbol{\xi}\in\mathbb{R}_+^N, b\in\mathbb{R} \\
\text{s.t.} & y_i\left(\boldsymbol{w}^\top\Phi(\boldsymbol{x}_i)+b\right)\geq 1-\xi_i, \forall i=1,\ldots,N
\end{array}
$$

MKL Optimization Problem I

MKL Primal formulation:

$$\min \quad \frac{1}{2}\left(\sum_{j=1}^{M}\beta_j \|\boldsymbol{w}_j\|_2\right)^2 + C\sum_{i=1}^{N}\xi_n$$

$$\text{w.r.t.} \quad \boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_M), \boldsymbol{w}_j \in \mathbb{R}^{k_j}, \boldsymbol{\xi} \in \mathbb{R}_+^N, \boldsymbol{\beta} \in \mathbb{R}_+^M, b \in \mathbb{R}$$

$$\text{s.t.} \quad y_i\left(\sum_{j=1}^{M}\beta_j \boldsymbol{w}_j^\top \Phi_j(\boldsymbol{x}_i) + b\right) \geq 1 - \xi_i, \forall i = 1, \ldots, N$$

$$\sum_{j=1}^{M}\beta_j = 1$$

Properties: equivalent to SVM for $M = 1$; solution sparse in “blocks”; each block j corresponds to one kernel

MKL Optimization Problem II

Dual Formulation (Bach, Lanckriet, Jordan 2004):

$$
\begin{aligned}
\min \quad & \frac{1}{2}\gamma^2 - \sum_{i=1}^{N} \alpha_i \\
\text{w.r.t.} \quad & \gamma \in \mathbb{R}, \boldsymbol{\alpha} \in \mathbb{R}^N \\
\text{s.t.} \quad & 0 \leq \boldsymbol{\alpha} \leq C, \sum_{i=1}^{N} \alpha_i y_i = 0 \\
& \underbrace{\sum_{r=1}^{N} \sum_{s=1}^{N} \alpha_r \alpha_s y_r y_s K_j(\boldsymbol{x}_r, \boldsymbol{x}_s)}_{=: S_j(\boldsymbol{\alpha})} - \gamma^2 \leq 0, \ \forall j = 1, \ldots, M
\end{aligned}
$$

"partial Lagrangian:"

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

MKL Optimization Problem II

Reformulation as Semi-Infinite Linear Program:

$$\max_{\boldsymbol{\beta}} \min_{\boldsymbol{\alpha}} \sum_{j=1}^{M} \beta_j \left(\frac{1}{2} S_j(\boldsymbol{\alpha}) - \sum_{i=1}^{N} \alpha_i \right)$$

$$\text{s.t. } 0 \leq \boldsymbol{\alpha} \leq C, \sum_{i=1}^{N} \alpha_i y_i = 0, \sum_{j=1}^{M} \beta_j = 1$$

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

$\Rightarrow$ **Linear, but infinitely many constraints**

The Semi-Infinite Linear Program

Properties:

- optimize a convex combination
- infinitely many constraints
- quite easy to identify violated constraints

Solving the SILP:

- Use Boosting like techniques: Arc-GV or AdaBoost*
- Column Generation
- SMO like algorithm

Column Generation I

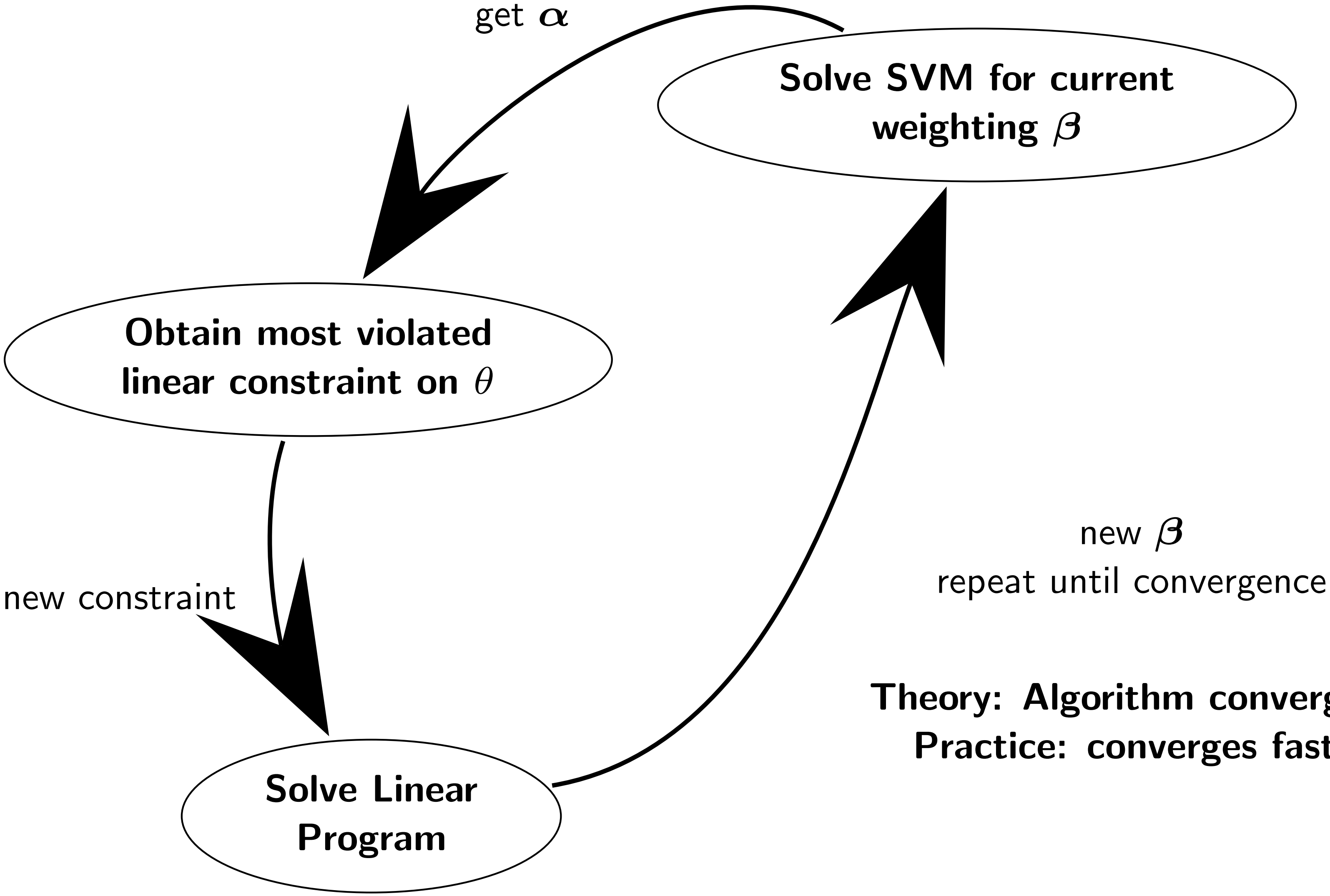
$$
\begin{aligned}
\max \quad & \theta \\
\text{w.r.t.} \quad & \theta \in \mathbb{R}, \boldsymbol{\beta} \in \mathbb{R}^M_+ \text{ with } \sum_{j=1}^{M} \beta_j = 1 \\
\text{s.t.} \quad & \sum_{j=1}^{M} \beta_j \left(\frac{1}{2} S_j(\boldsymbol{\alpha}) - \sum_{i=1}^{N} \alpha_i \right) \geq \theta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

- solved by taking set of most violated constraints into account
- most violated constraints given by SVM solution for fixed $\boldsymbol{\beta}$

$$
\sum_{j=1}^{M} \beta_j \left(\frac{1}{2} S_j(\boldsymbol{\alpha}) - \sum_{i=1}^{N} \alpha_i \right) = \frac{1}{2} \sum_{r=1}^{N} \sum_{s=1}^{N} \alpha_r \alpha_s y_r y_s \sum_{j=1}^{M} \beta_j k_j(\boldsymbol{x}_r, \boldsymbol{x}_s) - \sum_{i=1}^{N} \alpha_i,
$$

- iteratively find most violated constraints, solve linear program with current constraints, . . . , till convergence to the global optimum

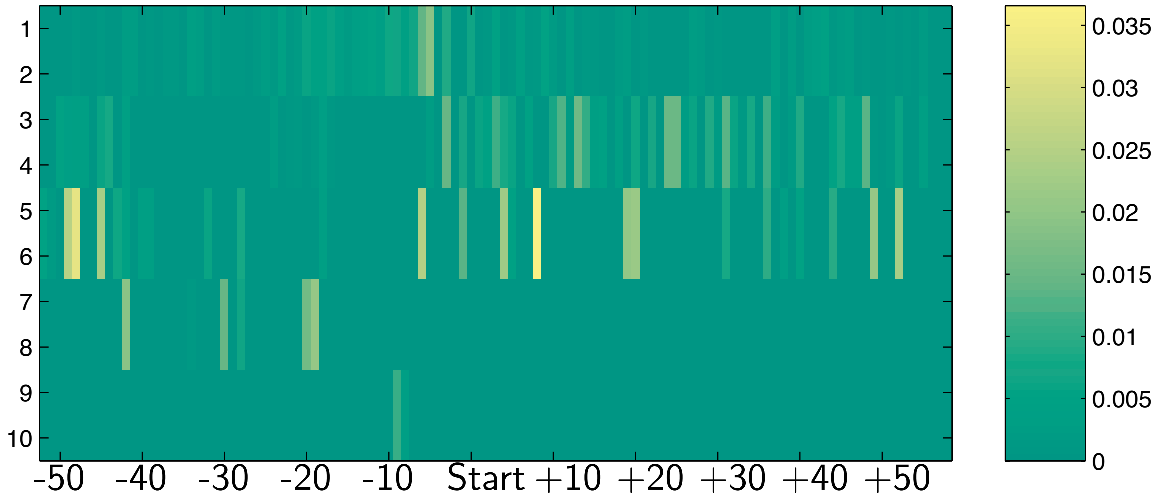
Column Generation II



Theory: Algorithm converges!
Practice: converges fast!

Results on Splice Sites

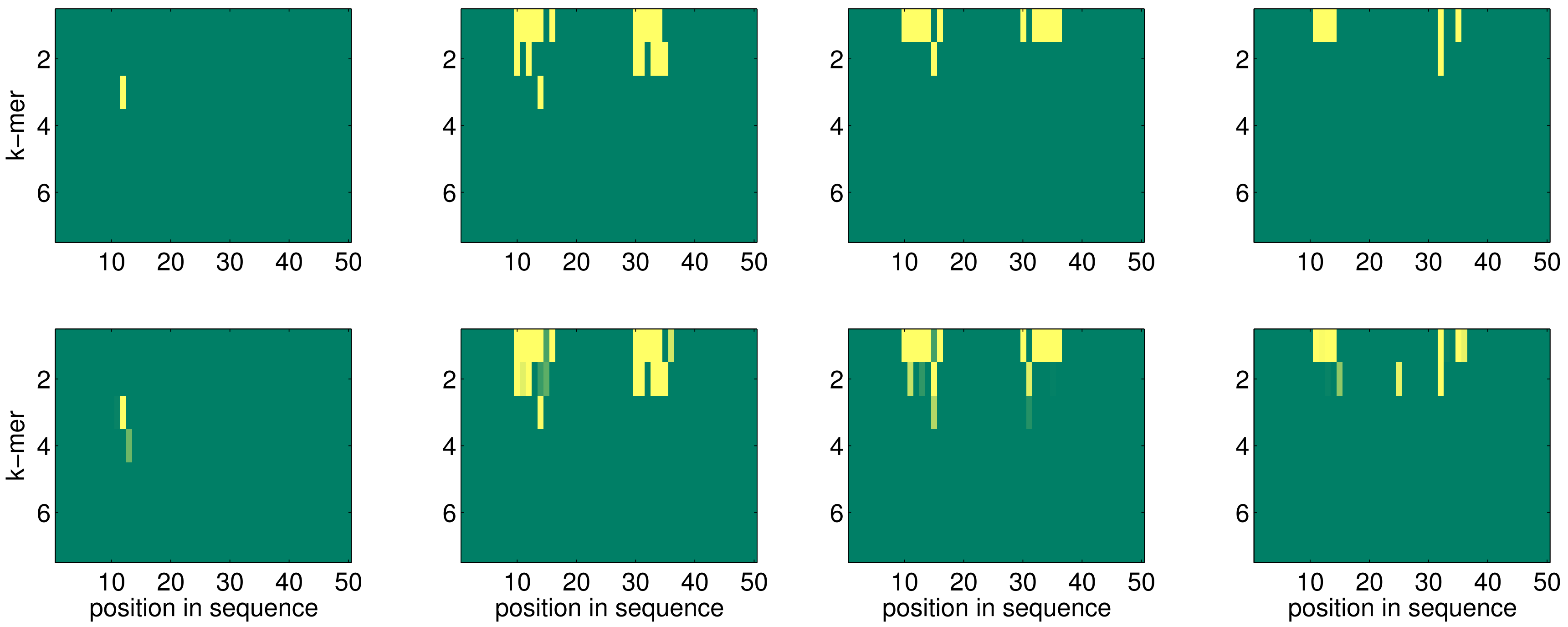
What characterizes the positions:



- Can we use that weighting to interpret the SVM solution? $\Rightarrow$ Not yet!
  - Stability of the weighting β?
  - Which weights are significant?

Use a statistical test to investigate significance of weights

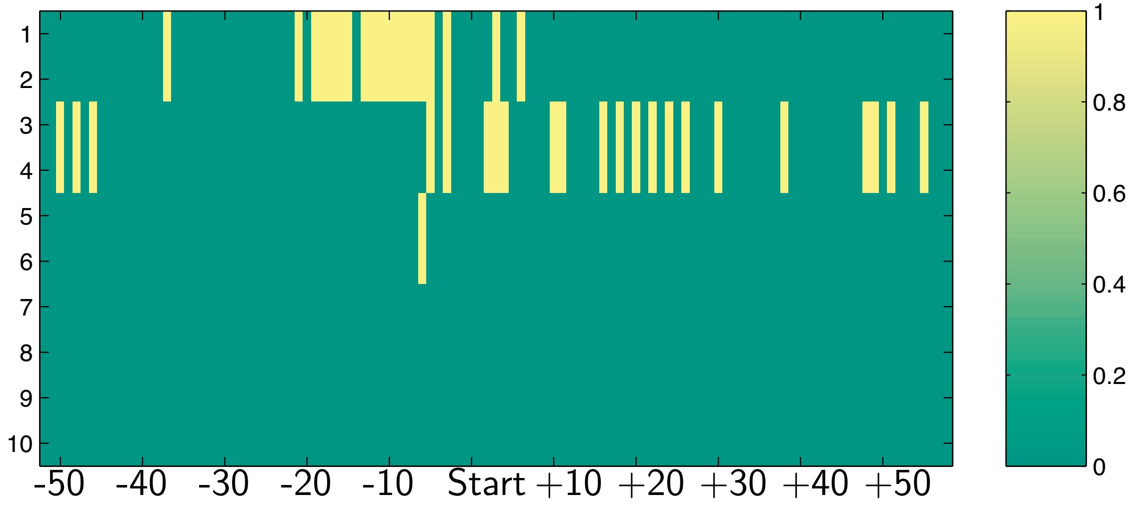
Toy Dataset



- DNA sequences of length 50 with hidden motifs at 10-16 and 30-36
- 8×50 string kernels with max. word- length 8
- compute significance level by bootstrapping
- columns $\equiv$ noise level
- subplot columns weights used at certain position, rows oligomer length

Results on Splice Sites

What characterizes the positions:



- Can we use that weighting to interpret the SVM solution? $\Rightarrow$ NOW !
  - Stability of the weighting β?
  - Which weights are significant?

Use a statistical test to investigate significance of weights

Conclusion

FIRST

Conclusion:

- MKL learns convex combination of kernels
  $\Rightarrow$ allows for interpreting SVM result
  $\Rightarrow$ matches prior knowledge about splice sites
- simple iterative algorithm
- suitable for large scale problems ($> 100,000$ examples)

Discussion:

- Can we improve classification using MKL ?
  $\Rightarrow$ $\|.\|_1 = 1$ good choice ?
- Does MKL overfit and if so when ?
  $\Rightarrow$ How to regularize complexity ?
- Can we do model selection via MKL ?