Genomic Signal Detection

. . . *using Support Vector Machines*

Sören Sonnenburg
TU Berlin

joint work with
Alexander Zien, Jonas Behr, Gabriele Schweikert,
Konrad Rieck, Petra Philips, Gunnar Rätsch, Vojtech Franc

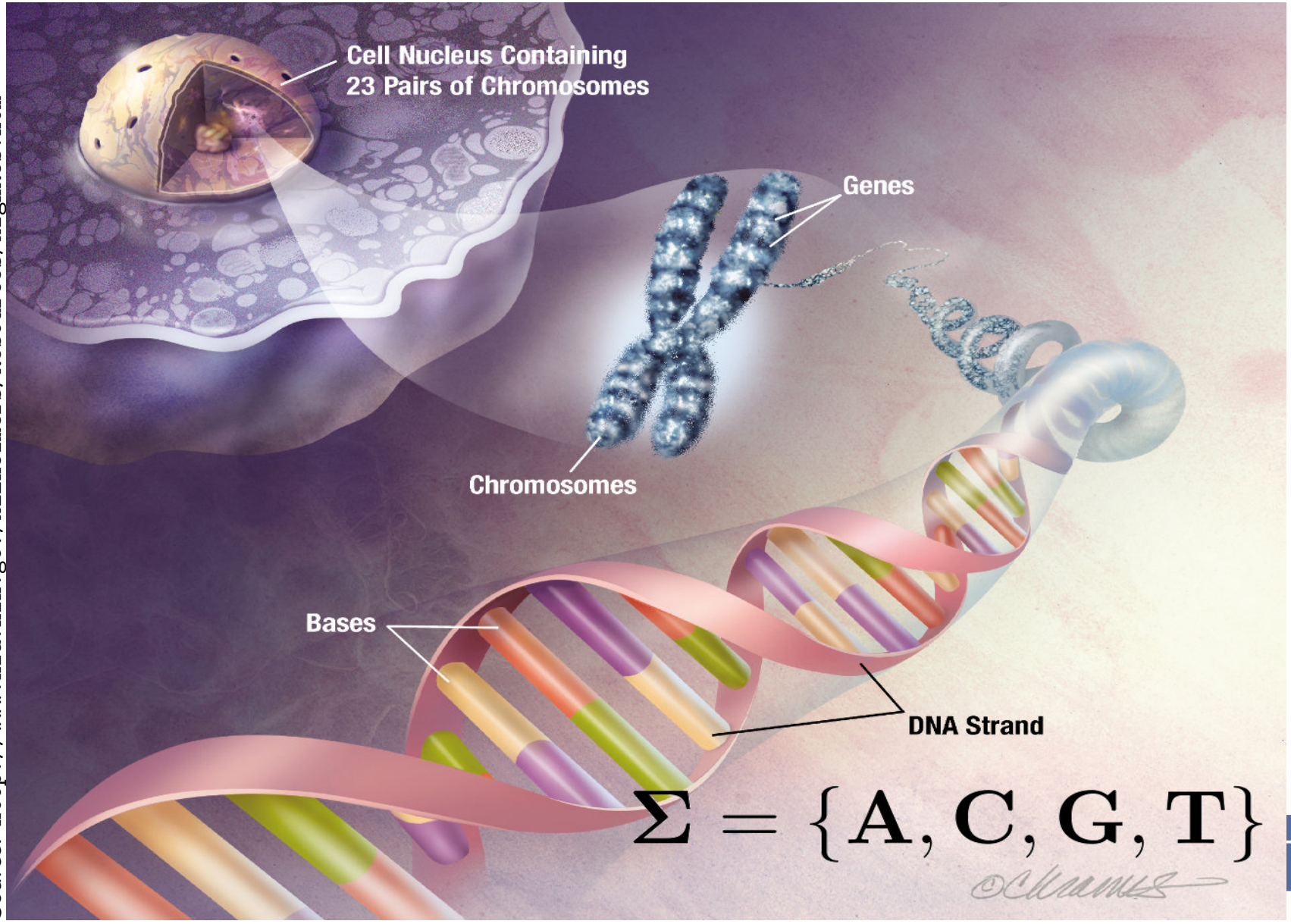
Outline

1. Introduction
2. Sequence Classification
3. Large Scale Learning
4. Explanation and Visualization
5. Discussion

Outline

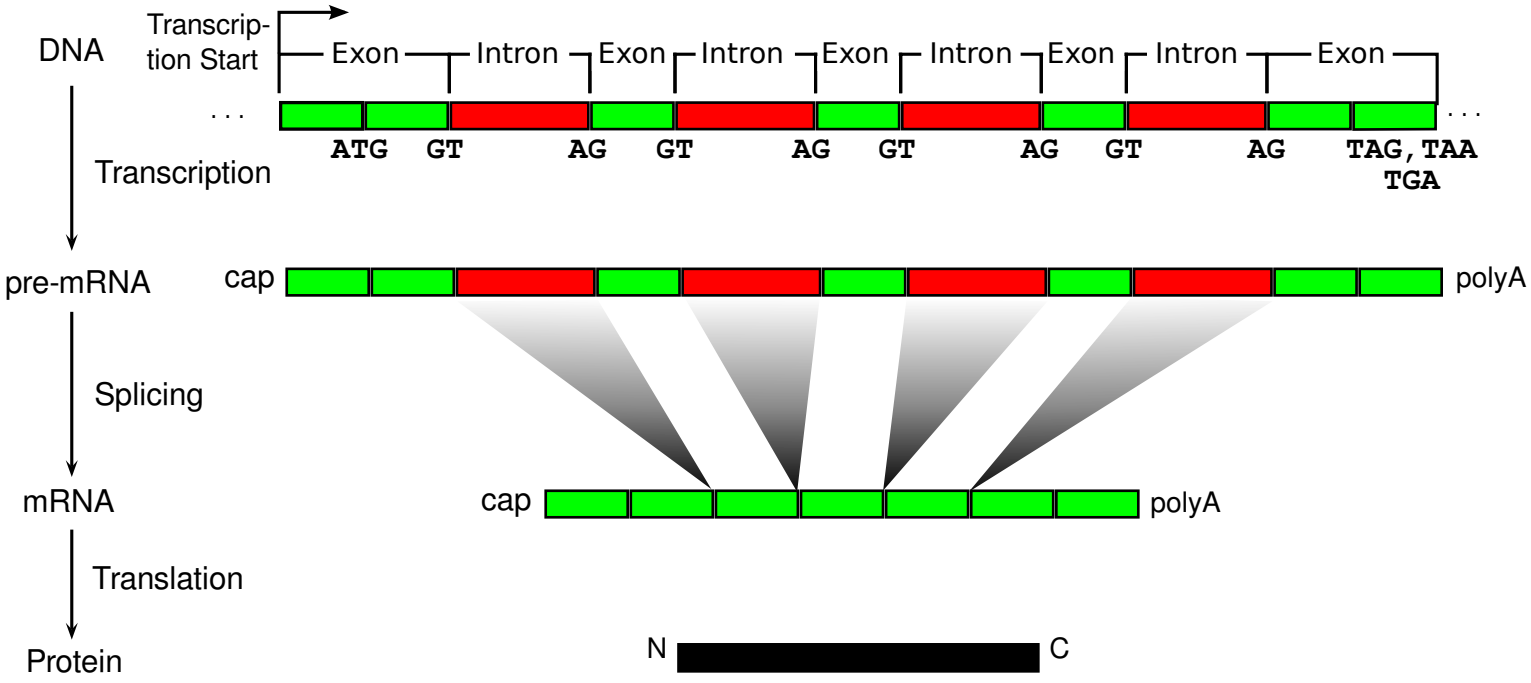
1. Introduction
   - Genomic Signals
2. Sequence Classification
   - Support Vector Machines
   - String Kernels
   - Example
3. Large Scale Learning
   - Application TSS recognition
4. Explanation and Visualization
   - Introduction
   - Definition
   - Applications
5. Discussion

Genome



Source: http://www.nia.nih.gov/Alzheimers/Resources/HighRes.htm

Genomic Signals

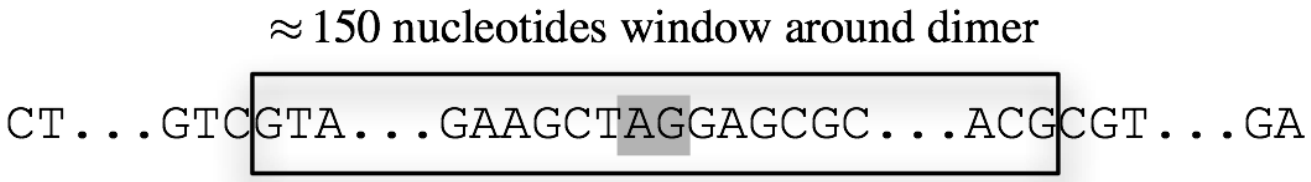


Genomic Signal Detection

- Start/Stop of Genes
- Donor Splice Site (Exon-Intron-Boundary)
- Acceptor Splice Site (Intron-Exon-Boundary)

Recognizing Genomic Signals

Discriminate true signal positions against all other positions



- True sites: fixed window around a true site
- Decoy sites: all other consensus sites

```
AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT
CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC
TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC
TTGTTTTAATATTCAATTTTTTACAGTAAGTTGCCAATTCAATGTTCCAC
TACCTAATTATGAAATTAAAATTCAGTGTGCTGATGGAAACGGAGAAGTC
```

Examples: Transcription start site finding, splice site prediction, alternative splicing prediction, trans-splicing, polyA signal

Types of Signal Detection Problems I

Vague categorization

(based on positional variability of motifs)

Position Independent

→ Motifs may occur anywhere,

e.g. tissue classification using promotor region

Types of Signal Detection Problems II

Vague categorization

(based on positional variability of motifs)

Position Dependent

→ Motifs very stiff, almost always at same position,

e.g. Splice Site Classification

Types of Signal Detection Problems III

Vague categorization

(based on positional variability of motifs)

Mixture Position Dependent/Independent

→ variable but still positional information

```
AAACAAATAAGTAACTAATCTTTTAAAGAGAACGTTTCAACCATTTTGAG
AAGATTAAAAAAAAACAAATTTCATTAAATACAGATATAATAATCTAATT
CACTCCCCAAATCAACGATATTTAAATTTCACTAACACATCCGTCTGTGC
```

e.g. Promoter Classification

Outline

1. Introduction
   - Genomic Signals
2. Sequence Classification
   - Support Vector Machines
   - String Kernels
   - Example
3. Large Scale Learning
   - Application TSS recognition
4. Explanation and Visualization
   - Introduction
   - Definition
   - Applications
5. Discussion

Classification - Learning based on examples

Given:

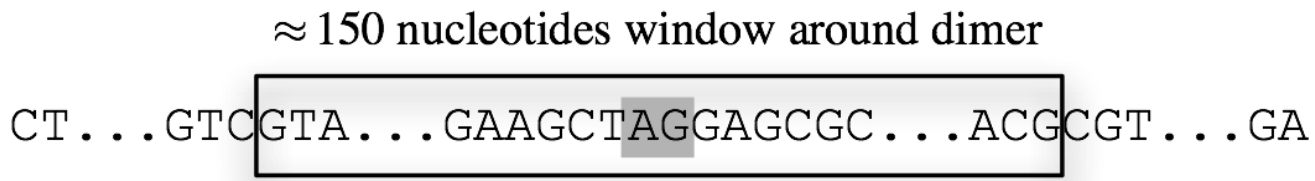
Training examples $(\mathbf{x}_i, y_i)_{i=1}^N \in (\{A, C, G, T\}^L, \{-1, +1\})^N$

AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG	$+1$
AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT	-1
CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC	$+1$
TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC	-1
TTGTTTTAATATTCAATTTTTTACAGTAAGTTGCCAATTCAATGTTCCAC	-1
TACCTAATTATGAAATTAAAATTCAGTGTGCTGATGGAAACGGAGAAGTC	-1

($\approx$ 1 billion neg. sequences; $<$ 200.000 positive sequences)

Wanted:

Function (Classifier) $f(\mathbf{x}) : \{A, C, G, T\}^L \mapsto \{-1, +1\}$



Aim: Accurate signal prediction for the whole genome

Classification - Learning based on examples

Given:

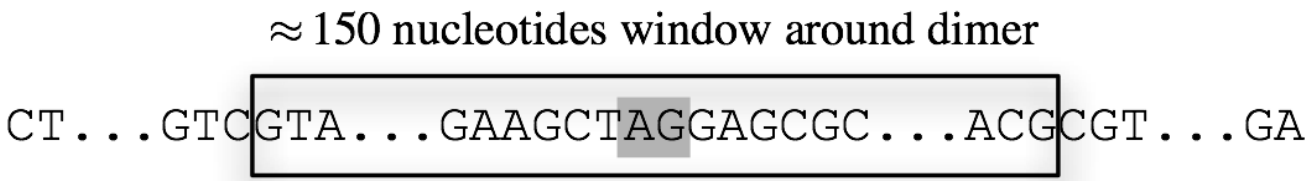
Training examples $(\mathbf{x}_i, y_i)_{i=1}^N \in (\{A, C, G, T\}^L, \{-1, +1\})^N$

AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG	$+1$
AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT	-1
CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC	$+1$
TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC	-1
TTGTTTTAATATTCAATTTTTTACAGTAAGTTGCCAATTCAATGTTCCAC	-1
TACCTAATTATGAAATTAAAATTCAGTGTGCTGATGGAAACGGAGAAGTC	-1

($\approx$ 1 billion neg. sequences; $<$ 200.000 positive sequences)

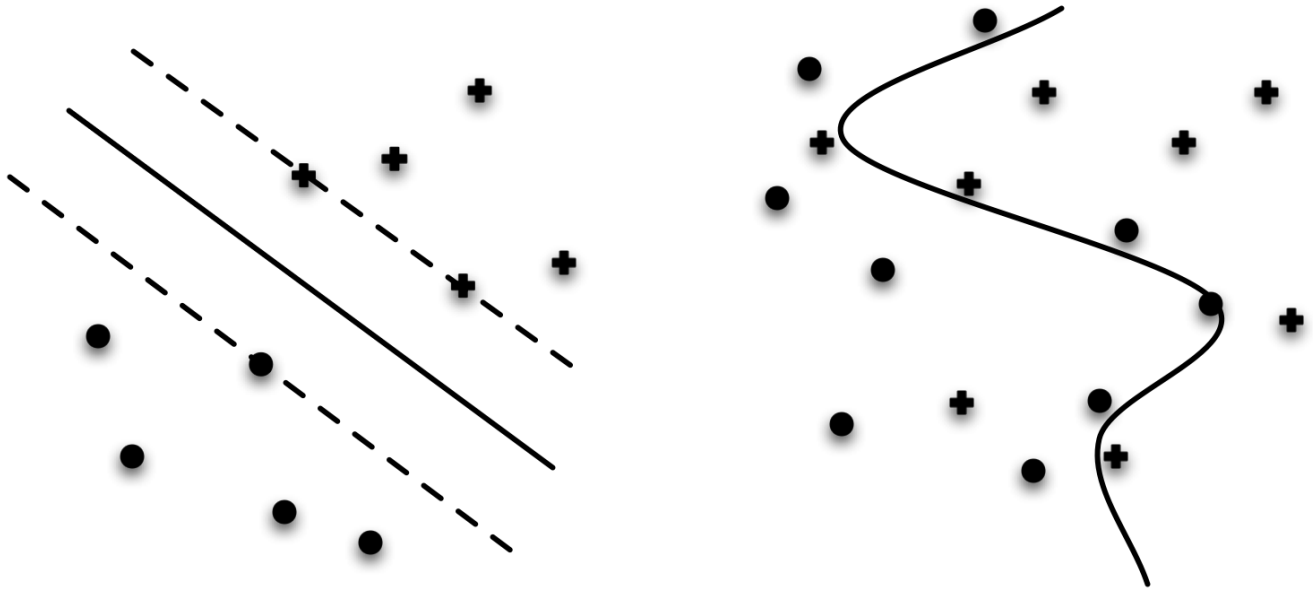
Wanted:

Function (Classifier) $f(\mathbf{x}) : \{A, C, G, T\}^L \mapsto \{-1, +1\}$



Aim: Accurate signal prediction for the whole genome

Support Vector Machines (SVMs)



- **Support Vector Machines** learn weights $\alpha \in \mathbb{R}^N$ over training examples in kernel feature space $\Phi : \mathbf{x} \mapsto \mathbb{R}^D$,

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

with kernel $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$

The Spectrum Kernel (Leslie et al. 2002)

Support Vector Machine

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

Spectrum Kernel (with mismatches, gaps)

$$K(\mathbf{x}, \mathbf{x}') = \Phi_{sp}(\mathbf{x}) \cdot \Phi_{sp}(\mathbf{x}')$$

Example $k = 3$:

x AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG

x' TACCTAATTATGAAATTAAATTTCAGTGTGCTGATGGAAACGGAGAAGTC

3-mer	AAA	AAC	...	CCA	CCC	...	TTT
# in $\mathbf{x}$	2	4	...	1	0	...	3
# in $\mathbf{x}'$	3	1	...	0	0	...	1

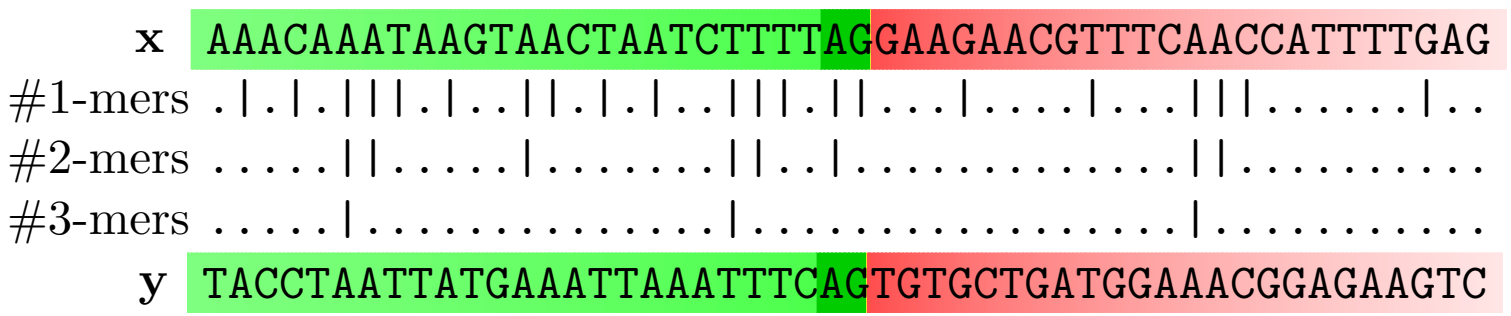
$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = 2 \cdot 3 + 4 \cdot 1 + \ldots 1 \cdot 0 + 0 \cdot 0 \ldots 3 \cdot 1$$

The Weighted Degree Kernel (Sonnenburg et al. 2005)

Support Vector Machine

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

$$\mathsf{k}(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{K} \beta_k \sum_{i=1}^{L-k+1} \mathbb{I}\left\{\mathbf{x}[i]^k = \mathbf{x}'[i]^k\right\}.$$

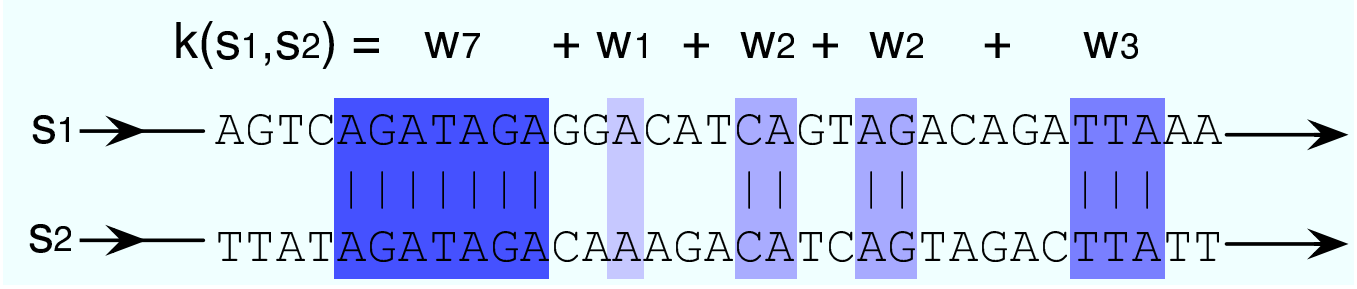


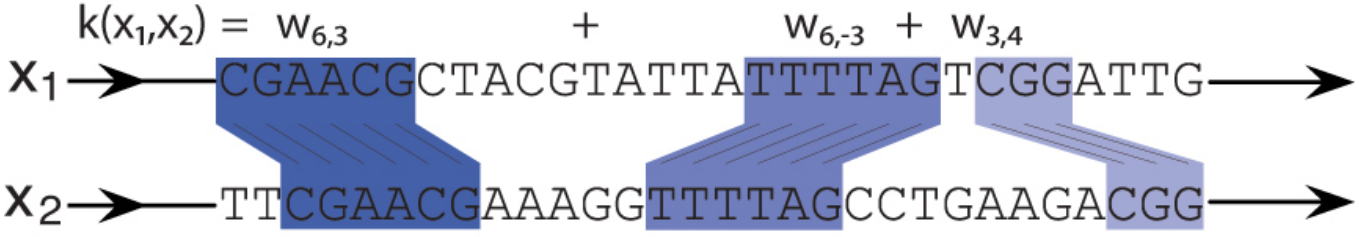
Example: $K = 3$: $\mathsf{k}(\mathbf{x}, \mathbf{x}') = \beta_1 \cdot 21 + \beta_2 \cdot 8 + \beta_3 \cdot 3$

The Weighted Degree Kernel with *shifts* (Raetsch, Sonnenburg et al. 2005)

Support Vector Machine

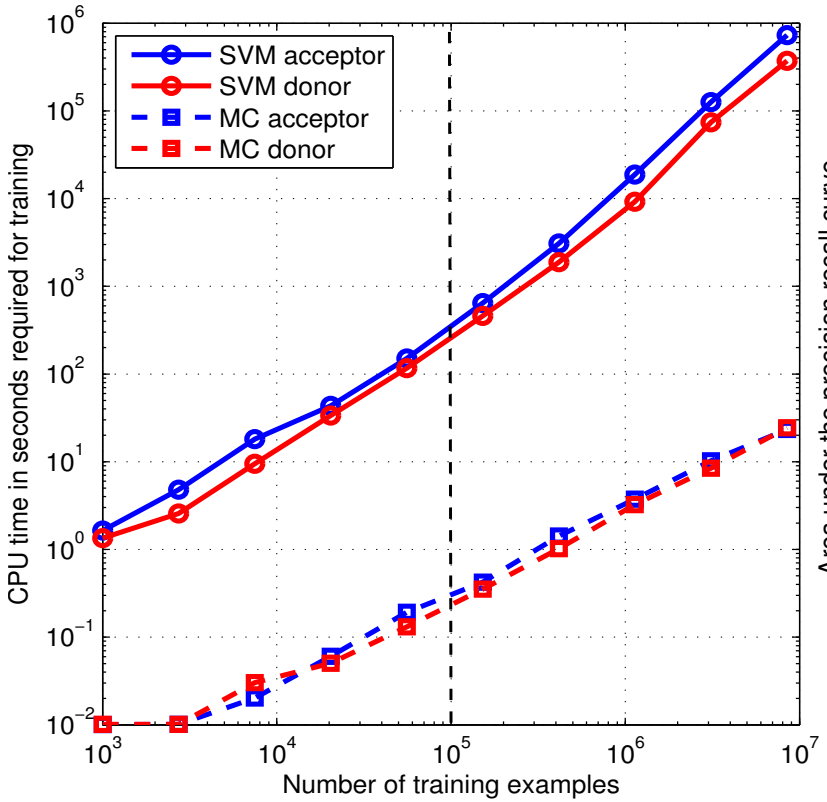
$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N} y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b\right),$$

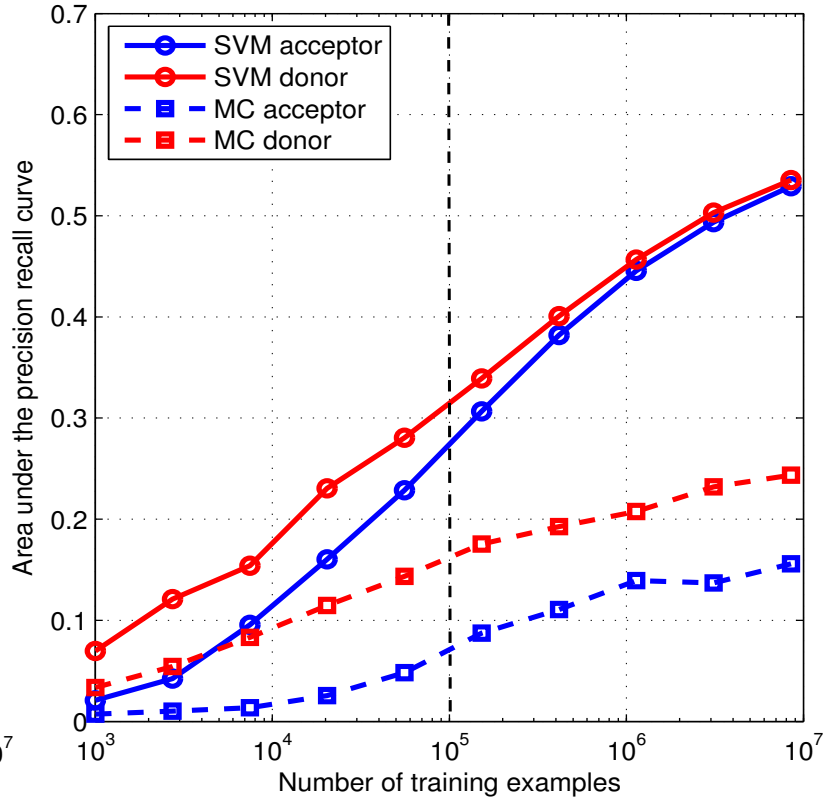




First Application Splice Sites

- Human splice sites: $5 \cdot 10^7$ strings of length ≈ 141
- Note: Raw data is already 7GB in size





Aim: Train string-kernel SVM on all available data

Outline

1. Introduction
   - Genomic Signals
2. Sequence Classification
   - Support Vector Machines
   - String Kernels
   - Example
3. Large Scale Learning
   - Application TSS recognition
4. Explanation and Visualization
   - Introduction
   - Definition
   - Applications
5. Discussion

The Curse of Support Vectors

SVMs deliver state-of-the-art results ...BUT...

To compute output on all M examples $\mathbf{x}_1, \ldots, \mathbf{x}_M$:

$$\forall j = 1, \ldots, M: \quad \sum_{i=1}^{N_s} \alpha_i y_i \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) + b$$

Computational effort:

- All $\mathcal{O}(N_s M T)$, (T time to compute the kernel)
- Effort Scales linearly with $N_s = \mathcal{O}(N) := \#\text{SVs}$

$\Rightarrow$ **Used in training and testing - worth tuning.**

$\Rightarrow$ **How to further speed up if $T = dim(\mathcal{X})$ already linear?**

Accelerating String-Kernel-SVMs

1. ~~Linear run-time of the kernel~~
2. Accelerating linear combinations of kernels

Idea of the `Linadd` Algorithm (Sonnenburg et al., 2005):

Store $\mathbf{w}$ and compute $\mathbf{w} \cdot \Phi(\mathbf{x})$ *efficiently*

$$f(\mathbf{x}_j) = \sum_{i=1}^{N_s} \alpha_i y_i \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \underbrace{\sum_{i=1}^{N_s} \alpha_i y_i \Phi(\mathbf{x}_i)}_{\mathbf{w}} \cdot \Phi(\mathbf{x}_j) = \mathbf{w} \cdot \Phi(\mathbf{x}_j)$$

Possible for low-dimensional or sparse

Effort: $\mathcal{O}(ML) \Rightarrow$ **speedup of factor** N_s (with $L := dim(\mathcal{X})$)

Technical Remark

Treating w

- **w** must be accessible by some index u (i.e. $u = 1 \ldots 4^8$ for 8-mers of Spectrum Kernel on DNA or word index for word-in-a-bag kernel)
- Needed Operations
  - `Clear:` $\mathbf{w} = \mathbf{0}$
  - `Add:` $w_u \leftarrow w_u + v$ (only needed $|W|$ times per iteration)
  - `Lookup:` obtain w_u (must be highly efficient)
- Storage
  - **Explicit Map** (store dense **w**); `Lookup` in $\mathcal{O}(1)$
  - **Sorted Array** (word-in-bag-kernel: all words sorted with value attached); `Lookup` in $\mathcal{O}(\log(\sum_u I(w_u \neq 0)))$
  - **Suffix Tries, Trees**; `Lookup` in $\mathcal{O}(K)$

Technical Remark

Treating w

- **w** must be accessible by some index u (i.e. $u = 1 \ldots 4^8$ for 8-mers of Spectrum Kernel on DNA or word index for word-in-a-bag kernel)
- Needed Operations
  - `Clear:` $\mathbf{w} = \mathbf{0}$
  - `Add:` $w_u \leftarrow w_u + v$ (only needed $|W|$ times per iteration)
  - `Lookup:` obtain w_u (must be highly efficient)
- Storage
  - **Explicit Map** (store dense **w**); Lookup in $\mathcal{O}(1)$
  - **Sorted Array** (word-in-bag-kernel: all words sorted with value attached); Lookup in $\mathcal{O}(\log(\sum_u I(w_u \neq 0)))$
  - **Suffix Tries, Trees**; Lookup in $\mathcal{O}(K)$

Technical Remark

Treating w

- **w** must be accessible by some index u (i.e. $u = 1 \ldots 4^8$ for 8-mers of Spectrum Kernel on DNA or word index for word-in-a-bag kernel)
- Needed Operations
  - `Clear:` $\mathbf{w} = \mathbf{0}$
  - `Add:` $w_u \leftarrow w_u + v$ (only needed $|W|$ times per iteration)
  - `Lookup:` obtain w_u (must be highly efficient)
- Storage
  - **Explicit Map** (store dense **w**); `Lookup` in $\mathcal{O}(1)$
  - **Sorted Array** (word-in-bag-kernel: all words sorted with value attached); `Lookup` in $\mathcal{O}(\log(\sum_u I(w_u \neq 0)))$
  - **Suffix Tries, Trees**; `Lookup` in $\mathcal{O}(K)$

Datastructures - Summary of Computational Costs

Comparison of worst-case run-times for operations

- `clear` of **w**
- `add` of all k-mers **u** from string **x** to **w**
- `lookup` of all k-mers **u** from $\mathbf{x}'$ in **w**

	Explicit map	Sorted arrays	Tries	Suffix trees
`clear`	$\mathcal{O}(\lvert\Sigma\rvert^K)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`add`	$\mathcal{O}(l_{\mathbf{x}})$	$\mathcal{O}(l_{\mathbf{x}} \log l_{\mathbf{x}})$	$\mathcal{O}(l_{\mathbf{x}} K)$	$\mathcal{O}(l_{\mathbf{x}})$
`lookup`	$\mathcal{O}(l_{\mathbf{x}'})$	$\mathcal{O}(l_{\mathbf{x}} + l_{\mathbf{x}'})$	$\mathcal{O}(l_{\mathbf{x}'} K)$	$\mathcal{O}(l_{\mathbf{x}'})$

Conclusions

- Explicit map ideal for small $\lvert\Sigma\rvert$
- Sorted Arrays for larger alphabets
- Suffix Arrays for large alphabets and order (overhead!)

Datastructures - Summary of Computational Costs

Comparison of worst-case run-times for operations

- `clear` of **w**
- `add` of all k-mers **u** from string **x** to **w**
- `lookup` of all k-mers **u** from $\mathbf{x}'$ in **w**

	Explicit map	Sorted arrays	Tries	Suffix trees
`clear`	$\mathcal{O}(\lvert\Sigma\rvert^K)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`add`	$\mathcal{O}(l_{\mathbf{x}})$	$\mathcal{O}(l_{\mathbf{x}} \log l_{\mathbf{x}})$	$\mathcal{O}(l_{\mathbf{x}} K)$	$\mathcal{O}(l_{\mathbf{x}})$
`lookup`	$\mathcal{O}(l_{\mathbf{x}'})$	$\mathcal{O}(l_{\mathbf{x}} + l_{\mathbf{x}'})$	$\mathcal{O}(l_{\mathbf{x}'} K)$	$\mathcal{O}(l_{\mathbf{x}'})$

Conclusions

- Explicit map ideal for small $\lvert\Sigma\rvert$
- Sorted Arrays for larger alphabets
- Suffix Arrays for large alphabets and order (overhead!)

Examples: Explicit map for WD kernel

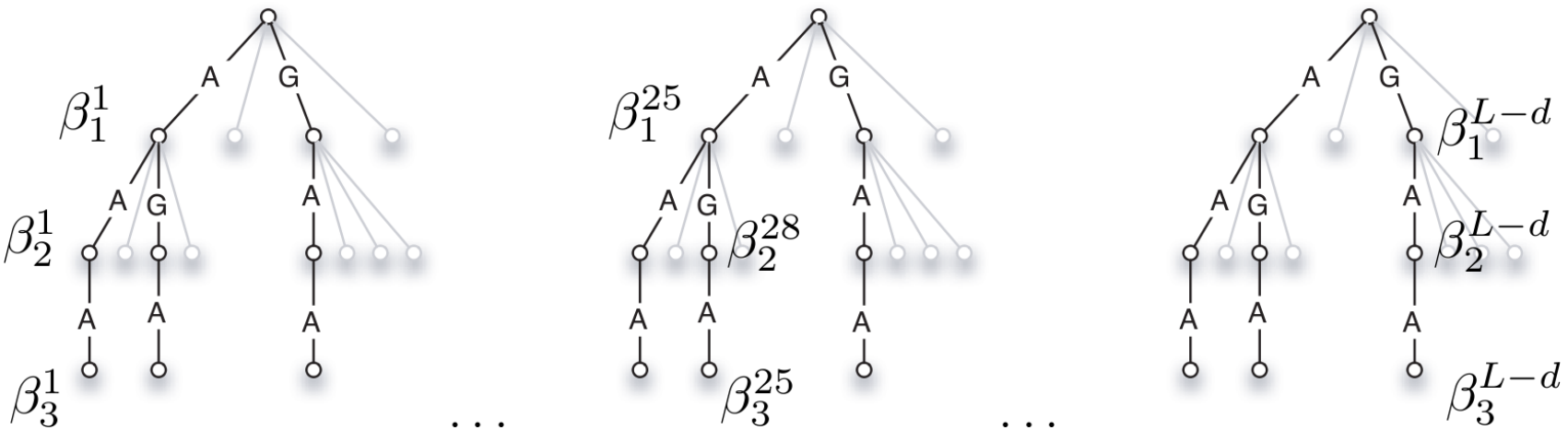
$$f(\mathbf{x}) = \sum_{k=1}^{K} \sum_{i=1}^{L-k+1} w\left(\mathbf{x}[i]^k, i\right) + b$$

k-mer	**pos. 1**	**pos. 2**	**pos. 3**	**pos. 4**	$\cdots$
A	+0.1	-0.3	-0.2	+0.2	$\cdots$
C	0.0	-0.1	+2.4	-0.2	$\cdots$
G	+0.1	-0.7	0.0	-0.5	$\cdots$
T	-0.2	-0.2	0.1	+0.5	$\cdots$
AA	+0.1	-0.3	+0.1	0.0	$\cdots$
AC	+0.2	0.0	-0.2	+0.2	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
TT	0.0	-0.1	+1.7	-0.2	$\cdots$
AAA	+0.1	0.0	0.0	+0.1	$\cdots$
AAC	0.0	-0.1	+1.2	-0.2	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
TTT	+0.2	-0.7	0.0	0.0	$\cdots$

Examples: Suffix Tries for WD kernel

- Use one tree of depth d per position in sequence
- For `lookup` traverse tree of depth d at a certain position in the sequence

Example $d = 3$:



It works! It is fast!

`Linadd` **speedup factor up to 100,000 when applying**

Further Speedup and Efficiency Considerations

- **w** may still be huge ⇒ fix by not constructing whole **w** but only blocks and computing batches
- Parallelize! ⇒ do `lookups` in parallel

What about training?

- Chunking based SVMs solve reduced problem on working set
- Update rule: $f_j \leftarrow f_j^{old} + \sum_{i \in W} (\alpha_i - \alpha_i^{old}) y_i \, \mathrm{k}(x_i, x_j)$
- Fast with kernel caching - but infeasible
  (for $N = 10^6$ only 125 kernel rows fit in 1GiB memory)
- No kernel caches necessary: Faster + Memory efficient

Training on 10 million examples ⇒ speedup factor up to 100

It works! It is fast!

`Linadd` **speedup factor up to 100,000 when applying**

Further Speedup and Efficiency Considerations

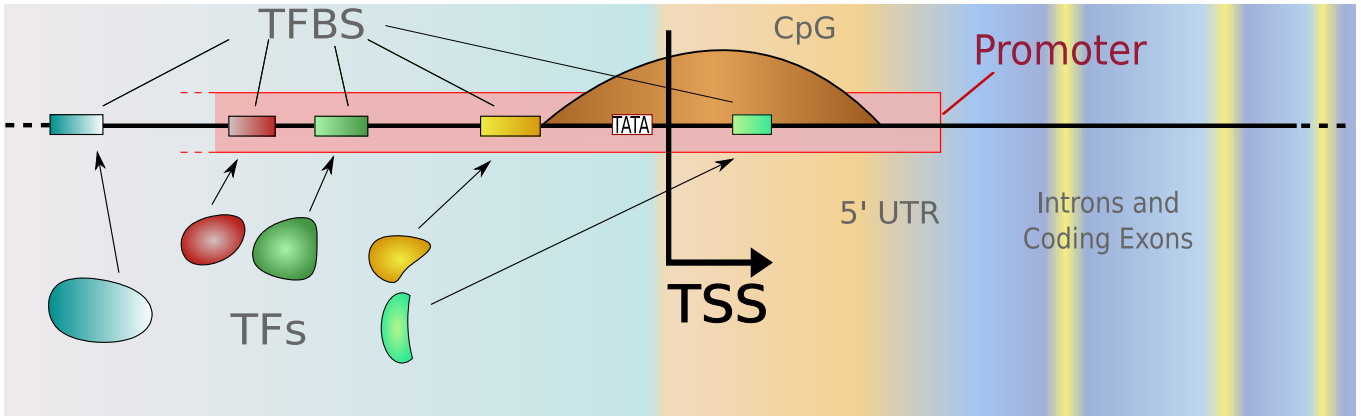
- **w** may still be huge ⇒ fix by not constructing whole **w** but only blocks and computing batches
- Parallelize! ⇒ do `lookups` in parallel

What about training?

- Chunking based SVMs solve reduced problem on working set
- Update rule: $f_j \leftarrow f_j^{old} + \sum_{i \in W} (\alpha_i - \alpha_i^{old}) y_i \, \mathrm{k}(x_i, x_j)$
- Fast with kernel caching - but infeasible
  (for $N = 10^6$ only 125 kernel rows fit in 1GiB memory)
- No kernel caches necessary: Faster + Memory efficient

Training on 10 million examples ⇒ speedup factor up to 100

Detecting Transcription Start Sites



Some features to describe TSS (weak)

- CpG islands (often over TSS/first exon; in most, but not all promoters)
- TSS with TATA box (≈ -30 bp upstream)
- TFBS in Promoter region, Exon content in UTR 5" region

Idea:

Combine weak features to build strong promoter predictor

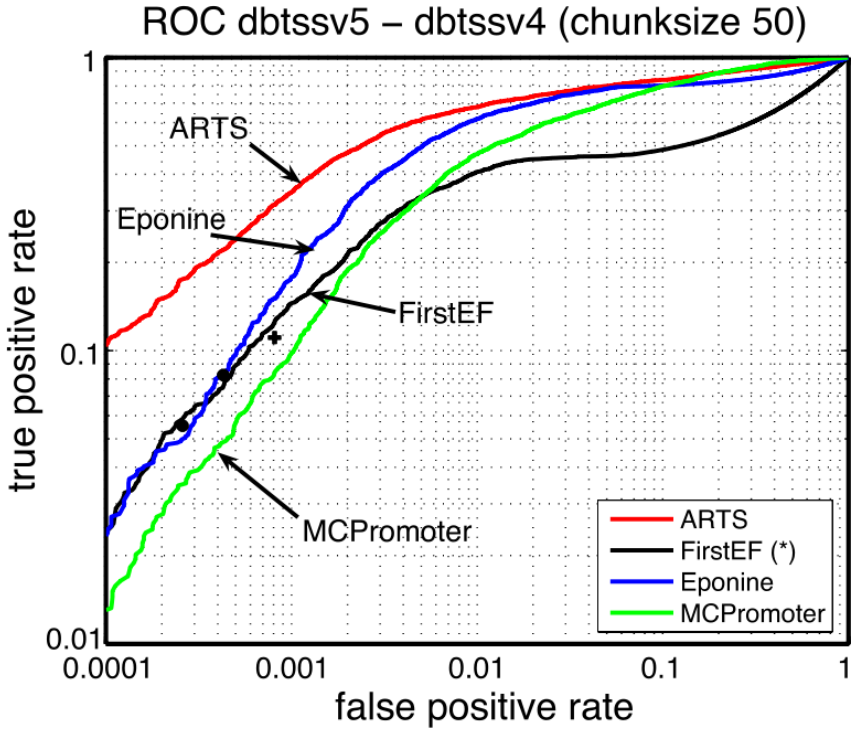
$$k(\mathbf{x}, \mathbf{x}') = k_{TSS}(\mathbf{x}, \mathbf{x}') + k_{CpG}(\mathbf{x}, \mathbf{x}') + k_{coding}(\mathbf{x}, \mathbf{x}') + k_{energy}(\mathbf{x}, \mathbf{x}') + k_{twist}(\mathbf{x}, \mathbf{x}')$$

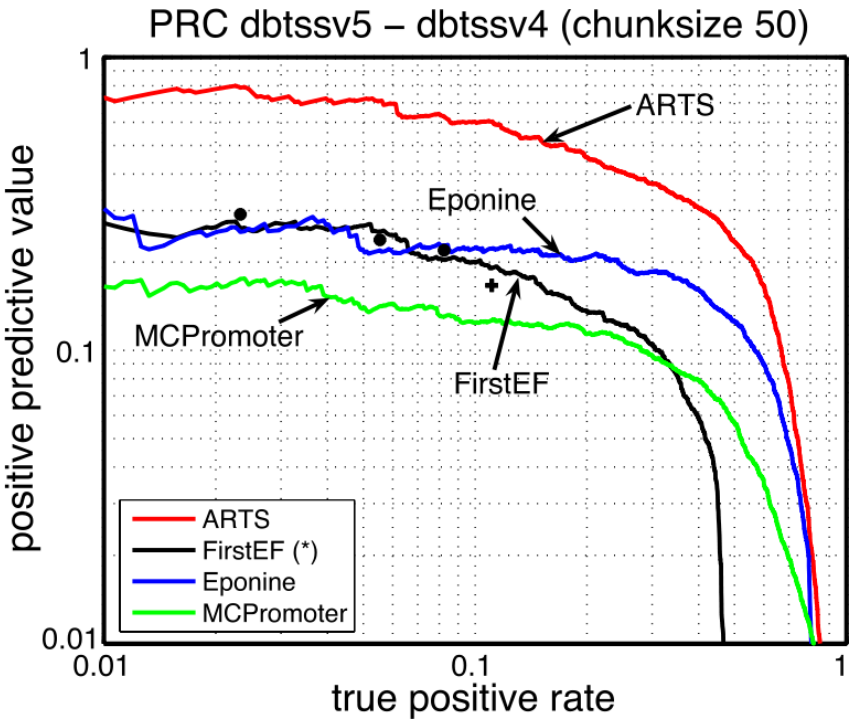
The 5 sub-kernels

1. TSS signal (including parts of core promoter with TATA box)
   – use **Weighted Degree Shift kernel**
2. CpG Islands, distant enhancers and TFBS upstream of TSS
   – use **Spectrum kernel** (large window upstream of TSS)
3. Model coding sequence TFBS downstream of TSS
   – use another **Spectrum kernel** (small window downstream of TSS)
4. Stacking energy of DNA
   – use *btwist* energy of dinucleotides with **Linear kernel**
5. Twistedness of DNA
   – use *btwist* angle of dinucleotides with **Linear kernel**

State-of-the-art Performance

Receiver Operator Characteristic and Precision Recall Curve





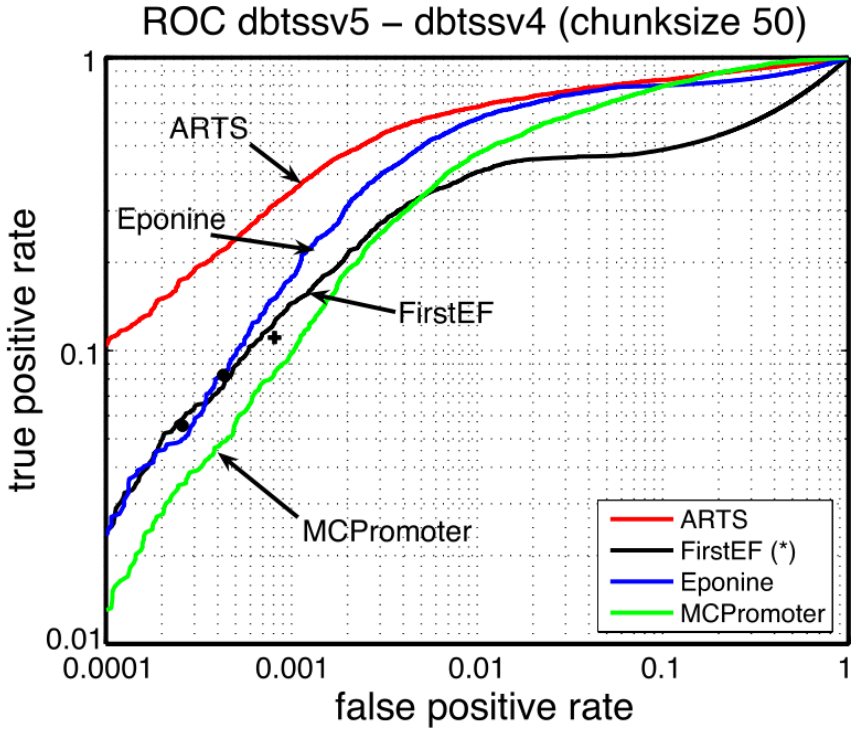
ARTS (Sonnenburg et al. 2006) twice as accurate!

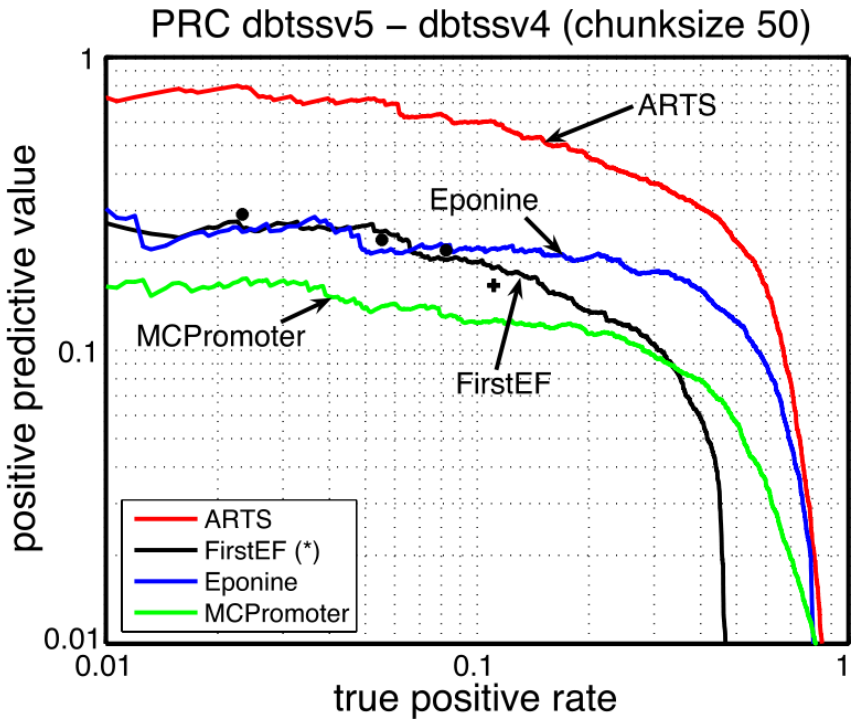
Independent evaluation of 17 methods (Abeel et al. ISMB, 2009)

TSS detector (ARTS) winner in evaluation of 17 methods.

State-of-the-art Performance

Receiver Operator Characteristic and Precision Recall Curve



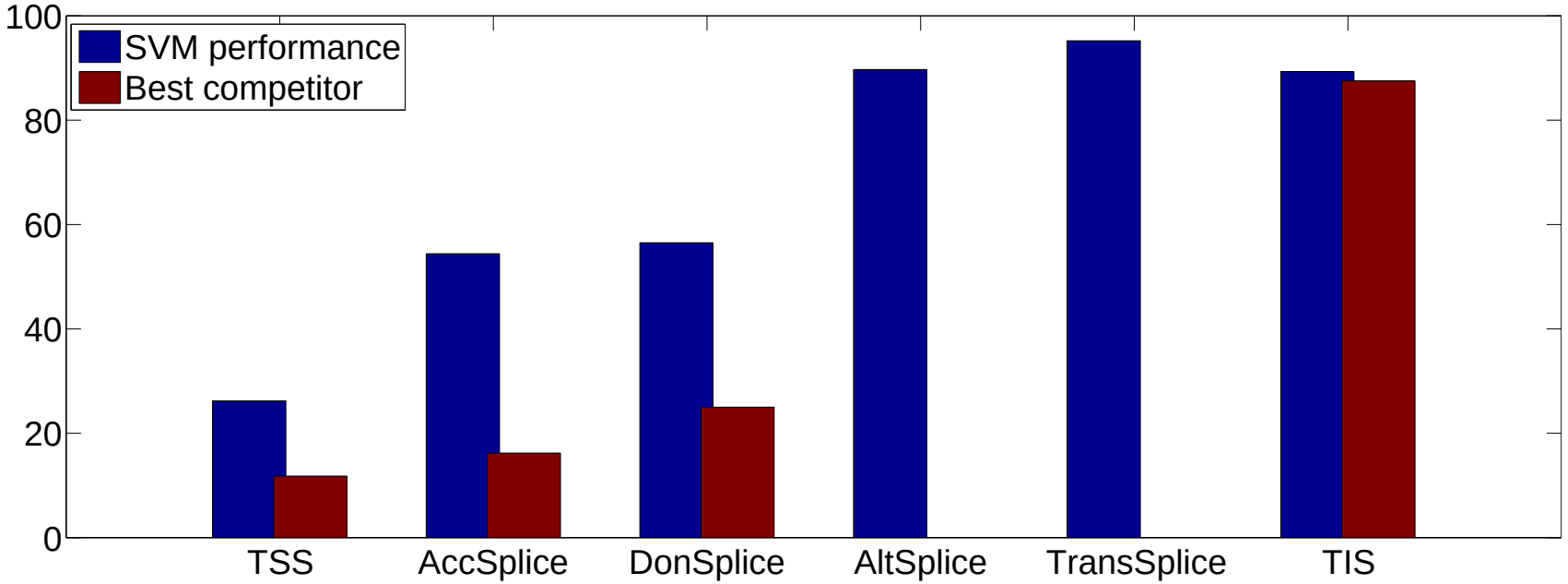


ARTS (Sonnenburg et al. 2006) twice as accurate!

Independent evaluation of 17 methods (Abeel et al. ISMB, 2009)

TSS detector (ARTS) winner in evaluation of 17 methods.

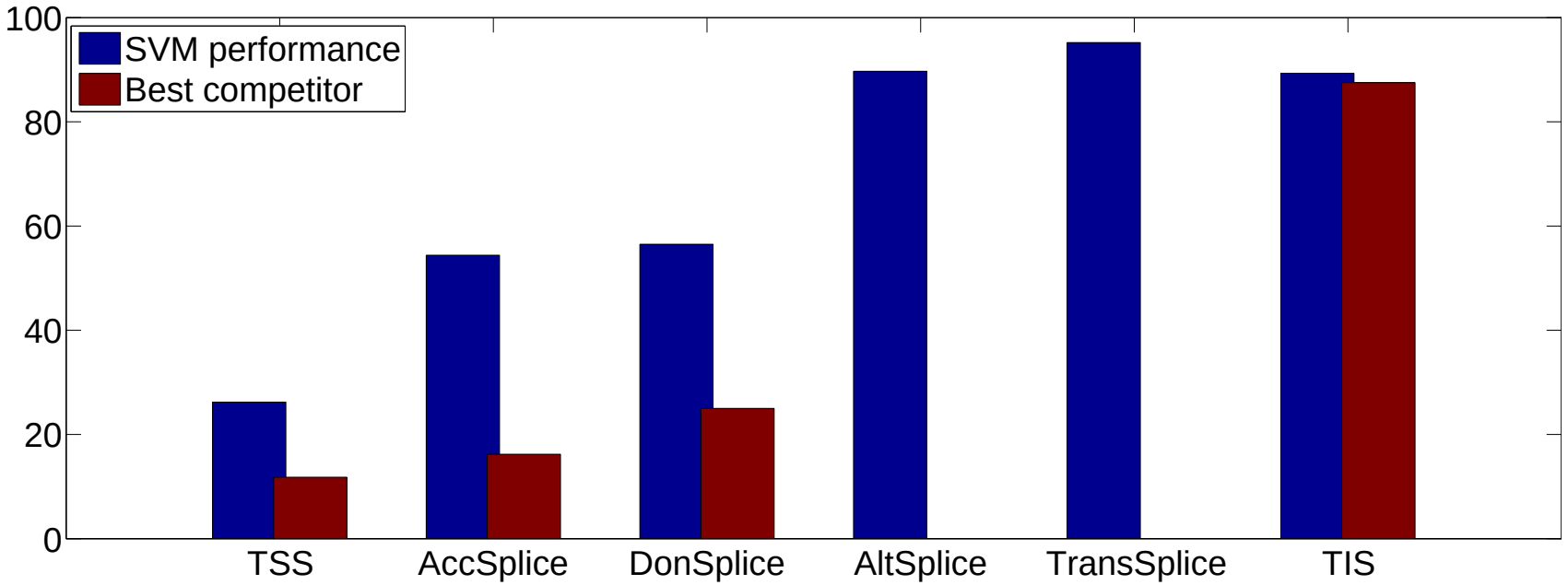
Beauty in Generality



- Transcription Start (Sonnenburg et al., 2006/Down et al. 2002)
- Acceptor Splice Site (Sonnenburg et al., 2007/Baten et al. 2006)
- Donor Splice Site (Sonnenburg et al., 2007/Baten et al. 2006)
- Alternative Splicing (Rätsch, Sonnenburg et al., 2005/-)
- Transsplicing (Schweikert, Sonnenburg et al., 2009/-)
- Translation Initiation (Sonnenburg et al., 2008/Saeys et al., 2007)

Drawback: SVM solution is hard to interpret!!

Beauty in Generality



- Transcription Start (Sonnenburg et al., 2006/Down et al. 2002)
- Acceptor Splice Site (Sonnenburg et al., 2007/Baten et al. 2006)
- Donor Splice Site (Sonnenburg et al., 2007/Baten et al. 2006)
- Alternative Splicing (Rätsch, Sonnenburg et al., 2005/-)
- Transsplicing (Schweikert, Sonnenburg et al., 2009/-)
- Translation Initiation (Sonnenburg et al., 2008/Saeys et al., 2007)

Drawback: SVM solution is hard to interpret!!

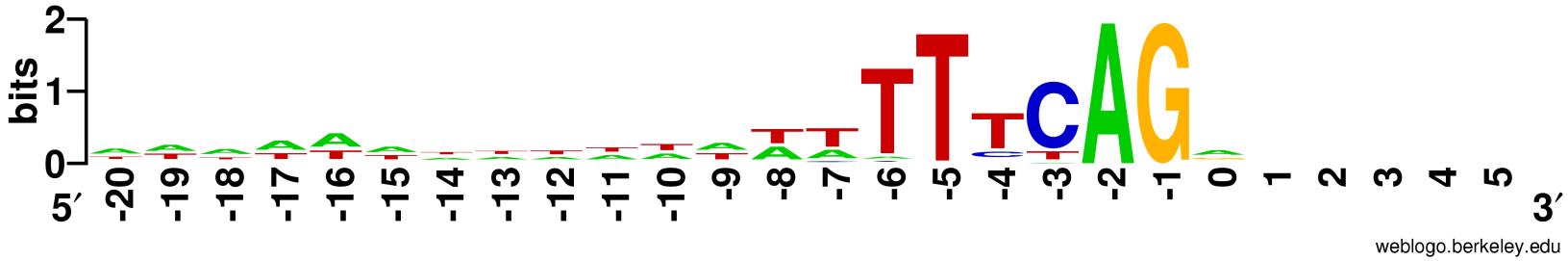
Outline

1. Introduction
   - Genomic Signals
2. Sequence Classification
   - Support Vector Machines
   - String Kernels
   - Example
3. Large Scale Learning
   - Application TSS recognition
4. Explanation and Visualization
   - Introduction
   - Definition
   - Applications
5. Discussion

Understanding Support Vector Machines

Goal

For PWMs we have sequence logos:

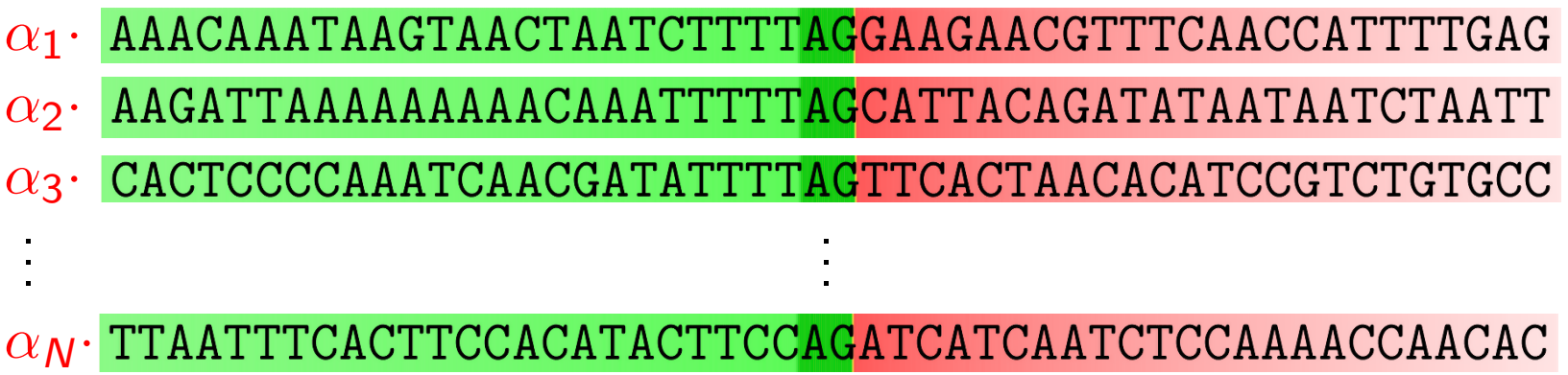


We would like to have **similar means to understand Support Vector Machines.**

Why Are SVM's Hard to Interpret?

SVM decision function is α weighting of training points

$$s(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}) + b$$



But we are interested in weights over features.

SVM Scoring Function

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \Phi(\mathbf{x}_i) \qquad s(\mathbf{x}) := \sum_{k=1}^{K} \sum_{i=1}^{L-k+1} w\left(\mathbf{x}[i]^k, i\right) + b$$

k-mer	pos. 1	pos. 2	pos. 3	pos. 4	$\cdots$
A	+0.1	-0.3	-0.2	+0.2	$\cdots$
C	0.0	-0.1	+2.4	-0.2	$\cdots$
G	+0.1	-0.7	0.0	-0.5	$\cdots$
T	-0.2	-0.2	0.1	+0.5	$\cdots$
AA	+0.1	-0.3	+0.1	0.0	$\cdots$
AC	+0.2	0.0	-0.2	+0.2	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
TT	0.0	-0.1	+1.7	-0.2	$\cdots$
AAA	+0.1	0.0	0.0	+0.1	$\cdots$
AAC	0.0	-0.1	+1.2	-0.2	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
TTT	+0.2	-0.7	0.0	0.0	$\cdots$

The Scoring System - Examples

$$s(\mathbf{x}) := \sum_{k=1}^{K} \sum_{i=1}^{L-k+1} w\left(\mathbf{x}[i]^k, i\right) + b$$

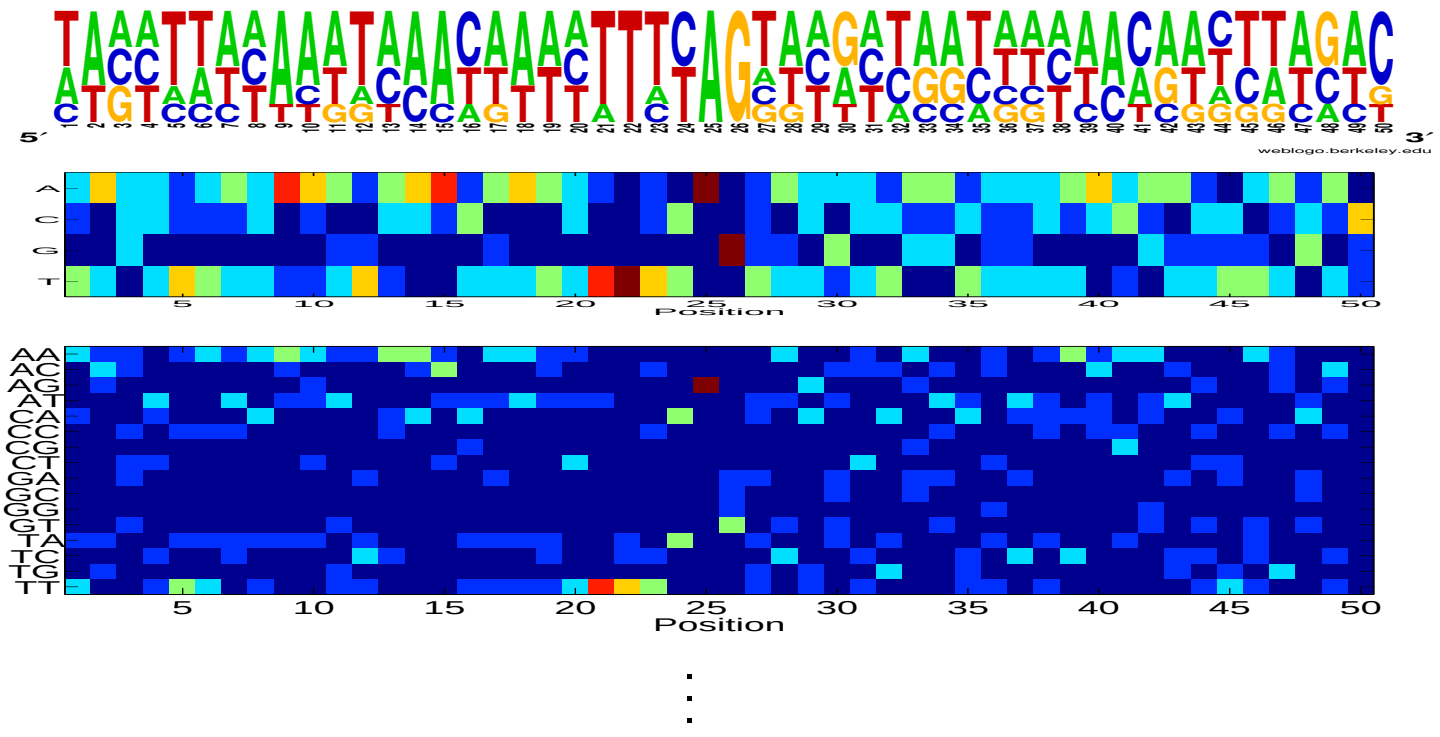
Examples:

- WD-kernel (Rätsch, Sonnenburg, 2005)
- WD-kernel with shifts (Rätsch, Sonnenburg, 2005)
- Spectrum kernel (Leslie, Eskin, Noble, 2002)
- Oligo Kernel (Meinicke et al., 2004)

Not limited to SVM's:

- Markov Chains (higher order/inhomogeneous/mixed order)

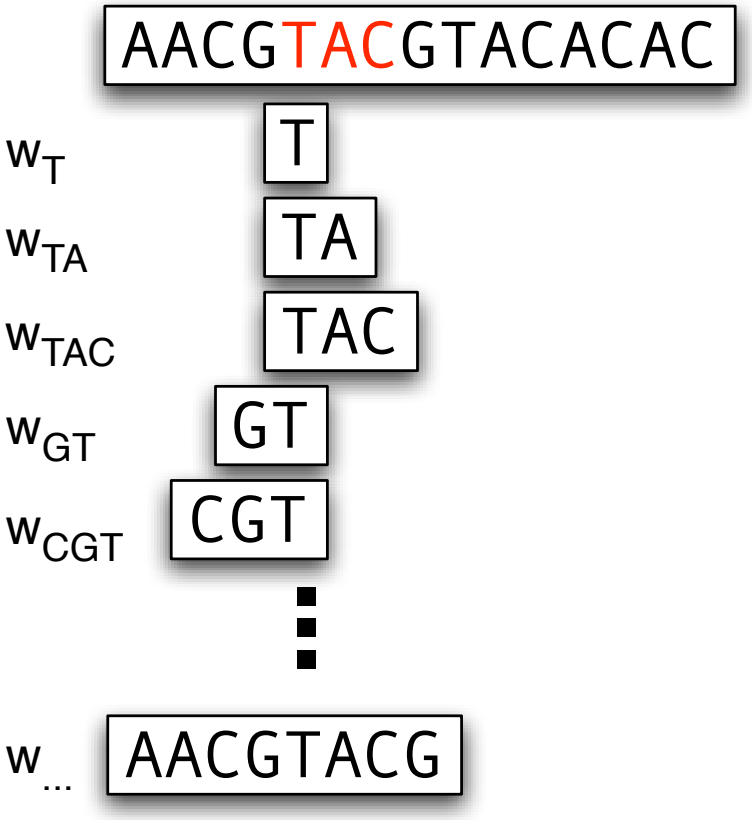
The SVM Weight Vector **w**



- Explicit representation of **w** allows for (some) interpretation!
- String kernel SVMs capable of efficiently dealing with large k-mers $k > 10$

But: Weights for substrings not independent

Interdependence of $k-$mer Weights



What is the score for TAC?

- Take w_{TAC}?
- But substrings and overlapping strings contribute too!

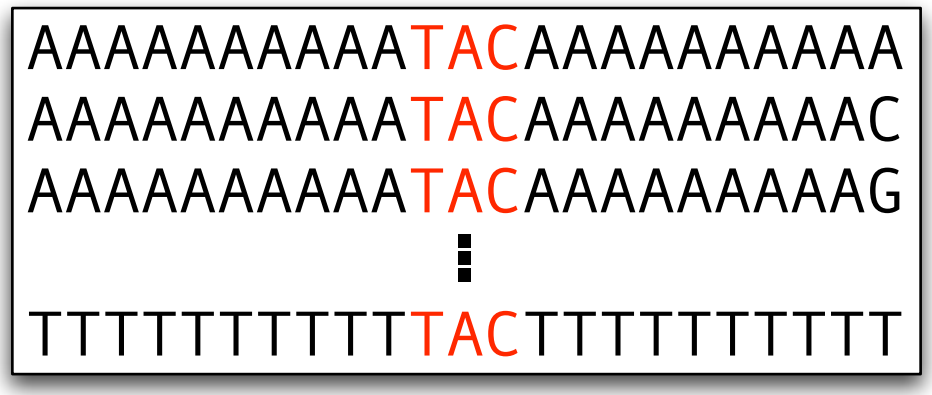
Problem

The SVM-w does NOT reflect the score for a motif

Positional Oligomer Importance Matrices (POIMs)

Idea:

- Given $k-$mer **z** at position j in the sequence, compute expected score $\mathbb{E}\left[\, s(\mathbf{x}) \mid \mathbf{x}[j] = \mathbf{z} \,\right]$ (**for small k**)



- Normalize with *expected score* over **all sequences**

POIMs (Sonnenburg et al. 2008)

$$Q(\mathbf{z}, j) \;:=\; \mathbb{E}\left[\, s(\mathbf{x}) \mid \mathbf{x}[j] = \mathbf{z} \,\right] - \mathbb{E}\left[\, s(\mathbf{x}) \,\right]$$

$\Rightarrow$ **Needs efficient algorithm for computation**

Efficient Computation

Effort of naive approach exponential $\mathcal{O}(|\Sigma|^L + L|\Sigma|^k)$
(e.g. Splice Sites 10^{120})

$$Q(\mathbf{z}, j) := \mathbb{E}[s(\mathbf{x}) \mid \mathbf{x}[j] = \mathbf{z}] - \mathbb{E}[s(\mathbf{x})]$$

- Number of k-mers grows linearly with size of input
- Only features which are dependent on $(\mathbf{z}, j)$ matter
- Computation can be split in contributions from 4 cases

Efficient Recursive Algorithm:

Effort linear in length of input: $\mathcal{O}(LN + L|\Sigma|^k)$

Deriving an Efficient Algorithm

All features which are independent of (z, j) vanish

$$
\begin{aligned}
Q(\mathbf{z}, j) &:= \mathbb{E}\left[s(\mathbf{x}) \mid \mathbf{x}[j] = \mathbf{z} \right] - \mathbb{E}\left[s(\mathbf{x}) \right]. \\
&= \sum_{(\mathbf{y},i)\in\mathcal{I}} w(\mathbf{y}, i) \left[Pr\left(\mathbf{x}[i] = \mathbf{y} \mid \mathbf{x}[j] = \mathbf{z} \right) - Pr\left(\mathbf{x}[i] = \mathbf{y} \right) \right] \\
&= u(\mathbf{z}, j) - \sum_{\mathbf{z}'\in\Sigma^{|\mathbf{z}|}} Pr\left(\mathbf{x}[j] = \mathbf{z}' \right) u(\mathbf{z}', j)
\end{aligned}
$$

Computation can be split in contributions from 4 cases:

$$
\begin{aligned}
u(\mathbf{z}, j) &:= \sum_{(\mathbf{y},i)\in\mathcal{I}(\mathbf{z},j)} Pr\left(\mathbf{x}[i] = \mathbf{y} \mid \mathbf{x}[j] = \mathbf{z} \right) w(\mathbf{y}, i) \\
&= u^{\vee}(\mathbf{z}, j) + u^{\wedge}(\mathbf{z}, j) + u^{<}(\mathbf{z}, j) + u^{>}(\mathbf{z}, j) - w(\mathbf{z}, j) \ ,
\end{aligned}
$$

For AATACGTAC: substring, superstring, left and right partial overlap

Efficient Recursive Algorithm

$$
\begin{aligned}
u^{\vee}(\sigma\mathbf{z}\tau, j) &= w_{(\sigma\mathbf{z}\tau, j)} + u^{\vee}(\sigma\mathbf{z}, j) + u^{\vee}(\mathbf{z}\tau, j+1) - u^{\vee}(\mathbf{z}, j+1) \quad \text{for } \sigma, \tau \in \Sigma \\
u^{\wedge}(\mathbf{z}, j) &= w_{(\mathbf{z}, j)} - \sum_{(\sigma,\tau)\in\Sigma^2} Pr\left(\mathbf{x}[j-1] = \sigma\right) Pr\left(\mathbf{x}[j+k] = \tau\right) u^{\wedge}(\sigma\mathbf{z}\tau, j-1) \\
&\quad + \sum_{\sigma\in\Sigma} Pr\left(\mathbf{x}[j-1] = \sigma\right) u^{\wedge}(\sigma\mathbf{z}, j-1) + \sum_{\tau\in\Sigma} Pr\left(\mathbf{x}[j+p] = \tau\right) u^{\wedge}(\mathbf{z}\tau, j) \\
u^{<}(\mathbf{z}, j) &= \sum_{\sigma\in\Sigma} Pr\left(\mathbf{x}[j-1] = \sigma\right) \sum_{l=1}^{\min\{k,K\}-1} L\left(\sigma(\mathbf{z}[1]^l), j-1\right) \\
u^{>}(\mathbf{z}, j) &= \sum_{\tau\in\Sigma} Pr\left(\mathbf{x}[j+k] = \tau\right) \sum_{l=1}^{\min\{k,K\}-1} R\left(\mathbf{z}[k-l+1]^l\tau, j+p-l\right) \quad ,
\end{aligned}
$$

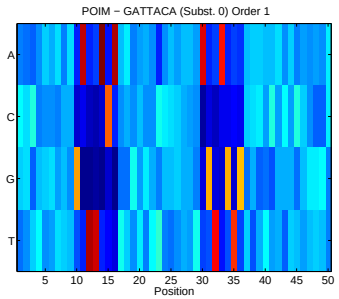
$$
\begin{aligned}
L(\mathbf{z}, j) &= w_{(\mathbf{z}, j)} + \sum_{\sigma\in\Sigma} Pr\left(\mathbf{x}\left[j-1\right] = \sigma\right) L(\sigma\mathbf{z}, j-1) \\
R(\mathbf{z}, j) &= w_{(\mathbf{z}, j)} + \sum_{\tau\in\Sigma} Pr\left(\mathbf{x}\left[j+p\right] = \tau\right) R(\mathbf{z}\tau, j)
\end{aligned}
$$

Ranking Features and Condensing Information

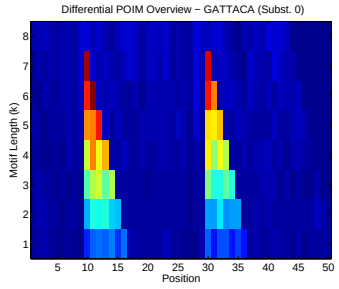
- Obtain highest scoring $\mathbf{z}$ from $Q(\mathbf{z}, i)$ (Enhancer or Silencer)

$\mathbf{z}$	i	$Q(\mathbf{z}, i)$
GATTACA	10	+30
AGTAGTG	30	+20
AAAAAAA	10	-10
...	...	...

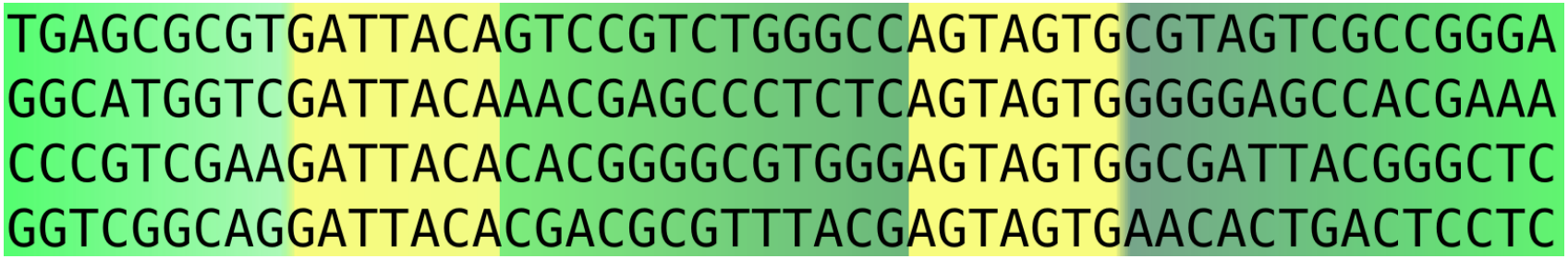
- Visualize POIM as heat map;
  x-axis: position
  y-axis: k-mer
  color: importance



- For large k: Differential POIMs;
  x-axis: position
  y-axis: k-mer length
  color: importance



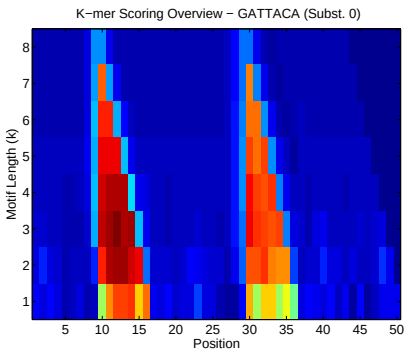
GATTACA and AGTAGTG at Fixed Positions 10 and 30



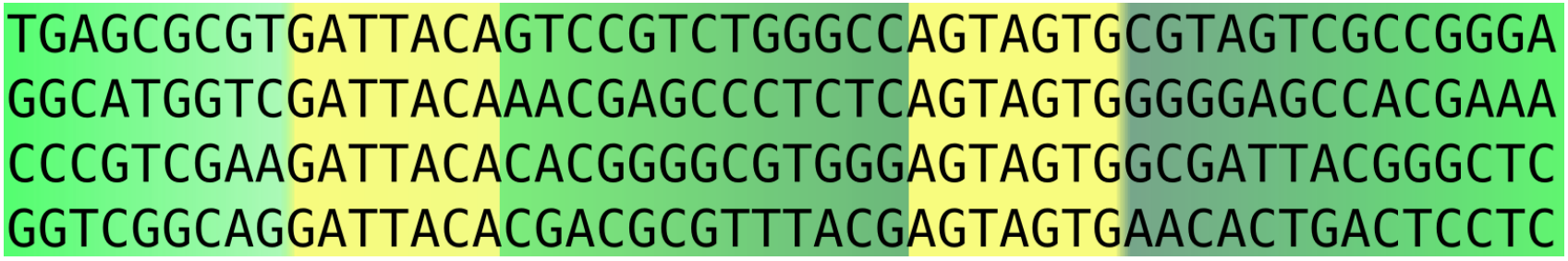
GATTACA and AGTAGTG at fixed positions 10 and 30

TGAGCGCGTGATTACAGTCCGTCTGGGCCAGTAGTGCGTAGTCGCCGGGA
GGCATGGTCGATTACAAACGAGCCCTCTCAGTAGTGGGGGAGCCACGAAA
CCCGTCGAAGATTACACACGGGGCGTGGGAGTAGTGGCGATTACGGGCTC
GGTCGGCAGGATTACACGACGCGTTTACGAGTAGTGAACACTGACTCCTC

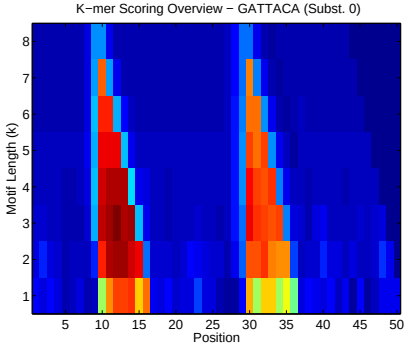
w



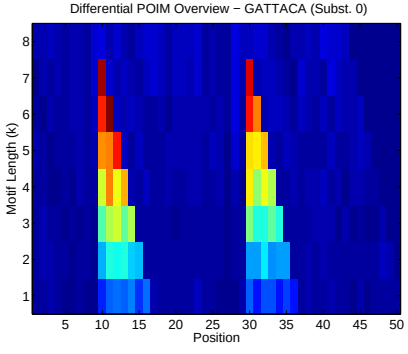
GATTACA and AGTAGTG at fixed positions 10 and 30



w



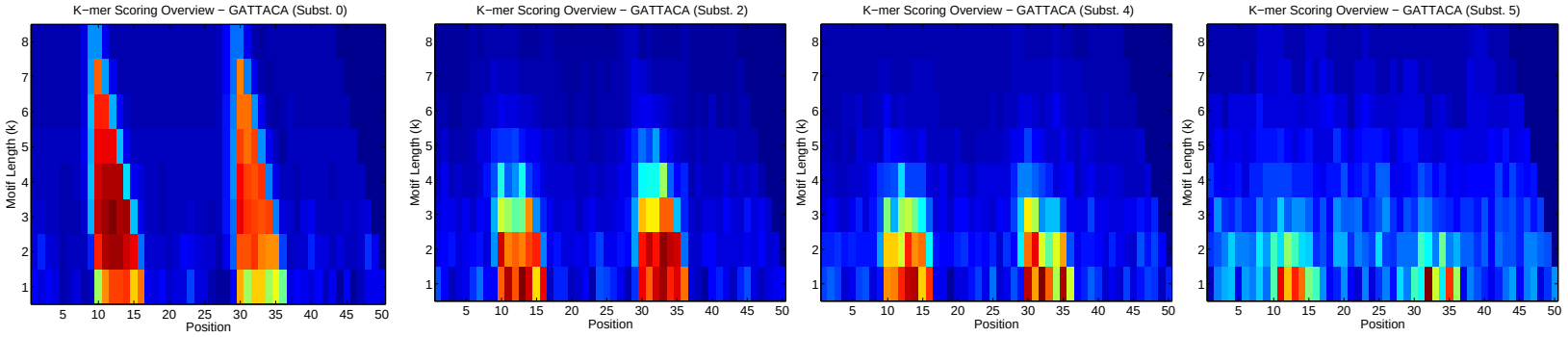
Q



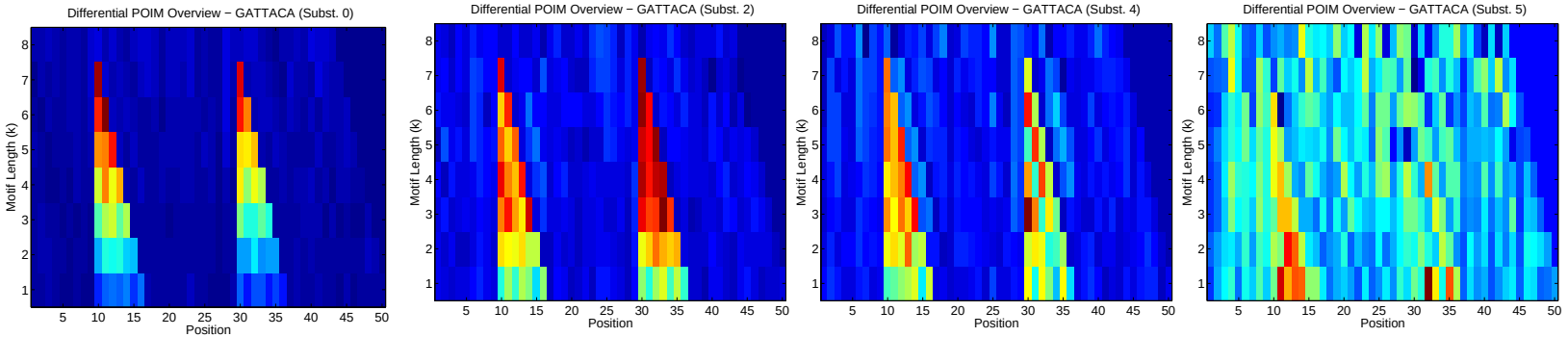
GATTACA and AGTAGTG at fixed positions 10 and 30

TGAGCGCGTGATTACAGTCCGTCTGGGCCAGTAGTGCGTAGTCGCCGGGA
GGCATGGTCGATTACAAACGAGCCCTCTCAGTAGTGGGGGAGCCACGAAA
CCCGTCGAAGATTACACACGGGGCGTGGGAGTAGTGGCGATTACGGGCTC
GGTCGGCAGGATTACACGACGCGTTTACGAGTAGTGAACACTGACTCCTC

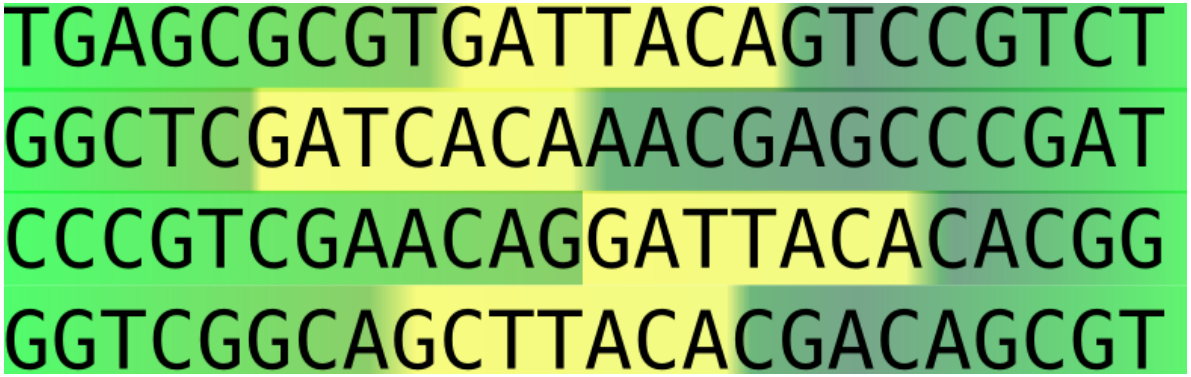
w



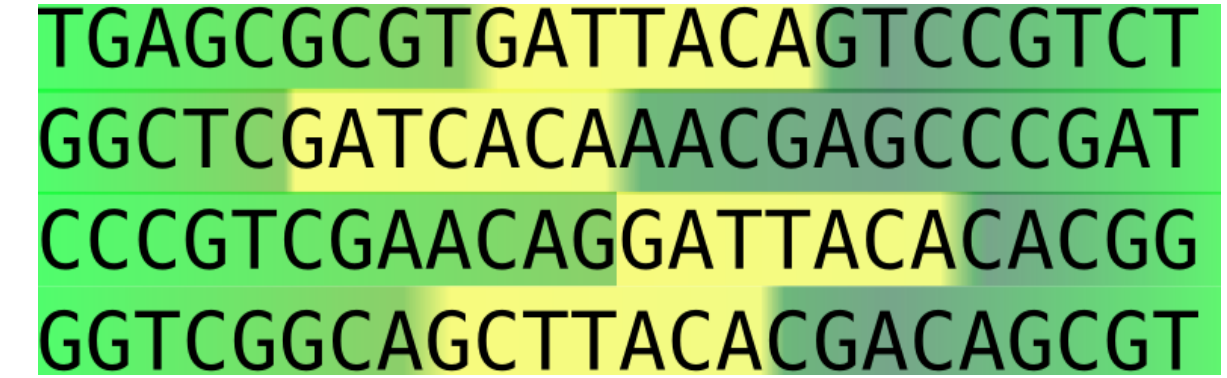
Q

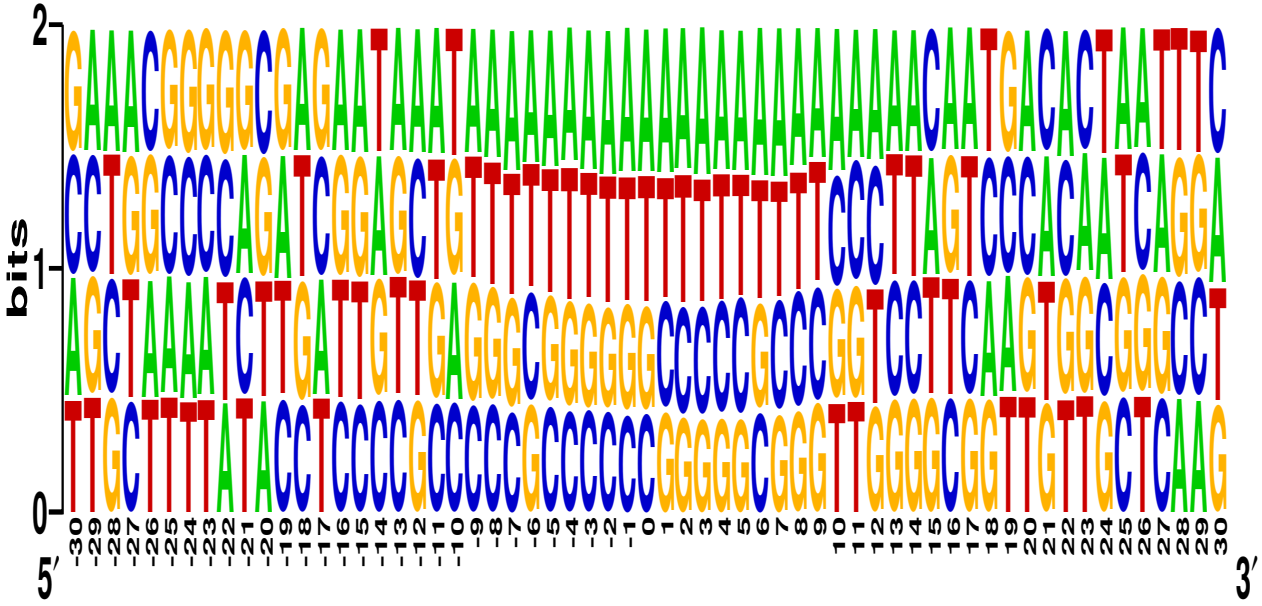


GATTACA at variable positions

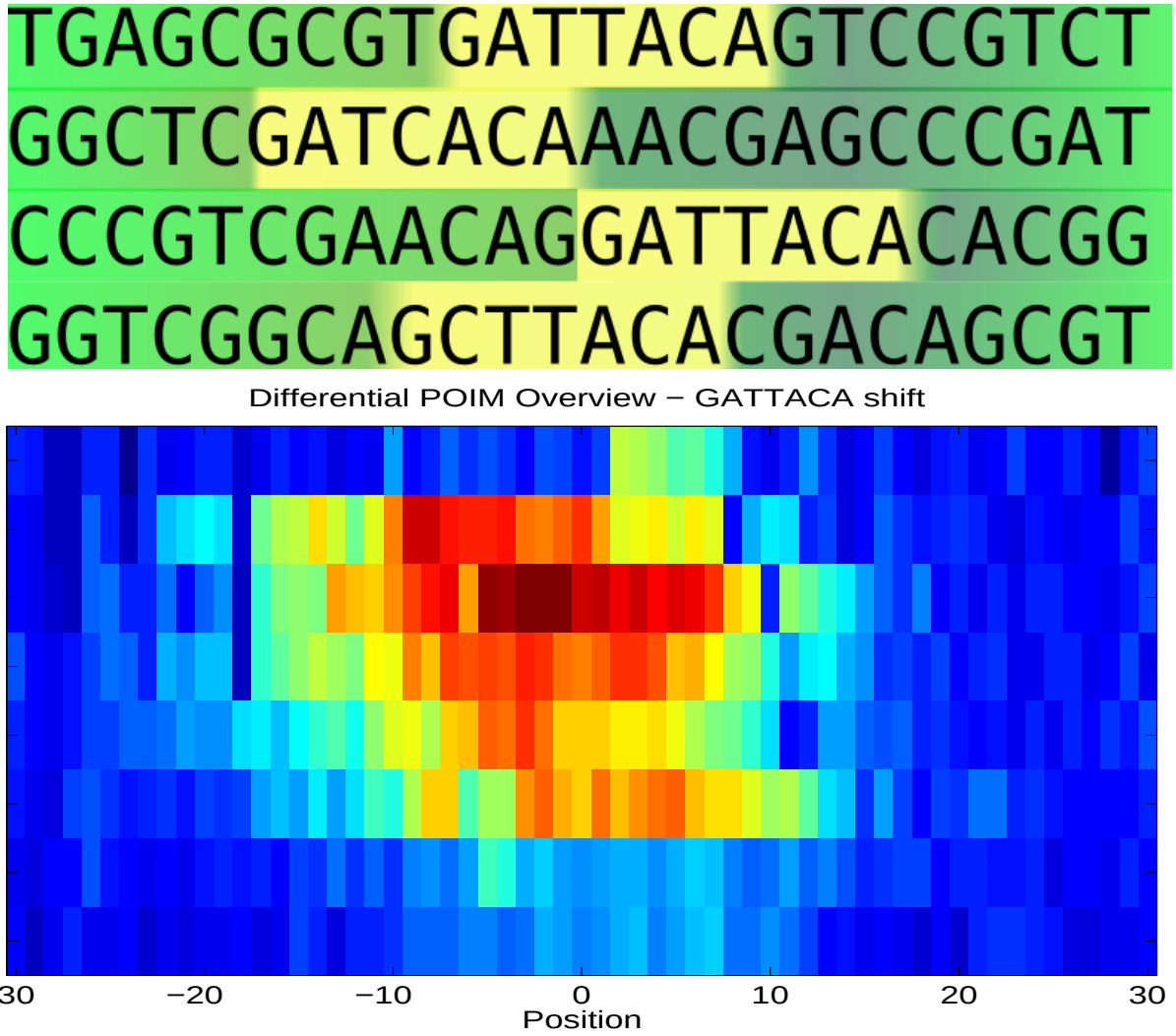


GATTACA at variable positions

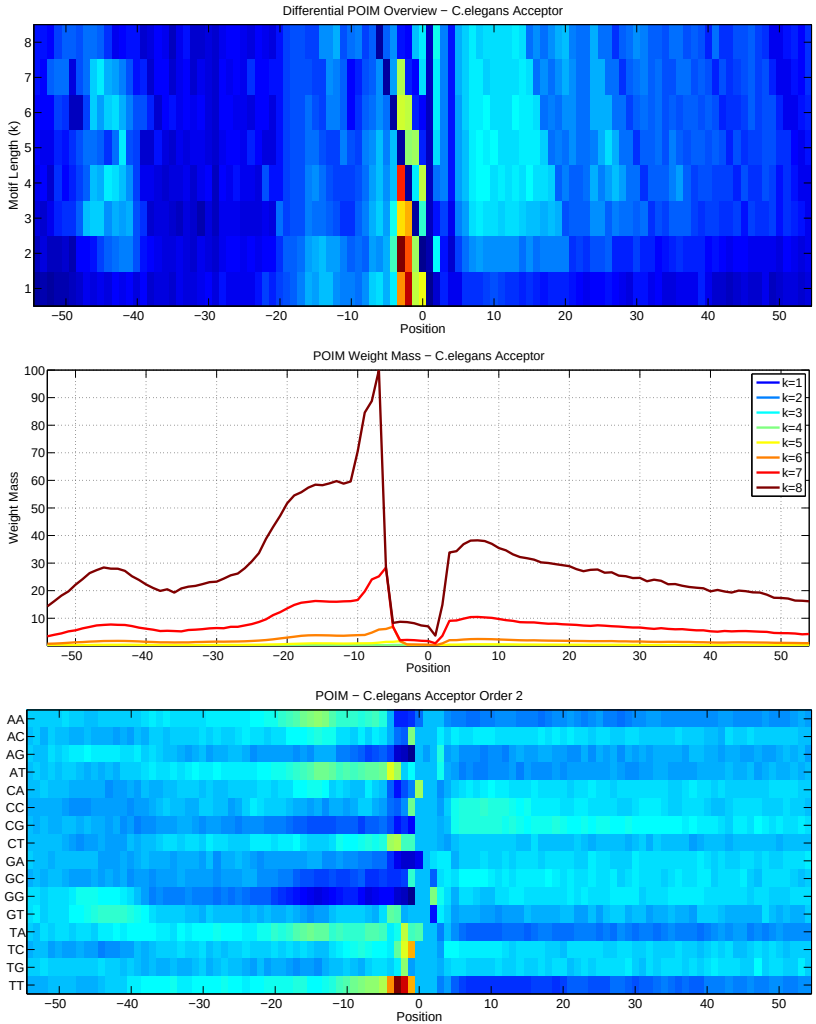




GATTACA at variable positions

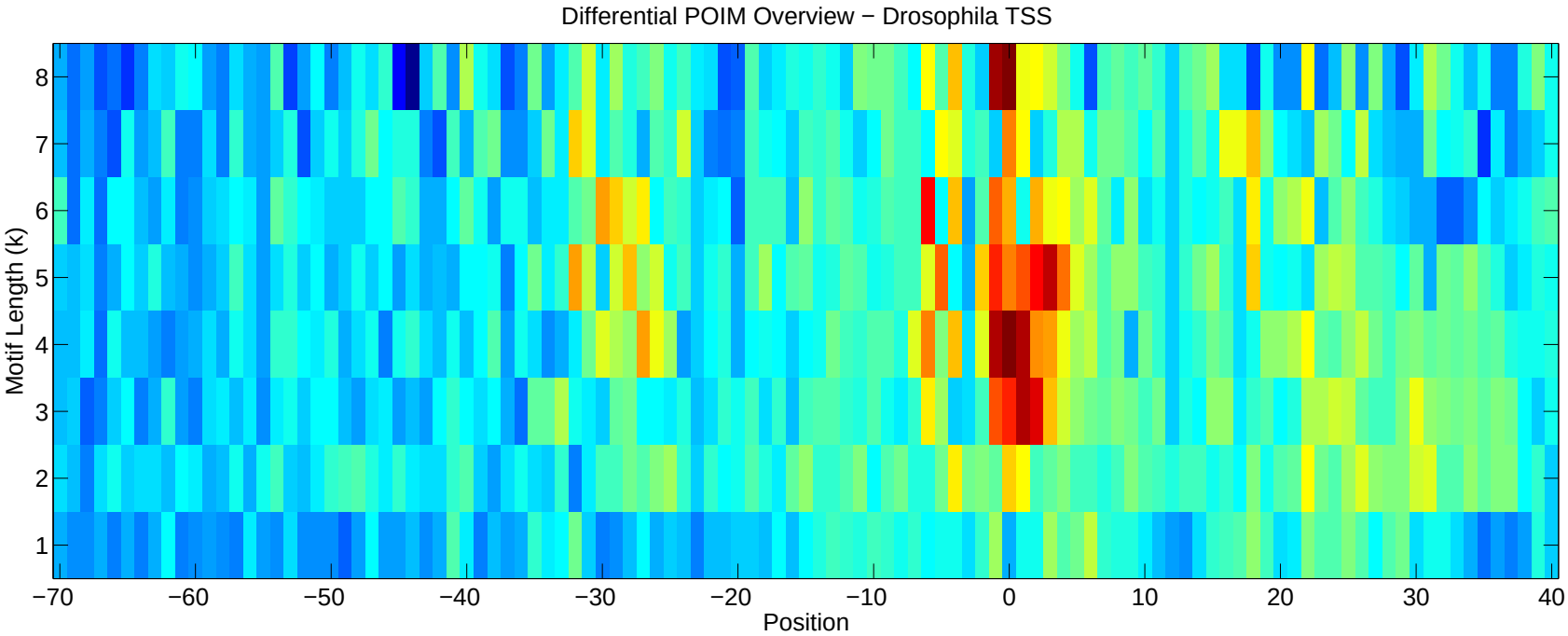


C.elegans Acceptor Splice Site Recognition



- Upstream
  - AG**GT**AAGT -44/++ Donor
  - GGGGGG -16/- - Silencer?
  - **TAAT**AA -16/++ Branch
- Central
  - TTTTTTC -06/+
  - TTTC**AG**$\frac{A}{G}$ -03/++ Acceptor
- Downstream
  - TTTTTTTT +07/- -
  - TTTTT +26/- -

Drosophila Transcription Starts



TATAAAA	-29/++	**CAGT**CAGT	-01/++	CGTCGCG	+18/++
GTATAAA	-30/++	T**CAGT**TGT	-01/++	GCGCGCG	+23/++
ATATAAA	-28/++	CGT**CAGT**T	-03/++	CGCGCGC	+22/++
TATA-box		**Inr** **TCA**$\frac{G}{T}$**T**$\frac{T}{C}$		**CpG**	

Outline

1. Introduction
   - Genomic Signals
2. Sequence Classification
   - Support Vector Machines
   - String Kernels
   - Example
3. Large Scale Learning
   - Application TSS recognition
4. Explanation and Visualization
   - Introduction
   - Definition
   - Applications
5. Discussion

Conclusions

Support Vector Machines with string kernels

- General and often state-of-the art signal detectors
- Applicable to genome-sized datasets
- Using POIMs SVMs are interpretable

Efficient implementation

`http://www.shogun-toolbox.org`
More machine learning software `http://mloss.org`

Discussion

- Multiple Kernel Learning for interpretability and improving Accuracy (Sonnenburg et al. 2004; Kloft, Sonnenburg et al. 2009)
- Learn string-kernel SVMs in the primal (Sonnenburg et al. 2010)