Overview

1. Introduction to Bioinformatics
   - Basic Biology and Central Dogma
   - Typical Data Types
   - Common Analysis Tasks
2. Sequence Analysis (with SVMs)
   - String Kernels
   - Large Scale Data Structures
   - Heterogeneous Data
3. Structured Output Learning
   - Hidden Markov Models & Dynamic Programming
   - Discriminative Approaches (CRFs & HMSVMs)
   - Large Scale Approaches
4. Some Applications
   - Spliced Alignments (PALMA)
   - Gene Finding (mGene)
   - Analysis of Resequencing Arrays (SNPs and Polymorphic regions)

Part II: Sequence Analysis with SVMs

Part II: Sequence Analysis with SVMs

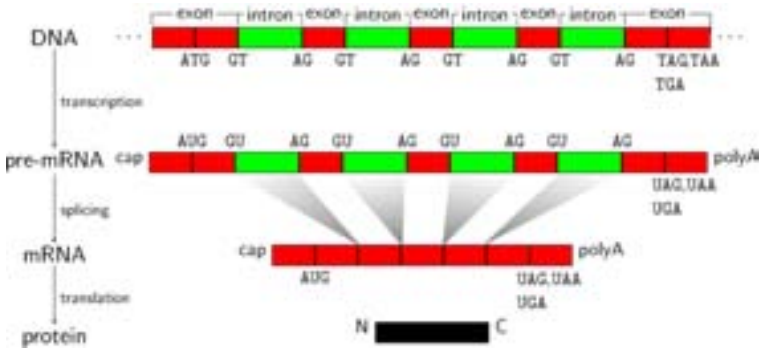
- **Running Example: Splicing**
- String kernels
- Large Scale Data Structures
- Heterogeneous Data

NB: Large parts are from the tutorial at the German Conference for Bioinformatics 2006 with Cheng Soon Ong.

Running Example: Splicing

Central Dogma (Remember!)

- Transcription: Copy DNA to RNA
- Splicing (remove introns), ...
- Translation (produce protein)



A few facts about splicing

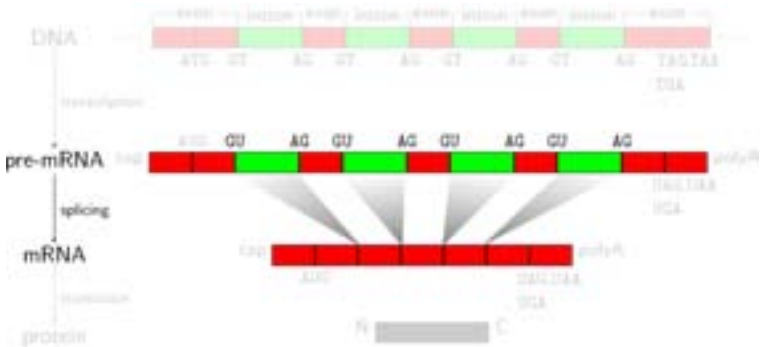
- Transcript may contain between zero and about 100 introns
- Intron typically ...
  - starts with letters `GT` (donor splice site)
  - ends with letters `AG` (acceptor splice site)
  - is between 30 nt and 100,000 nt long

For graphical illustration of splicing see for instance
`http://vcell.ndsu.nodak.edu/animations/mrnasplicing`.

Running Example: Splicing

Central Dogma (Remember!)

- Transcription: Copy DNA to RNA
- Splicing (remove introns), ...
- Translation (produce protein)



A few facts about splicing

- Transcript may contain between zero and about 100 introns
- Intron typically ...
  - starts with letters `GT` (donor splice site)
  - ends with letters `AG` (acceptor splice site)
  - is between 30 nt and 100,000 nt long

For graphical illustration of splicing see for instance
`http://vcell.ndsu.nodak.edu/animations/mrnasplicing`.

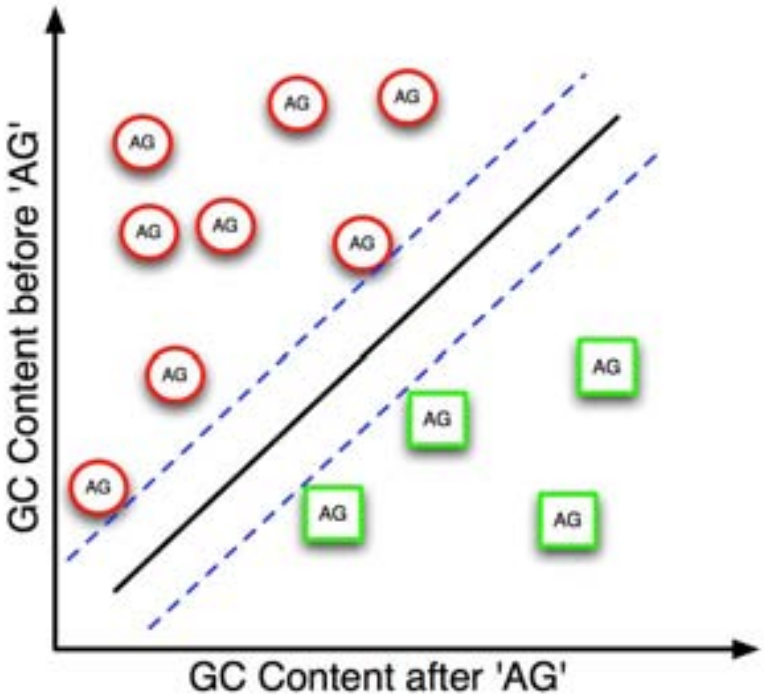
Recognition of Splice Sites

- Given: Potential acceptor splice sites

```
AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT
CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC
TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC
```

intron **exon**

- Goal: Rule that distinguishes true from false ones



e.g. exploit that exons have higher GC content

or

that certain motifs appear near splice site

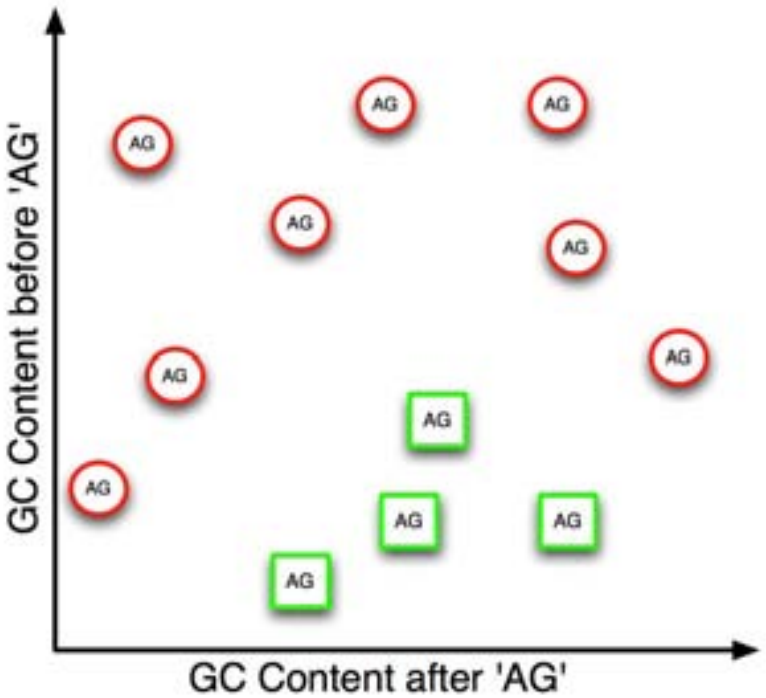
Recognition of Splice Sites

- Given: Potential acceptor splice sites

```
AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT
CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC
TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC
```

intron **exon**

- Goal: Rule that distinguishes true from false ones



More realistic problem!?

- Not linearly separable!
- Need nonlinear separation!?
- Need more features!

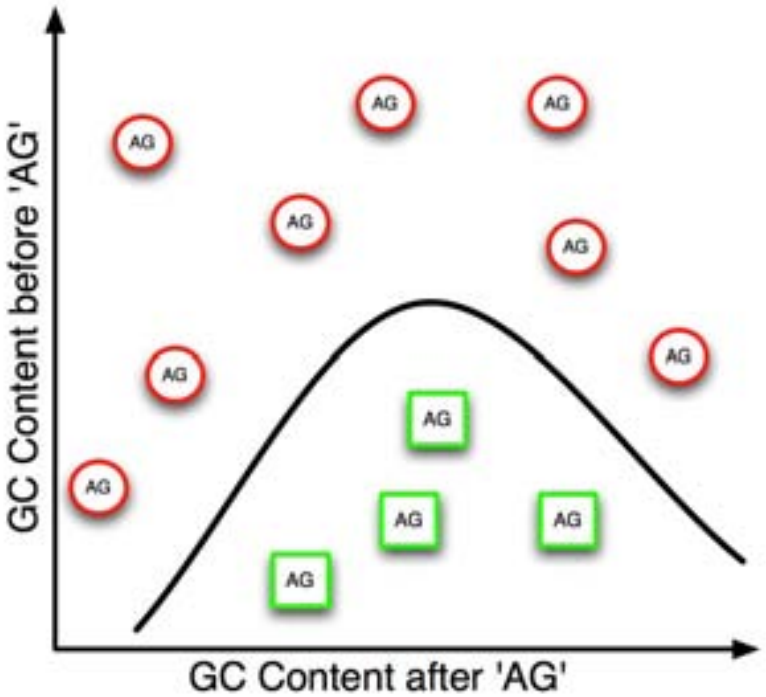
Recognition of Splice Sites

- Given: Potential acceptor splice sites

```
AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT
CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC
TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC
```

intron **exon**

- Goal: Rule that distinguishes true from false ones



More realistic problem!?

- Not linearly separable!
- **Need nonlinear separation!?**
- Need more features!

Part II: Sequence Analysis with SVMs

Part II: Sequence Analysis with SVMs

- Running Example: Splicing
- **String kernels**
- Large Scale Data Structures
- Heterogeneous Data

NB: Large parts are from the tutorial at the German Conference for Bioinformatics 2006 with Cheng Soon Ong.

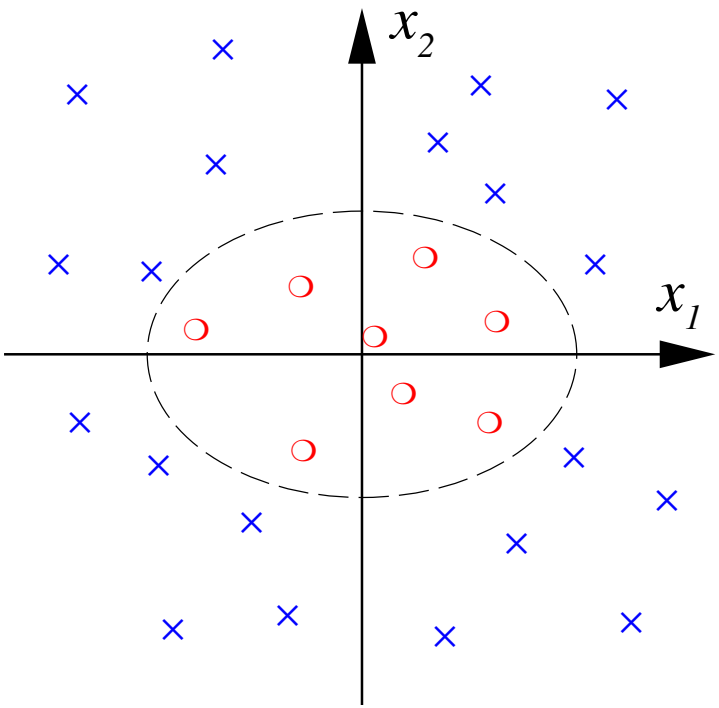
Nonlinear Algorithms in Feature Space

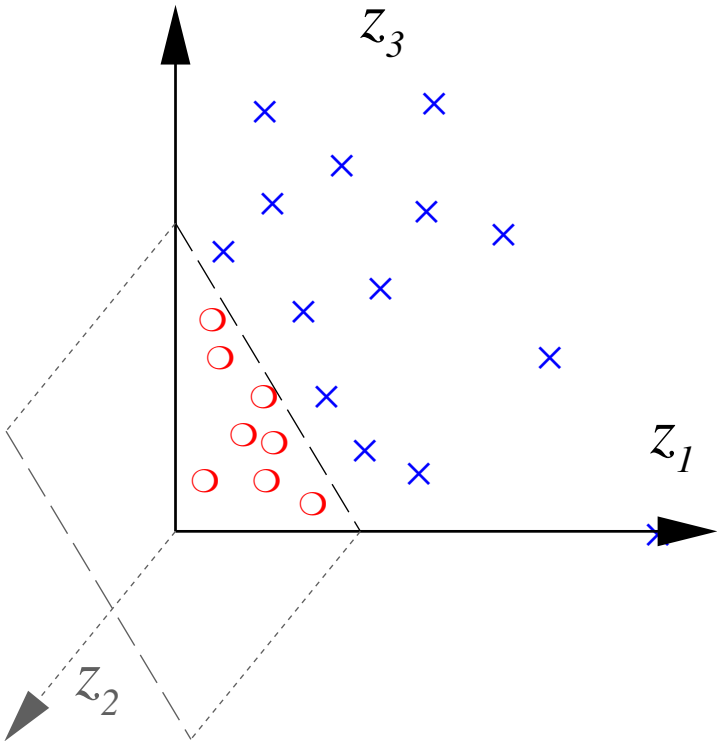
Linear separation might be not sufficient!

$\Rightarrow$ Map into a higher dimensional feature space

Example: All second order monomials

$$\begin{aligned} \Phi : \mathbf{R}^2 &\rightarrow \mathbf{R}^3 \\ (x_1, x_2) &\mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2) \end{aligned}$$





Summary "Kernel Trick"

- Representer Theorem: $\mathbf{w} = \sum_{i=1}^{N} \alpha_i \Phi(\mathbf{x}_i)$.
- Hyperplane in $\mathcal{F}$: $y = \operatorname{sgn}\left(\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b\right)$
- Putting things together

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sgn}\left(\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{N} \alpha_i \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \rangle + b\right) \\ &= \operatorname{sgn}\left(\sum_{i:\alpha_i \neq 0} \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b\right) \qquad \textbf{sparse!} \end{aligned}$$

- Trick: $k(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$, i.e. **do not use Φ, but k!**

See e.g. Vapnik [1995], Müller et al. [2001], Schölkopf and Smola [2002] for details.

Common Kernels

Common kernels: [Vapnik, 1995, Müller et al., 2001, Schölkopf and Smola, 2002]

$$
\begin{aligned}
\text{Polynomial} \quad & k(\mathbf{x},\mathbf{y}) &&= (\langle \mathbf{x},\mathbf{y}\rangle + c)^d \\
\text{Sigmoid} \quad & k(\mathbf{x},\mathbf{y}) &&= \tanh(\kappa\langle \mathbf{x},\mathbf{y}\rangle + \theta) \\
\text{RBF} \quad & k(\mathbf{x},\mathbf{y}) &&= \exp\left(-\|\mathbf{x}-\mathbf{y}\|^2/(2\,\sigma^2)\right) \\
\text{Convex combinations} \quad & k(\mathbf{x},\mathbf{y}) &&= \beta_1 k_1(\mathbf{x},\mathbf{y}) + \beta_2 k_2(\mathbf{x},\mathbf{y}) \\
\text{Normalization} \quad & k(\mathbf{x},\mathbf{y}) &&= \frac{k'(\mathbf{x},\mathbf{y})}{\sqrt{k'(\mathbf{x},\mathbf{x})k'(\mathbf{y},\mathbf{y})}}
\end{aligned}
$$

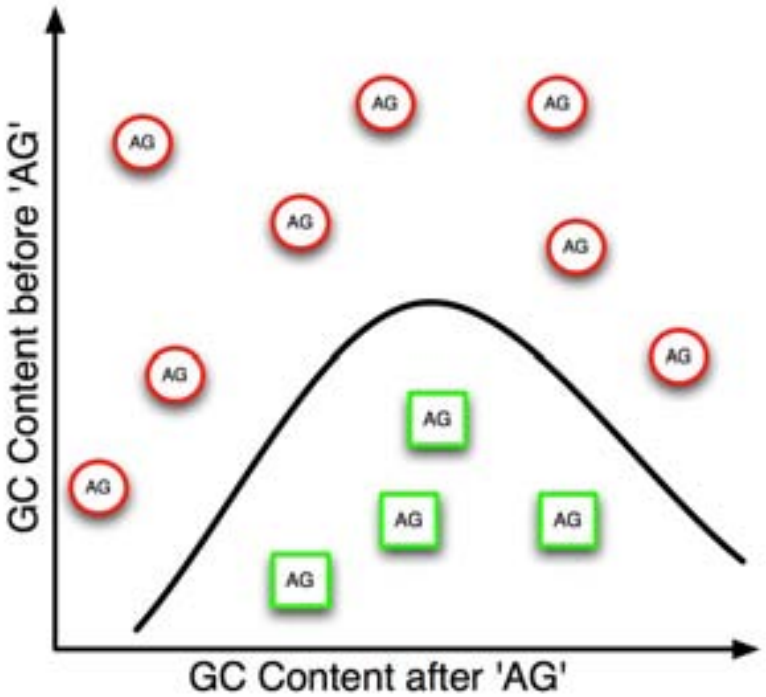
Recognition of Splice Sites

- Given: Potential acceptor splice sites

```
AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT
CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC
TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC
```

intron **exon**

- Goal: Rule that distinguishes true from false ones



More realistic problem!?

- Not linearly separable!
- Need nonlinear separation!?
- **Need more features!?**

  ⇒ **Need new kernel!**

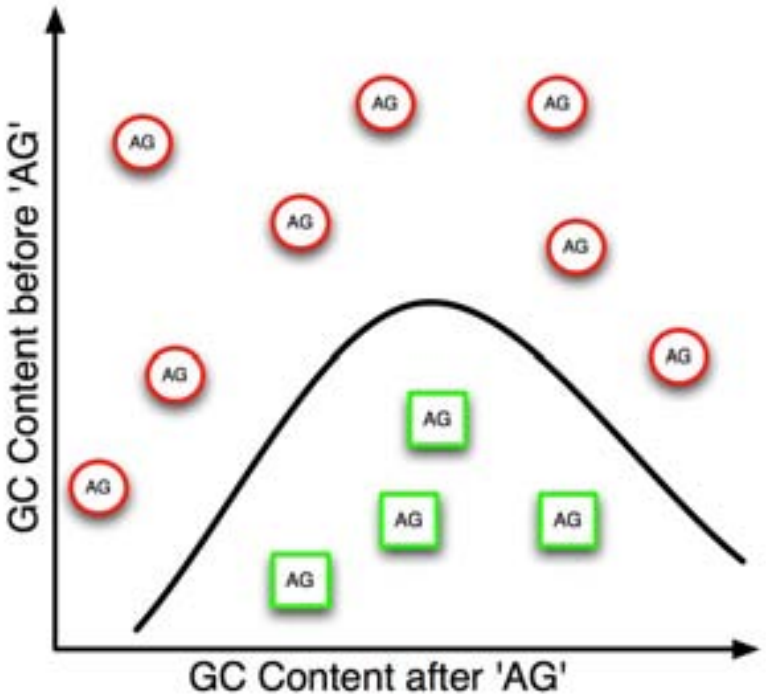
Recognition of Splice Sites

- Given: Potential acceptor splice sites

```
AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT
CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC
TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC
```

intron **exon**

- Goal: Rule that distinguishes true from false ones



More realistic problem!?

- Not linearly separable!
- Need nonlinear separation!?
- **Need more features!?**
  $\Rightarrow$ **Need new kernel!**

How to construct a kernel

At least two ways to get to a kernel

- Construct Φ and think about efficient ways to compute $\langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$
- Construct similarity measure, show positiveness and think about what it means

What is a good kernel?

- It should be mathematically valid (symmetric, pos. def.),
- fast to compute and
- adapted to the problem (yields good performance).

What can you do if kernel is not positive definite?

- Ignore it (but then optimization problem is not convex!)
- Add constant to diagonal (cheap)
- Exponentiate kernel matrix $\Rightarrow$ all eigenvalues become positive

How to construct a kernel

At least two ways to get to a kernel

- Construct Φ and think about efficient ways to compute $\langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$
- Construct similarity measure, show positiveness and think about what it means

What is a good kernel?

- It should be mathematically valid (symmetric, pos. def.),
- fast to compute and
- adapted to the problem (yields good performance).

What can you do if kernel is not positive definite?

- Ignore it (but then optimization problem is not convex!)
- Add constant to diagonal (cheap)
- Exponentiate kernel matrix $\Rightarrow$ all eigenvalues become positive

How to construct a kernel

At least two ways to get to a kernel

- Construct Φ and think about efficient ways to compute $\langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$
- Construct similarity measure, show positiveness and think about what it means

What is a good kernel?

- It should be mathematically valid (symmetric, pos. def.),
- fast to compute and
- adapted to the problem (yields good performance).

What can you do if kernel is not positive definite?

- Ignore it (but then optimization problem is not convex!)
- Add constant to diagonal (cheap)
- Exponentiate kernel matrix $\Rightarrow$ all eigenvalues become positive

More Features?

Some ideas:

- statistics for all four letters (or even dimer/codon usage),
- appearance of certain motifs,
- information content,
- secondary structure, ...

Approaches:

- Manually generate a few *strong* features
  - Requires background knowledge
  - Nonlinear decisions often beneficial
- Include many potentially useful *weak* features
  - Requires more training examples

Best in practice: Combination of both

More Features?

Some ideas:

- statistics for all four letters (or even dimer/codon usage),
- appearance of certain motifs,
- information content,
- secondary structure, ...

Approaches:

- Manually generate a few *strong* features
  - Requires background knowledge
  - Nonlinear decisions often beneficial
- Include many potentially useful *weak* features
  - Requires more training examples

Best in practice: Combination of both

Spectrum Kernel

- General idea [Leslie et al., 2002]
  - For each k-mer $s \in \Sigma^k$, the coordinate indexed by s will be the number of times s occurs in sequence x.
  - Then the k-spectrum feature map is

$$\Phi_k^{\text{Spectrum}}(\mathbf{x}) = (\phi_s(\mathbf{x}))_{s \in \Sigma^k}$$

  - Here $\phi_s(\mathbf{x})$ is the # occurrences of s in x.
  - The spectrum kernel is now the inner product in the feature space defined by this map:

$$k^{\text{Spectrum}}(\mathbf{x}, \mathbf{x}') = \langle \Phi_k^{\text{Spectrum}}(\mathbf{x}), \Phi_k^{\text{Spectrum}}(\mathbf{x}') \rangle$$

  - Dimensionality: Exponential in k: $|\Sigma|^k$

Spectrum Kernel

- Principle
  - Spectrum kernel: Count shared k-mers

  - $\Phi(\mathbf{x})$ has only very few non-zero dimensions
    $\Rightarrow$ Efficient kernel computations possible

Substring Kernels

- General idea
  - Count common substrings in two strings
  - Sequences are deemed the more similar, the more common substrings they contain
- Variations
  - Allow for gaps
  - (Include wildcards)
  - Allow for mismatches
  - (Include substitutions)
  - Motif Kernels
  - (Assign weights to substrings)

Gappy Kernel

- General idea [Lodhi et al., 2002, Leslie and Kuang, 2004]
  - Allow for gaps in common substrings
    → "subsequences"
  - A g-mer then contributes to all its k-mer subsequences

$$\phi^{\text{Gap}}_{(g,k)}(s) = (\phi_\beta(s))_{\beta\in\Sigma^k}$$

  - For a sequence x of any length, the map is then extended as

$$\phi^{\text{Gap}}_{(g,k)}(\mathbf{x}) = \sum_{g\text{-mers } s \text{ in } \mathbf{x}} (\phi^{\text{Gap}}_{(g,k)}(s))$$

  - The gappy kernel is now the inner product in feature space defined by:

$$k^{\text{Gap}}_{(g,k)}(\mathbf{x},\mathbf{x}') = \left\langle \Phi^{\text{Gap}}_{(g,k)}(\mathbf{x}), \Phi^{\text{Gap}}_{(g,k)}(\mathbf{x}') \right\rangle$$

  - Related to the *Locality improved kernel* [Zien et al., 2000]

Gappy Kernel

- Principle
  - Gappy kernel: Count common k-subsequences of g-mers

g=3, l=2

Protein A: ILVFMC

ILV: IL, IV, LV
LVF: LV, VF, LF
VFM: VF, FM, VM
FMC: FM, FC, MC

Protein B: WLVFQC

WLV: WL, LV, WV
LVF: LV, VF, LF
VFQ: VF, FQ, VQ
FQC: FQ, QC, FC

Mismatch Kernel

- General idea [Leslie et al., 2003]
  - Do not enforce strictly exact matches
  - Define mismatch neighborhood of k-mer s with up to m mismatches:

$$\phi^{\text{Mismatch}}_{(l,m)}(s) = (\phi_\beta(s))_{\beta \in \Sigma^k}$$

  - For a sequence x of any length, the map is then extended as

$$\phi^{\text{Mismatch}}_{(l,m)}(\mathbf{x}) = \sum_{k\text{-mers } s \text{ in } \mathbf{x}} (\phi^{\text{Mismatch}}_{(l,m)}(s))$$

  - The mismatch kernel is now the inner product in feature space defined by:

$$k^{\text{Mismatch}}_{(l,m)}(\mathbf{x}, \mathbf{x}') = \left\langle \Phi^{\text{Mismatch}}_{(l,m)}(\mathbf{x}), \Phi^{\text{Mismatch}}_{(l,m)}(\mathbf{x}') \right\rangle$$

Mismatch Kernel

- Principle
  - Mismatch kernel: Count common k-mers with max. m mismatches

Protein A: ILVFMC

Protein B: WLVFQC

Common 3-mers
with max. 1 mismatch:

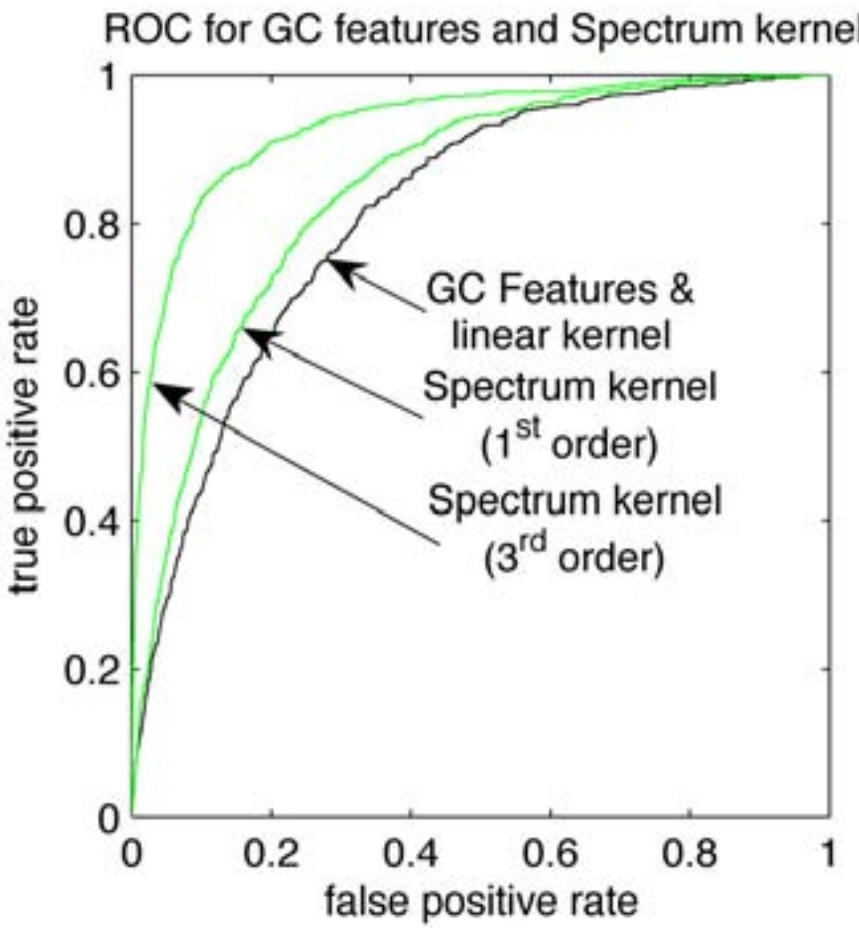
ILV - WLV
LVF - LVF
VFM - VFQ
FMC - FQC

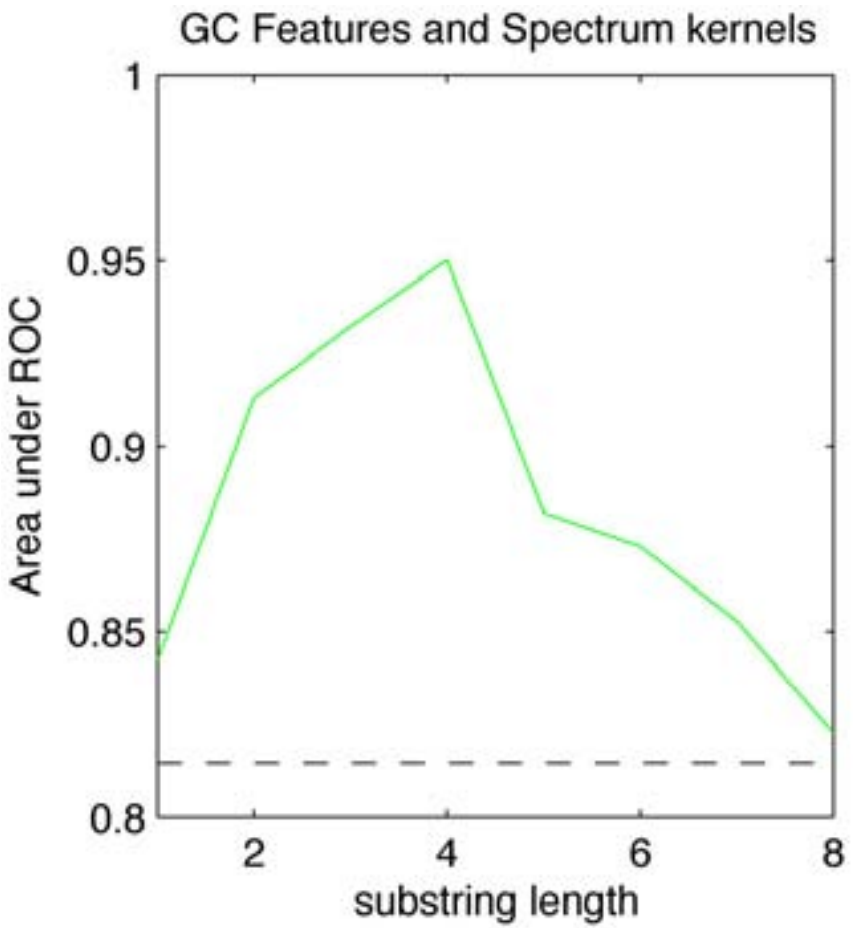
Motif kernels

- General idea [Logan et al., 2001]
  - Conserved motifs in sequences indicate structural and functional characteristics
  - Model sequence as feature vector representing motifs
  - i-th vector component is 1 $\Leftrightarrow$ $\mathbf{x}$ contains i-th motif
- Motif databases
  - Protein: Pfam, PROSITE, ...
  - DNA: Transfac, Jaspar, ...
  - RNA: Rfam, Structures, Regulatory sequences, ...
- Generated by
  - manual construction/prior knowledge
  - multiple sequence alignment (do not use test set!)

Simulation Example (Acceptor Splice Sites)

- Linear Kernel on GC-content features
- Spectrum kernel $k_k^{\text{Spectrum}}(\mathbf{x}, \mathbf{x}')$





Position Dependence

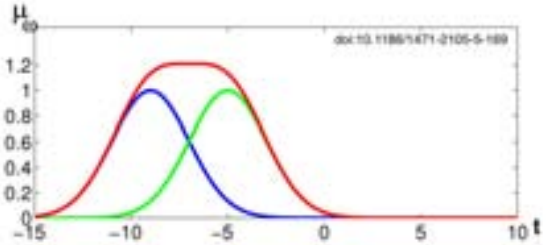
- Given: Potential acceptor splice sites

- Goal: Rule that distinguishes true from false ones
- Position of motif is important ('T' rich just before 'AG')
- Spectrum kernel is *blind* w.r.t. positions
- New kernels for sequences with constant length
  - Substring kernel per position (sum over positions)
    - Oligo kernel
    - Weighted Degree kernel
  - Can detect motifs at specific positions
    - weak if positions vary
    - Extension: allow "shifting"

Oligo Kernel [Meinicke et al., 2004]

- Consider k-mers in fixed length sequences $s \in \Sigma^L$
- Define **oligo functions** for every k-mer ω at positions S_ω

$$\mu_\omega(t) := \sum_{p \in S_\omega} \exp\left(-\frac{(t-p)^2}{2\sigma^2}\right)$$



- Feature space defined by concatenation: $\Phi(s) = (\mu_{\omega_1}, \ldots, \mu_{\omega_m})^T$
- **Oligo kernel** defined by scalar product:

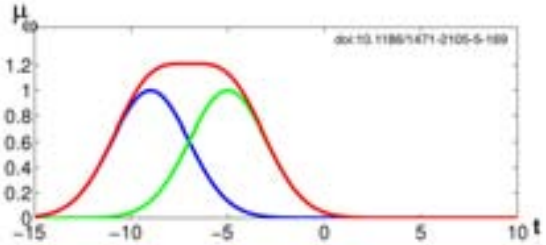
$$\langle \phi(s_i), \phi(s_j) \rangle = \sum_{\omega \in \Sigma^k} \sum_{p \in S_\omega^i} \sum_{q \in S_\omega^j} \int e^{\left(-\frac{(t-p)^2}{2\sigma^2}\right)} e^{\left(-\frac{(t-q)^2}{2\sigma^2}\right)} dt$$

$$= \sqrt{\pi}\sigma \sum_{\omega \in \Sigma^k} \sum_{p \in S_\omega^i} \sum_{q \in S_\omega^j} \exp\left(-\frac{(q-p)^2}{4\sigma^2}\right)$$

Oligo Kernel [Meinicke et al., 2004]

- Consider k-mers in fixed length sequences $s \in \Sigma^L$
- Define **oligo functions** for every k-mer ω at positions S_ω

$$\mu_\omega(t) := \sum_{p \in S_\omega} \exp\left(-\frac{(t-p)^2}{2\sigma^2}\right)$$



- Feature space defined by concatenation: $\Phi(s) = (\mu_{\omega_1}, \ldots, \mu_{\omega_m})^T$
- **Oligo kernel** defined by scalar product:

$$\begin{aligned}\langle \phi(s_i), \phi(s_j) \rangle &= \sum_{\omega \in \Sigma^k} \sum_{p \in S_\omega^i} \sum_{q \in S_\omega^j} \int \mathrm{e}^{\left(-\frac{(t-p)^2}{2\sigma^2}\right)} \mathrm{e}^{\left(-\frac{(t-q)^2}{2\sigma^2}\right)} dt \\ &= \sqrt{\pi}\sigma \sum_{\omega \in \Sigma^k} \sum_{p \in S_\omega^i} \sum_{q \in S_\omega^j} \exp\left(-\frac{(q-p)^2}{4\sigma^2}\right)\end{aligned}$$

Weighted Degree Kernel [Rätsch and Sonnenburg, 2004]

- Equivalent to a mixture of spectrum kernels (up to order K) at every position for appropriately chosen β's:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{K} \sum_{l=1}^{L-k+1} \beta_k k_k^{\text{Spectrum}}(\mathbf{u}_{l:l+k}(\mathbf{x}_i), \mathbf{u}_{l:l+k}(\mathbf{x}_j))$$

  where $\beta_k = \frac{K-k+1}{\sum_k (K-k+1)} = 2\frac{K-k+1}{k\cdot(k+1)}$.

- Can be equivalently computed by

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

  for appropriately chosen β_k.

- Equivalent to oligo kernel for $\sigma \longrightarrow 0$

Weighted Degree Kernel with Shifts [Rätsch et al., 2005]

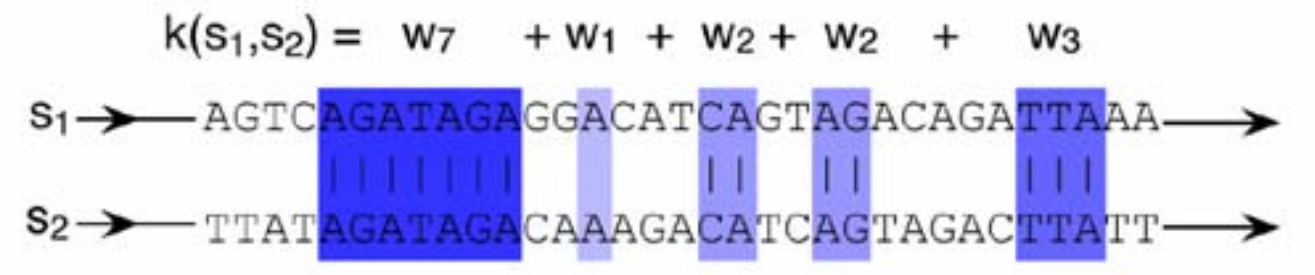
- Weighted Degree kernel is inefficient if motif position varies
  - Motif may not have appeared exactly at a specific position
  - Needs many examples to sample all positions
- Idea: Shift sequences against each other, i.e.

$$k_{WDS}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{s=0}^{S} \kappa_s \left(k_{WD}(\mathbf{u}_{1:L-s}(\mathbf{x}_i), \mathbf{u}_{1+s:L}(\mathbf{x}_j)) + \right.$$
$$\left. + k_{WD}(\mathbf{u}_{1+s:L}(\mathbf{x}_i), \mathbf{u}_{1:L-s}(\mathbf{x}_j)) \right)$$

- Choose waiting $\kappa_s = 1/(2(s+1))$: Larger deviations in position get lower weights
- Shifting may depend on sequence position [Rätsch et al., 2005]

Weighted Degree Kernel Block Formulation

- Without shifts: Compare two sequences by identifying the largest matching blocks:

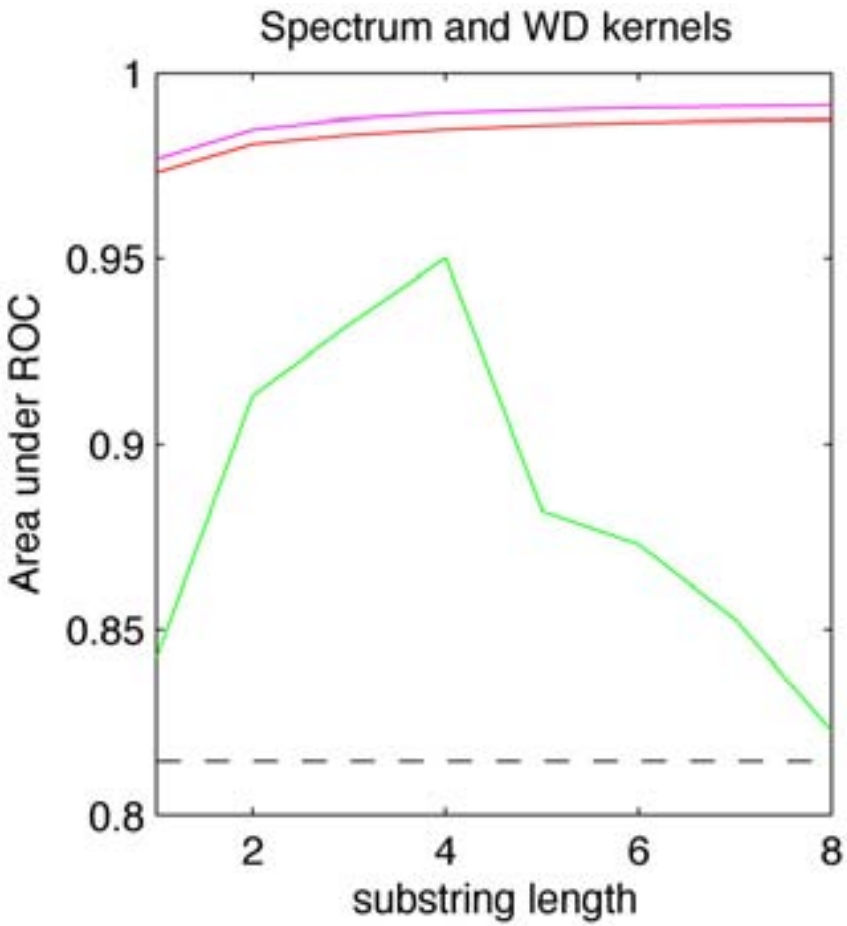


where a matching block of length k implies many shorter matches:

$$w_k = \sum_{j=1}^{\min(k,K)} \beta_j \cdot (k - j + 1).$$

- With shifts: Allows matching subsequences with offsets

Substring Kernel Comparison

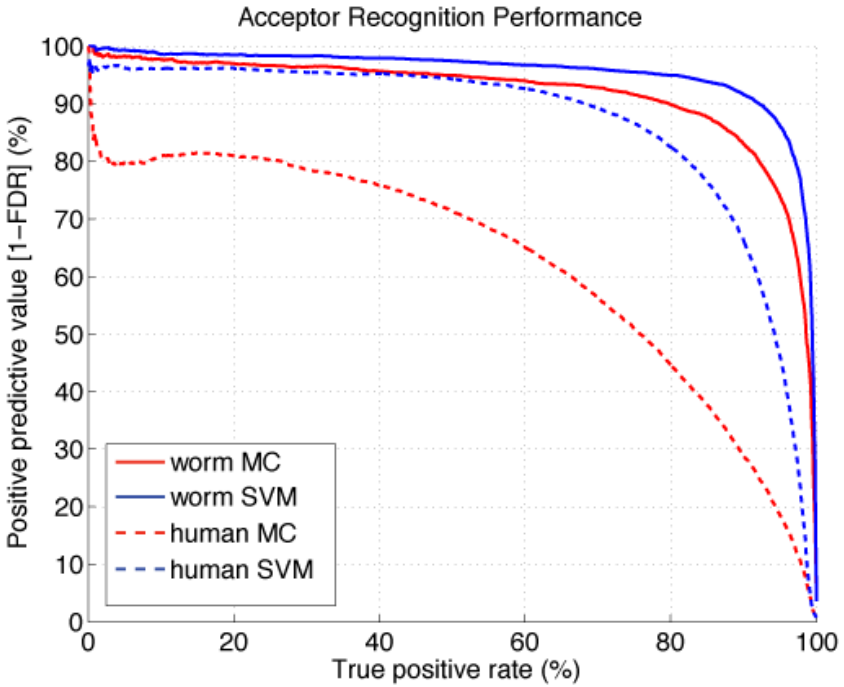


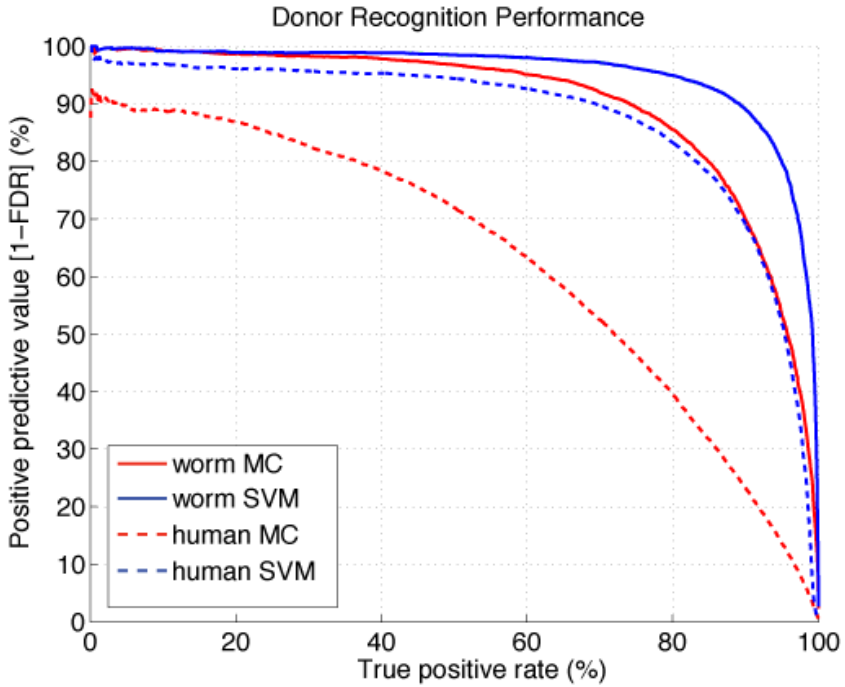
- Linear kernel on GC-content features
- Spectrum kernel
- Weighted degree kernel
- Weighted degree kernel with shifts

Remark: Higher order substring kernels typically exploit that correlations appear locally and not between arbitrary parts of the sequence (other than e.g. the polynomial kernel).

Example: SVMs for Splice Site Recognition

	Worm		Fly		Cress		Fish		Human	
	Acc	**Don**	**Acc**	**Don**	**Acc**	**Don**	**Acc**	**Don**	**Acc**	**Don**
Markov Chain										
auROC(%)	99.62	99.55	98.78	99.12	99.12	99.44	98.98	99.19	96.03	97.78
auPRC(%)	92.09	89.98	80.27	78.47	87.43	88.23	63.59	62.91	16.20	24.98
WD-SVM										
auROC(%)	99.80	99.82	99.12	99.51	99.43	99.68	99.38	99.61	97.86	98.57
auPRC(%)	95.89	95.34	86.67	87.47	92.16	92.88	86.58	86.94	54.42	56.86
WDS-SVM										
auROC(%)	99.80	99.82	99.12	99.51	99.43	99.68	99.38	99.61	97.86	*98.57*
auPRC(%)	95.89	95.34	86.67	87.47	92.16	92.88	86.58	86.94	54.42	*56.86*





Simulation Example

- Generate 100 examples (50 positives/50 negatives)
- Place motif "AAAA" at central position
- Train SVM with
  - Spectrum kernel
  - WD kernel
  - WDS kernel
- Observe generalization performance (auROC)
- Use varying motif positions – What do we expect?
- Retrain and observe generalization performance (auROC)

The scripts can be downloaded from the Wiki pages. It uses the Shogun toolbox available at `http://www.shogun-toolbox.org`.

Simulation Example

- Generate 100 examples (50 positives/50 negatives)
- Place motif "AAAA" at central position
- Train SVM with
  - Spectrum kernel
  - WD kernel
  - WDS kernel
- Observe generalization performance (auROC)
- Use varying motif positions – What do we expect?
- Retrain and observe generalization performance (auROC)

The scripts can be downloaded from the Wiki pages. It uses the Shogun toolbox available at `http://www.shogun-toolbox.org`.

Simulation Example

- Generate 100 examples (50 positives/50 negatives)
- Place motif "AAAA" at central position
- Train SVM with
  - Spectrum kernel
  - WD kernel
  - WDS kernel
- Observe generalization performance (auROC)
- Use varying motif positions – What do we expect?
- Retrain and observe generalization performance (auROC)

The scripts can be downloaded from the Wiki pages. It uses the Shogun toolbox available at `http://www.shogun-toolbox.org`.

Simulation Example

- Generate 100 examples (50 positives/50 negatives)
- Place motif "AAAA" at central position
- Train SVM with
  - Spectrum kernel
  - WD kernel
  - WDS kernel
- Observe generalization performance (auROC)
- Use varying motif positions – What do we expect?
- Retrain and observe generalization performance (auROC)

The scripts can be downloaded from the Wiki pages. It uses the Shogun toolbox available at `http://www.shogun-toolbox.org`.

Simulation Example

- Generate 100 examples (50 positives/50 negatives)
- Place motif "AAAA" at central position
- Train SVM with
  - Spectrum kernel
  - WD kernel
  - WDS kernel
- Observe generalization performance (auROC)
- Use varying motif positions – What do we expect?
- Retrain and observe generalization performance (auROC)

The scripts can be downloaded from the Wiki pages. It uses the Shogun toolbox available at `http://www.shogun-toolbox.org`.

Simulation Example

- Generate 100 examples (50 positives/50 negatives)
- Place motif "AAAA" at central position
- Train SVM with
  - Spectrum kernel
  - WD kernel
  - WDS kernel
- Observe generalization performance (auROC)
- Use varying motif positions – What do we expect?
- Retrain and observe generalization performance (auROC)

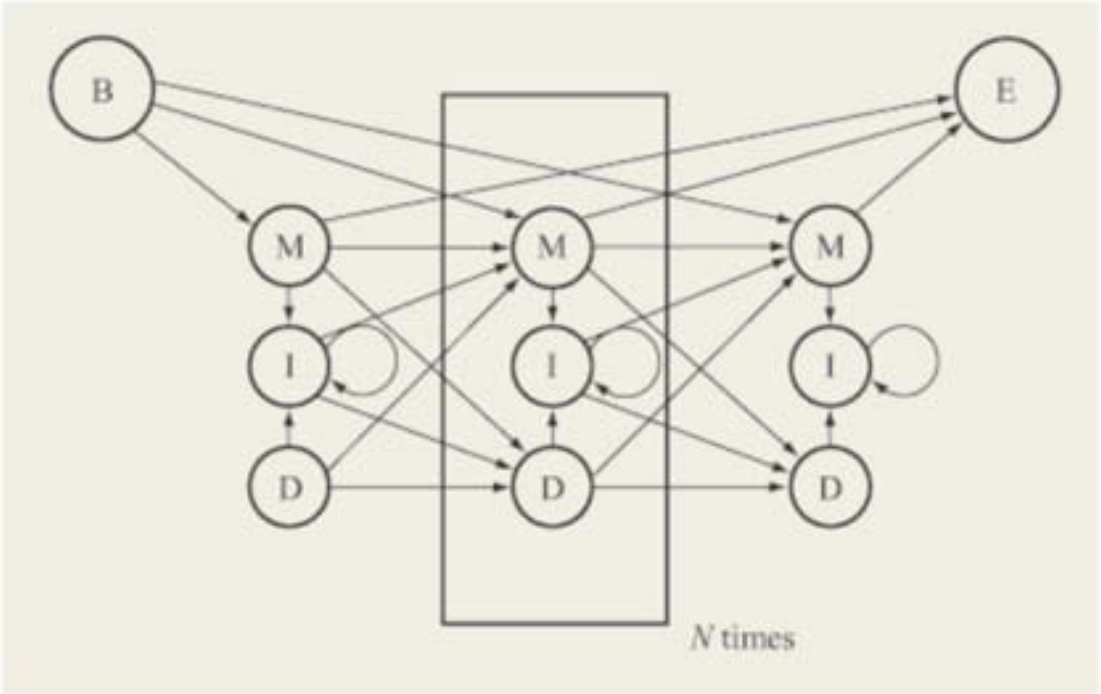
The scripts can be downloaded from the Wiki pages. It uses the Shogun toolbox available at `http://www.shogun-toolbox.org`.

Kernel Engineering

- Define a (possibly high-dimensional) feature space of interest
  - Spectrum, mismatch, substring kernels
  - Position specific kernels
  - Physico-chemical kernels
- Derive a kernel from a generative model
  - Fisher kernel
  - Mutual information kernel
  - Marginalized kernel
- Derive a kernel from a similarity measure
  - Pairwise alignments
  - Local alignment kernel

Probabilistic models for sequences

Probabilistic modeling of biological sequences is older than kernel designs. Important models include Markov Models (MM), Hidden Markov Models (HMM) and Stochastic Context Free Grammars (SCFG)



Parametric model is a family of distributions:

$$\{P_\theta \mid \theta \in \mathbf{R}^m\}$$

Simple Example: PSSM

PSSM = Position Specific Scoring Matrices

- Identical to zeroth order Markov Chains
- Fixed length sequences $s \in \Sigma^N$, assume independence of positions
- One parameter per letter and position:

$$P_\theta(s) = \prod_{i=1}^{N} \theta_{i,s_i}$$

- "Training" = Counting:
  - Training set $s^1, \ldots, s^N$
  - $\theta_{i,\sigma} = \frac{1}{N} \sum_{n=1}^{N} I(s_i^n = \sigma)$
- Estimate PSSM for positive and negative class separately
- Use logodds ratio for discrimination

$$\log\left(\frac{P_{\theta^+}(s)}{P_{\theta^-}(s)}\right)$$

Fisher Kernel

- Fix a parameter $\theta_0 \in \mathbf{R}^m$ (e.g., by maximum likelihood over a training set of sequences)
- For each sequence x, compute the Fisher score vector:

$$\Phi_{\theta_0}(x) = \nabla \log P_\theta(x)|_{\theta=\theta_0}$$

- Form the kernel [Jaakkola et al., 2000]

$$K(x, x') = \Phi_{\theta_0}(x)^\top I(\theta_0)^{-1} \Phi_{\theta_0}(\mathbf{x}')$$

where $I(\theta_0) = E_{\theta_0}[\Phi_{\theta_0}(x)\Phi_{\theta_0}(x)^\top]$ is the Fisher information matrix.

Fisher Kernel Properties

- The Fisher score describes how each parameter contributes to the process of generating a particular example
- The Fisher kernel is invariant under change of parametrization of the model
- A kernel classifier employing the Fisher kernel derived from a model that contains the label as a latent variable is, asymptotically, at least as good a classifier as the MAP labelling based on the model (under several assumptions, Tsuda et al. [2002a]).

Fisher Kernel in Practice

- $\Phi_{\theta_0}(x)$ can be computed explicitly for many models (e.g., HMMs)
- $I(\theta_0)$ is often replaced by the identity matrix
- Several different models (i.e., different θ_0) can be trained and combined
- Feature vectors are explicitly computed

Example: Fisher Kernel on PSSMs

- Fixed length sequences $s \in \Sigma^N$
- PSSMs: $p(s|\boldsymbol{\theta}) = \prod_{i=1}^{N} \theta_{i,s_i}$
- Fisher scores features: $(\Phi(s))_{i,\sigma} = \frac{\mathrm{d}p(s|\boldsymbol{\theta})}{\mathrm{d}\theta_{i,\sigma}} = \mathrm{Id}(s_i = \sigma)$
- Kernel: $k(s, s') = \langle \Phi(s), \Phi(s') \rangle = \sum_{i=1}^{N} \mathrm{Id}(s_i = s'_i)$
- Identical to WD kernel with order 1

Note: Marginalized-count kernels [Tsuda et al., 2002b] can be understood as a generalization of Fisher kernels.

Pairwise comparison kernels

- General idea [Liao and Noble, 2002]
  - Employ empirical kernel map on Smith-Waterman/Blast scores
- Advantage
  - Utilizes decades of practical experience with Blast
- Disadvantage
  - High computational cost ($O(N^3)$)
- Alleviation
  - Employ Blast instead of Smith-Waterman
  - Use a smaller subset for empirical map

Local Alignment Kernel

In order to compute the score of an alignment, one needs

- substitution matrix $S \in \mathbf{R}^{\Sigma \times \Sigma}$
- gap penalty $g : \mathbf{N} \rightarrow \mathbf{R}$

An alignment π is then scored as follows:

```
CGGSLIAMM----WFGV
|...|||||....||||
C---LIVMMNRLMWFGV
```

$$s_{S,g}(\pi) = S(C,C) + S(L,L) + S(I,I) + S(A,V) + 2S(M,M)$$
$$+S(W,W) + S(F,F) + S(G,G) + S(V,V) - g(3) - g(4)$$

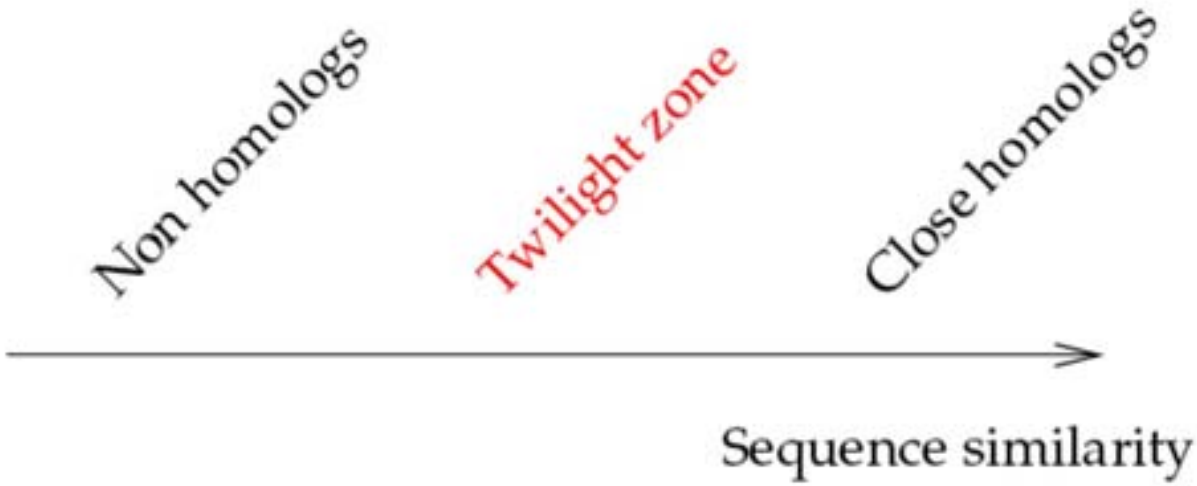
Smith-Waterman score (not positive definite)

$$SW_{S,g}(\mathbf{x},\mathbf{y}) := \max_{\pi \in \Pi(\mathbf{x},\mathbf{y})} s_{S,g}(\pi)$$

Local Alignment Kernel [Vert et al., 2004]

$$K^{\beta}(\mathbf{x},\mathbf{y}) = \sum_{\pi \in \Pi(\mathbf{x},\mathbf{y})} \exp(\beta s_{S,g}(\pi))$$

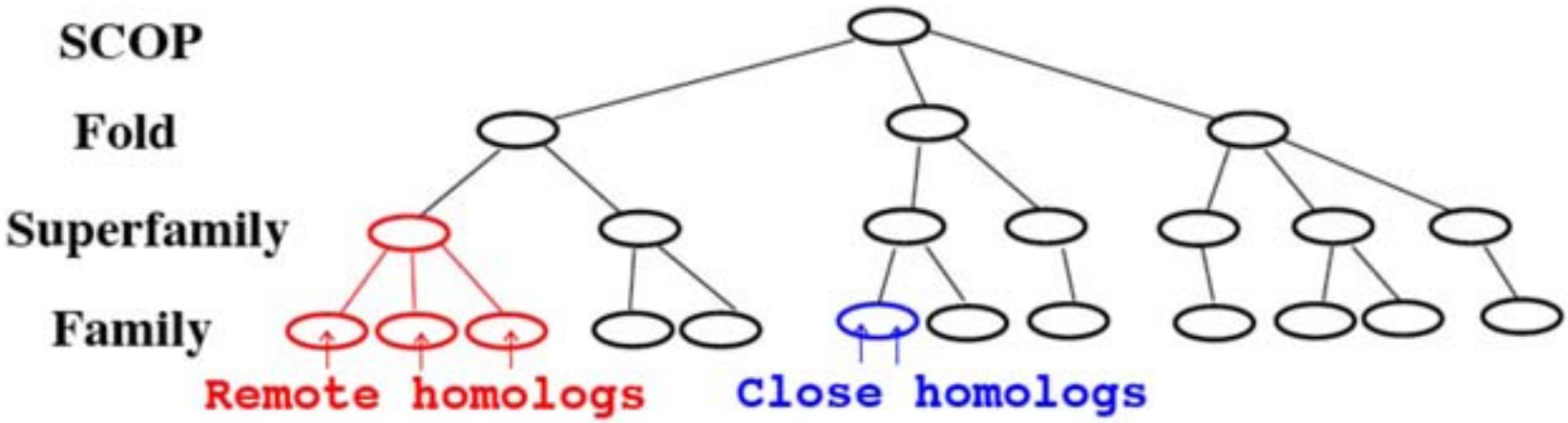
Example: Remote Homology



- Homologs have common ancestors
- Structures and functions are more conserved than sequences
- Remote homologs can not easily be detected by direct sequence comparison

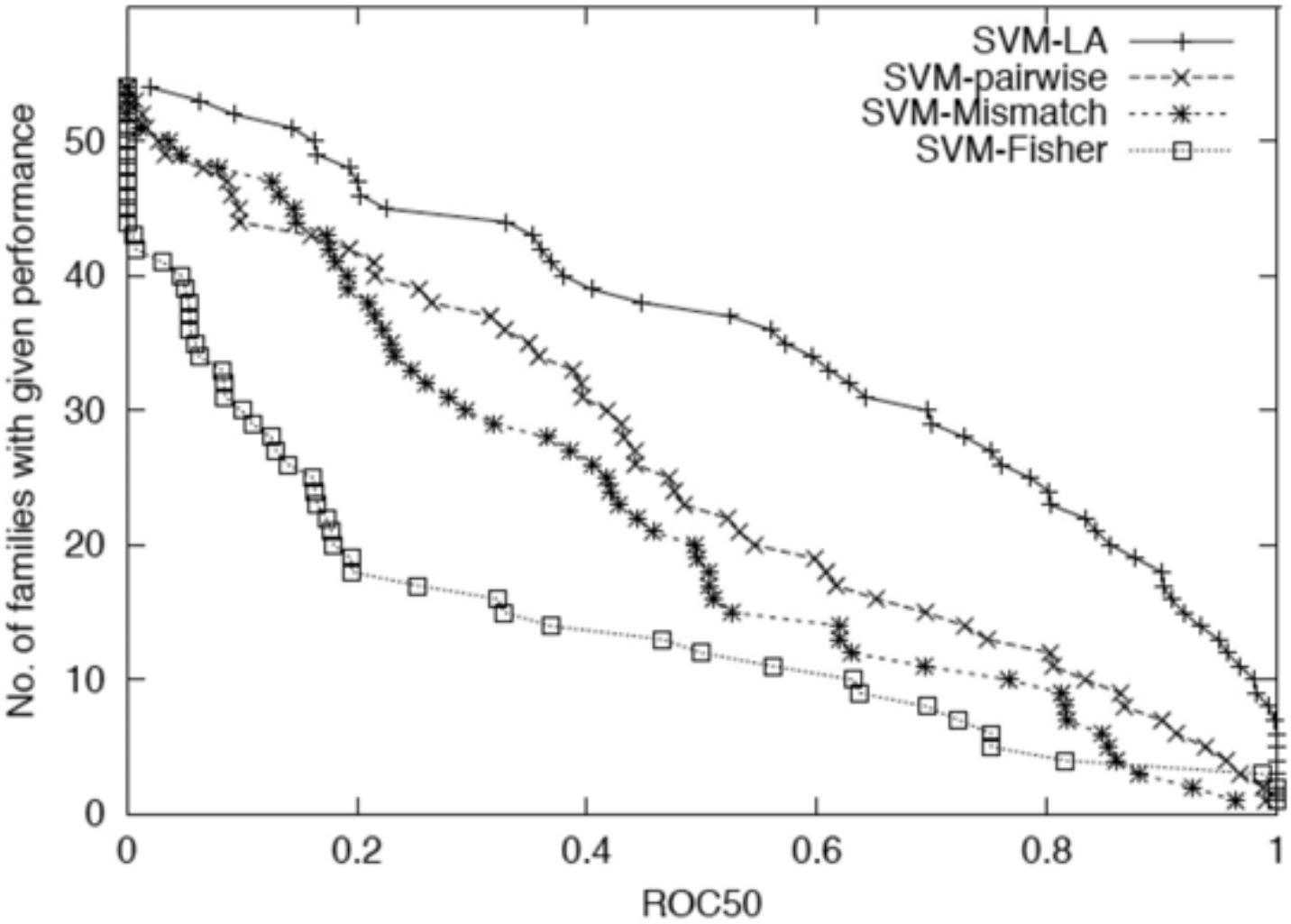
(Thanks to J.-P. Vert for providing the slides on remote homology detection.)

SCOP Database & Experiment



- Goal: Recognize the superfamily
- Training: for a sequence, positive examples come from the same superfamily, but different family. Negative examples come from other superfamilies
- Test: Predict the superfamily

Difference in Performance



- Performance on SCOP superfamily benchmark [Vert et al., 2004]
- ROC50 is the area under the ROC curve up to the first 50 FPs

Kernel Summary

- Kernel extend SVMs to nonlinear decision boundaries, while keeping the simplicity of linear classification
- Good kernel design is important for every single data analysis task
- String kernels perform computations in very high dimensional feature space
- Kernels on strings can be:
  - Substring kernels (e.g. Spectrum & WD kernel)
  - Based on probabilistic methods (e.g. Fisher Kernel)
  - Derived from similarity measures (e.g. Alignment kernels)
- Not mentioned: Kernels on graphs, images, structures
- Application goes far beyond computational biology

Part II: Sequence Analysis with SVMs

Part II: Sequence Analysis with SVMs

- String kernels
- **Large Scale Data Structures**
- Heterogeneous Data

NB: Large parts are from the tutorial at the German Conference for Bioinformatics 2006 with Cheng Soon Ong.

Spectrum Kernel

- General idea [Leslie et al., 2002]
  - For each k-mer $s \in \Sigma^k$, the coordinate indexed by s will be the number of times s occurs in sequence x.
  - Then the k-spectrum feature map is

$$\Phi_k^{\text{Spectrum}}(\mathbf{x}) = (\phi_s(\mathbf{x}))_{s \in \Sigma^k}$$

  - Here $\phi_s(\mathbf{x})$ is the # occurrences of s in x.
  - The spectrum kernel is now the inner product in the feature space defined by this map:

$$k^{\text{Spectrum}}(\mathbf{x}, \mathbf{x}') = \langle \Phi_k^{\text{Spectrum}}(\mathbf{x}), \Phi_k^{\text{Spectrum}}(\mathbf{x}') \rangle$$

  - Dimensionality: exponential in k: $|\Sigma|^k$

Fast string kernels?

- Use index structures to speed up computation
  - Single kernel computation $k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$
  - Kernel (sub-)matrix $k(\mathbf{x}_i, \mathbf{x}_j), i \in I, j \in J$
  - Linear combination of kernel elements

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}_i, \mathbf{x}) = \left\langle \sum_{i=1}^{N} \alpha_i \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \right\rangle$$

- Idea: Exploit that $\Phi(\mathbf{x})$ and also $\sum_{i=1}^{N} \alpha_i \Phi(\mathbf{x}_i)$ is sparse:
  - Explicit maps
  - Sorted lists
  - (Suffix) trees/tries/arrays

Efficient data structures

- $\mathbf{v} = \Phi(\mathbf{x})$ is very sparse
- Computation with $\mathbf{v}$ requires efficient operations on single dimensions, e.g.

$$\texttt{lookup } v_s \quad \text{or } \texttt{update } v_s = v_s + \alpha$$

- Use trees or arrays to store only non-zero elements
  $\Rightarrow$ Substring is the index into the tree or array
- Leads to more efficient optimization algorithms:
  - Precompute $v = \sum_{i=1}^{N} \alpha_i \Phi(\mathbf{x}_i)$
  - Compute $\sum_{i=1}^{N} \alpha_i k(\mathbf{x}_i, \mathbf{x})$ by

$$\sum_{s \text{ substring in } \mathbf{x}} v_s$$

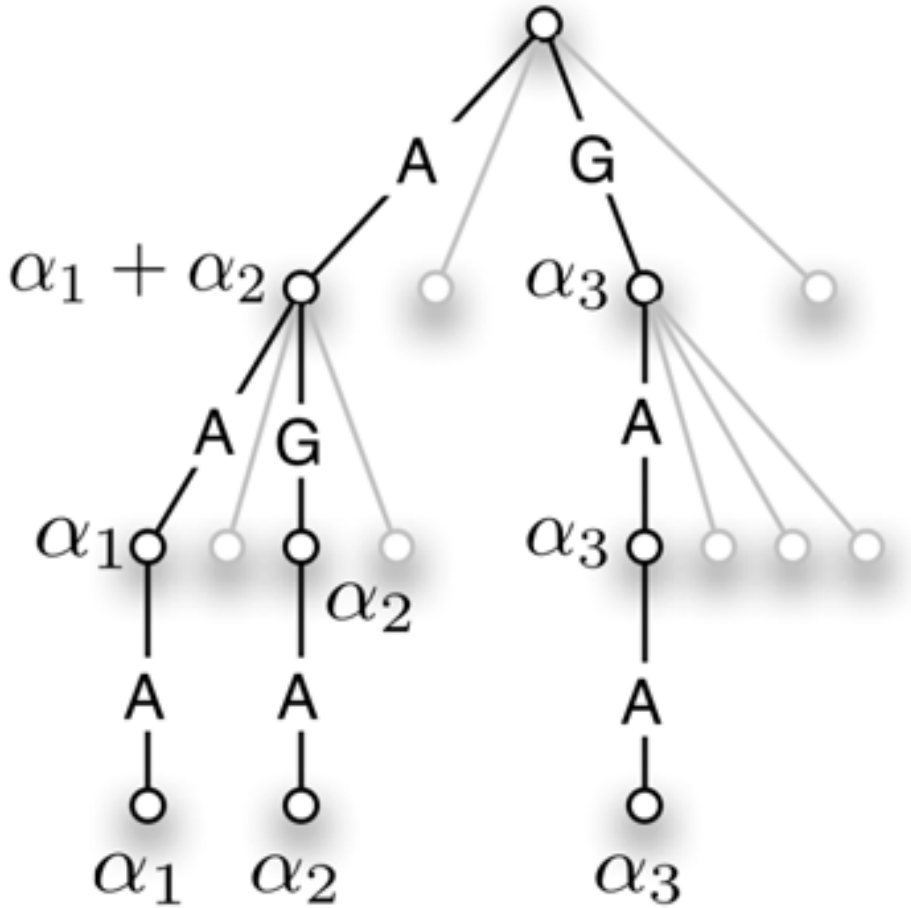
Explicit Maps

- Require $\mathcal{O}(|\Sigma|^k)$ memory
- Explicitly store $\mathbf{w} = \sum_i \alpha_i \Phi(\mathbf{x}_i)$
- `lookup` and `update` operations are $\mathcal{O}(1)$
- Updating all $f(\mathbf{x}_i)$ takes $\mathcal{O}(Q \cdot L \cdot k + N \cdot L \cdot k)$
- Very efficient, but only work for small k

Sorted Lists

- Generate a sorted list with pairs $(\mathbf{u}, \alpha)$ of length $Q \cdot L$
  - $\mathcal{O}(Q \cdot L \cdot \log(Q \cdot L))$
- Requires $\mathcal{O}(Q \cdot L \cdot k)$ memory
- Iterate trough list and k-mer list of example (pre-sorted)
  - identify co-occuring k-mers
- Single $f(\mathbf{x}_i)$ requires $\mathcal{O}((Q \cdot L \cdot \log(Q \cdot L) + L) \cdot k)$
- All $f(\mathbf{x}_i)$ require $\mathcal{O}((Q \cdot L \cdot \log(Q \cdot L) + N \cdot L \cdot \log(N \cdot L)) \cdot k)$
- Requires additional sorting of long lists
- Also works for large k

Example: Trees & Tries



Tree (trie) data structure stores sparse weightings on sequences (and their subsequences).

Illustration: Three sequences AAA, AGA, GAA were added to a trie (α's are the weights of the sequences).

- Building tree: $\mathcal{O}(Q \cdot L \cdot k)$
- Compute all $f(\mathbf{x}_i)$: $\mathcal{O}(N \cdot L \cdot k)$

Solving the SVM Dual

$$\begin{array}{ll} \text{maximize}_\alpha & \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \, k(x_i, x_j) \\ \text{s.t.} & \sum_{i=1}^N \alpha_i y_i = 0 \\ & 0 \leqslant \alpha_i \leqslant C \text{ for } i = 1, 2, \ldots, N. \end{array}$$

Requires N^2 kernel computations

- expensive to compute ($\mathcal{O}(k \cdot L \cdot N^2)$)
- expensive to store matrix ($\mathcal{O}(N^2)$)

Solving QP using interior point methods is expensive: $\mathcal{O}(N^3)$

Idea: Chunking based methods: Iterate

- Select small number of variables
- Optimize w.r.t. to these variables
- Stop if converged

Chunking

$$\begin{aligned} F(\boldsymbol{\alpha}) := \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & 0 \leqslant \alpha_i \leqslant C \text{ for } i = 1, 2, \ldots, N. \end{aligned}$$

- Select Q variables $i_1, \ldots, i_Q$
  - Random (inefficient)
  - Sequential (inefficient)
  - Heuristic selection motivated by KKT conditions
    - Requires $f(\mathbf{x}_j) = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}_i, \mathbf{x}_j)$ for all j
    - Points that have too small margin, but $\alpha_i < C$
    - Points that are outside margin area, but $\alpha_i > 0$
    - Points with $0 \leq \alpha_i \leq C$
- Solve QP of size Q ($\mathcal{O}(Q^3)$)
- Update $f(\mathbf{x}_j)$ if necessary

Chunking

- What do we need per iteration
  - Compute $f(\mathbf{x}_j) = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}_i, \mathbf{x}_j)$ for all j
  - Solve QP of size Q
- Complexity: $\mathcal{O}(Q \cdot N + Q^3)$
- First part very expensive for large N

- Can we speedup computing $f(\mathbf{x}_j)$?
- So far for string kernels: $\mathcal{O}(Q \cdot N \cdot L \cdot k)$
- With new data structures: $\mathcal{O}(N \cdot L \cdot k)$

Algorithm

INITIALIZATION

$f_i = 0$, $\alpha_i = 0$ for $i = 1, \ldots, N$

LOOP UNTIL CONVERGENCE

`For` $t = 1, 2, \ldots$

Check optimality conditions and stop if optimal

Select working set W based on $\mathbf{g}$ and $\boldsymbol{\alpha}$, store $\boldsymbol{\alpha}^{old} = \boldsymbol{\alpha}$

Solve reduced QP and update $\boldsymbol{\alpha}$

`clear` $\mathbf{w}$

$\mathbf{w} \leftarrow \mathbf{w} + (\alpha_j - \alpha_j^{old}) y_j \Phi(\mathbf{x}_j)$ for all $j \in W$

Update $f_i = f_i + \langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle$ for all $i = 1, \ldots, N$

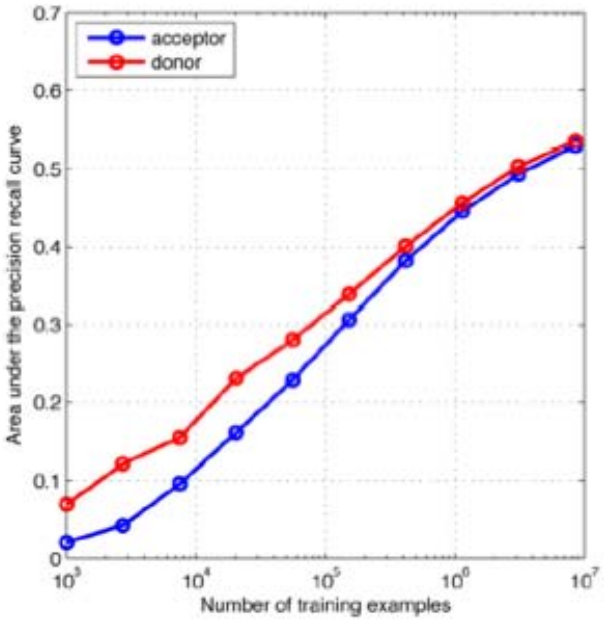
See Sonnenburg et al. [2007a] for more details.

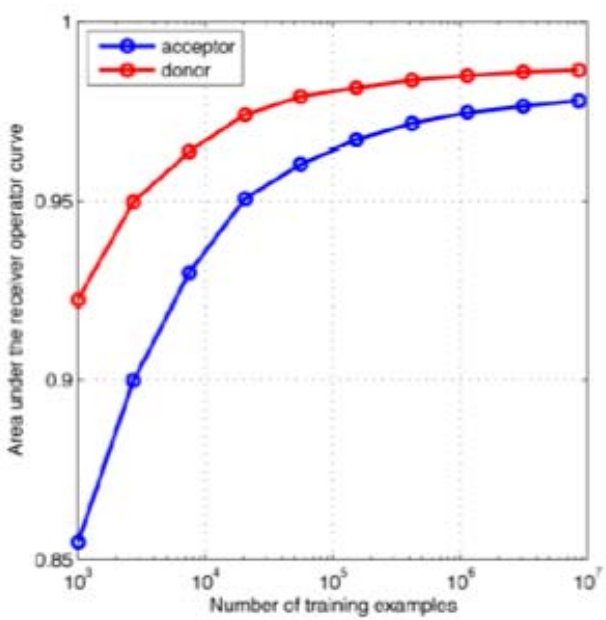
All implemented in Shogun toolbox (`http://www.shogun-toolbox.org`)

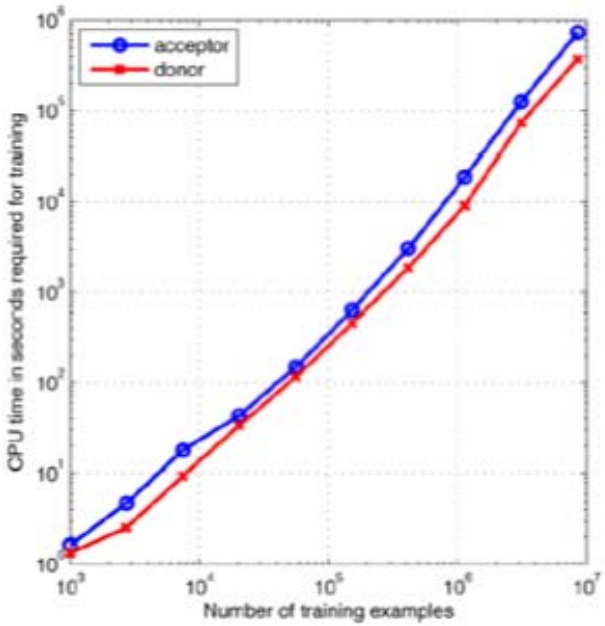
Results with WD Kernel (Human Acceptors)

N	Computing time (s)		ROC (%)
	WD	*WD w/ tries*	
500	17	83	75.61
1,000	17	83	79.70
5,000	28	105	90.38
10,000	47	134	92.79
30,000	195	266	94.73
50,000	441	389	95.48
100,000	1,794	740	96.13
500,000	31,320	7,757	96.93
1,000,000	102,384	26,190	97.20
2,000,000	-	(115,944)	97.36
5,000,000	-	(764,144)	97.52
10,000,000	-	(2,825,816)	97.64
10,000,000	PSSMs		96.03

Result with WD Kernel (Human Splice Sites)







Part II: Sequence Analysis with SVMs

Part II: Sequence Analysis with SVMs

- String kernels
- Large Scale Data Structures
- **Heterogeneous Data**

NB: Large parts are from the tutorial at the German Conference for Bioinformatics 2006 with Cheng Soon Ong.

Heterogeneous Data

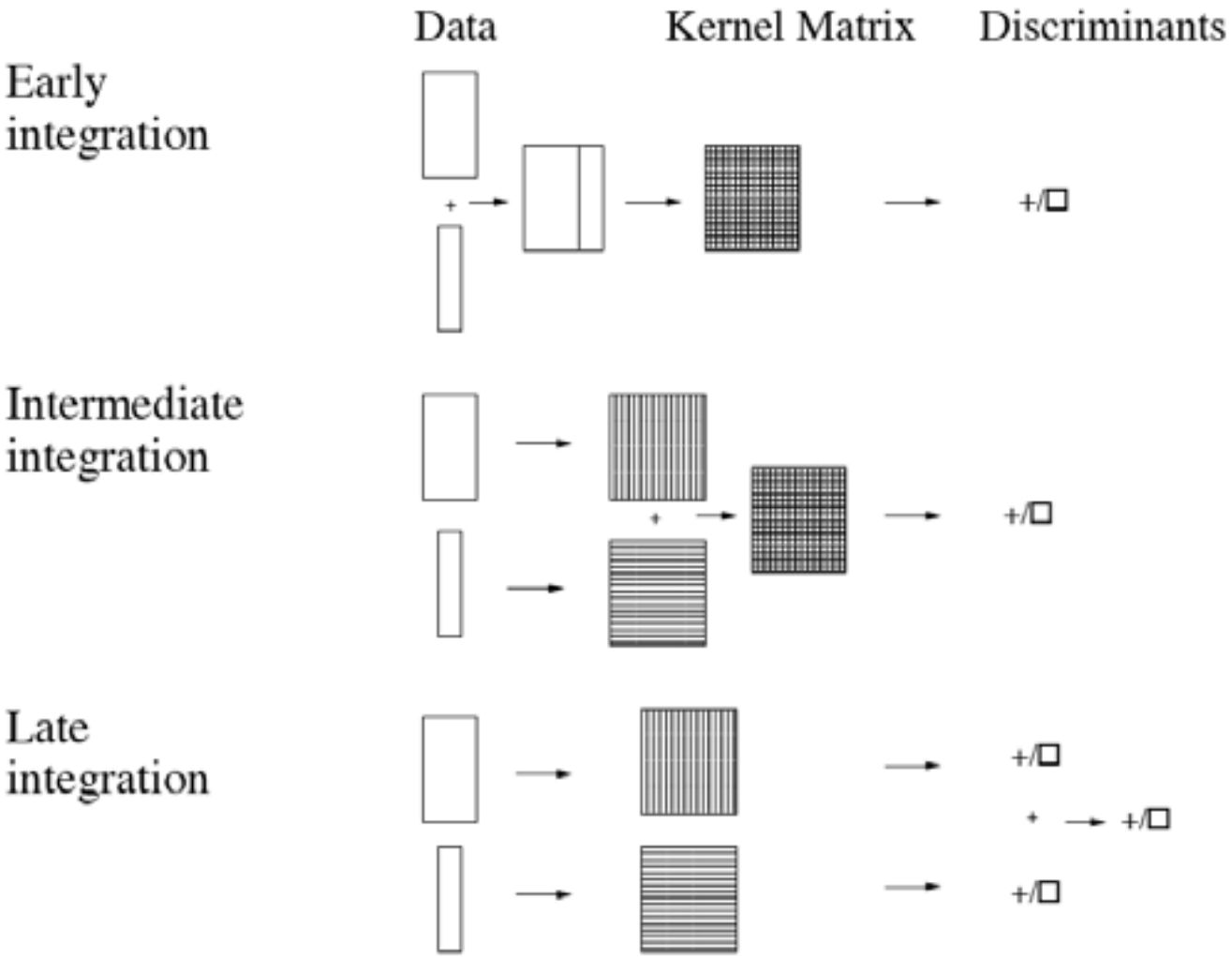
Heterogeneous Data about the same object:

- Different data sources
- Different data types
- Each may require a specific kernel

Heterogeneous Data appears, e.g. when

- Classifying proteins/genes (e.g. Lanckriet et al. [2004], Zien and Ong [2007a])
  - Using similarity to other proteins
  - Gene expression levels
  - Network information [Vert and Kanehisa, 2003]
  - Phylogenetic information
- Can be incomplete (cf. [Tsuda et al., 2003])

Data Fusion



from [Noble, 2004]

Multiple Kernel Learning (MKL)

- Possible solution We can add the two kernels, that is

$$k(\mathbf{x}, \mathbf{x}') := k_{sequence}(\mathbf{x}, \mathbf{x}') + k_{structure}(\mathbf{x}, \mathbf{x}').$$

- Better solution We can mix the two kernels,

$$k(\mathbf{x}, \mathbf{x}') := (1 - t)k_{sequence}(\mathbf{x}, \mathbf{x}') + t\,k_{structure}(\mathbf{x}, \mathbf{x}'),$$

  where t should be estimated from the training data.
- In general: Use the data to find best convex combination.

$$k(\mathbf{x}, \mathbf{x}') = \sum_{p=1}^{K} \beta_p k_p(\mathbf{x}, \mathbf{x}').$$

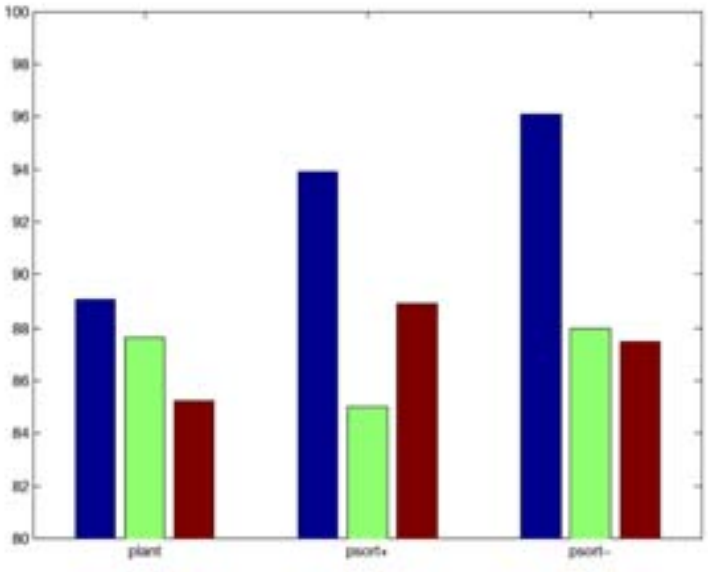
- Needs *semi-definite programming* to simultaneously find α's and β's

Applications

- "Optimal" Data Fusion
- Improving interpretability

Example: Subcellular Localization

- Predict to which location a protein will be transported
- Available data/kernels:
  - Amino acid kernel
  - Motif composition kernel
  - BLAST kernel
  - Phylogeny kernel
- Use Multiclass version of MKL [Zien and Ong, 2007b] on PSORTb data base [Gardy et al., 2004]



Cross-validated F1-scores for

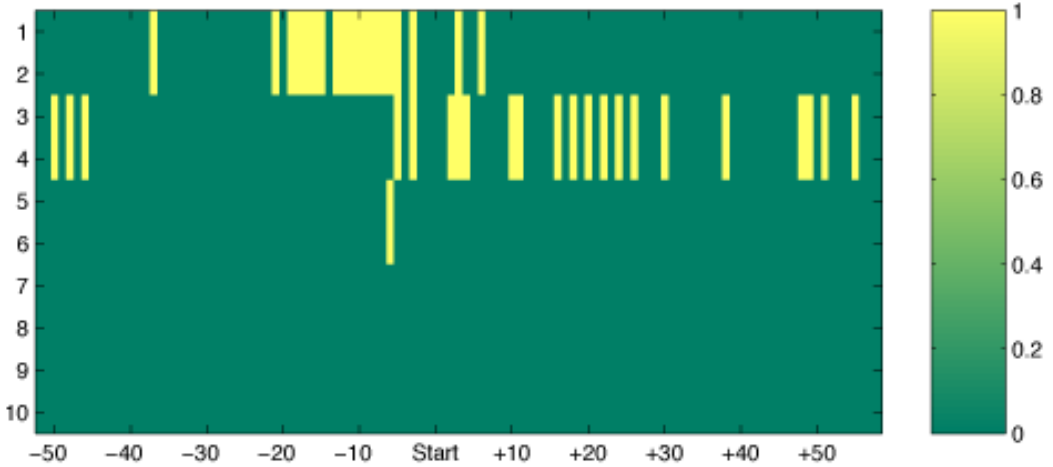
- MKL method (blue) [Zien and Ong, 2007a]
- Uniform kernel weighting (green)
- State of the art (red) [Gardy et al., 2004, Höglund et al., 2006]

Method for Interpreting SVMs

- Weighted Degree kernel: linear comb. of $L \cdot K$ kernels

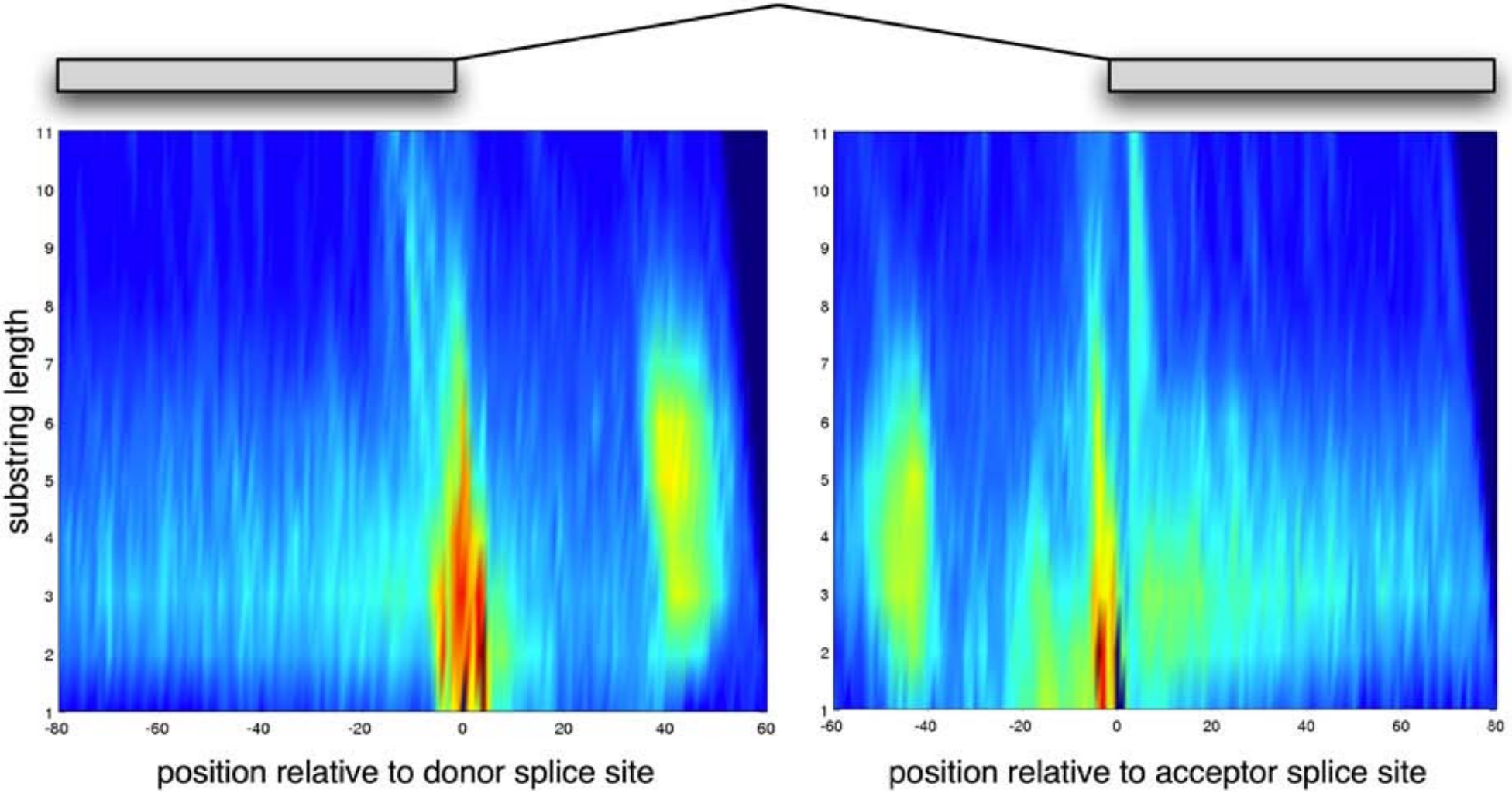
$$k(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{K} \sum_{l=1}^{L-k+1} \gamma_{l,k} \mathbf{I}(\mathbf{u}_{l,k}(\mathbf{x}) = \mathbf{u}_{l,k}(\mathbf{x}'))$$

- Example: Classifying splice sites



See Rätsch et al. [2006] for more details.

POIMs for Splicing



Color-coded importance scores of substrings near splice sites.
(cf. Rätsch et al. [2007])