Kernel Methods for Sequence Analysis: Introduction

Gunnar Rätsch[1], Cheng Soon Ong[1,2], Petra Philips[1]

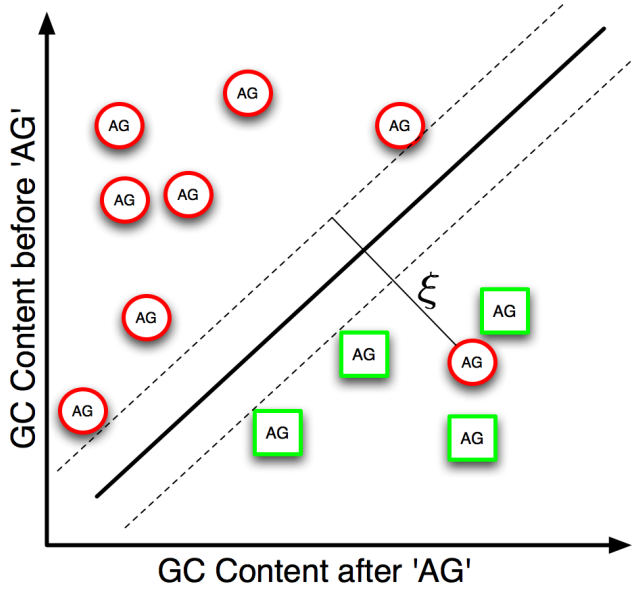
[1] Friedrich Miescher Laboratory, Tübingen
[2] MPI for Biological Cybernetics, Tübingen

Vorlesung WS 2007/2008
Eberhard-Karls-Universität Tübingen

4 December 2007

http://www.fml.mpg.de/raetsch/lectures/amsa07

SVMs: Geometric View



Minimize

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

Subject to

$$y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) \geqslant 1 - \xi_i$$
$$\xi_i \geqslant 0$$
for all $i = 1, \ldots, N$.

- The examples on the margin are called support vectors [Vapnik, 1995]
- Called the soft margin SVM or the C-SVM [Cortes and Vapnik, 1995]

SVM: Loss View

$$\underset{\mathbf{w},b}{\text{minimize}}\, \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N} \ell(f_{\mathbf{w},b}(\mathbf{x}), y_i),$$

where

$$\ell(f_{\mathbf{w},b}(\mathbf{x}), y_i) := C \max\{0, 1 - y_i(f_{\mathbf{w},b}(\mathbf{x}))\}$$
$$f_{\mathbf{w},b}(\mathbf{x}) := \mathbf{w}^\top \mathbf{x}_i + b.$$

The above loss function is known as the hinge loss.

Regularizer = $\frac{1}{2}\|\mathbf{w}\|^2$.

Empirical Risk = $\sum_{i=1}^{N} \ell(\mathbf{w}^\top \mathbf{x}_i + b, y_i)$.

How much does a mistake cost us?

Optimization

We have two versions of the same problem

Primal

$$\text{minimize}_{w,b} \frac{1}{2}\|w\|^2 + \sum_{i=1}^{N} \ell(w^\top x_i + b, y_i).$$

For a convex differentiable loss function, we can solve this directly using gradient methods or Newton's method.

Dual (with $\ell = C$ times hinge loss)

$$\begin{aligned} \text{maximize}_\alpha \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ \text{subject to} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & 0 \leqslant \alpha_i \leqslant C \text{ for } i = 1, 2, \ldots, N. \end{aligned}$$

We can use primal dual interior point methods to solve the convex optimization problem.

Summary "Kernel Trick"

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

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

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

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

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

Spectrum Kernel

General idea [Leslie et al., 2002]

- For each ℓ-mer $s \in \Sigma^\ell$, the coordinate indexed by s will be the number of times s occurs in sequence x.
- Then the ℓ-spectrum feature map is

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

- 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:

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

- Dimensionality: exponential in ℓ: $|\Sigma|^\ell$

Spectrum Kernel

Principle

- Spectrum kernel: Count exactly common ℓ-mers

Protein A: ILVFMC

Protein B: WLVFQC

Common 1-mers
L, V, F, C

Common 2-mers
LV
VF

Common 3-mers
LVF

- $\Phi(\mathbf{x})$ has only very few non-zero dimensions
  $\Rightarrow$ efficient kernel computations possible ($\mathcal{O}(|\mathbf{x}| + |\mathbf{x}'|)$)

Single Spectrum Kernel Computation

To compute $\mathrm{k}_D^{\mathrm{Spectrum}}(\mathbf{x}_i, \mathbf{x}_j)$

- Sorted Lists
  - Extract all $L_i - D + 1$ possible D-mers ($L_i = |\mathbf{x}_i|$)
  - Sort D-mer lists ($\mathcal{O}(D \cdot L_i \cdot \log L_i)$)
  - Iterate through both lists
    - Identify duplicated D-mers
    - $\mathcal{O}(D \cdot (L_i + L_j))$
- Sorting can be precomputed, i.e. final computation $\mathcal{O}(D \cdot (L_i + L_j))$

Weighted Degree Kernel

Weighted degree kernel Equivalent to a mixture of spectrum kernels (up to order D) at every position for appropriately chosen β's:

$$\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{d=1}^{D} \sum_{l=1}^{L-d+1} \beta_d \mathrm{k}_d^{\mathrm{Spectrum}}(\mathbf{u}_{l:l+d}(\mathbf{x}_i), \mathbf{u}_{l:l+d}(\mathbf{x}_j))$$

where $\beta_d = \frac{D-d+1}{\sum_d (D-d+1)} = 2\frac{D-d+1}{d\cdot(d+1)}$. [Rätsch and Sonnenburg, 2004]

Can be equivalently computed by

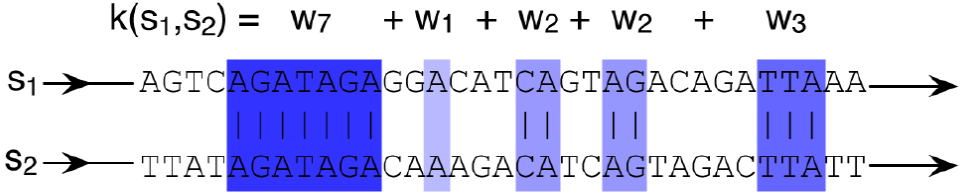
$$\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{d=1}^{D} \sum_{l=1}^{L-d+1} \beta_d \mathrm{I}(\mathbf{u}_{l:l+d}(\mathbf{x}_i) = \mathbf{u}_{l:l+d}(\mathbf{x}_j))$$

for appropriately chosen β_d.

Complexity: $\mathcal{O}(D \cdot L)$

Weighted Degree Kernel

Weighted degree kernel compares two sequences by identifying the largest matching blocks which contribute depending on their length ($\mathcal{O}(L)$).



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

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

If $k \leq D$, then $w_k = \frac{k(-k^2+3D\cdot k+3D+1)}{3D(D+1)}$,
otherwise $w_k = \frac{3k-D+1}{3}$

Solving the SVM Dual

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

Requires N^2 kernel computations

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

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

Idea: Chunking based methods

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

Coordinate Descent

$$F(\boldsymbol{\alpha}) := \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$$

- Select single variable
  - Random
  - Sequential
  - With largest gradient

$$\frac{\partial F(\boldsymbol{\alpha})}{\partial \alpha_i} = 1 - \frac{1}{2} \sum_{i \neq j = 1}^{N} \alpha_j y_i y_j \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) - \alpha_i \mathrm{k}(\mathbf{x}_i, \mathbf{x}_i)$$

- Solve $\Delta_t = \mathrm{argmax}_\Delta \, F(\alpha_1^t, \ldots, \alpha_n^t + \Delta, \ldots, \alpha_N^t)$
- Update $\boldsymbol{\alpha}^{t+1} = (\alpha_1^t, \ldots, \alpha_n^t + \Delta_n, \ldots, \alpha_N^t)^\top$
- Convergence guarantee ($\beta < 1$)

$$F(\boldsymbol{\alpha}^*) - F(\boldsymbol{\alpha}^t) \leq \exp(-\beta t)$$

Chunking

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

- Select Q variables $i_1, \ldots, i_Q$
  - Random (inefficient)
  - Sequential (inefficient)
  - Heuristic selection motivated by KKT conditions
    - Requires $f(\mathbf{x}_j) = \sum_{i=1}^{N} \alpha_i \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$ 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 $f(\mathbf{x}_j) = \sum_{i=1}^{N} \alpha_i \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$ for all j
- Solve QP of size Q

Complexity: $\mathcal{O}(Q \cdot N + Q^3)$
First step very expensive for large N

Can we speedup computing $f(\mathbf{x}_j)$?

So far for string kernels: $\mathcal{O}(Q \cdot N \cdot L \cdot D)$

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
- Computing 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 $\sum_{i=1}^{N} \alpha_i \mathrm{k}(\mathbf{x}_i, \mathbf{x})$ by

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

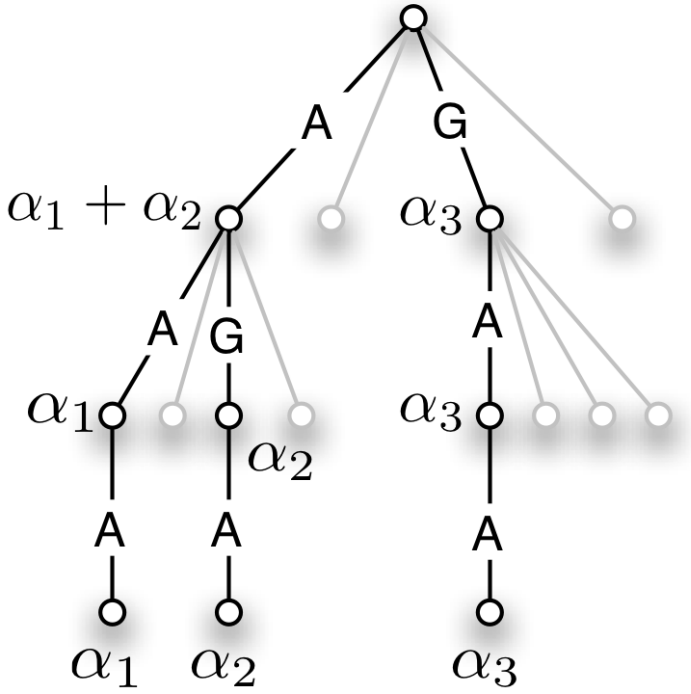
Explicit Maps

- Require $\mathcal{O}(|\Sigma|^D)$ 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 $\mathcal{O}(Q \cdot L \cdot D + N \cdot L \cdot D)$
- Very efficient, but only works for small D

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

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: $\mathcal{O}(Q \cdot L \cdot D)$
- Compute all $f(\mathbf{x}_i)$: $\mathcal{O}(N \cdot L \cdot D)$
- Memory: $\mathcal{O}(Q \cdot L \cdot D \cdot |\Sigma|)$
- Works for any D

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. [2007] for more details (available on website)

All implemented in Shogun toolbox

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	PWMs		96.03

References

C. Cortes and V.N. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575, 2002.

K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001.

G. Rätsch and S. Sonnenburg. Accurate splice site detection for *Caenorhabditis elegans*. In K. Tsuda B. Schoelkopf and J.-P. Vert, editors, *Kernel Methods in Computational Biology*. MIT Press, 2004.

B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

S. Sonnenburg, G. Rätsch, and K. Rieck. *Large Scale Kernel Machines*, chapter Large Scale Learning with String Kernels. MIT Press, 2007.

V.N. Vapnik. *The nature of statistical learning theory*. Springer Verlag, New York, 1995.