Support Vector Machines and Multiple Kernel Learning

(lets do it large scale with shogun)

Sören Sonnenburg

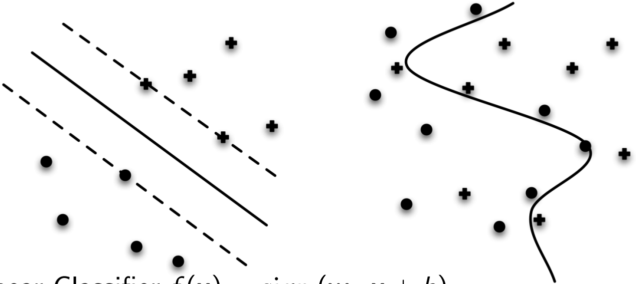
Shogun Toolbox Data Days 2013, Juli 12, Berlin

将軍

Outline

Classification

Given training examples $(\mathbf{x}_i, y_i)_{i=1}^N \in (\mathcal{X}, \{-1, +1\})^N$



- Linear Classifier $f(\mathbf{x}) = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b)$
- Kernel Machine (e.g. Support Vector Machine), learn weighting $\boldsymbol{\alpha} \in \mathbb{R}^N$ on training examples in kernel feature space $\Phi(\mathbf{x})$

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

where Kernel $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$

将軍

Classification using Support Vector Machines

Via kernel: non-linear discrimination in input space

- learn weighting $\boldsymbol{\alpha} \in \mathbb{R}^N$ on training examples $(\mathbf{x}_i, y_i)_{i=1}^N$ in kernel feature space $\Phi(\mathbf{x})$

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

- where Kernel $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$
- still linear classifier $f(\mathbf{x}) = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b)$ in kernel feature space, with weighting $\mathbf{w} = \sum_{i=1}^N y_i \alpha_i \Phi(\mathbf{x}_i)$ and examples $\mathbf{x} \mapsto \Phi(\mathbf{x})$

SVMs rock!

- Kernels - flexible!
- In many applications SVMs define the state-of-the-art!

Classification using Support Vector Machines

Via **kernel: non-linear** discrimination in input space

- learn weighting $\boldsymbol{\alpha} \in \mathbb{R}^N$ on training examples $(\mathbf{x}_i, y_i)_{i=1}^N$ in kernel feature space $\Phi(\mathbf{x})$

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

- where Kernel $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$
- still linear classifier $f(\mathbf{x}) = \mathrm{sign}\left(\mathbf{w} \cdot \mathbf{x} + b\right)$ in kernel feature space, with weighting $\mathbf{w} = \sum_{i=1}^N y_i \alpha_i \Phi(\mathbf{x}_i)$ and examples $\mathbf{x} \mapsto \Phi(\mathbf{x})$

SVMs rock!

- Kernels - flexible!
- In many applications SVMs define the state-of-the-art!

Support Vector Machines in Shogun I

Implementations supporting Kernels

- LibSVM, LibSVMOneClass, MulticlassLibSVM, LibSVR (sequential minimal optimization)
- GNPPSVM (ℓ_2-SVM)
- GPBTSVM (chunking algorithm based SVM)
- GMNPSVM (true multi-class SVM)
- LaRank (true multi-class SVM)
- MPDSVM (minimal primal-dual SVM)
- SVMLight, SVMLightOneClass, SVRLight (chunking based, thread-parallel non-GPL)
- ScatterSVM (multi-class approximation)

Support Vector Machines in Shogun I

Implementations supporting Kernels

- **LibSVM, LibSVMOneClass, MulticlassLibSVM, LibSVR** (sequential minimal optimization)
- GNPPSVM (ℓ_2-SVM)
- GPBTSVM (chunking algorithm based SVM)
- **GMNPSVM** (true multi-class SVM)
- LaRank (true multi-class SVM)
- MPDSVM (minimal primal-dual SVM)
- **SVMLight, SVMLightOneClass, SVRLight** (chunking based, thread-parallel non-GPL)
- ScatterSVM (multi-class approximation)

Kernels

Popular Kernels

- Gaussian, Polynomial, String Kernels
- Custom Kernel (bring your own)

Kernel Normalizers

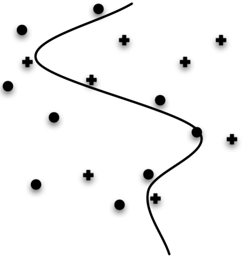
- SqrtDiagKernelNormalizer (points on hypersphere with $R = 1$)
- ZeroMeanCenterKernelNormalizer (centered in feature space)
- AvgDiagKernelNormalizer, more `http://bit.ly/133aG2V`

Most comprehensive Kernel/SVM Framework

- Over 60 kernels available
- `http://bit.ly/15zlCWY`
- normalizers can be attached to kernels

将軍

Kernel based Support Vector Machines



- SVMs learn weights $\boldsymbol{\alpha} \in \mathbb{R}^m$ over training examples in kernel feature space $\Phi : \mathbf{x} \mapsto \mathbb{R}^n$

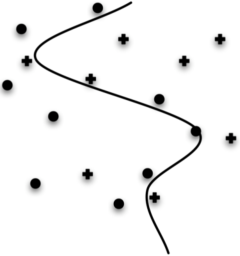
- Decision function $f(\mathbf{x}) = \mathrm{sign}\left(\sum_{i=1}^{m} y_i \alpha_i \mathrm{k}(\mathbf{x}, \mathbf{x}_i) + b\right)$,
  with kernel $k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$

SVMs rock!

- Kernels - flexible!
- In many applications SVMs define the state-of-the-art!
- **But not large scale!**

将軍

Kernel based Support Vector Machines



- SVMs learn weights $\boldsymbol{\alpha} \in \mathbb{R}^m$ over training examples in kernel feature space $\Phi : \mathbf{x} \mapsto \mathbb{R}^n$

- Decision function $f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{m} y_i \alpha_i \mathrm{k}(\mathbf{x}, \mathbf{x}_i) + b\right)$,
  with kernel $k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$

SVMs rock!

- Kernels - flexible!
- In many applications SVMs define the state-of-the-art!
- **But not large scale!**

将軍

Kernel based Support Vector Machines are a Dead End

The Curse of Support Vectors

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

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

Computational effort:

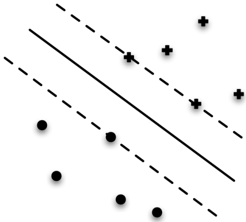
- All $\mathcal{O}(m_s m T)$, (T time to compute the kernel)
- Effort Scales linearly with $m_s = \mathcal{O}(m) := \#\mathrm{SVs}$

$\Rightarrow$ **SVM's in bigO are not faster than standard k-NN.**

$\Rightarrow$ **Kernel Machines are just not large-scale!**

將軍

What about Linear SVMs ?



- **Linear** Support Vector Machines learn weights $\mathbf{w} \in \mathbb{R}^n$
- Decision function $\mathbf{f}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$

Recent Progress in Linear SVM solvers

- SGD (Bottou 2007), SGD-QN (Bordes et al., 2009)
- SVM^{perf} (Joachims 2006), liblinear (Fan et al. 2008)
- BMRM (Teo et.al. 2007), OCAS (Franc, Sonnenburg 2009)

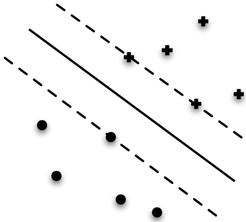
$\Rightarrow$ **Linear training Effort** $\mathcal{O}(m)$

$\Rightarrow$ **Computing Outputs Linear Effort** $\mathcal{O}(nm)$

. . . already linear time but just linear

将軍

What about Linear SVMs ?



- **Linear** Support Vector Machines learn weights $\mathbf{w} \in \mathbb{R}^n$
- Decision function $\mathbf{f}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$

Recent Progress in Linear SVM solvers

- SGD (Bottou 2007), SGD-QN (Bordes et al., 2009)
- SVM^{perf} (Joachims 2006), liblinear (Fan et al. 2008)
- BMRM (Teo et.al. 2007), OCAS (Franc, Sonnenburg 2009)

$\Rightarrow$ **Linear training Effort** $\mathcal{O}(m)$

$\Rightarrow$ **Computing Outputs Linear Effort** $\mathcal{O}(nm)$

... already linear time but just linear

将軍

Support Vector Machines in Shogun II

Linear SVMs

- OnlineLibLinear, LibLinear, MulticlassLibLinear (dual coordinate descent)
- NewtonSVM (newton primal svm)
- OnlineSVMSGD, SVMSGD (stochastic gradient descent)
- SGDQN (stochastic gradient descent with quasi-newton steps)
- SVMLin
- SVMOcas, MulticlassOCAS (cutting plane/bundle-method)

Support Vector Machines in Shogun II

Linear SVMs

- **OnlineLibLinear, LibLinear, MulticlassLibLinear** (dual coordinate descent)
- **NewtonSVM** (newton primal svm)
- **OnlineSVMSGD, SVMSGD** (stochastic gradient descent)
- **SGDQN** (stochastic gradient descent with quasi-newton steps)
- SVMLin
- **SVMOcas, MulticlassOCAS** (cutting plane / bundle-method)

Shogun's computational framework for linear SVMs

AIM: Development of a large scale learning framework for SVMs

> *"Algorithm [linear SVM solver] improvements do not improve the order of test error convergence. They can simply improve constant factors and therefore* **compete evenly with the implementation improvements***. Time spent refining the implementation is time well spent."*

from: Bordes, Bottou, Gallinari: SQD-QN: Careful Quasi-Newton Stochastic Gradient Descent. JMLR 2009.

Towards a computational framework for linear SVMs

Linear SVM solvers like liblinear, SGD, BMRM, OCAS only require two operations to access data:

(i) dot product between feature vector and the vector **w**:
$r \leftarrow \langle \mathbf{x}, \mathbf{w} \rangle$ **DOT**

(ii) multiplication with scalar $\alpha \in \Re$ and addition to vector $\mathbf{v} \in \Re^n$
$\mathbf{v} \leftarrow \alpha \mathbf{x} + \mathbf{v}$ **ADD**

将軍

COFFIN

COFFIN really is just two simple ideas:

On demand compute. . .

1. Features $\Phi(\mathbf{x})$ (only non-zero dims)
   - Non-Linearity Possible
   - Examples: Low Degree Polynomial Kernel, Spectrum Kernel, Weighted Degree Kernel
   - On-the-fly (de)compression
2. Virtual Examples
   - Incorporating Invariances possible
   - Examples: Image translation, rotation, etc

Needs efficient data structure for w!

- . . ., dense, sorted array, trees, hashes
- fast only when $|\Phi_{\neq 0}(\mathbf{z})| \sim dim(\mathbf{z})$

将軍

COFFIN

COFFIN really is just two simple ideas:

On demand compute...

1. Features $\Phi(\mathbf{x})$ (only non-zero dims)
   - Non-Linearity Possible
   - Examples: Low Degree Polynomial Kernel, Spectrum Kernel, Weighted Degree Kernel
   - On-the-fly (de)compression
2. Virtual Examples
   - Incorporating Invariances possible
   - Examples: Image translation, rotation, etc

Needs efficient data structure for w!

- ..., dense, sorted array, trees, hashes
- fast only when $|\Phi_{\neq 0}(\mathbf{z})| \sim dim(\mathbf{z})$

将軍

Data Structures

Effort of **ADD** and **DOT** for $\mathbf{z}$ and memory requirement of $\mathbf{w}$.

	Dense	Sorted Array	Tree
Add	$\mathcal{O}(\vert\Phi_{\neq 0}(\mathbf{z})\vert)$	$\mathcal{O}(\vert\mathbf{w}\vert_{\neq 0}) + \vert\Phi_{\neq 0}(\mathbf{z})\vert)$	$\mathcal{O}(\vert\Phi_{\neq 0}(\mathbf{z})\vert)$ to $\mathcal{O}(K\vert\Phi_{\neq 0}(\mathbf{z})\vert)$
Dot	$\mathcal{O}(\vert\Phi_{\neq 0}(\mathbf{z})\vert)$	$\mathcal{O}(\vert\mathbf{w}\vert_{\neq 0}) + \vert\Phi_{\neq 0}(\mathbf{z})\vert)$	$\mathcal{O}(\Phi_{\neq 0}(\mathbf{z})\vert)$ to $\mathcal{O}(K\vert\Phi_{\neq 0}(\mathbf{z})\vert)$
Mem	$\mathcal{O}(n)$	$\mathcal{O}(\sum_{i=1}^m \vert\Phi_{\neq 0}(\mathbf{z}_i)\vert)$	$\mathcal{O}(\sum_{i=1}^m \vert\Phi_{\neq 0}(\mathbf{z}_i)\vert)$

Sparse data structures have huge overhead!

Hashing to the Rescue (Shi et al (2009))

- Always use *dense* $\mathbf{w}$ with "compressed index"
- Hash function $h(J) \mapsto 1, \ldots, 2^\gamma$,
- $(\widehat{\Phi}(\mathbf{z}))_j = \sum_{i \in J; h(i)=j} (\Phi(\mathbf{z}))_i$

将軍

Data Structures

Effort of **ADD** and **DOT** for $\mathbf{z}$ and memory requirement of $\mathbf{w}$.

	Dense	Sorted Array	Tree
Add	$\mathcal{O}(\lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\lvert\mathbf{w}\rvert_{\neq 0}) + \lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$ to $\mathcal{O}(K\lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$
Dot	$\mathcal{O}(\lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\lvert\mathbf{w}\rvert_{\neq 0}) + \lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\Phi_{\neq 0}(\mathbf{z})\rvert)$ to $\mathcal{O}(K\lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$
Mem	$\mathcal{O}(n)$	$\mathcal{O}(\sum_{i=1}^{m}\lvert\Phi_{\neq 0}(\mathbf{z}_i)\rvert)$	$\mathcal{O}(\sum_{i=1}^{m}\lvert\Phi_{\neq 0}(\mathbf{z}_i)\rvert)$

Sparse data structures have huge overhead!

Hashing to the Rescue (Shi et al (2009))

- Always use *dense* $\mathbf{w}$ with "compressed index"
- Hash function $h(J) \mapsto 1, \ldots, 2^{\gamma}$,
- $(\widehat{\Phi}(\mathbf{z}))_j = \sum_{i \in J; h(i)=j} (\Phi(\mathbf{z}))_i$

将軍

COFFIN is implemented in Shogun's DotFeatures

DotFeatures

- Dense-,SparseFeatures
- PolyFeatures, BinnedDotFeatures
- CombinedDotFeatures
- ExplicitSpecFeatures, ImplicitWeightedSpecFeatures
- HashedDocDotFeatures

You can

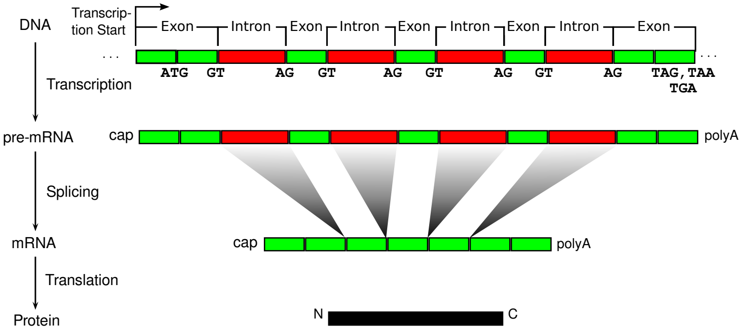
- Stack together features (dense, sparse,...)
- Provides abstraction to data representation

Use it!

Implemented for all linear SVMs

将軍

Splice Site Predictions



Application to Human Acceptor Splice Site Prediction

将軍

Splice Site Prediction

Discriminate true signal positions against all other positions

$\approx$ **150 nucleotides window around dimer**

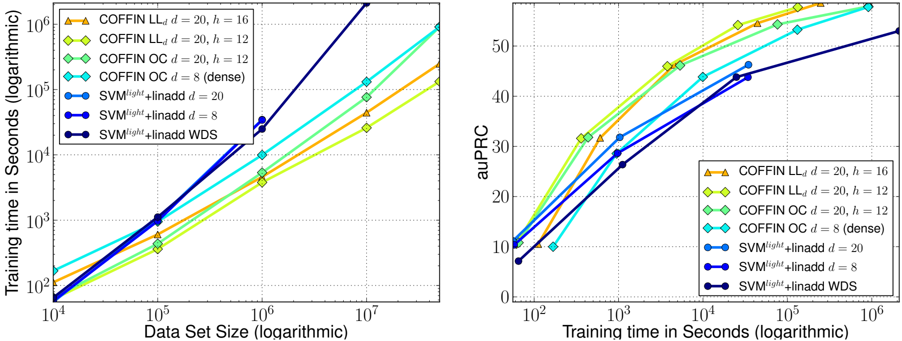
```
CT...GTCGTA...GAAGCTAGGAGCGC...ACGCGT...GA
```

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

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

- 50 million training examples
- COFFIN with kernels: weighted spectrum and weighted degree (explicit and hashed representation) $\approx 200,000,000$ dims

Applications



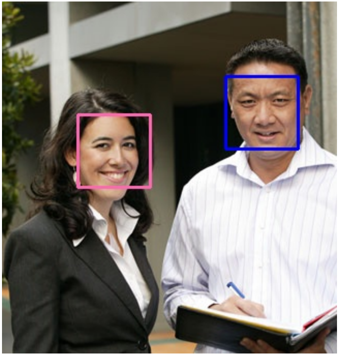
It's fast and works!

- Factor 47 faster on $10 \cdot 10^6$ examples than linadd
- New state-of-the-art results auPRC 58.57% vs. 53.01%

将軍

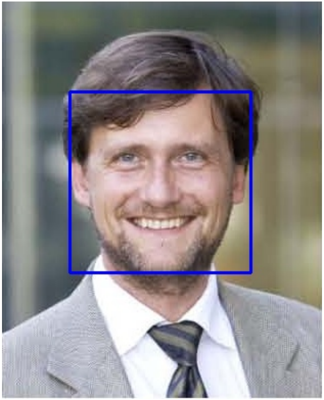
Gender Classification

Distinguish Females from Males solely based on Faces



- learn COFFIN on labelled faces
- virtual examples: translation, rotation, scale
- train $\approx$ 5 million sample (that would require 50GB) on Vojtechs notebook

Gender Classification I



It's fast and works - again!

- auROC 95.44%
- (vs. auROC 89.57% without VE)

Klaus-Robert Müller is a male!

将軍

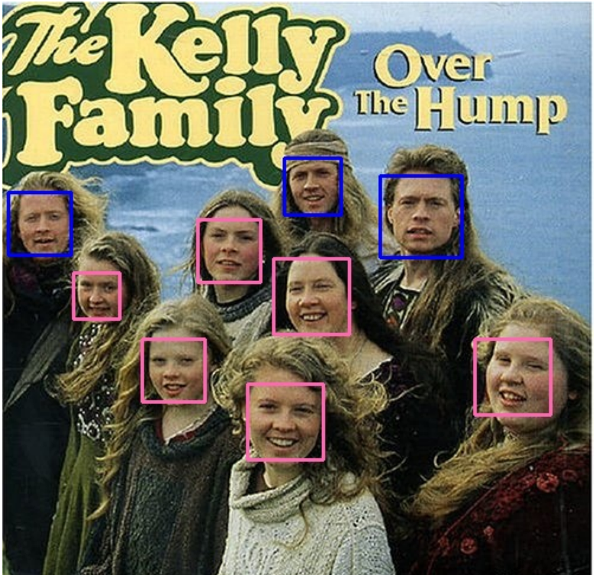
Gender Classification II

将軍

Gender Classification III

将軍

Gender Classification IV



Summary Support Vector Machines

Kernel SVMs

- Allows non-linear and complex models
- Applicable to mid-size datasets (1e6)
- General and often state-of-the art detectors

Linear SVMs

- Allows non-linearity with COFFIN/DotFeatures
- Applicable to huge datasets
- General and often state-of-the art detectors

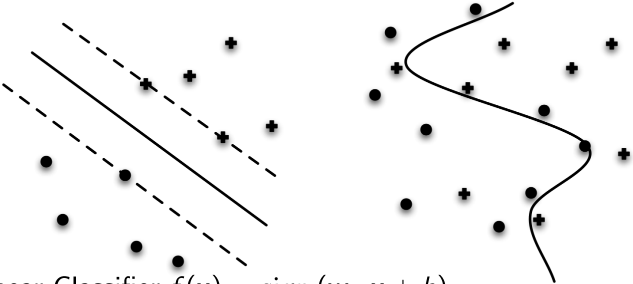
Datasets, Scripts, Efficient implementation

- Data and Scripts `http://sonnenburgs.de/soeren/coffin`
- Implementation `http://www.shogun-toolbox.org`
- More machine learning software `http://mloss.org`

将軍

Multiple Kernel Learning

Given training examples $(\mathbf{x}_i, y_i)_{i=1}^N \in (\mathcal{X}, \{-1,+1\})^N$



- Linear Classifier $f(\mathbf{x}) = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b)$
- Kernel Machine (e.g. Support Vector Machine), learn weighting $\boldsymbol{\alpha} \in \mathbb{R}^N$ on training examples in kernel feature space $\Phi(\mathbf{x})$

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

where Kernel $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$

将軍

Classification using Kernel Machines I

Single Kernel

- Kernel Machine (e.g. Support Vector Machine)
  - learn weighting $\boldsymbol{\alpha} \in \mathbb{R}^N$ on training examples $(\mathbf{x}_i, y_i)_{i=1}^N$ in kernel feature space $\Phi(\mathbf{x})$

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

  - where Kernel $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$
  - still linear classifier $f(\mathbf{x}) = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b)$ in kernel feature space, with weighting $\mathbf{w} = \sum_{i=1}^{N} y_i \alpha_i \Phi(\mathbf{x}_i)$ and examples $\mathbf{x} \mapsto \Phi(\mathbf{x})$

via kernel: non-linear discrimination in input space

Classification using Kernel Machines II

Multiple Kernels

- Linear combination of kernels $k(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{M} \beta_j \, k_j(\mathbf{x}, \mathbf{x}')$, $\beta_j \geq 0$. Learn $\boldsymbol{\alpha}$ *and* $\boldsymbol{\beta}$. Resulting classifier:

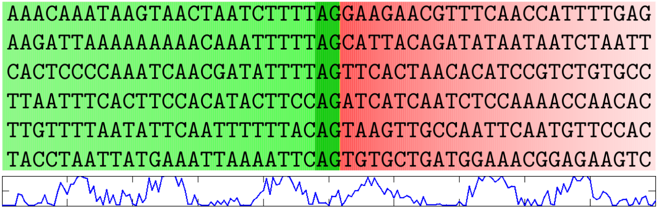
$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{j=1}^{M} \beta_j \sum_{i=1}^{N} y_i \alpha_i k_j(\mathbf{x}, \mathbf{x}_i) + b \right)$$

Learn weighting over training examples α and kernels β

将軍

Combining Heterogeneous Data

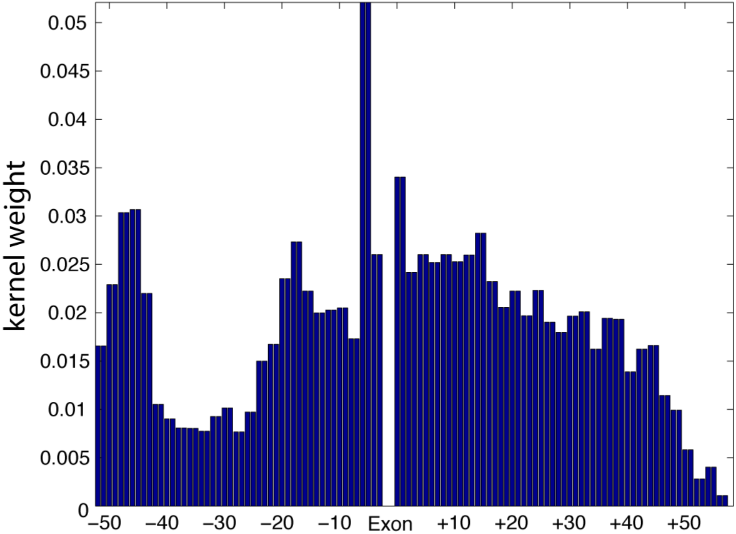
- Consider data from different domains: e.g Bioinformatics features: DNA-strings, binding energies, conservation, structure,...



$$k(\mathbf{x}, \mathbf{x}') =$$
$$\beta_1 \, k_{dna}(\mathbf{x}_{dna}, \mathbf{x}'_{dna}) + \beta_2 \, k_{nrg}(\mathbf{x}_{nrg}, \mathbf{x}'_{nrg}) + \beta_3 \, k_{3d}(\mathbf{x}_{3d}, \mathbf{x}'_{3d}) + \cdots$$

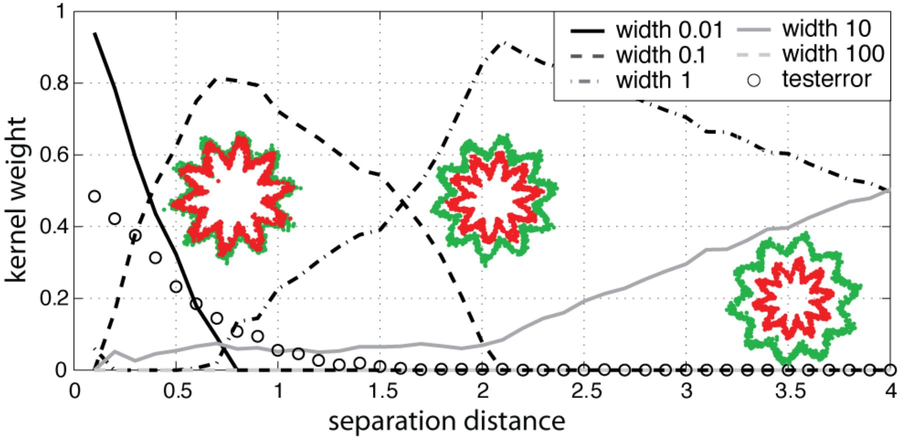
Interpretability

- Bioinformatics: One weight per position in sequence



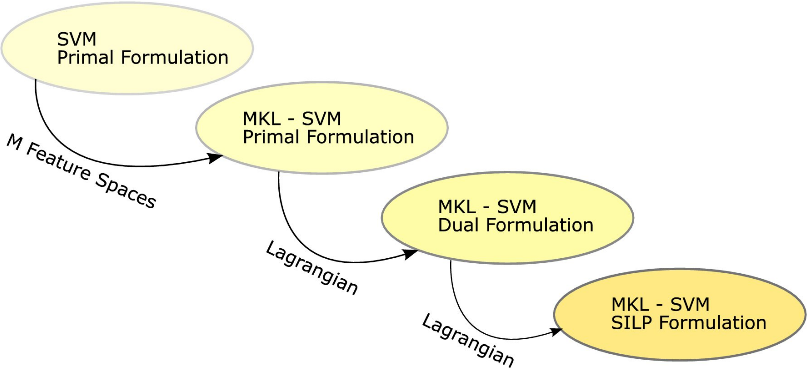
将軍

Automated Model Selection



将軍

Derivation



For details see Sonnenburg, Rätsch, Schäfer, Schölkopf 2006

将軍

SVM Primal Formulation

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

将軍

MKL Primal Formulation

$$
\begin{aligned}
\min \quad & \frac{1}{2}\left(\sum_{j=1}^{M} \beta_j \|\mathbf{w}_j\|_2\right)^2 + C\sum_{i=1}^{N} \xi_n \\
\text{w.r.t.} \quad & \mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_M),\ \mathbf{w}_j \in \mathbb{R}^{D_j}, \quad \forall j = 1 \ldots M \\
& \boldsymbol{\beta} \in \mathbb{R}_+^M,\ \boldsymbol{\xi} \in \mathbb{R}_+^N,\ b \in \mathbb{R} \\
\text{s.t.} \quad & y_i\left(\sum_{j=1}^{M} \beta_j \mathbf{w}_j{}^{\mathsf{T}} \Phi_j(\mathbf{x}_i) + b\right) \geq 1 - \xi_i,\ \forall i = 1, \ldots, N \\
& \sum_{j=1}^{M} \beta_j = 1
\end{aligned}
$$

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

将軍

MKL Dual Formulation

Bach, Lanckriet, Jordan 2004:

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

Properties: equivalent to SVM for $M = 1$

The Semi-Infinite Linear Program I

$$
\begin{array}{rl}
\max & \theta \\
\text{w.r.t.} & \theta \in \mathbb{R}, \boldsymbol{\beta} \in \mathbb{R}^M_+ \text{ with } \sum_{j=1}^{M} \beta_j = 1 \\
\text{s.t.} & \sum_{j=1}^{M} \beta_j \left(\underbrace{\frac{1}{2} \sum_{r=1}^{N} \sum_{s=1}^{N} \alpha_r \alpha_s y_r y_s K_j(\mathbf{x}_r, \mathbf{x}_s)}_{=:S_j(\boldsymbol{\alpha})} - \sum_{i=1}^{N} \alpha_i \right) \geq \theta \\
& \text{for all } \boldsymbol{\alpha} \text{ with } 0 \leq \boldsymbol{\alpha} \leq C \text{ and } \sum_{i=1}^{N} y_i \alpha_i = 0
\end{array}
$$

Properties:

- linear, optimize over a convex combination of $\boldsymbol{\beta}$
- infinitely many constraints, one for each $0 \leq \boldsymbol{\alpha} \leq C$
- can use standard SVM to identify most violated constraints

将軍

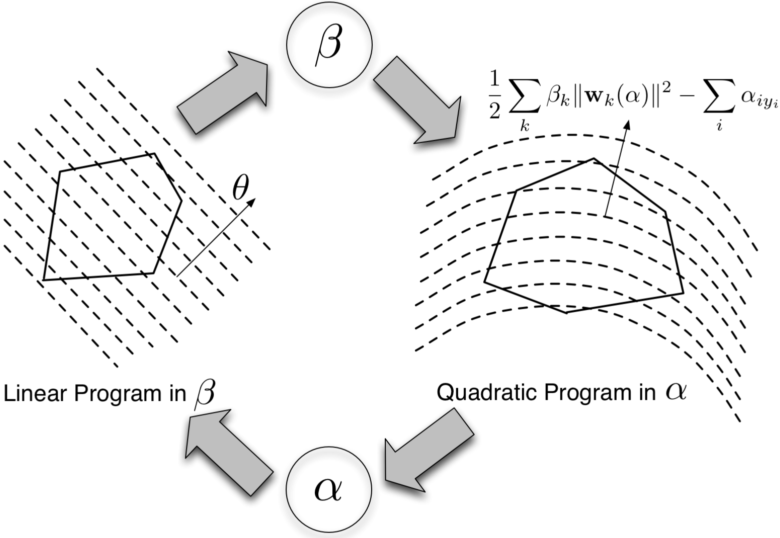
The Semi-Infinite Linear Program II

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

Solving the SILP:

- Column Generation
  - fast, but no known convergence rate
- Use Boosting like techniques: Arc-GV or AdaBoost*
  - known convergence rate $\mathcal{O}(\log(M)/\varepsilon^2)$
- Chunking like algorithm
  - consider suboptimal SVM solutions: empirically 3-5 times faster

Solving the SILP: Column Generation I



将軍

Solving the SILP: Column Generation II

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

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

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

- solved by taking set of most violated constraints into account
- most violated constraints given by SVM solution for fixed β

Regression

Primal Formulation:

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

$$\text{w.r.t.} \quad \mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_M), \mathbf{w}_j \in \mathbb{R}^{D_j}, \quad \forall j = 1 \ldots M$$

$$\boldsymbol{\beta} \in \mathbb{R}_+^M, \ \boldsymbol{\xi} \in \mathbb{R}^N, \ \boldsymbol{\xi}^* \in \mathbb{R}_+^N, \ b \in \mathbb{R}$$

$$\text{s.t.} \quad \sum_{j=1}^{M} \beta_j \mathbf{w}_j{}^{\mathsf{T}} \Phi_j(\mathbf{x}_i) + b \leq y_i + \varepsilon + \xi_i, \forall i = 1 \ldots N$$

$$\sum_{j=1}^{M} \beta_j \mathbf{w}_j{}^{\mathsf{T}} \Phi_j(x_i) + b \geq y_i - \varepsilon - \xi_i^*, \forall i = 1 \ldots N$$

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

将軍

One Class

Primal Formulation:

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

$$\text{w.r.t.} \quad \mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_M),\ \mathbf{w}_j \in \mathbb{R}^{D_j}, \quad \forall j = 1 \ldots M$$

$$\boldsymbol{\beta} \in \mathbb{R}_+^M,\ \boldsymbol{\xi} \in \mathbb{R}_+^N$$

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

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

Generalized for arbitrary strictly convex differentiable loss functions (Sonnenburg, Rätsch, Schäfer, Schölkopf 2006)

将軍

Multiclass

Primal Formulation (Zien, Ong 2007):

$$\begin{aligned}
\min \quad & \frac{1}{2}\left(\sum_{j=1}^{M} \beta_j \|\mathbf{w}_j\|_2\right)^2 + C\sum_{i=1}^{N}\xi_n \\
\text{w.r.t.} \quad & \mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_M),\ \mathbf{w}_j \in \mathbb{R}^{k_j}, \quad \forall j = 1 \ldots M \\
& \boldsymbol{\beta} \in \mathbb{R}_+^M,\ \mathbf{s} \in \mathbb{R}^{N\times c},\ \boldsymbol{\xi} \in \mathbb{R}_+^N,\ b \in \mathbb{R} \\
\text{s.t.} \quad & \xi_i = \max_{u \neq y_i} s_{iu},\ s_{iu} \geq 0, \\
& \sum_{j=1}^{M} \beta_j \mathbf{w}_j^{\mathsf{T}} \left(\Phi_j(\mathbf{x}_i, y_i) - \Phi_j(\mathbf{x}_i, u)\right) + b_{y_i} - b_u \geq 1 - s_{iu}, \\
& \quad \forall i = 1 \ldots N,\ \forall u = 1 \ldots c
\end{aligned}$$

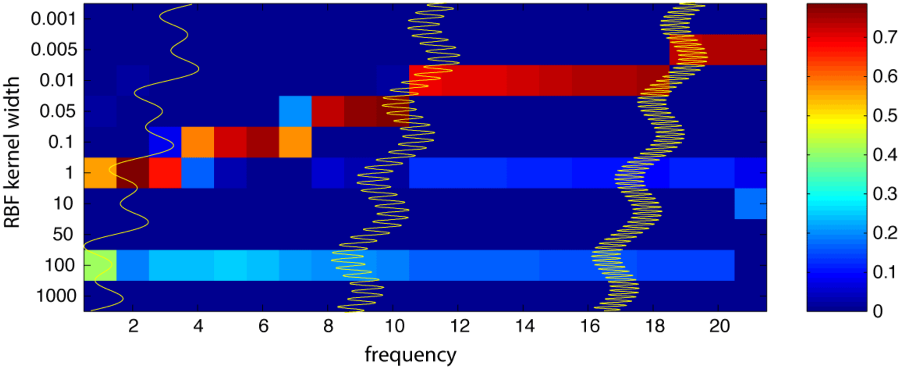
Further Extension

Extension

- ℓ_p norm MKL
- faster algorithms
- convergence bounds
- and many more

$\vdots$

Automated Model Selection - Regression

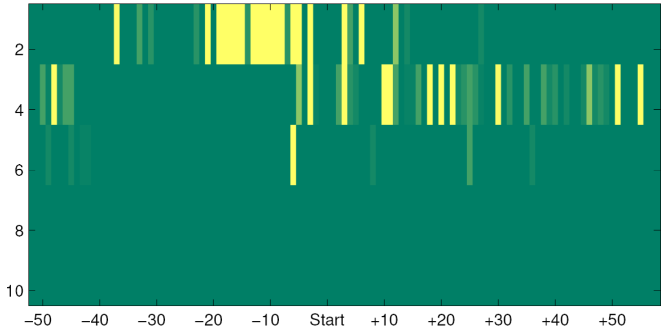


- $f(x) = \sin(ax) + \sin(bx) + cx$ for varying a
- Support Vector Regression with 10 RBF-Kernels of different width

Knowledge discovery

将軍

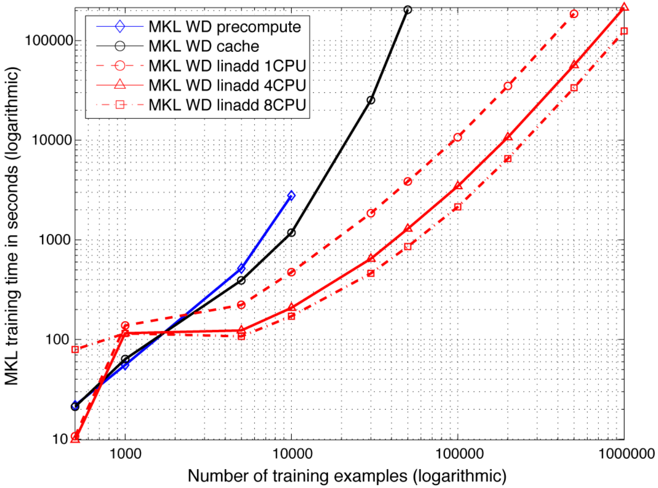
Feature Extraction



- Support Vector Classification on Bioinformatics problem, distinguish “splice sites” form “fake sites” (aligned DNA sequences)
- One weight β_j per position and per sub-sequence length
- Displayed: Learned weights of 500 kernels

将軍

A word about speed.



将軍

Summary and Outlook

MKL learns convex combination of kernels

$\Rightarrow$ allows (to some extend) for automated model selection

$\Rightarrow$ allows for interpreting SVM result

$\Rightarrow$ matches prior knowledge on real-world bioinformatics problem

- **Simple:** iterative wrapper algorithm around single kernel SVM
- **General:** same technique applicable to a wide range of problems (1-class, 2-class, Multiclass, Regression, ...)
- **Fast:** suitable for large scale problems ($> 100,000$ examples)

Download free source `http://www.shogun-toolbox.org`**.**