Cmput 466/551

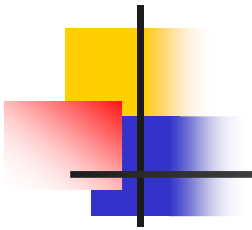
Support Vector Machines: Kernels

Covering chapters (HTF): 4.5, 12

R Greiner
Department of Computing Science
University of Alberta

Much of this is taken from Andrew W. Moore; CMU
http://www.cs.cmu.edu/~awm/tutorials

+ Nello Cristianini, Ron Meir, Ron Parr, Barnabas Poczos



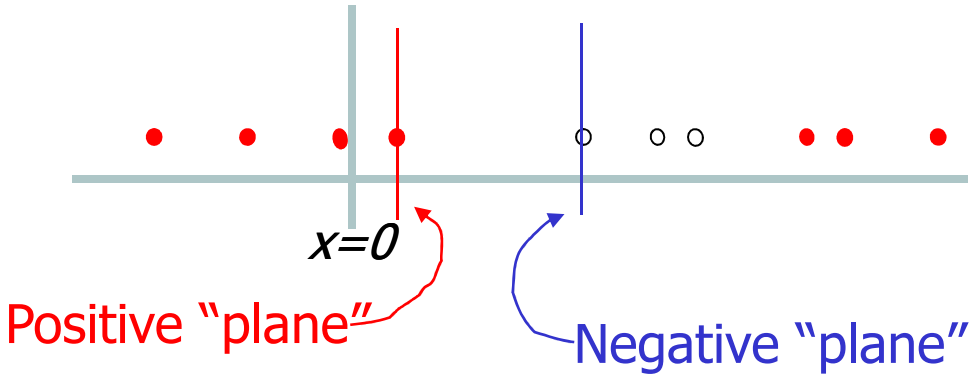
Outline

- Foundations
  - ~~Linear Programming;Primal/Dual~~
  - Constrained Optimization
    - Lagrange Multipliers
    - KKT
  - Perceptron Factoids
    - Dual Representation
- "Best" Linear Separator: Max Margin!
- Coping with Non-Linearly Separated Data
  - Slack Variables
- Kernel Trick
- Regression

Hard 1-dimensional Dataset

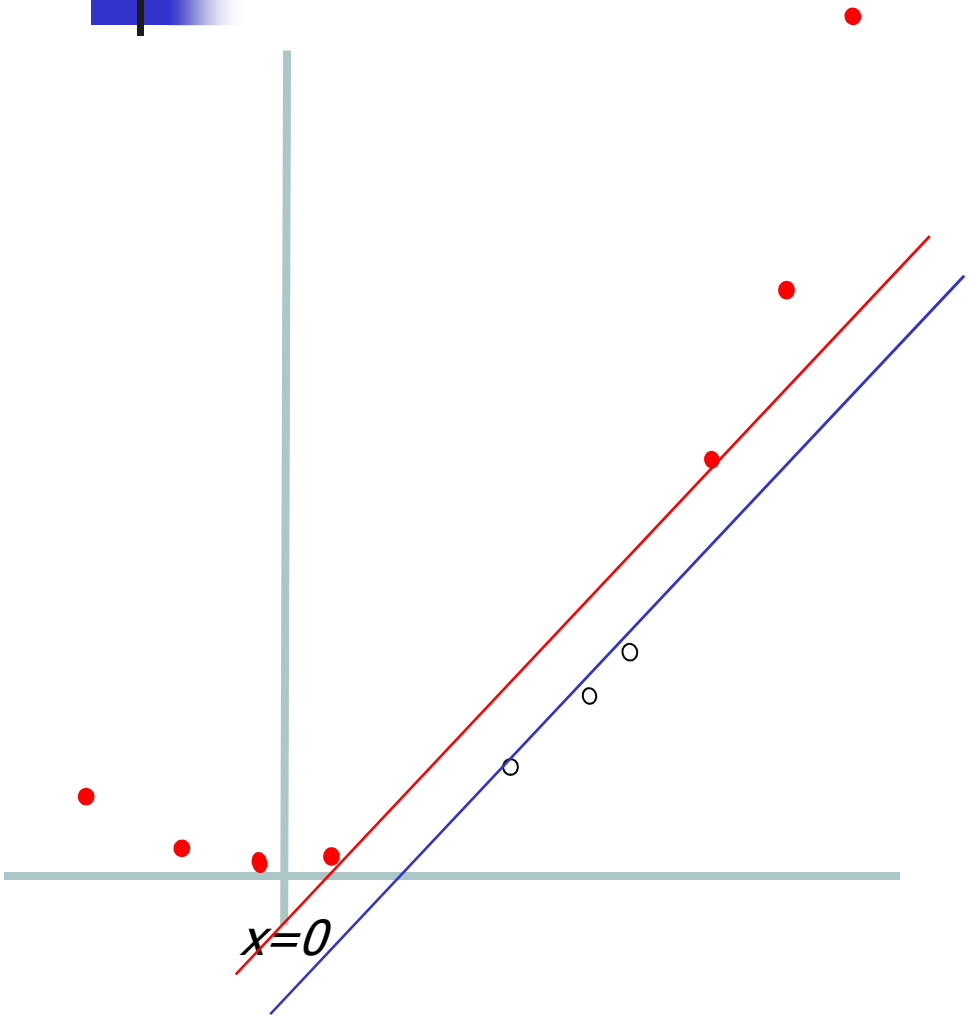
What would SVMs do with this data?

Not a big surprise



Doesn't look like slack variables will save us this time…

Hard 1-dimensional Dataset



M*ake up a new feature!*

$$z_k = (x_k, x_k^2)$$

… computed from original feature(s)

Separable! … MAGIC!

Now run linear SVM on this "augmented" data

… New Features from Old …

- Here: mapped $\Re \rightarrow \Re^2$ by $\Phi: x \rightarrow [x, x^2]$
  - Using "extra dimensions" $\Rightarrow$ linearly separable!
- In general,
  - Start with vector $\mathbf{x} \in \Re^k$
  - Want to add in x_1^2 , x_2^2, …
  - Probably want other terms – eg $x_2 \cdot x_7$, …
  - Which ones to include?
- Why not ALL OF THEM?
  (If some features are linearly-separable, then any SUPERSET is)
- $(x_1, x_2, x_3) \rightarrow (1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3)$
  - $\Re^3 \rightarrow \Re^{10}$
  - In general, #features grows:

$$1 + m \rightarrow 1 + m + m + \binom{m}{2} = \frac{(m+2)(m+1)}{2} \approx \frac{m^2}{2}$$

Implied Algorithm

- Training: Given R training instances, each in $\Re^m$
  1. Map each $\Re^m$ –tuple $\mathbf{x}_i$ to $\Re^{\frac{m^2}{2}}$ –tuple $\Phi(\mathbf{x}_i)$
  2. Learn SVM classifier wrt these $\Phi(\mathbf{x}_i)$ tuples

- Performance: Given new $\Re^m$ –tuple $\mathbf{x}$
  1. Map this $\mathbf{x}$ to $\Re^{\frac{m^2}{2}}$ –tuple $\Phi(\mathbf{x})$
  2. Apply learned SVM classifier to $\Phi(\mathbf{x})$

Original "Linear" QP

$$\max_{\lambda} \sum_{k=1}^{R} \lambda_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{r=1}^{R} \lambda_k \lambda_r Q_{kr} \qquad Q_{kr} = y_k y_r (\mathbf{x}_k \cdot \mathbf{x}_r)$$

Subject to the constraints:

$$0 \le \lambda_k \le C \quad \forall k \qquad \sum_{k=1}^{R} \lambda_k y_k = 0$$

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \lambda_k y_k \mathbf{x}_k$$

$$b = y_K (1 - \varepsilon_K) - \mathbf{w} \cdot \mathbf{x}_K$$

where $K = \arg\max_k \lambda_k$

Then classify with:

f(**x**,**w**,b) = *sign*(***w*** · ***x*** + *b*)

QP using Basis Functions

$$\max_{\lambda} \sum_{k=1}^{R} \lambda_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{r=1}^{R} \lambda_k \lambda_r Q_{kr} \qquad Q_{kr} = y_k y_r (\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_r))$$

Subject to the constraints:

$$0 \le \lambda_k \le C \quad \forall k \qquad \sum_{k=1}^{R} \lambda_k y_k = 0$$

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \lambda_k y_k \, \Phi(\mathbf{x}_k)$$

$$b = y_K (1 - \varepsilon_K) - \mathbf{w} \cdot \Phi(\mathbf{x}_K)$$

where $K = \arg\max_k \lambda_k$

Then classify with:

f(**x**,**w**,b) = sign(**w** · $\Phi(\mathbf{x})$ + b)

QP using Basis Functions

$$\max_{\lambda} \sum_{k=1}^{R} \lambda_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{r=1}^{R} \lambda_k \lambda_r Q_{kr} \qquad Q_{kr} = y_k y_r (\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_r))$$

Subject to the constraints: $0 \le \lambda_k \le C \quad \forall k \qquad \sum_{k=1}^{R} \lambda_k y_k$

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \lambda_k y_k \, \Phi(\mathbf{x}_k)$$

$$b = y_K(1 - \varepsilon_K) - \mathbf{w} \cdot \Phi(\mathbf{x}_K)$$

where $K = \arg\max_k \lambda_k$

$\Phi(\mathbf{x}_k)$ only appears within dot product!

Then classify with:

f(**x**,**w**,b) = sign(**w** ·) + b)

$$= \text{sign}\left(\sum_k \lambda_k y_k \, \boldsymbol{\phi}(\mathbf{x}_k) \cdot \boldsymbol{\phi}(\mathbf{x}) + \mathbf{b} \right)$$

$$= \text{sign}\left(\sum_k \lambda_k y_k \, [\boldsymbol{\phi}(\mathbf{x}_k) \cdot \boldsymbol{\phi}(\mathbf{x})] + \mathbf{b} \right)$$

Implied Algorithm

- Training: Given R training instances, each in $\Re^m$
  1. Map each $\Re^m$ –tuple $\mathbf{x}_i$ to $\Re^{m*m/2}$ –tuple $\Phi(\mathbf{x}_i)$
  2. Learn SVM classifier wrt these $\Phi(\mathbf{x}_i)$ tuples

- Performance: Given new $\Re^m$ –tuple $\mathbf{x}$
  1. Map this $\mathbf{x}$ to $\Re^{m*m/2}$ –tuple $\Phi(\mathbf{x})$
  2. Apply learned SVM classifier to $\Phi(\mathbf{x})$

One more trick!

- Issue:
  - This $\Phi(.)$ operation is expensive!! – $O(m^2)$
  - What if want $\Phi'(.)$ that deals with "x^3", or "x^4", or ...

Higher Dimensional Space

- Given
  - $\mathbf{x} = (x_1, x_2, x_3)^T$
  - $\mathbf{y} = (y_1, y_2, y_3)^T$
- Standard model:
  - $\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i = x_1y_1 + x_2y_2 + x_3y_3$
- What if some squared terms might matter:
  - $\Phi(\mathbf{x}) = (x_1x_2, x_1x_3, x_2x_3, {x_1}^2, {x_2}^2, {x_3}^2)^T$?

Quadratic Basis Function

$$(\boldsymbol{a}^T \boldsymbol{b})^2 = (a_1 b_1 + a_2 b_2)^2$$

$$= a_1^2 b_1^2 + 2 a_1 b_1 a_2 b_2 + a_2^2 b_2^2$$

$$= (a_1^2 , \sqrt{2}\, a_1 a_2 , a_2^2) \begin{pmatrix} b_1^2 \\ \sqrt{2}\, b_1 b_2 \\ b_2^2 \end{pmatrix}$$

$$= \phi(\mathbf{a}) \bullet \phi(\mathbf{b})$$

where

$$\phi(\mathbf{z}) = \phi([z_1, z_2]) = (z_1^2 , \sqrt{2}\, z_1 z_2 , z_2^2)$$

Higher Dimensional Space

- Want to consider

$$\Phi(\mathbf{x}) = \left(\sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3, \ x_1^2, \ x_2^2, x_3^2\right)^T$$

$$\Phi(\mathbf{x})^T\Phi(\mathbf{y}) = 2x_1x_2y_1y_2 + 2x_1x_3y_1y_3 + 2x_2x_3y_2y_3 + x_1^2y_1^2 + x_2^2y_2^2 + x_3^2y_3^2$$

- Computation complexity:
  - Compute $\Phi(\mathbf{x})$ – $O(m^2)$ multiplications
  - Compute $\Phi(\mathbf{y})$ – $O(m^2)$ multiplications
  - Compute $\Phi(\mathbf{x})^T\Phi(\mathbf{y})$ – $O(m^2)$ multiplications & additions

  $\Rightarrow$ $O(m^2)$ time
- Small tweak
  - Note $\Phi(\mathbf{x})^T\Phi(\mathbf{y}) = \left(\mathbf{x}^T\mathbf{y}\right)^2$
  - Computational complexity: $O(m)$

Higher Dimensional Space

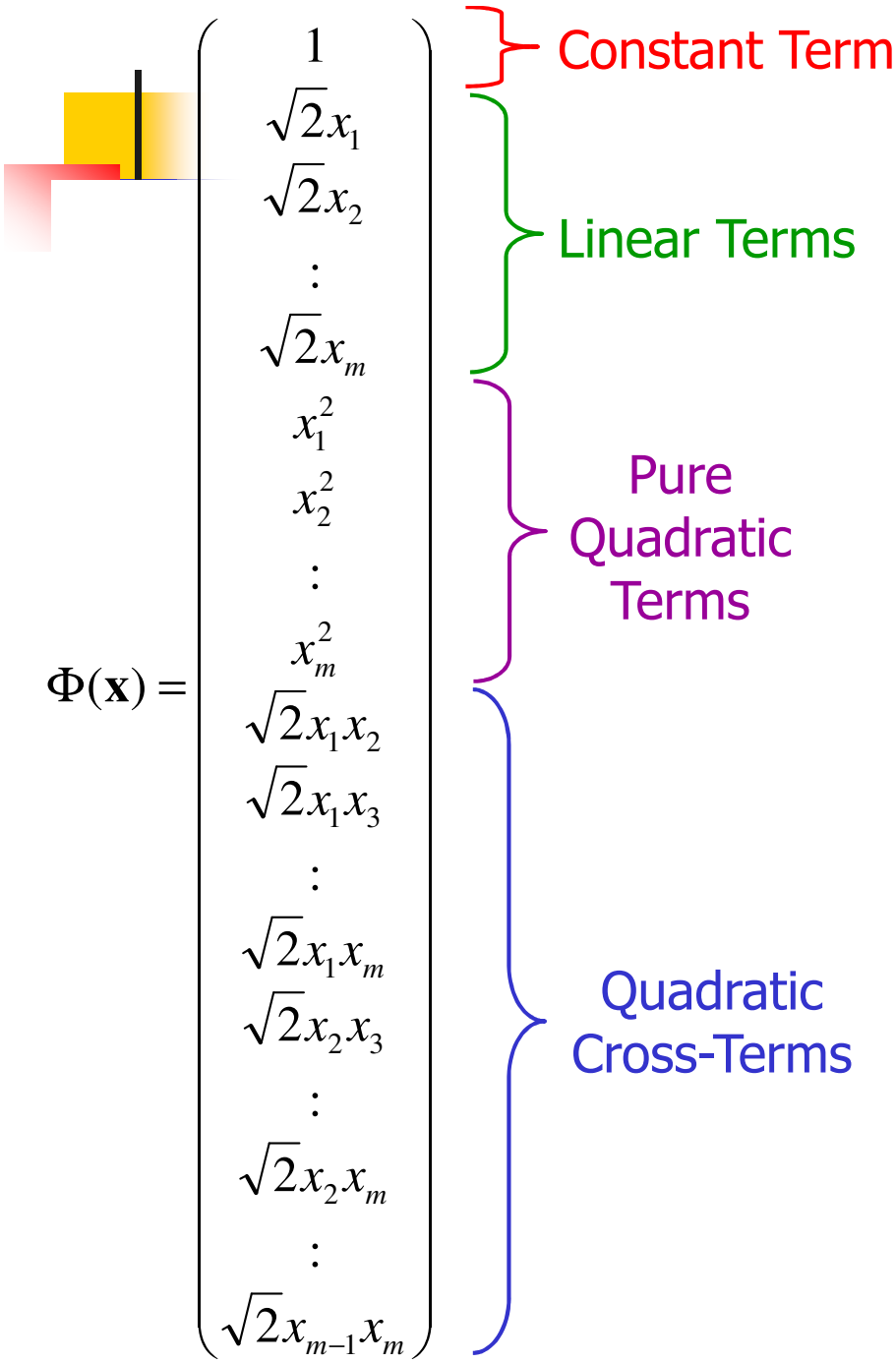
- Want to consider

$$\Phi(\mathbf{x}) = \left(\sqrt{2}x_1x_2, \sqrt{2}\,x_1x_3, \sqrt{2}x_2x_3, x_1^2, x_2^2, x_3^2\right)^T$$

$$\Phi(\mathbf{x})^T\Phi(\mathbf{y}) = 2x_1x_2y_1y_2 + 2x_1x_3y_1y_3 + 2\,x_2x_3y_2y_3 + x_1^2y_1^2 + x_2^2y_2^2 + x_3^2y$$

  - Note $\Phi(\mathbf{x})^T\Phi(\mathbf{y}) = \left(\mathbf{x}^T\mathbf{y}\right)^2$

- Advantage:
  - Computational complexity: $O(m)$ [not $O(m^2)$]
- But…
  - Only considering THIS COMBINATION of terms
  - Why not $(17\,x_1x_2,\ 5\ x_1x_3,\ 2\ x_2x_3,\ 4\,x_1^2, x_2^2, 0\,x_3^2)^T$



Another Quadratic Basis Function

What about those $\sqrt{2}$ s ??

... stay tuned

Quadratic Dot Products

$$\Phi(\mathbf{a}) \bullet \Phi(\mathbf{b}) = \begin{pmatrix} 1 \\ \sqrt{2}a_1 \\ \sqrt{2}a_2 \\ \vdots \\ \sqrt{2}a_m \\ a_1^2 \\ a_2^2 \\ \vdots \\ a_m^2 \\ \sqrt{2}a_1a_2 \\ \sqrt{2}a_1a_3 \\ \vdots \\ \sqrt{2}a_1a_m \\ \sqrt{2}a_2a_3 \\ \vdots \\ \sqrt{2}a_2a_m \\ \vdots \\ \sqrt{2}a_{m-1}a_m \end{pmatrix} \bullet \begin{pmatrix} 1 \\ \sqrt{2}b_1 \\ \sqrt{2}b_2 \\ \vdots \\ \sqrt{2}b_m \\ b_1^2 \\ b_2^2 \\ \vdots \\ b_m^2 \\ \sqrt{2}b_1b_2 \\ \sqrt{2}b_1b_3 \\ \vdots \\ \sqrt{2}b_1b_m \\ \sqrt{2}b_2b_3 \\ \vdots \\ \sqrt{2}b_2b_m \\ \vdots \\ \sqrt{2}b_{m-1}b_m \end{pmatrix}$$

$$1 + \sum_{i=1}^{m} 2a_ib_i + \sum_{i=1}^{m} a_i^2b_i^2 + \sum_{i=1}^{m}\sum_{j=i+1}^{m} 2a_ia_jb_ib_j$$

Quadratic Dot Products

$$\Phi(\mathbf{a}) \bullet \Phi(\mathbf{b}) =$$

$$1 + 2\sum_{i=1}^{m} a_i b_i + \sum_{i=1}^{m} (a_i b_i)^2 + \sum_{i=1}^{m} \sum_{j=i+1}^{m} 2 a_i a_j b_i b_j$$

Now consider another fn of ***a*** and ***b*** :

$$(\mathbf{a} \cdot \mathbf{b} + 1)^2$$

$$= (\mathbf{a} \cdot \mathbf{b})^2 + 2\mathbf{a} \cdot \mathbf{b} + 1$$

$$= \left(\sum_{i=1}^{m} a_i b_i \right)^2 + 2\sum_{i=1}^{m} a_i b_i + 1$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{m} a_i b_i a_j b_j + 2\sum_{i=1}^{m} a_i b_i + 1$$

$$= \sum_{i=1}^{m} (a_i b_i)^2 + 2\sum_{i=1}^{m} \sum_{j=i+1}^{m} a_i b_i a_j b_j + 2\sum_{i=1}^{m} a_i b_i + 1$$

They're the same!

But this is only O(m) to compute… not O(m^2)

Higher Order Polynomials

$$Q_{kr} = y_k y_r \phi(\mathbf{x_k}) \cdot \phi(\mathbf{x_r})$$

Poly-nomial	$\phi(\mathbf{x})$	Cost to build Q_{kr} matrix: *traditional*	Cost if m=100 inputs	$\phi(\mathbf{a})\cdot\phi(\mathbf{b})$	Cost to build Q_{kr} matrix: *sneaky*	Cost if m=100 inputs
Quadratic	All $m^2/2$ terms ≤ degree 2	$m^2 R^2/4$	2 500 R^2	$(\mathbf{a}\cdot\mathbf{b}+1)^2$	$m R^2/2$	50 R^2
Cubic	All $m^3/6$ terms ≤ degree 3	$m^3 R^2/12$	83 000 R^2	$(\mathbf{a}\cdot\mathbf{b}+1)^3$	$m R^2/2$	50 R^2
Quartic	All $m^4/24$ terms ≤ degree 4	$m^4 R^2/48$	1 960 000 R^2	$(\mathbf{a}\cdot\mathbf{b}+1)^4$	$m R^2/2$	50 R^2

Kinda … coefficients for terms are not independent …

QP using Quintic Basis Functions

This matrix requires $R^2/2$ dot products.

In 100-d, each dot product requires 103 ops, … not 75 million

But still worries…

$$Q_{kr} = y_k y_r (\Phi(\boldsymbol{x}_k) \cdot \Phi(\boldsymbol{x}_r))$$

The use of Maximum Margin magically reduces this problem

Subject to the constraints:

- Overfitting due to enormous number of terms
- The evaluation phase (doing a predictions on a test instance **x**) seems expensive…

  as **w**·ϕ(**x**) needs 75 million operations

Then define:

$$\boldsymbol{w} = \sum_{k=1}^{R} \lambda_k y_k \, \Phi(\boldsymbol{x}_k)$$

$$\boldsymbol{w} \cdot \Phi(\boldsymbol{x}) = \sum_{k \; s.t. \; \lambda_k > 0} \lambda_k y_k \Phi(\boldsymbol{x}_k) \Phi(\boldsymbol{x})$$

Only *S m* operations (*S*=#support vectors)

$$f(\mathbf{x},\mathbf{w},b) = \text{sign}(\mathbf{w} \cdot \Phi(\mathbf{x}) + b)$$

$$= \text{sign}\left(\sum_k \lambda_k y_k \, \boldsymbol{\phi}(\mathbf{x}_k) \cdot \boldsymbol{\phi}(\mathbf{x}) + \mathbf{b}\right)$$

$$= \text{sign}\left(\sum_k \lambda_k y_k \, [\boldsymbol{\phi}(\mathbf{x}_k) \cdot \boldsymbol{\phi}(\mathbf{x})] + \mathbf{b}\right)$$

The "Kernel Trick"!

$$\max_{\lambda} \sum_{k=1}^{R} \lambda_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{r=1}^{R} \lambda_k \lambda_r y_k y_r \, \mathrm{K}(\mathbf{x}_k, \mathbf{x}_r)$$

$$\mathrm{K}(\mathbf{x}_k, \mathrm{x}_r) = \boldsymbol{\phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\phi}(\boldsymbol{x}_j)$$

$$\sum_i \lambda_i y_i = 0 \qquad 0 \le \lambda_k \le C$$

$$\boldsymbol{w} = \sum_i \lambda_i y_i \, \boldsymbol{\phi}(\boldsymbol{x}_i)$$

$$b = y_k - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_k)$$

for any k s.t. $0 < \lambda_k < C$

- Never represent features explicitly
  - Compute dot products in closed form

- ∃ constant-time high-dimensional dot-products for many classes of features

… at classification time

- Recall classifier computes: sign($\mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x})$ +b)
- For a new input $\boldsymbol{x}$, if we need to represent $\boldsymbol{\phi}(\boldsymbol{x})$, we are in trouble!
- Using kernels: we are cool!

$$K(\boldsymbol{u}, \boldsymbol{v}) = \Phi(\boldsymbol{u}) \cdot \Phi(\boldsymbol{v})$$

$$\boldsymbol{w} = \sum_k \lambda_k y_k \, \boldsymbol{\phi}(\boldsymbol{x}_k)$$

$$b = y_k - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_k)$$

for any k s.t. $0 < \lambda_k < C$

$$\boldsymbol{w} \cdot \Phi(\boldsymbol{x}) = k \sum \lambda_k y_k K(\boldsymbol{x}, \boldsymbol{x}_k)$$

$$b = y_k - \sum_i \lambda_i y_i K(\boldsymbol{x}_k, \boldsymbol{x}_i)$$

for any $k \; s.t. \; \lambda_k > 0$

Classify as

sign($\sum_k \lambda_k \, y_k \, [\phi(\mathbf{x}_k) \cdot \phi(\mathbf{x})] + b$)

= sign($\sum_k \lambda_k y_k \, \mathbf{K}(\mathbf{x}_k, \mathbf{x}) + b$)

Nb: never have to deal with $\boldsymbol{\phi}(\boldsymbol{x})$ … only $\mathbf{K}(\mathbf{x}_k, \mathbf{x}_i)$

Classifying using SVMs with Kernels

- Training time: Given $\{ [\mathbf{x}_i , y_i] \}$
  - Choose kernel function $\phi(\cdot)$ of kernel $K(\cdot, \cdot)$
  - Solve dual problem to obtain parameter values λ_i (of support vectors)
  - Compute $b = y_k - \sum_i \lambda_i y_i \mathbf{K}(\mathbf{x_i}, \mathbf{x_k})$ for any k s.t. $0 < \lambda_k < C$

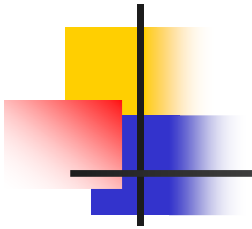
- Performance time, given $\mathbf{x}$, compute (and return):

$$\text{sign}(\sum_k \lambda_k y_k \mathbf{K}(\mathbf{x}_k, \mathbf{x}) + b)$$

What makes a valid kernel?

- A ***sufficient*** (but not necessary) condition is for $K(\cdot, \cdot)$ to behave like a distance metric
  - Non-negative: $K(x,y) \geq 0$
    - $K(x,x) = 0$
  - Symmetric $K(x,y) = K(y,x)$
  - Obeys triangle inequality $K(x,y) + K(y,z) \geq K(x,z)$

- In general, $K(\cdot, \cdot)$ matrix must be symmetric, positive semi-definite
  - Given $\mathbf{X} = \{\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}\}$, $K(\mathbf{X}) = [K(\boldsymbol{x_i}, \boldsymbol{x_j})]$
  - Matrix $K(\boldsymbol{X})$ is positive semi-definite iff $\forall \mathrm{y} \quad \mathrm{y}^{\mathrm{T}}\mathrm{K}(\mathbf{X})\mathrm{y} \geq 0$



Mercer Theorem

- Any symmetric positive definite matrix K can be a kernel matrix
  - ie, $\exists\ \phi(\cdot)$ s.t. $\mathrm{K}(\mathrm{x},\mathrm{y}) = \phi(\mathrm{x}) \cdot \phi(\mathrm{y})$
  - as an inner product matrix in some space

- Matrix $\mathbf{M}$ is:
  - *symmetric* iff $\mathbf{M} = \mathbf{M}^{\mathrm{T}}$
  - *positive definite* iff $\forall y \quad y^T \mathbf{M}\, y \geq 0$

Common Kernels

- Polynomials of degree d

$$K_d^{poly}(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d$$

- Polynomials of degree up to d

$$K_{\leq d}^{poly}(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$$

- Sigmoid

$$K_{\alpha,\beta}^{sigmoid}(\mathbf{u}, \mathbf{v}) = \tanh(\alpha \mathbf{u} \cdot \mathbf{v} + \beta)$$

- Gaussian kernels

$$K_\sigma^{gaussian}(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{|\mathbf{u}-\mathbf{v}|^2}{2\sigma^2}\right)$$

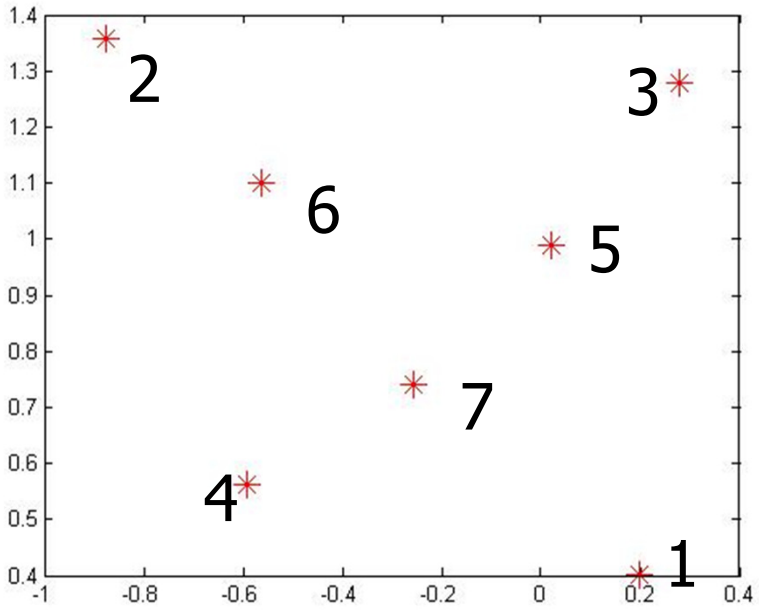
Corresponding $\phi(x)$ is of infinite dimensionality!

Finite example

- Given
  - a kernel function $k: D \times D \to \Re$
  - FINITE set $X = \{x_1, \ldots, x_r\} \subset D$
- construct associated $\phi(\cdot)$
- Build Gram matrix $G \in \Re^{r \times r}$ where

$$G_{i,j} = k(x_i, x_j)$$

- Note G is symmetric, and positive semi-definite



Finite example

Given

- 7 2D-points
- Choose a kernel k

Calculate $G_{i,j} = \exp(-\frac{|x_i - x_j|^2}{10})$

G =

1.0000	0.8131	0.9254	0.9369	0.9630	0.8987	0.9683
0.8131	1.0000	0.8745	0.9312	0.9102	0.9837	0.9264
0.9254	0.8745	1.0000	0.8806	0.9851	0.9286	0.9440
0.9369	0.9312	0.8806	1.0000	0.9457	0.9714	0.9857
0.9630	0.9102	0.9851	0.9457	1.0000	0.9653	0.9862
0.8987	0.9837	0.9286	0.9714	0.9653	1.0000	0.9779
0.9683	0.9264	0.9440	0.9857	0.9862	0.9779	1.0000

From B Poczos

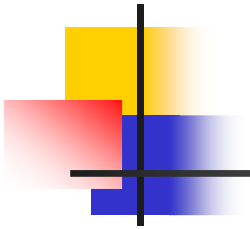
Source of Kernels

- $K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T A \mathbf{x}'$
  - for any positive-semidefinite matrix A
- The set of kernels is closed under certain operations

  $\Rightarrow$ can make complex kernels from simple ones: modularity !
- If K, K' are kernels, then so are:
  - $K + K'$
  - cK for $c>0$
    - $aK+bK'$ for $a,b>0$
  - ...

Source of Kernels

- Can generate new kernels from old:
- If $k_1(\mathbf{x},\mathbf{x}')$, $k_2(\mathbf{x},\mathbf{x}')$ are kernels, then so is:
  - $\mathbf{x}^T A \mathbf{x}'$ – A any positive semidefinite matrix
  - $c\, k_1(\mathbf{x},\mathbf{x}')$ – $c \in \Re^+$
  - $f(\mathbf{x})\, k_1(\mathbf{x},\mathbf{x}')\, f(\mathbf{x}')$ – $f(.)$ any function
  - $q(\, k_1(\mathbf{x},\mathbf{x}')\,)$ – $q(.)$ any poly function w/ coeff's ≥ 0
  - $\exp(\, k_1(\mathbf{x},\mathbf{x}')\,)$
  - $k_1(\mathbf{x},\mathbf{x}') + k_2(\mathbf{x},\mathbf{x}')$
  - $k_a(\mathbf{x}_a,\mathbf{x}'_a) + k_b(\mathbf{x}_b,\mathbf{x}'_b)$ – $\mathbf{x} = (\mathbf{x}_a\ \mathbf{x}_b)$ and $k_a(.,.)$ kernel over "a" space, $k_b(.,.)$ kernel over "b" space
  - …

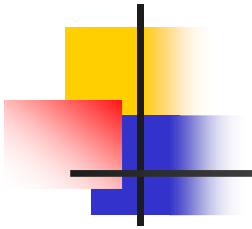


Must Domain be $\Re^n$?

- So far, considering $x \in D = \Re^n$
- But note:

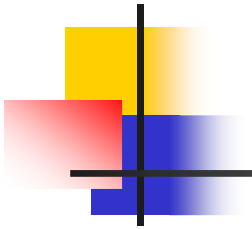
  no computation uses x, just $K(x, y)$

  $\Rightarrow$ so can have domain = strings, graphs, …
- So Kernels for …
  - Text
  - DNA strings
  - Graphs
  - …



Overfitting?

- Huge feature space with kernels, … what about overfitting???
  - Maximizing margin leads to **sparse** set of support vectors
  - Some interesting theory says that SVMs search for **simple** hypothesis with large margin
  - Often robust to overfitting



SVM Performance

- Anecdotally SVMs do work very very well indeed.
  - *Eg1:* The best-known classifier on a well-studied hand-written-character recognition benchmark
  - *Eg2:* Many people doing practical real-world work claim that SVMs have saved them…
    when their other favorite classifiers did poorly.
- Lots of excitement and religious fervor about SVMs as of 2001…
- Still… some practitioners are a little skeptical…

Kernels in Logistic Regression

$$P(Y = 1 \mid x, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \Phi(\mathbf{x}) + b)}}$$

- Define weights in terms of support vectors:

$$\mathbf{w} = \sum_i \alpha_i \Phi(\mathbf{x}_i)$$

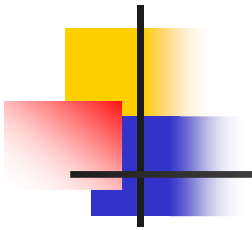
$$P(Y = 1 \mid x, \mathbf{w}) = \frac{1}{1 + e^{-(\sum_i \alpha_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) + b)}}$$

$$= \frac{1}{1 + e^{-(\sum_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b)}}$$

- Derive simple gradient descent rule on α_i

Differences between SVM and Logistic Regression

	SVMs	Logistic Regression
Loss function	Hinge loss	Log-loss
High dimensional features with kernels	**Yes!**	No
Solution sparse	Often yes!	Almost always no!
Semantics of output	"Margin"	Real probabilities

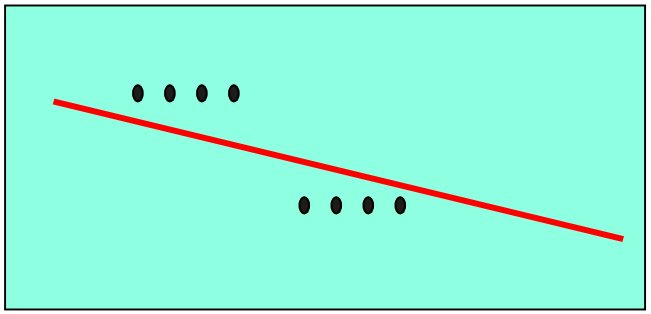


Outline

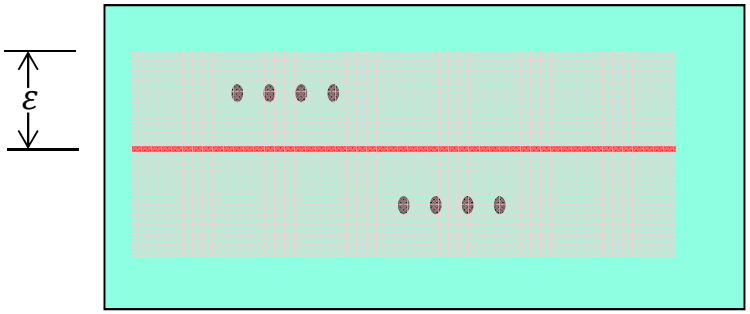
- Foundations
  - ~~Linear Programming;Primal/Dual~~
  - Constrained Optimization
    - Lagrange Multipliers
    - KKT
  - Perceptron Factoids
    - Dual Representation
- "Best" Linear Separator: Max Margin!
- Coping with Non-Linearly Separated Data
  - Slack Variables
- Kernel Trick
- Regression

Simple Example

- What is best linear fit:
  - Minimizes the sum of the squared error: $\sum_i (y_i - t_i)^2$

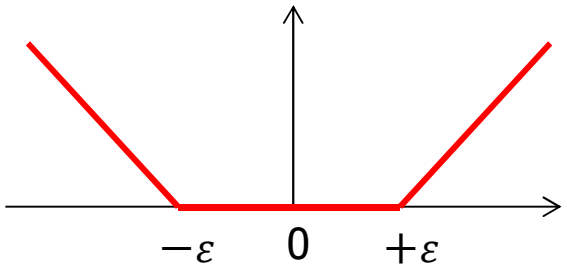


- What if no penalty if within ε

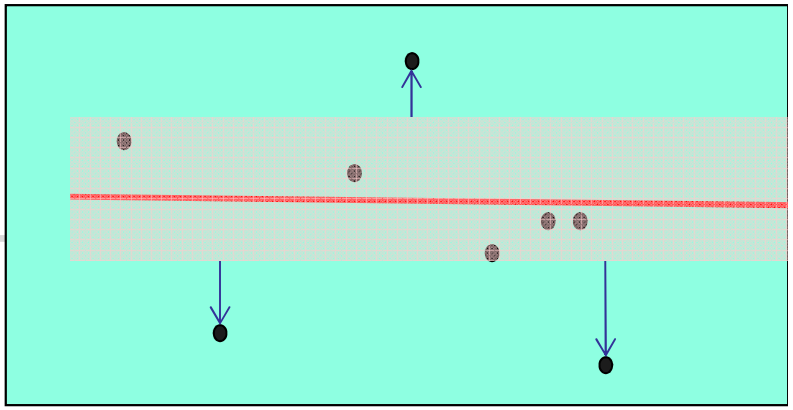


- For each point,

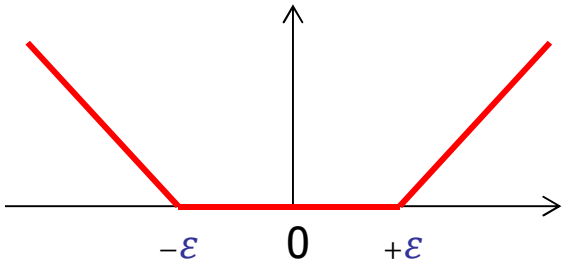
$$E_\varepsilon(y,t) = \begin{cases} 0 & \text{if } |y-t| < \varepsilon \\ |y-t| - \varepsilon & otherwise \end{cases}$$



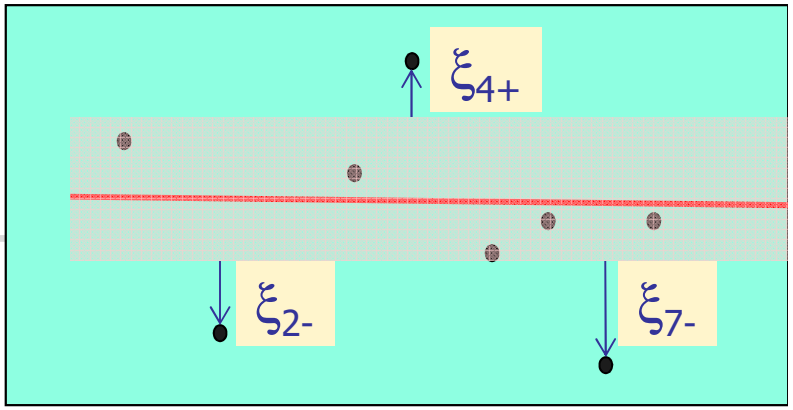
Error Function



- If point is within ε of line: 0 error
- If point is NOT within ε of line:
  Just that additional distance: $|y - t| - \varepsilon$
- $E_\varepsilon(y, t) = \begin{cases} 0 & \text{if } |y - t| < \varepsilon \\ |y - t| - \varepsilon & \text{otherwise} \end{cases}$
- Error is $\sum_i E_\varepsilon(y_i, t_i)$



Loss Function

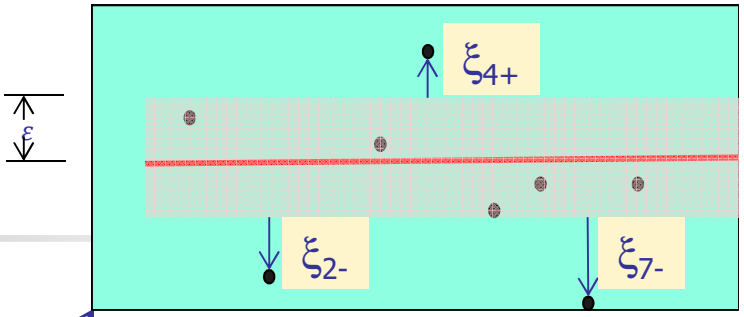


- Want "best" $y(\mathbf{x}) = \boldsymbol{w}^T\mathbf{x}$
- Typically: $L_{LMS}(\boldsymbol{w}) = C\ \sum_i (y_i(\mathbf{x}) - t_i)^2 + \frac{1}{2}|\boldsymbol{w}|^2$
- Instead: $L_{SVR}(\boldsymbol{w}) = C\ \sum_i\ E_\varepsilon(y_i(\mathbf{x}),\ t_i) \quad + \frac{1}{2}|\mathbf{w}|^2$

$$\sum_i \xi_{i+} + \xi_{i-}$$

- wrt $E_\varepsilon(y_i(x),\ t_i)$:
  - No penalty if $t_i \in [\, y_i - \varepsilon\ ,\ y_i + \varepsilon\ ]$
  - Slack variables: $\{\ \xi_{i+}\ ,\ \xi_{i-}\ \}$
    - $t_i \le y_i + \varepsilon + \xi_{i+}$
    - $t_i \ge y_i - \varepsilon - \xi_{i-}$

SVR Formulation



- $\min_{w} L_{SVR}(\boldsymbol{w}) = C\sum_i (\xi_{i+} + \xi_{i-}) + \frac{1}{2}|w|^2$

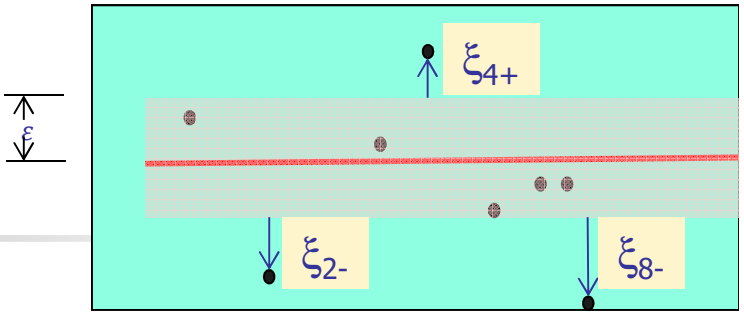
  s.t.

  - $t_i \le y_i + \varepsilon + \xi_{i+}$
  - $t_i \ge y_i - \varepsilon - \xi_{i-}$
  - $\xi_{i+}\ ,\ \xi_{i-} \ge 0$

- Use Lagrange Multipliers $\{a_{n+}, a_{n-}, \mu_{n+}, \mu_{n-}\} \ge 0$

- $$L_{SVR}(\boldsymbol{w}, \dots) = C\sum_i (\xi_{i+} + \xi_{i-}) + \frac{1}{2}|\boldsymbol{w}|^2 - \sum_i(\mu_{i+}\xi_{i+} + \mu_{i-}\xi_{i-}) - \sum_i a_{i+}(\epsilon + \xi_{i+} + y_i - t_i) - \sum_i a_{i-}(\epsilon + \xi_{i-} - y_i + t_i)$$

SVR Solution



$$L_{SVR}(\boldsymbol{w},\dots) = C\sum_i(\xi_{i+}+\xi_{i-}) + \frac{1}{2}|\boldsymbol{w}|^2 - \sum_i(\mu_{i+}\xi_{i+}+\mu_{i-}\xi_{i-})$$

$$-\sum_i a_{i+}(\epsilon+\xi_{i+}+y_i-t_i) - \sum_i a_{i-}(\epsilon+\xi_{i-}-y_i+t_i)$$

- Set derivatives to 0, solve for $\{a_{n+}, a_{n-}, \mu_{n+}, \mu_{n-}\}$
- $L'(a_+,a_-) = \frac{1}{2}\sum_i\sum_j(a_{i+}-a_{i-})(a_j-a_j)\ \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$

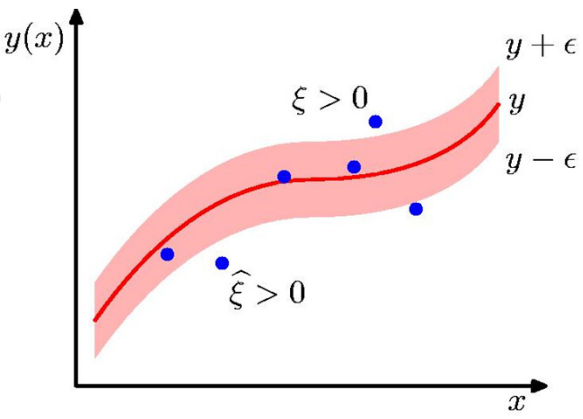
  $-\epsilon\sum_i(a_{i+}-a_{i-}) + \sum_i(a_{i+}-a_{i-})t_i$
  - s.t. $0 \le a_{i+} \le C$, $0 \le a_{i-} \le C$
- Predictions for new **x**:

$$y(\mathbf{x}) = \sum_i(a_{i+}-a_{i-})\ \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x})$$

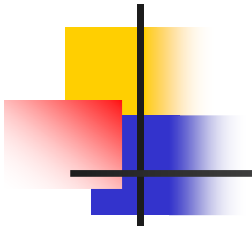
Quadratic Program!!

SVM Regression, con't

$$y(\mathbf{x}) = \sum_i (a_{i+} - a_{i-}) K(\mathbf{x}_i, \mathbf{x})$$

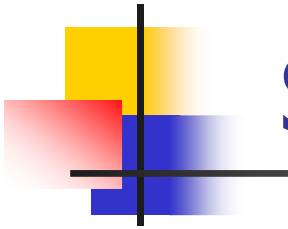


- Can ignore $\mathbf{x}_n$ unless either $a_{n+}>0$ or $a_{n-}>0$
  - $a_{n+}>0$ only if $t_n = y_n + \varepsilon + \xi_{n+}$
    ie, if on upper boundary of ε-tube ($\xi_{n+}=0$)
    or above ($\xi_{n+}>0$)
  - $a_{n-}>0$ only if $t_n = y_n - \varepsilon - \xi_{n-}$
    ie, if on lower boundary of ε-tube ($\xi_{n-}=0$)
    or below ($\xi_{n-}>0$)



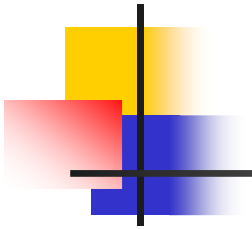
SVM Implementations

- Sequential Minimal Optimization, SMO [Platt]
  - efficient implementation of SVMs
  - in Weka
- SVMlight
  - http://svmlight.joachims.org/



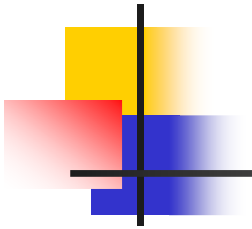
Summary I: **Advantages**

- Systematic implementation through quadratic programming
  - $\exists$ *very efficient* implementations
- Excellent data-dependent *generalization bounds*
- *Regularization* built into cost function
- Statistical performance is *independent* of dimensions of feature space
  - Runtime typically quadratic in the # of **data points**
    - … not #dimensions
  - … less if # support vectors is small
- Theoretically related to widely studied fields of regularization theory and sparse approximation
- Fully adaptive procedures available for determining hyper-parameters



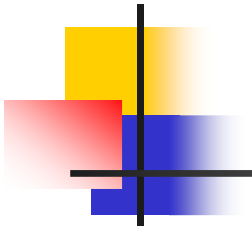
Summary II: **Drawbacks**

- Treatment of non-separable case somewhat heuristic
- Number of support vectors may depend strongly on
  - the kernel type and
  - the hyper-parameters (σ for RBF, k for poly)
- Systematic choice of kernels is difficult (prior information)
  - … some ideas exist …
- Optimization may require clever heuristics for large problems



Summary III: **Extensions**

- Online algorithms
- Systematic choice of kernels using generative statistical models
- Applications to
  - Clustering
  - Non-linear principal component analysis
  - Independent component analysis
- Generalization bounds constantly improving
  - (some even practically useful!)



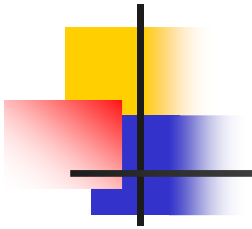
Key SVM Ideas

- Maximize the margin between + and – examples
  - connects to PAC theory
- Sparse:
  Only the support vectors contribute to solution
- Handles non-separable case … with penalties
- Kernels map examples into a new space
  … usually nonlinear
  - Implicitly: dot products in this new space
    (in the "dual" form of the SVM program)
  - Kernels ≠ SVMs

    … but they combine very nicely with SVMs

What You Should Know

- Definition of a maximum margin classifier
- Sparse version: (Linear) SVMs
- How Maximum Margin = a QP problem
- What QP can do for you
  - Better if you know how it works !
- How to deal with noisy (non-separable) data
  - Slack variables
- How to permit "non-linear boundaries"
  - Kernel trick
- How SVM Kernel functions permit us to pretend we're working with a zillion features



References

- An excellent tutorial on VC-dimension and Support Vector Machines:

  C.J.C. Burges. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2(2):955-974, 1998. http://citeseer.nj.nec.com/burges98tutorial.html

- The VC/SRM/SVM Bible:

  Statistical Learning Theory by Vladimir Vapnik, Wiley-Interscience; 1998.

  But not easy reading …