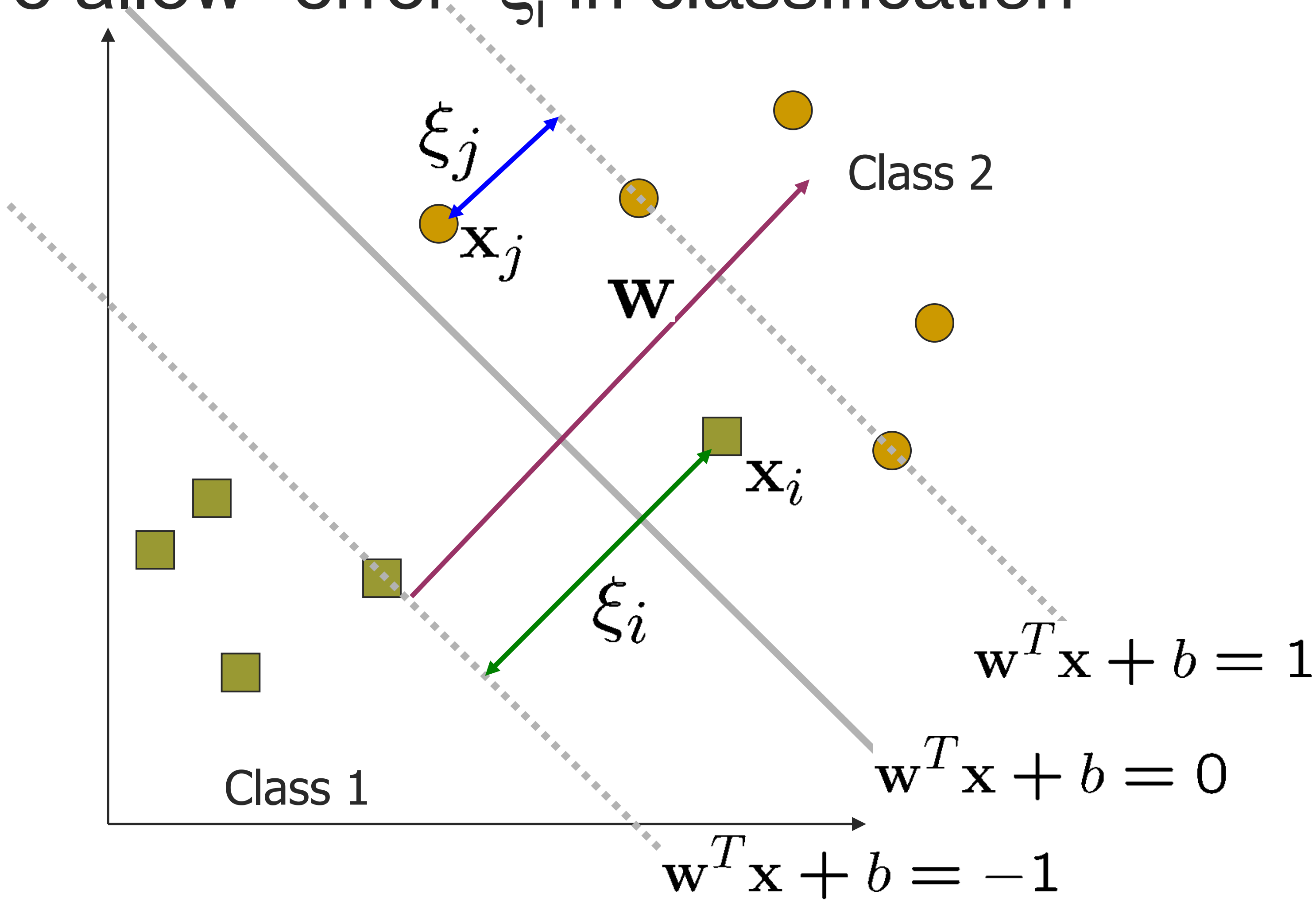
Introduction to Statistical Learning Theory, Kernel Methods, and Support Vector Machines (SVM) Part II

By: **Kaveh Kavousi**
Department of Bioinformatics
IBB (Institute of Biochemistry and Biophysics)
University of Tehran

If Not Linearly Separable

- We allow "error" ξ_i in classification



Soft Margin Hyperplane

- In practice, a separating hyperplane may not exist, e.g., if a high noise level causes a large overlap of the classes. To allow for the possibility of examples violating

$$y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, l$$

we introduce *slack variables*

- $\xi_i \geq 0$ for all $i = 1, \ldots, l$

Slack variables are positive (or zero), local quantities that relax the stiff condition of linear separability, where each training point is seeing the same marginal hyperplane.

Soft Margin Hyperplane

- Define ξ_i=0 if there is no error for x_i
  - ξ_i are just "slack variables" in optimization theory

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

- We want to minimize $\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\xi_i$
  - *C* : tradeoff parameter between error and margin
- The optimization problem becomes

$$\text{minimize} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\xi_i$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Soft Margin Hyperplane

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{l}\xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, l \\ & \xi_i \geq 0, \quad i = 1, \ldots, l \ . \end{aligned}$$

The parameter C controls the relative weighting between the twin goals of making

the $||w||^2$ small (which we saw earlier makes the margin large) and of ensuring

that most examples have functional margin at least 1. we can form the Lagrangian:

$$\mathcal{L}(w,b,\xi,\alpha,r) = \frac{1}{2}w^T w + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i\left[y^{(i)}(x^T w + b) - 1 + \xi_i\right] - \sum_{i=1}^{l} r_i\xi_i.$$

Here, the α_i's and r_i's are our Lagrange multipliers (constrained to be ≥ 0).

Lagrange dual form

$$\mathcal{L}(w,b,\xi,\alpha,r)=\frac{1}{2}w^Tw+C\sum_{i=1}^{l}\xi_i-\sum_{i=1}^{l}\alpha_i\left[y^{(i)}(x^Tw+b)-1+\xi_i\right]-\sum_{i=1}^{l}r_i\xi_i.$$

Here, the α_i's and r_i's are our Lagrange multipliers (constrained to be ≥ 0).

We won't go through the derivation of the dual again in detail, but after setting the derivatives with respect to ω and b to zero as before, substituting them back in, and simplifying, we obtain the following dual form of the problem:

$$\begin{aligned}\max_\alpha \quad & W(\alpha)=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{l}y^{(i)}y^{(j)}\alpha_i\alpha_j\langle x^{(i)},x^{(j)}\rangle \\ \text{s.t.} \quad & 0\leq\alpha_i\leq C,\quad i=1,\ldots,l \\ & \sum_{i=1}^{l}\alpha_i y^{(i)}=0,\end{aligned}$$

Lagrange dual form

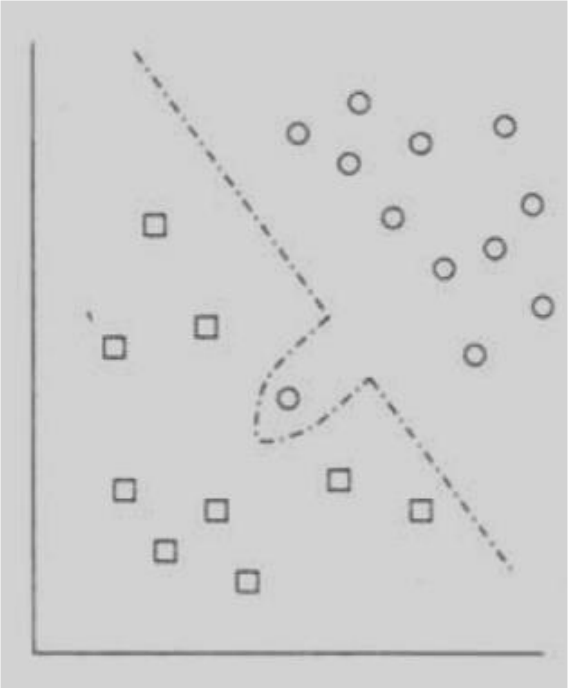
$$
\begin{aligned}
\max_\alpha \quad & W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\
\text{s.t.} \quad & 0 \le \alpha_i \le C, \quad i = 1, \ldots, l \\
& \sum_{i=1}^{l} \alpha_i y^{(i)} = 0,
\end{aligned}
$$

Also, the KKT dual-complementarity conditions are:

$$
\begin{aligned}
\alpha_i = 0 \quad & \Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \ge 1 \\
\alpha_i = C \quad & \Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \le 1 \\
0 < \alpha_i < C \quad & \Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) = 1.
\end{aligned}
$$

Linear SVM: non-separable case (cont'd)

- The constant c controls the trade-off between the margin and misclassification errors.
- Aims to prevent outliers from affecting the optimal hyperplane.



The New Optimization Problem

- The dual of the problem is

$$\text{max. } W(\mu) = \sum_{i=1}^{n} \mu_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \mu_i \mu_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } C \geq \mu_i \geq 0, \sum_{i=1}^{n} \mu_i y_i = 0$$

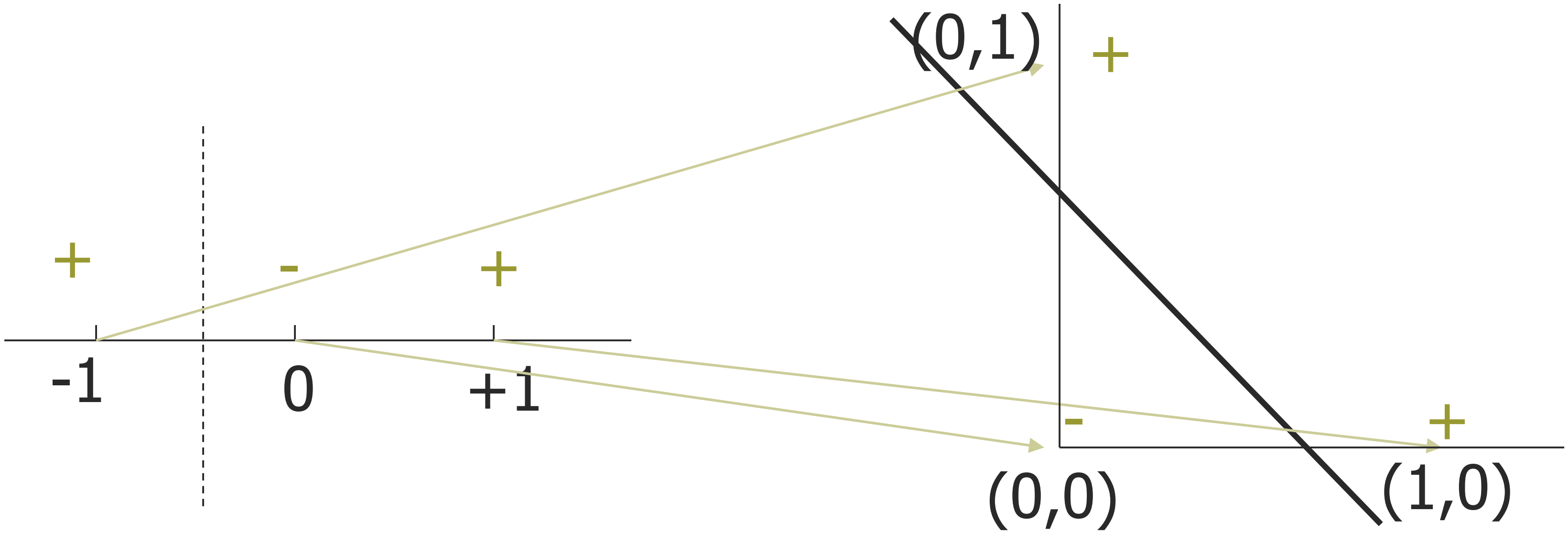
- **w** is also recovered as $\mathbf{w} = \sum_{j=1}^{s} \mu_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- The only difference with the linear separable case is that there is an upper bound C on α_i
- A QP solver can be used to find $\boldsymbol{\mu}_i$'s

Extension to Non-linear Decision Boundary

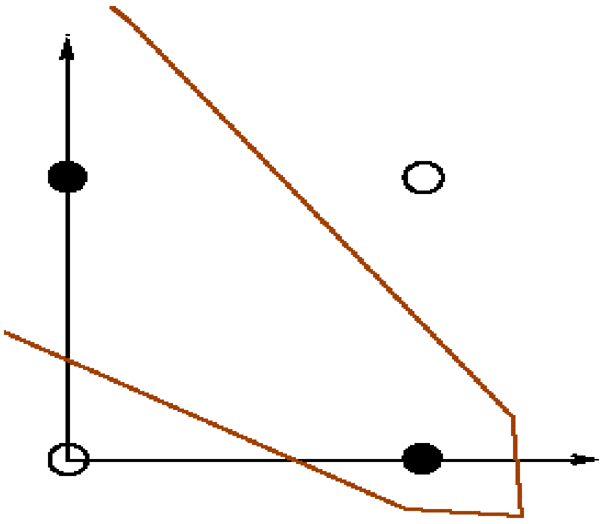
- Key idea: transform $\mathbf{x}_i$ to a higher dimensional space to “make classes linearly separable”
  - Input space: the space $\mathbf{x}_i$ are in
  - Feature space: the space of $\phi(\mathbf{x}_i)$ after transformation
- Why transform?
  - Linear operation in the feature space is equivalent to non-linear operation in input space
  - The classification task can be “easier” with a proper transformation. Example: XOR

Higher Dimensions

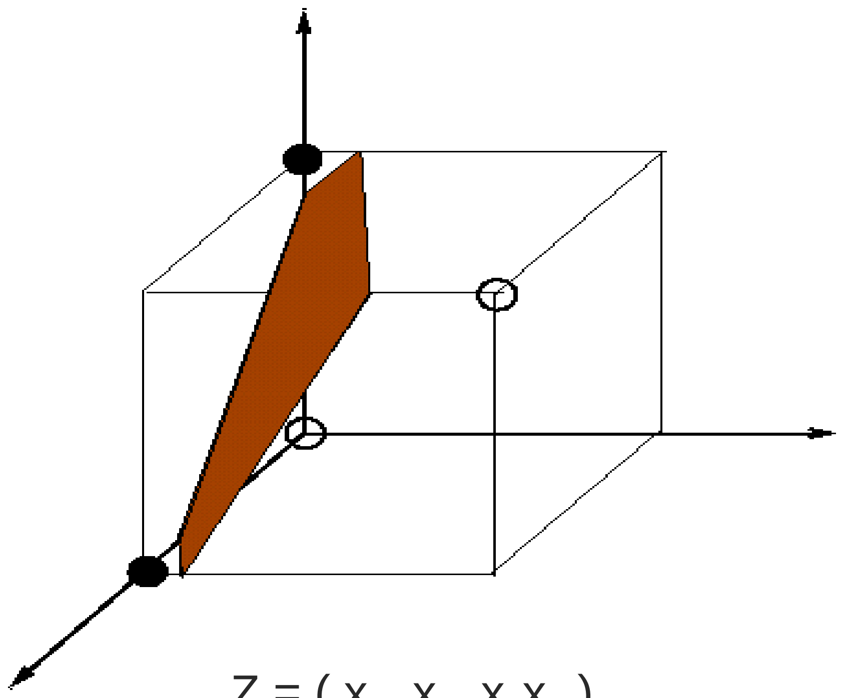
- Project the data to high dimensional space where it is linearly separable and then we can use linear SVM – (Using Kernels)



The XOR problem



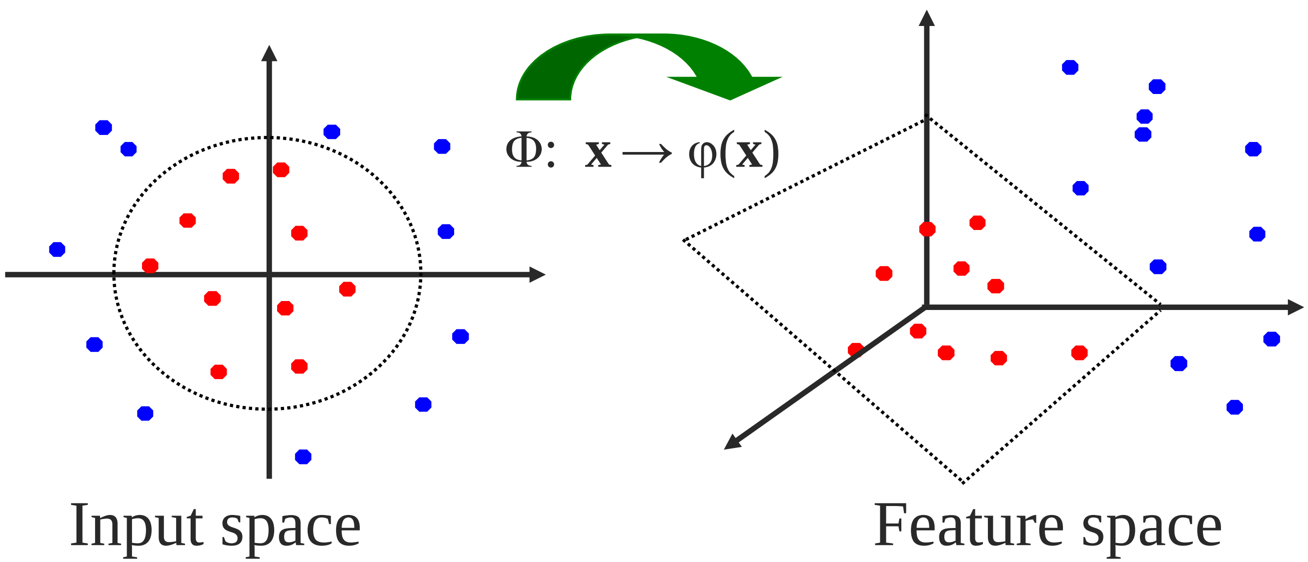
$X = (x_1, x_2)$



$Z = (x_1, x_2, x_1x_2)$

Extension to Non-linear Decision Boundary

- Possible problem of the transformation
  - High computation burden and hard to get a good estimate
- SVM solves these two issues simultaneously
  - Kernel tricks for efficient computation
  - Minimize $||\mathbf{w}||^2$ can lead to a "good" classifier



Nonlinear SVM

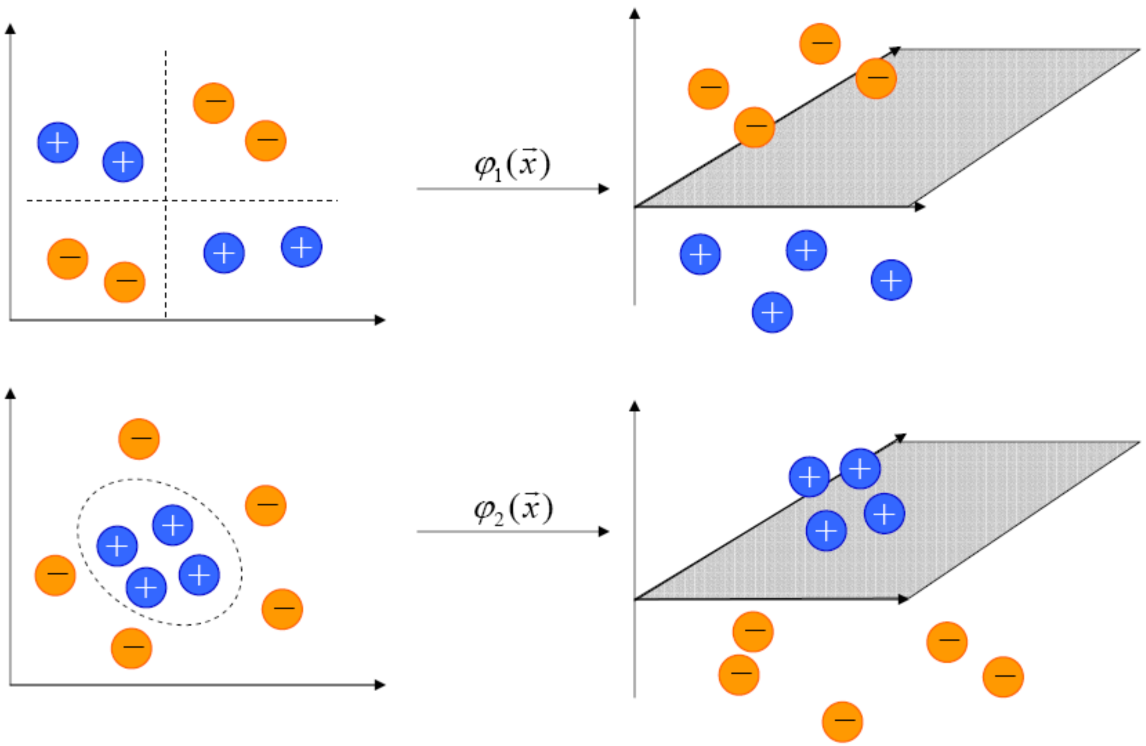
- Extending these concepts to the non-linear case involves mapping the data to a high-dimensional space *h*:

$$\mathbf{x}_k \rightarrow \mathbf{\Phi}(\mathbf{x}_k) = \begin{bmatrix} \varphi_1(\mathbf{x}_k) \\ \varphi_2(\mathbf{x}_k) \\ \dots \\ \varphi_h(\mathbf{x}_k) \end{bmatrix}$$

- Nonlinear Mapping the data to a new (sufficiently high dimensional) space is likely to cast the data linearly separable in that space.

Nonlinear SVM (cont'd)

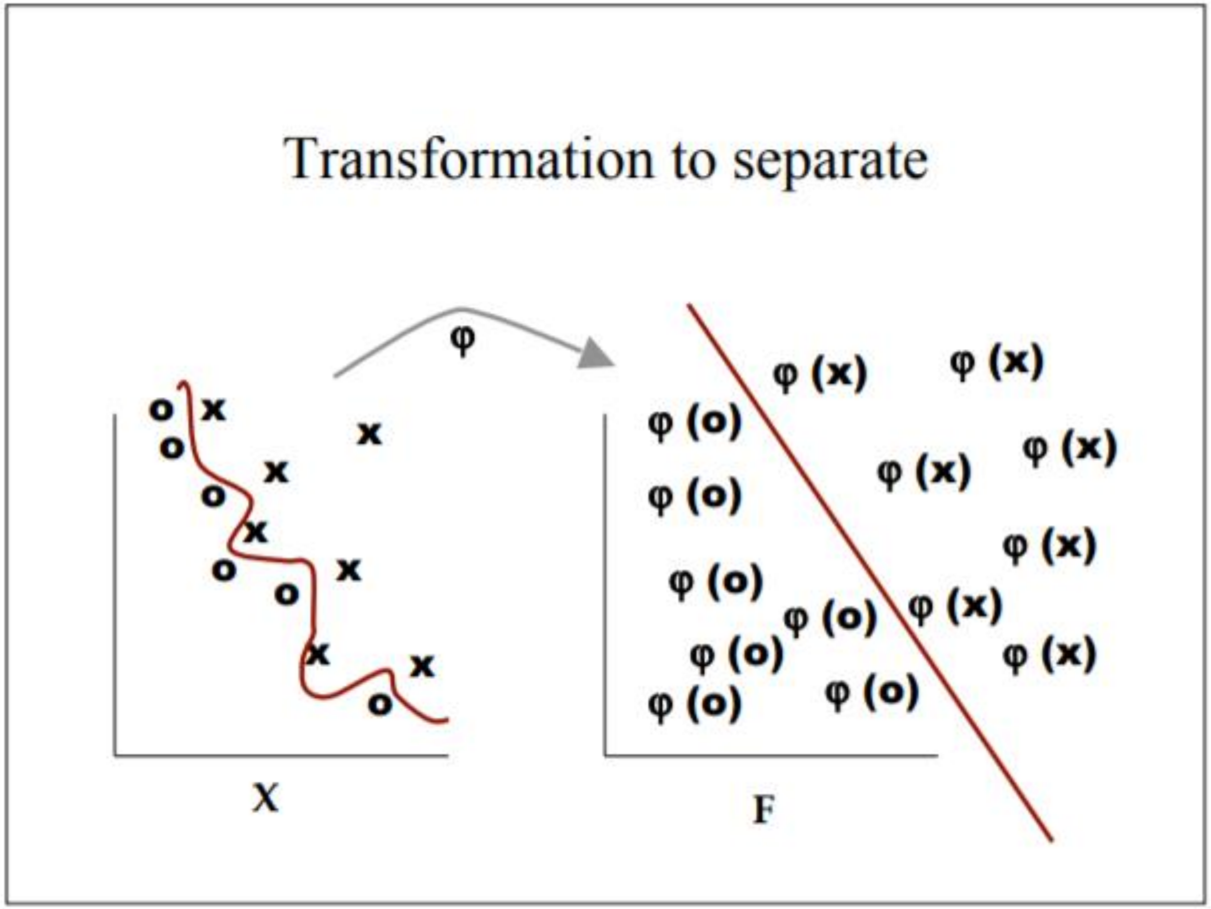
Example:



Linearly Separable in Higher Dimension

Nonlinear SVM (cont'd)

Example:



Nonlinear SVM (cont'd)

$$\langle \Phi(x), \Phi(x^{(i)}) \rangle$$

linear SVM: $$f(x) = \sum_{i=1}^{l} y^{(i)} \alpha_i \langle x, x^{(i)} \rangle + b$$

l is the number of training data points

non-linear SVM: $$f(x) = \sum_{i=1}^{l} y^{(i)} \alpha_i \langle \Phi(x), \Phi(x^{(i)}) \rangle + b$$

Advantage of solving Lagrange dual form over primal problem

classification rule:

Decide ω_1 if $f(\mathbf{x}) > 0$ and ω_2 if $f(\mathbf{x}) < 0$

Nonlinear SVM (cont'd)

non-linear SVM: $$f(x) = \sum_{i=1}^{l} y^{(i)} \alpha_i \langle \Phi(x), \Phi(x^{(i)}) \rangle + b$$

- The disadvantage of this approach is that the mapping

$$\mathbf{x}_k \rightarrow \Phi(\mathbf{x}_k)$$

might be very computationally intensive to compute! $\Phi(\mathbf{x}).\Phi(\mathbf{x}_k)$

- Is there an efficient way to compute?

Quadratic Basis Functions

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \vdots \\ \sqrt{2}x_m \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_m^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \sqrt{2}x_2x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \vdots \\ \sqrt{2}x_{m-1}x_m \end{pmatrix}$$

Constant Term

Linear Terms

Pure Quadratic Terms

Quadratic Cross-Terms

Number of terms (assuming m input dimensions) = (m+2)-choose-2

= (m+2)(m+1)/2

= (as near as makes no difference) $m^2/2$

You may be wondering what those $\sqrt{2}$'s are doing.

- You should be happy that they do no harm
- You'll find out why they're there soon.

Quadratic Dot Products

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

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

Quadratic Dot Products

m is new feature dimension. So, ***a*** and ***b*** are m dimensional vectors.

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

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

Just out of casual, innocent, interest, let's look at another function of ***a*** and ***b***:

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

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

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

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

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

Quadratic Dot Products

m is input dimension. So, ***a*** and ***b*** are m dimensional vectors.

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

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

Just out of casual, innocent, interest, let's look at another function of ***a*** and ***b***:

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

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

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

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

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

They're the same!

And this is only O(m) to compute!

Higher Order Polynomials as Kernel

Poly-nomial	$\phi(x)$	Cost to build Q_{kl} matrix traditionally	Cost if 100 inputs	$\phi(a).\phi(b)$	*Cost to direct calculation*	Cost if 100 inputs
Quadratic	All $m^2/2$ terms up to degree 2	$m^2 n^2/4$	$2{,}500\, n^2$	$(a.b+1)^2$	$mn^2/2$	$50n^2$
Cubic	All $m^3/6$ terms up to degree 3	$m^3 n^2/12$	$83{,}000\, n^2$	$(a.b+1)^3$	$mn^2/2$	$50n^2$
Quartic	All $m^4/24$ terms up to degree 4	$m^4 n^2/48$	$1{,}960{,}000\, n^2$	$(a.b+1)^4$	$mn^2/2$	$50n^2$

n is the number of training data points and
m is input dimension

Polynomial Kernel

$$K(x,y)=(x\,.\,y)^{d}$$

* It can be shown for the case of polynomial kernels that the data is mapped to a space of dimension $h = \binom{m+d-1}{d}$ where m is the original dimensionality.

* Suppose m=256 and $d = 4$, then h=183,181,376 !!

* A dot product in the high dimensional space would require $O(h)$ computations while the kernel requires only $O(m)$ computations.

Choice of Φ is not unique

$$\textit{Example}\text{: consider } x \in R^2, \Phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix} \in R^3, \text{ and } K(x,y) = (x.\,y)^2$$

$$(x.\,y)^2 = (x_1y_1 + x_2y_2)^2$$

$$\Phi(x).\,\Phi(y) = x_1^2y_1^2 + 2x_1y_1x_2y_2 + x_2^2y_2^2 = (x_1y_1 + x_2y_2)^2$$

- Note that neither the mapping $\Phi()$ nor the high dimensional space are unique.

$$\Phi(x) = \frac{1}{\sqrt{2}}\begin{pmatrix} (x_1^2 - x_2^2) \\ 2x_1x_2 \\ (x_1^2 + x_2^2) \end{pmatrix} \in R^3 \quad \text{or} \quad \Phi(x) = \begin{pmatrix} x_1^2 \\ x_1x_2 \\ x_1x_2 \\ x_2^2 \end{pmatrix} \in R^4$$

The kernel trick

- Compute dot products using a kernel function

$$K\left(x, x^{(i)}\right) = \left\langle \Phi(x), \Phi\left(x^{(i)}\right) \right\rangle$$

$$f(x) = \sum_{i=1}^{l} y^{(i)} \alpha_i \left\langle \Phi(x), \Phi\left(x^{(i)}\right) \right\rangle + b$$

$$\Downarrow$$

$$f(x) = \sum_{i=1}^{l} y^{(i)} \alpha_i K\left(x, x^{(i)}\right) + b$$

The kernel trick (cont'd)

- The relationship between the kernel function K and the mapping $\phi(.)$ is

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$$

  - This is known as the kernel trick
- In practice, we specify K, thereby specifying $\phi(.)$ indirectly, instead of choosing $\phi(.)$
- $K(\mathbf{x},\mathbf{y})$ needs to satisfy Mercer condition in order for $\phi(.)$ to exist

The kernel trick (cont'd)

- Comments
  - Kernel functions which can be expressed as a dot product in some space satisfy the Mercer's condition (see Burges' paper)
  - The Mercer's condition does not tell us how to construct Φ() or even what the high dimensional space is.
- Advantages of kernel trick
  - no need to know Φ()
  - computations remain feasible even if the feature space has high dimensionality.

Mercer's condition

In mathematics, a real-valued function *K(x,y)* is said to fulfill **Mercer's condition** if for all integrable functions *g*(*x*) one has

$$\iint K(x,y)g(x)g(y)\,dxdy \geq 0.$$

Examples

The constant function

$$K(x,y) = 1$$

satisfies Mercer's condition, as then the integral becomes by Fubini's theorem

$$\iint g(x)g(y)\,dxdy = \int g(x)\,dx \int g(y)\,dy = \left(\int g(x)\,dx\right)^2$$

which is indeed non-negative.

Example Transformation

- Define the kernel function *K* (**x**,**y**) as

$$K(\mathbf{x},\mathbf{y}) = (1 + x_1 y_1 + x_2 y_2)^2$$

- Consider the following transformation

$$\phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)$$

$$\phi(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}) = (1, \sqrt{2}y_1, \sqrt{2}y_2, y_1^2, y_2^2, \sqrt{2}y_1 y_2)$$

$$\begin{aligned} \langle \phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}), \phi(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}) \rangle &= (1 + x_1 y_1 + x_2 y_2)^2 \\ &= K(\mathbf{x},\mathbf{y}) \end{aligned}$$

- The inner product can be computed by *K* without going through the map $\phi(.)$

Examples of Kernel Functions

- Polynomial kernel with degree *d*

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + 1)^d$$

- Radial basis function kernel with width σ

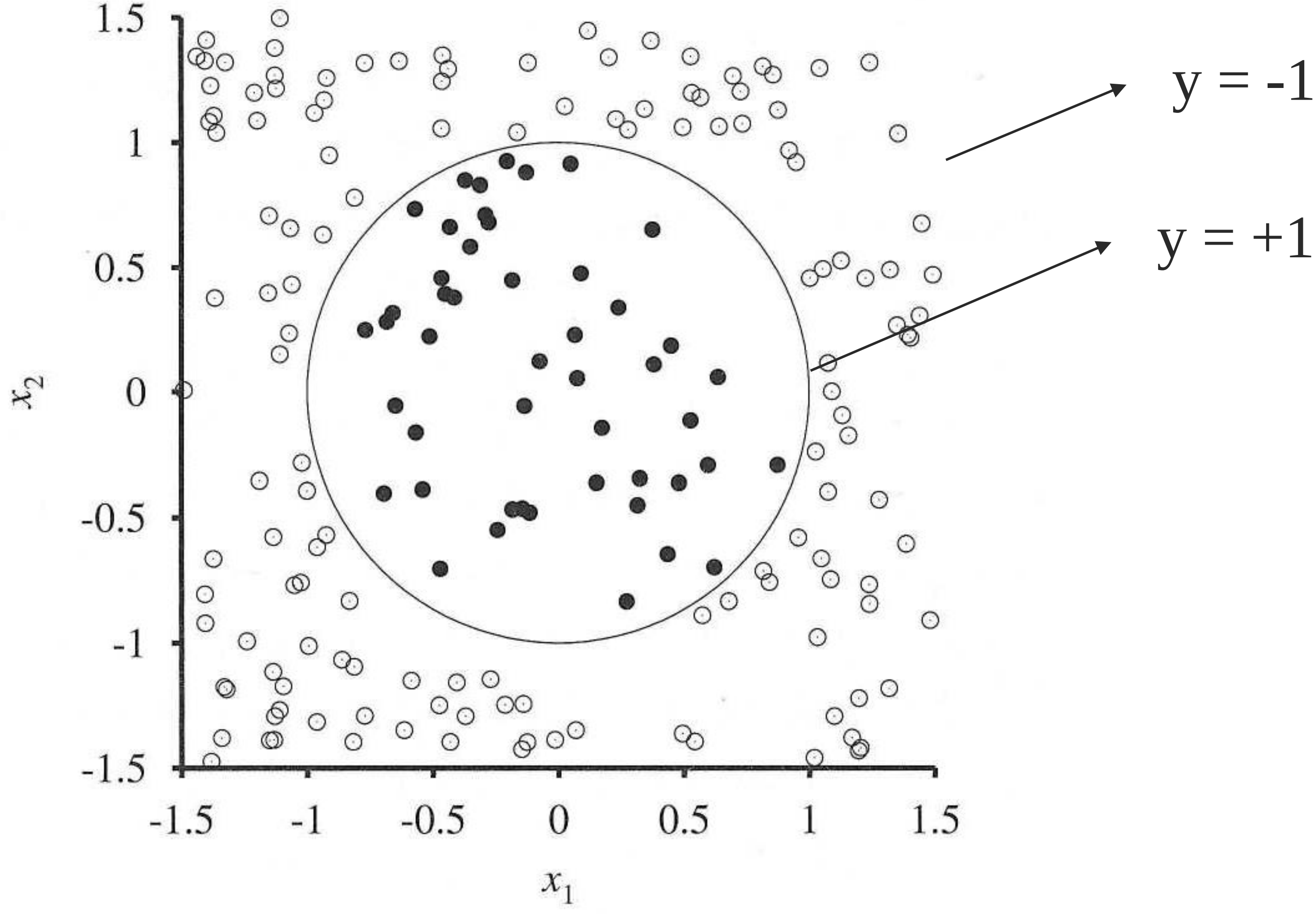
$$K(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||^2/(2\sigma^2))$$

  - Closely related to radial basis function neural networks
- Sigmoid with parameter κ and θ

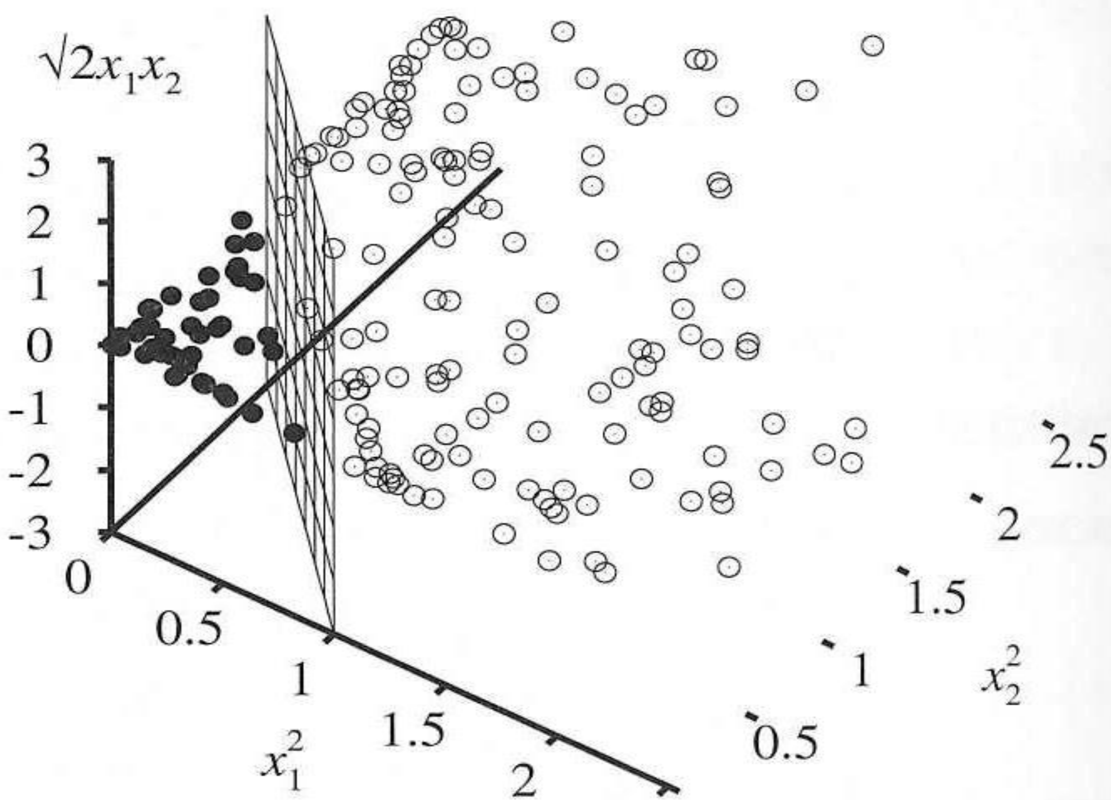
$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa\mathbf{x}^T\mathbf{y} + \theta)$$

  - It does not satisfy the Mercer condition on all κ and θ

(x_1, x_2)



$$(x_1^2, x_2^2, \sqrt{2x_1x_2})$$



Optimization Algorithms

- Most popular optimization algorithms for SVMs are SMO [Platt '99] and SVM^{light} [Joachims' 99], both use *decomposition* to hill-climb over a subset of μ_i's at a time.
- Idea behind SMO
  - Adjusting only 2 μ_i's at each step

$$\text{Maximize } \mathrm{L}(\mu_1, \mu_2) = \sum_{i=1}^{l} \mu_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \mu_i \mu_i y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to } \mu_1 y_1 + \mu_2 y_2 = const \text{ and } 0 \le \mu_i \le C,\ i = 1, 2$$

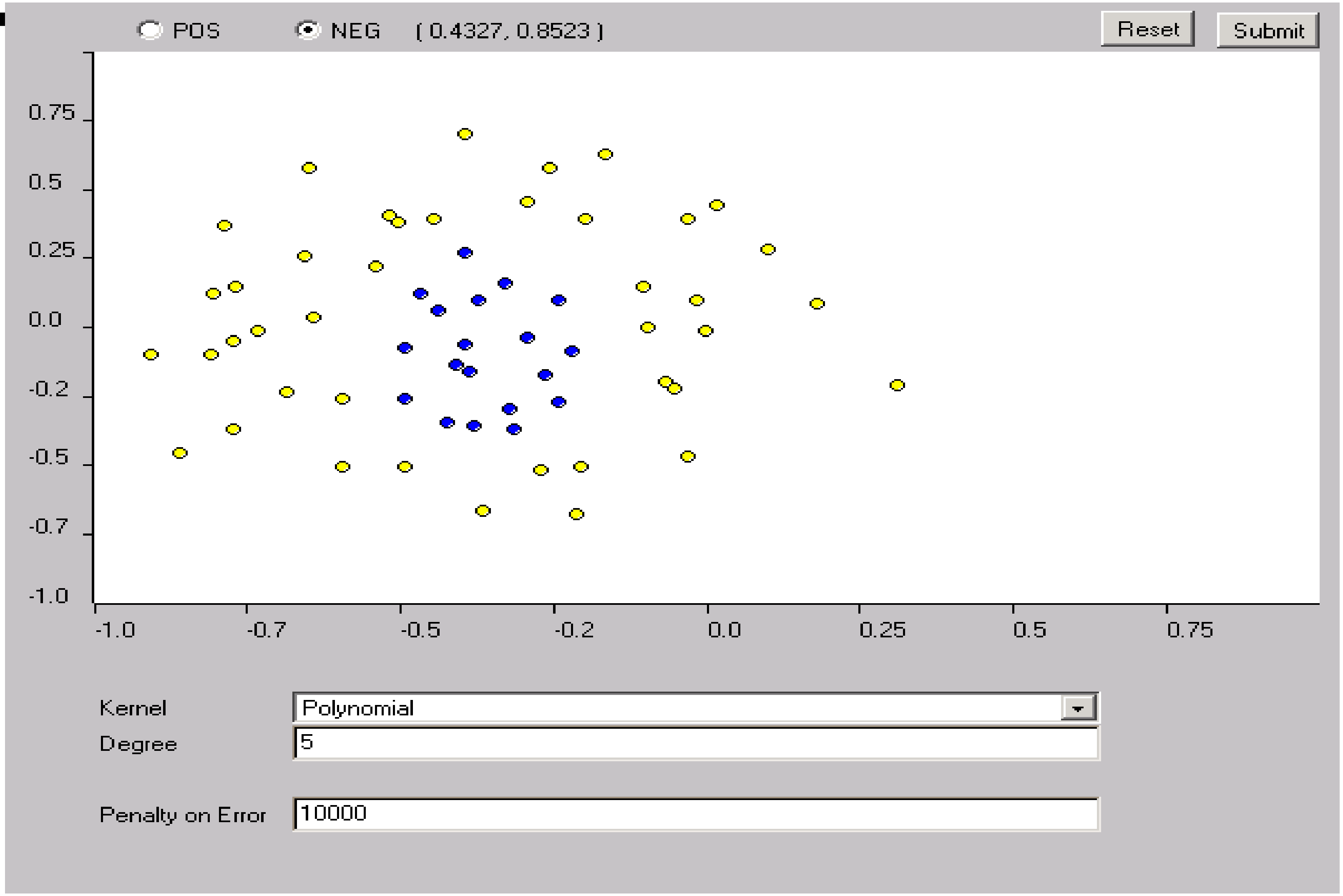
  - All μ_i's are initialized to be zero

SVM vs. Neural Networks

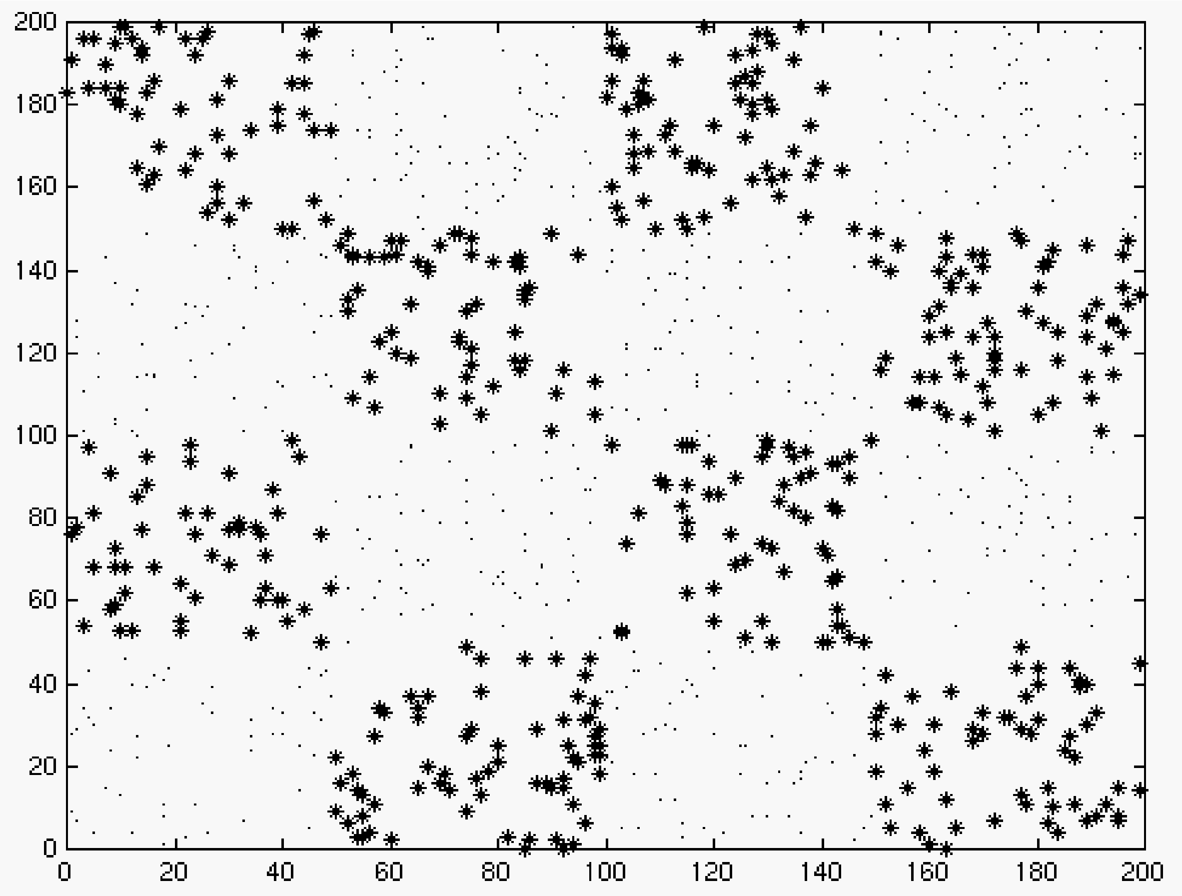
- SVM
  - Relatively new concept
  - Nice Generalization properties
  - Hard to learn – learned in batch modes using QP techniques
  - Using kernels can learn very complex functions

- Neural Networks
  - Generalizes well but doesn't have mathematical foundation
  - Can easily be learnt in incremental fashion
  - To learn complex function – use complex multi layer structure.

Example of Non-linear SVM



A Nonlinear Kernel Application



Checkerboard Training Set: 1000 Points in ***R2***
Separate 486 Asterisks from 514 Dots

Some useful links for libsvm

http://www.csie.ntu.edu.tw/~cjlin/libsvm/

http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/

http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/multilabel/

http://www.csie.ntu.edu.tw/~cjlin/bsvm/

https://sites.google.com/site/kittipat/libsvm_matlab

http://www.cnblogs.com/lvpengms/archive/2012/04/11/2442277.html

References

- [1] B. Schölkopf, A. J. Smola, “Learning with Kernels”, 2001
- [2] C.J.C. Burges, “A Tutorial on Support Vector Machines for Pattern Recognition”, 1998
- [3] P.S. Sastry, “An Introduction to Support Vector Machine”
- [4] J. Platt, “Sequential minimal optimization: A fast algorithm for training support vector machines”, 1999

- **Theorem** Consider some set of m points in $\mathbb{R}^n$. Choose any one of the points as origin. Then the m points can be shattered by oriented hyperplanes if and only if the position vectors of the remaining points are linearly independent.

- **Corollary**: The VC dimension of the set of oriented hyperplanes in $\mathbb{R}^n$ is n+1.

  Proof: we can always choose n + 1 points, and then choose one of the points as origin, such that the position vectors of the remaining n points are linearly independent, but can never choose n + 2 such points (since no n + 1 vectors in $\mathbb{R}^n$ can be linearly independent).