Support Vector Machines

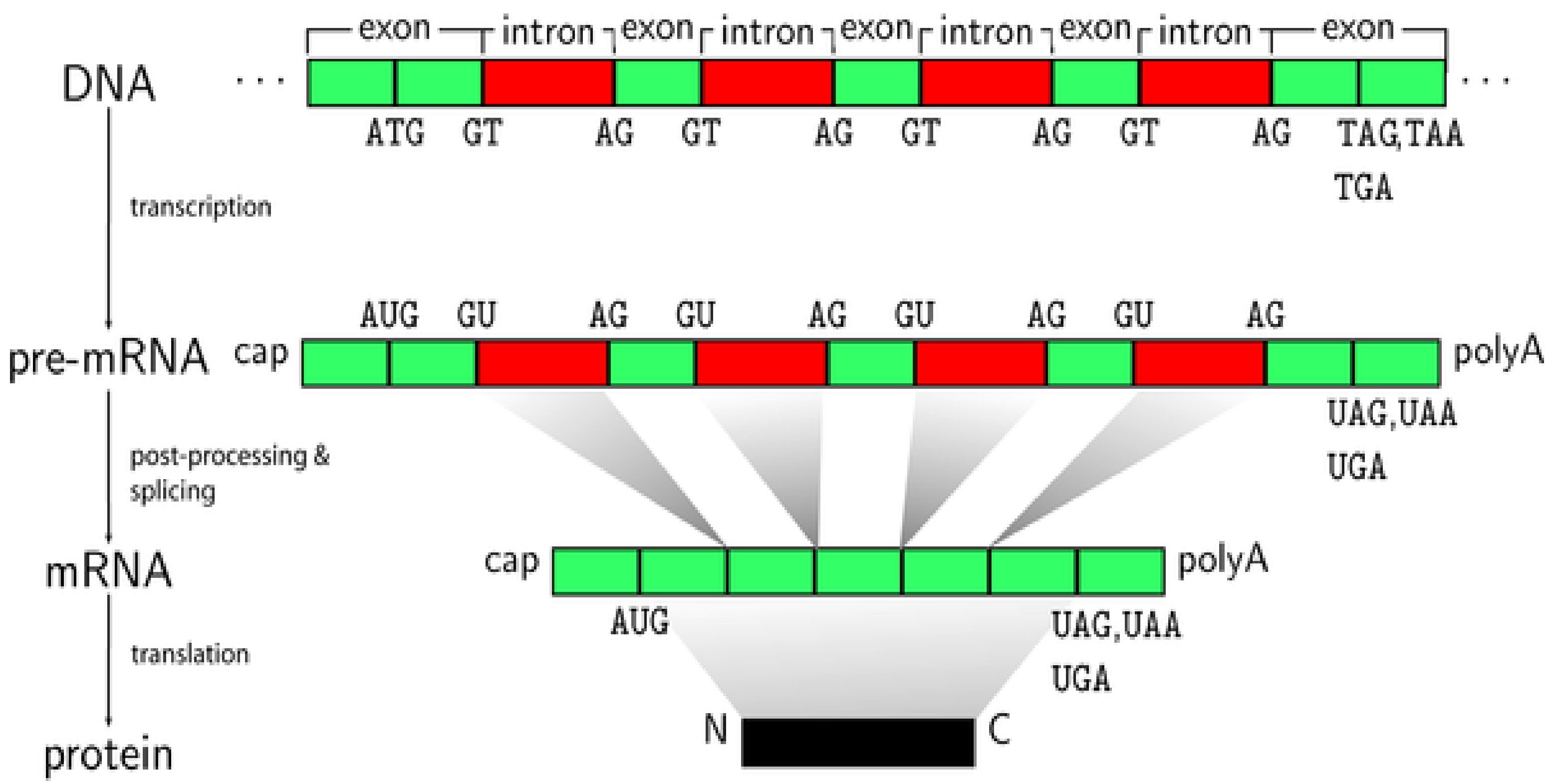
Sourangshu Bhattacharya
sourangshu@cse.iitkgp.ernet.in

Computer Science and Engineering
IIT Kharagpur

Outline

- Classification Problem.
- Linear Classifiers.
  - Max-margin principle.
  - Dual problem.
  - Soft-margin SVMs.
- Non-linear classifiers.
  - Kernel methods.
  - Examples.

mRNA Splicing

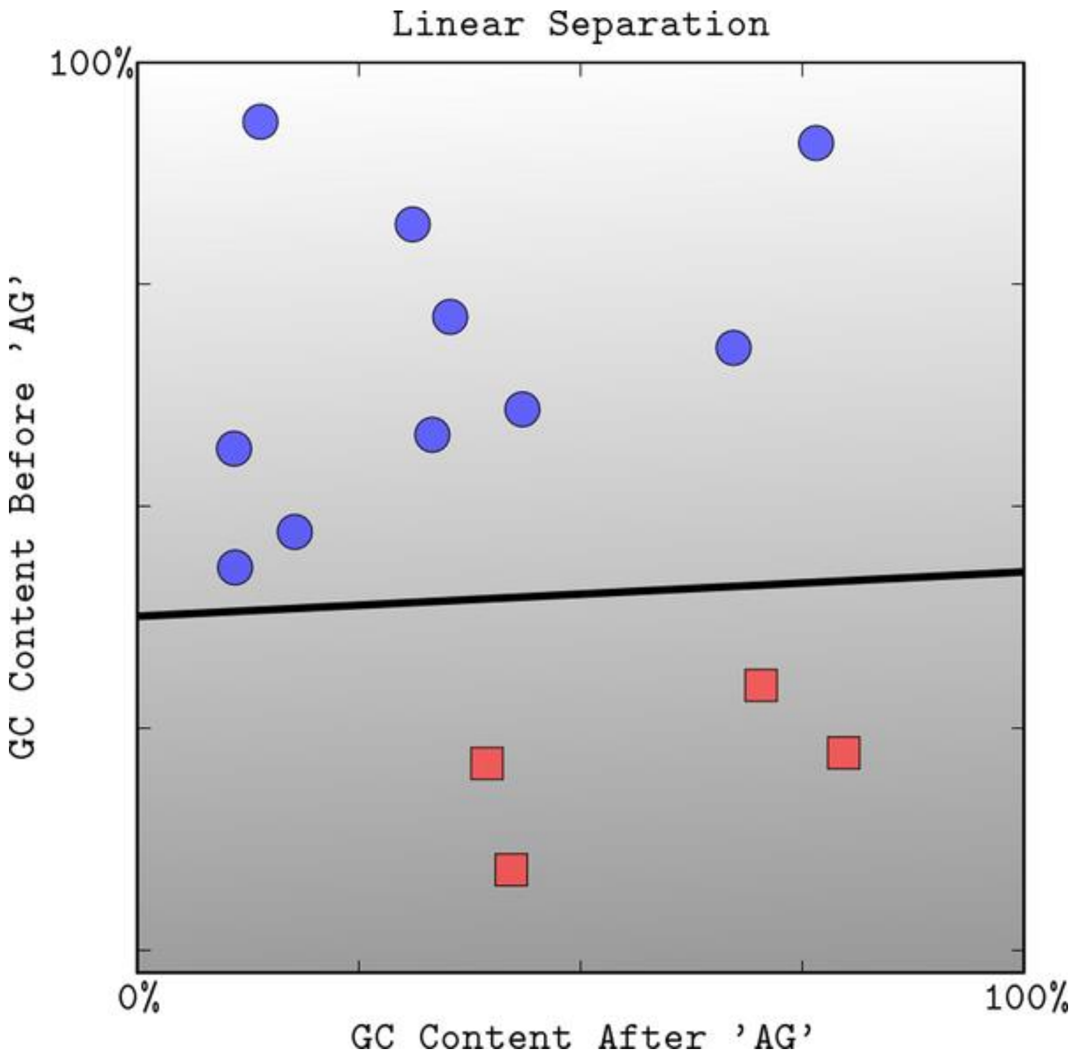


Ben-Hur et al, PLOS computational Biology 4 (2008)

Splice site recognition

- Donor sites contain GT on the intron side.
- Acceptor sites contain AG on the intron side.
- Task is to classify AG as acceptor or not.
- GC content of exons is higher than introns.
- Use GC content before and after AG to classify it as acceptor.

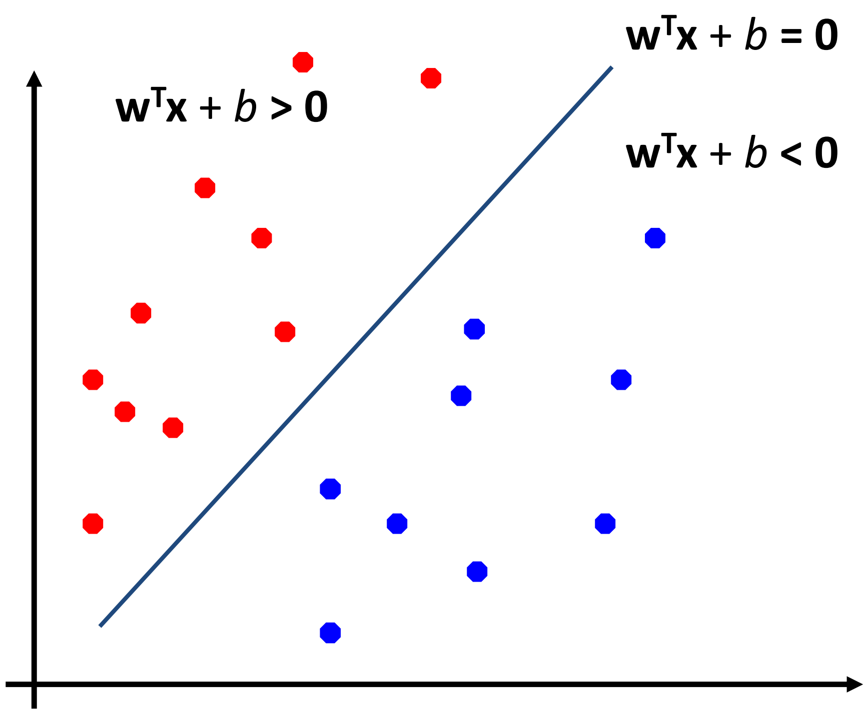
Slice site recognition



Ben-Hur et al,
PLOS computational Biology,
4 (2008)

Linear Separators

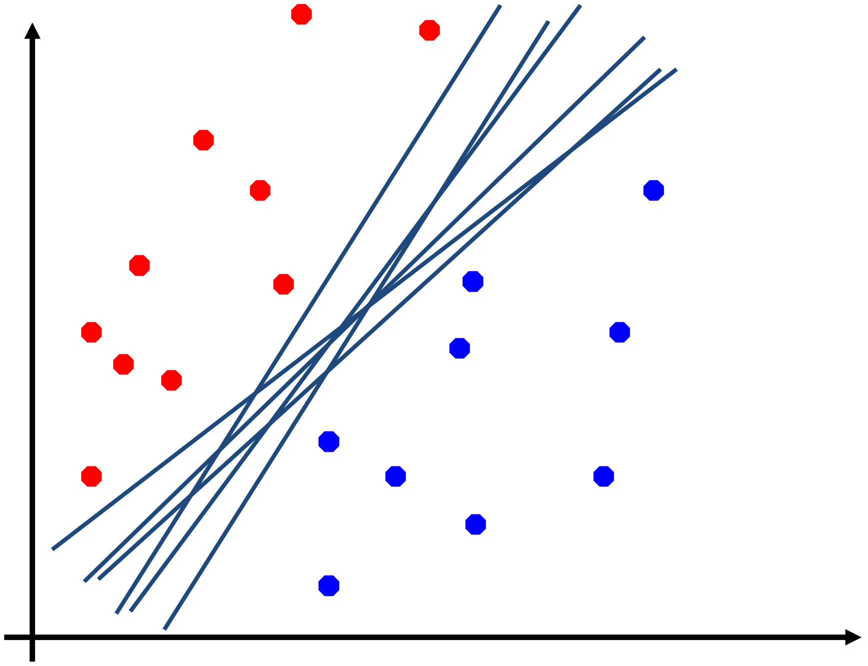
- Binary classification can be viewed as the task of separating classes in feature space:



$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$

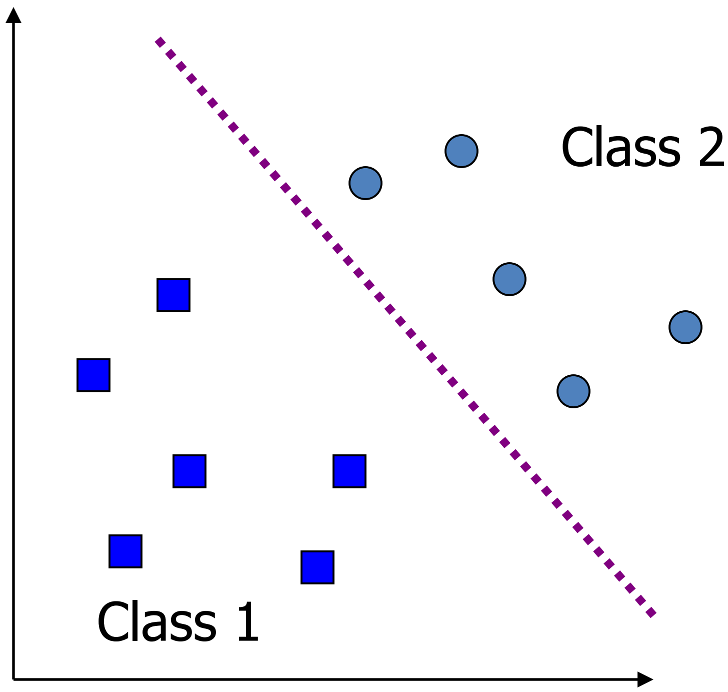
Linear Separators

- Which of the linear separators is optimal?

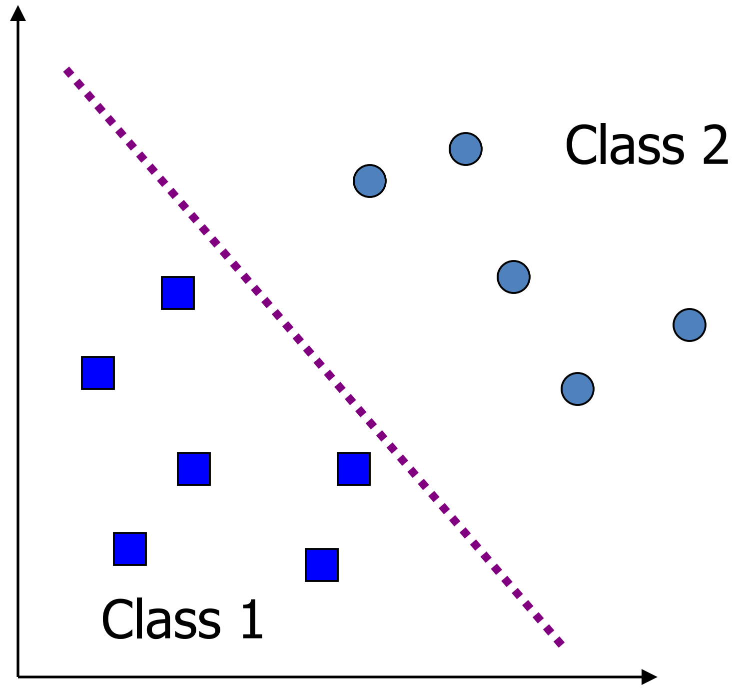


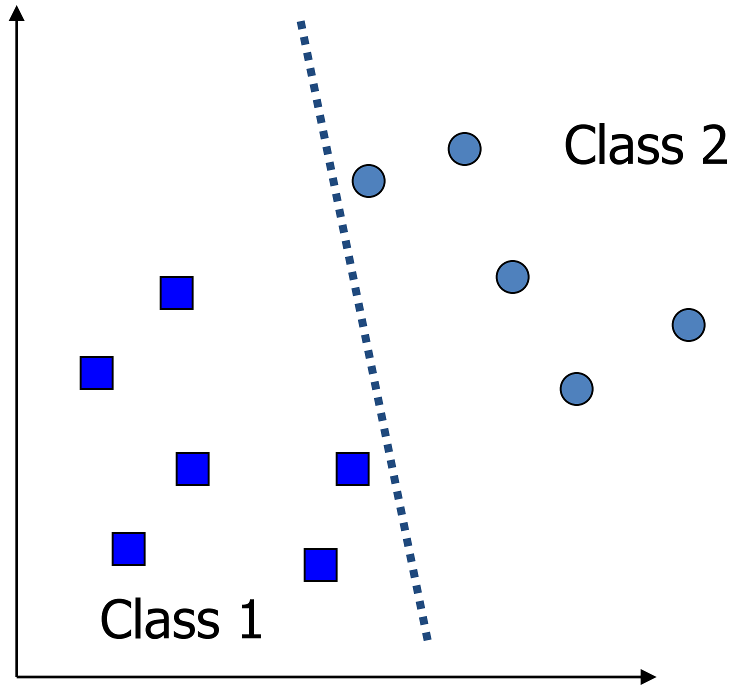
What is a good Decision Boundary?

- Many decision boundaries!
  - The Perceptron algorithm can be used to find such a boundary
- Are all decision boundaries equally good?



Examples of Bad Decision Boundaries





Finding the Decision Boundary

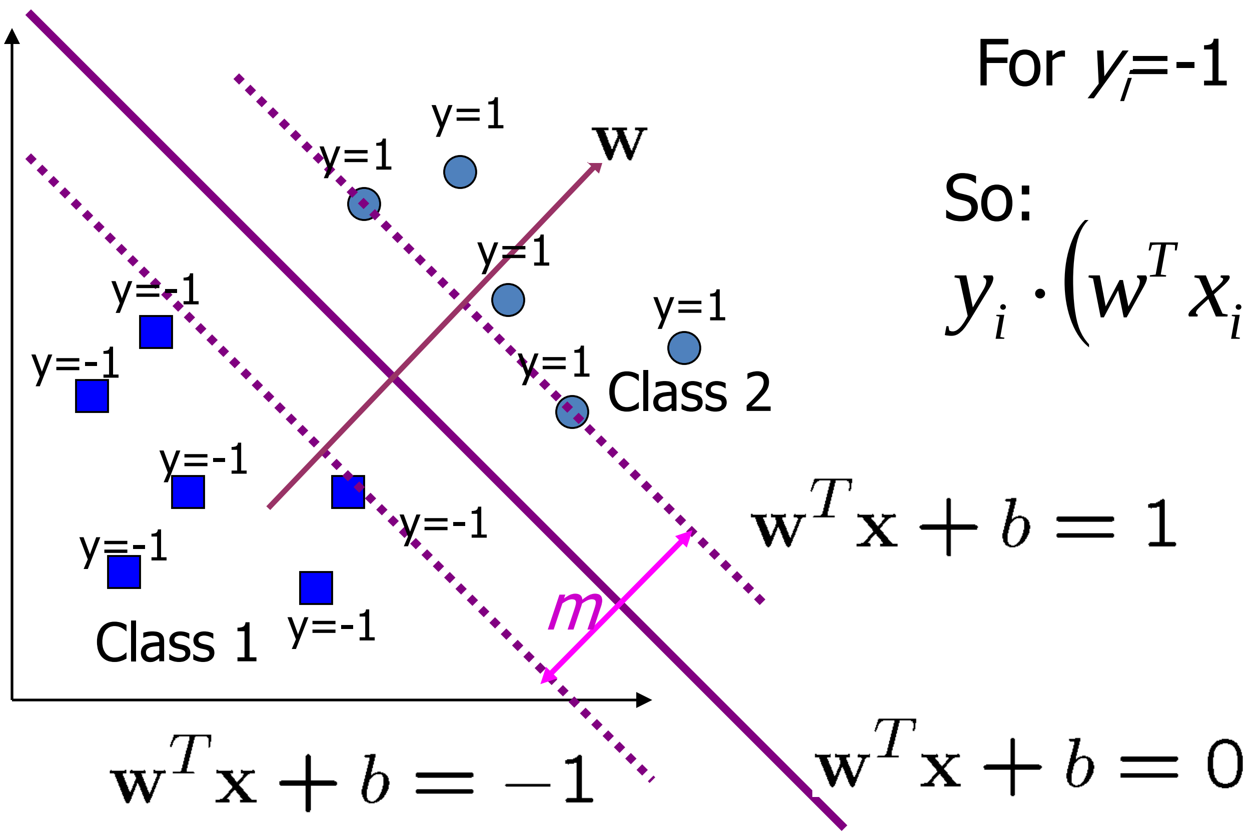
- Let $\{x_1, \ldots, x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i

For $y_i=1$ $\quad w^T x_i + b \geq 1$

For $y_i=-1$ $\quad w^T x_i + b \leq -1$

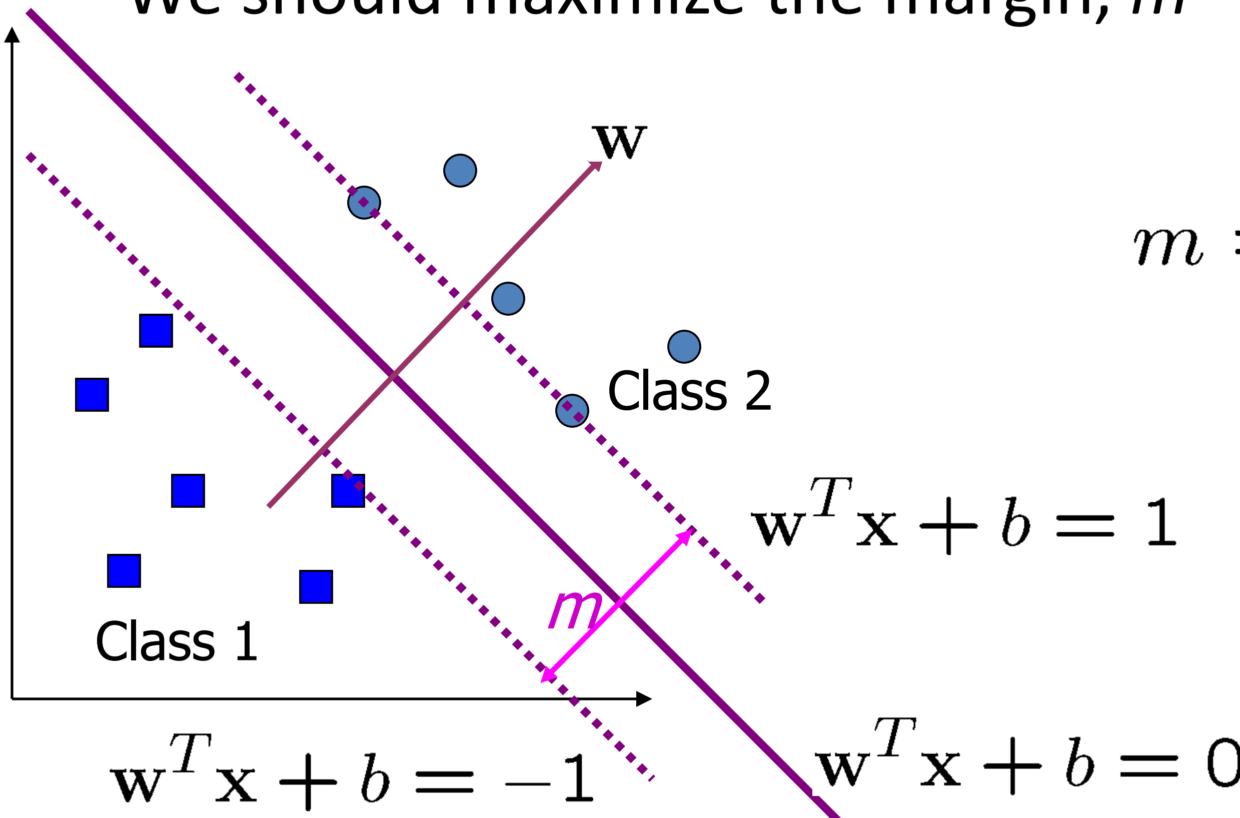
So:

$$y_i \cdot \left(w^T x_i + b\right) \geq 1, \forall \left(x_i, y_i\right)$$



Large-margin Decision Boundary

- The decision boundary should be as far away from the data of both classes as possible
  - We should maximize the margin, *m*



$$m = \frac{2}{||\mathbf{w}||}$$

Finding the Decision Boundary

- The decision boundary should classify all points correctly $\Rightarrow$

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \qquad \forall i$$

- The decision boundary can be found by solving the following constrained optimization problem

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i$$

- This is a constrained optimization problem. Solving it requires to use Lagrange multipliers

Finding the Decision Boundary

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$

$$\text{subject to } 1-y_i(\mathbf{w}^T\mathbf{x}_i+b) \leq 0 \qquad \text{for } i = 1, \ldots, n$$

- The Lagrangian is

$$\mathcal{L} = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \alpha_i \left(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\right)$$

  - $\alpha_i \geq 0$
  - Note that $||\mathbf{w}||^2 = \mathbf{w}^T\mathbf{w}$

Gradient with respect to *w* and *b*

- Setting the gradient of $\mathcal{L}$ w.r.t. **w** and b to zero, we have

$$L = \frac{1}{2} w^T w + \sum_{i=1}^{n} \alpha_i \left(1 - y_i \left(w^T x_i + b\right)\right) =$$

$$= \frac{1}{2} \sum_{k=1}^{m} w^k w^k + \sum_{i=1}^{n} \alpha_i \left(1 - y_i \left(\sum_{k=1}^{m} w^k x_i{}^k + b \right) \right)$$

n: no of examples, m: dimension of the space

$$\begin{cases} \dfrac{\partial L}{\partial w^k} = 0, \forall k \\ \dfrac{\partial L}{\partial b} = 0 \end{cases}$$

$$\mathbf{w} + \sum_{i=1}^{n} \alpha_i (-y_i) \mathbf{x}_i = \mathbf{0} \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

The Dual Problem

- If we substitute $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$ to $\mathcal{L}$, we have

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T \sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i \left(1 - y_i(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i + b)\right)$$

$$= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i y_i \sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i - b \sum_{i=1}^{n} \alpha_i y_i$$

$$= -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i$$

Since $\sum_{i=1}^{n} \alpha_i y_i = 0$

- This is a function of α_i only

The Dual Problem

- The new objective function is in terms of α_i only
- It is known as the dual problem: if we know **w**, we know all α_i; if we know all α_i, we know **w**
- The original problem is known as the primal problem
- The objective function of the dual problem needs to be maximized (comes out from the KKT theory)
- The dual problem is therefore:

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

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

Properties of α_i when we introduce the the Lagrange multipliers

The result when we differentiate the original Lagrangian w.r.t. b

The Dual Problem

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

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

- This is a quadratic programming (QP) problem
  - A global maximum of α_i can always be found
- **w** can be recovered by $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$

Characteristics of the Solution

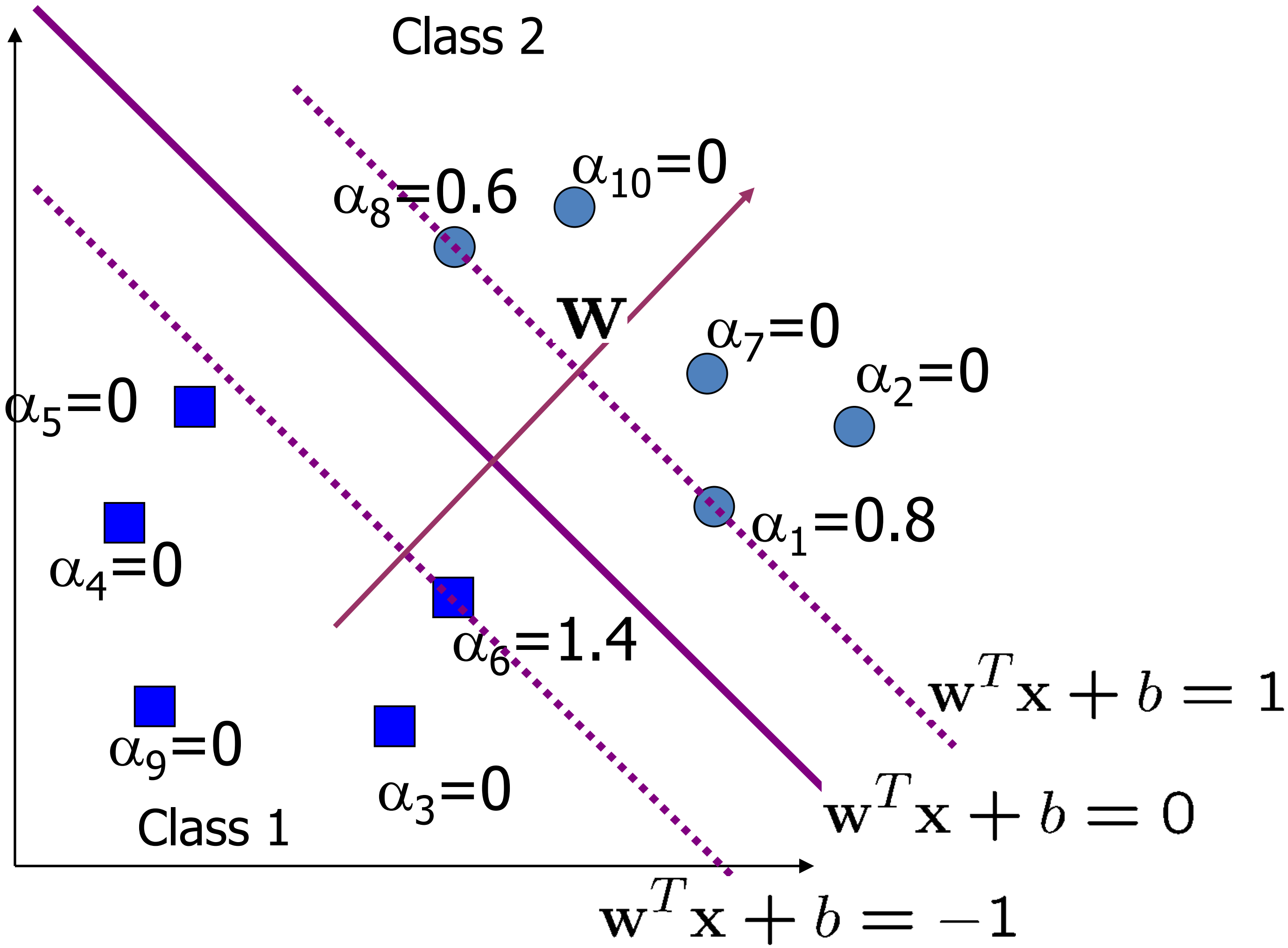
- Many of the α_i are zero
  - **w** is a linear combination of a small number of data points
  - This "sparse" representation can be viewed as data compression as in the construction of knn classifier

- $\mathbf{x}_i$ with non-zero α_i are called support vectors (SV)
  - The decision boundary is determined only by the SV
  - Let t_j (j=1, ..., s) be the indices of the s support vectors. We can write

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

  - Note: **w** need not be formed explicitly

A Geometrical Interpretation



Characteristics of the Solution

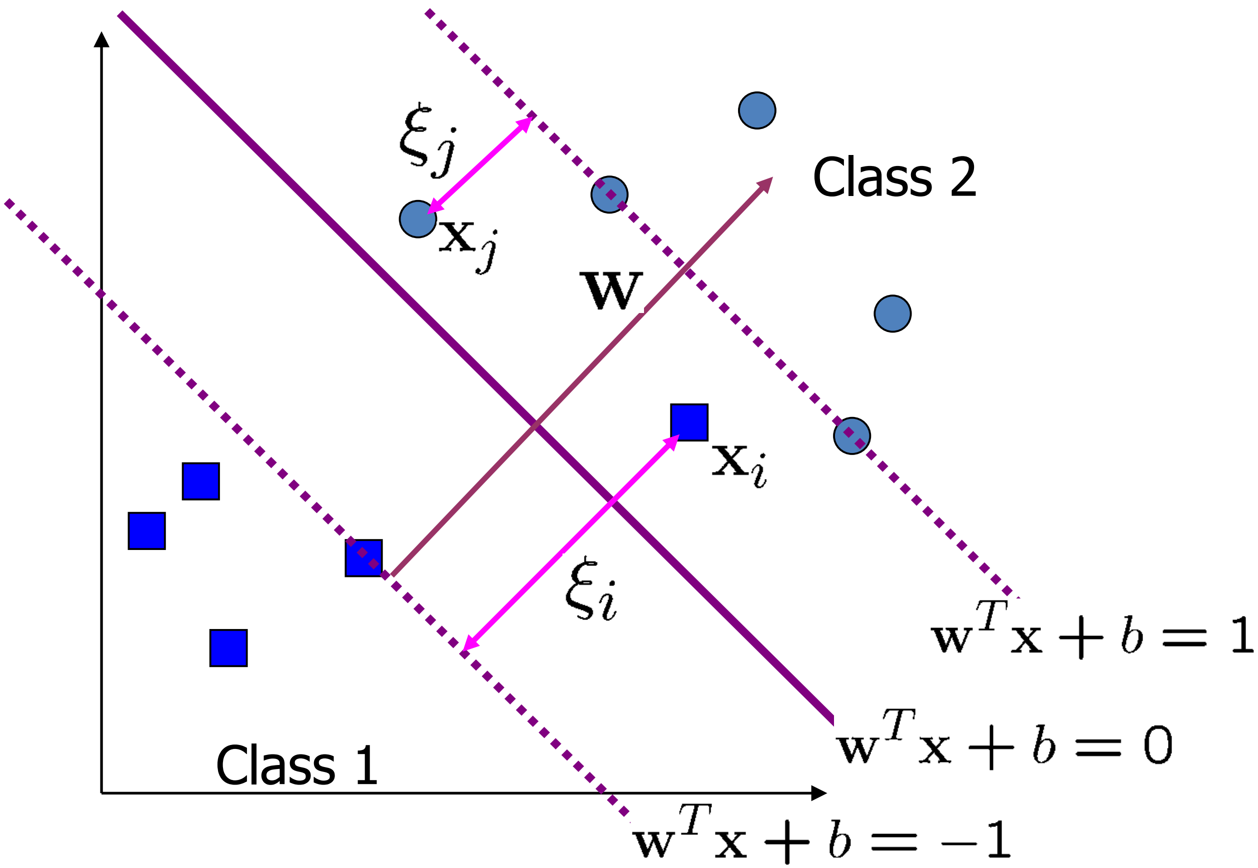
- For testing with a new data **z**
  - Compute $\mathbf{w}^T\mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$ and classify **z** as class 1 if the sum is positive, and class 2 otherwise
  - Note: **w** need not be formed explicitly

The Quadratic Programming Problem

- Many approaches have been proposed
  - Loqo, cplex, etc. (see http://www.numerical.rl.ac.uk/qp/qp.html)
- Most are "interior-point" methods
  - Start with an initial solution that satisfies the constraints
  - Improve this solution by optimizing the objective function
  - For SVM, sequential minimal optimization (SMO) is the most popular
  - A QP with two variables is trivial to solve
  - Each iteration of SMO picks a pair of (α_i,α_j) and solve the QP with these two variables; repeat until convergence
- In practice, we can just regard the QP solver as a "black-box" without bothering how it works

Non-linearly Separable Problems

- We allow "error" ξ_i in classification; it is based on the output of the discriminant function $\boldsymbol{w}^T\boldsymbol{x}+b$
- ξ_i approximates the number of misclassified samples



Soft Margin Hyperplane

- The new conditions become

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

  - ξ_i are "slack variables" in optimization
  - Note that ξ_i=0 if there is no error for $\mathbf{x}_i$
  - ξ_i is an upper bound of the number of errors
- We want to minimize

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

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

- *C* : tradeoff parameter between error and margin

The Optimization Problem

$$L = \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \alpha_i \left(1 - \xi_i - y_i \left(w^T x_i + b\right)\right) - \sum_{i=1}^{n} \mu_i \xi_i$$

With α and μ Lagrange multipliers, POSITIVE

$$\frac{\partial L}{\partial w_j} = w_j - \sum_{i=1}^{n} \alpha_i y_i x_{ij} = 0$$

$$\vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i = 0$$

$$\frac{\partial L}{\partial \xi_j} = C - \alpha_j - \mu_j = 0$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} y_i \alpha_i = 0$$

The Dual Problem

$$L = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \vec{x}_i^{\,T}\vec{x}_j + C\sum_{i=1}^{n}\xi_i +$$

$$+\sum_{i=1}^{n}\alpha_i\left(1-\xi_i - y_i\left(\sum_{j=1}^{n}\alpha_j y_j x_j^{\,T} x_i + b\right)\right) - \sum_{i=1}^{n}\mu_i\xi_i$$

With $\sum_{i=1}^{n} y_i\alpha_i = 0$ and $C = \alpha_j + \mu_j$

$$L = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \vec{x}_i^{\,T}\vec{x}_j + \sum_{i=1}^{n}\alpha_i$$

The Optimization Problem

- The dual of this new constrained optimization problem is

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

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

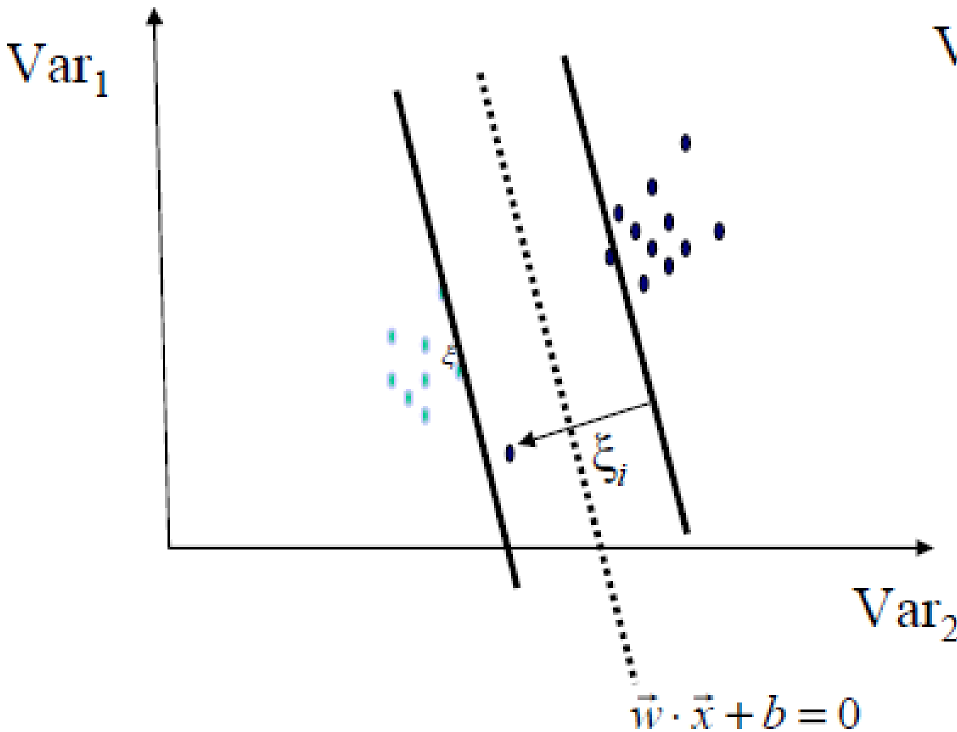
- New constraints derived from $C = \alpha_j + \mu_j$ since μ and α are positive.
- **w** is recovered as $\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- This is very similar to the optimization problem in the linear separable case, except that there is an upper bound *C* on α_i now
- Once again, a QP solver can be used to find α_i

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

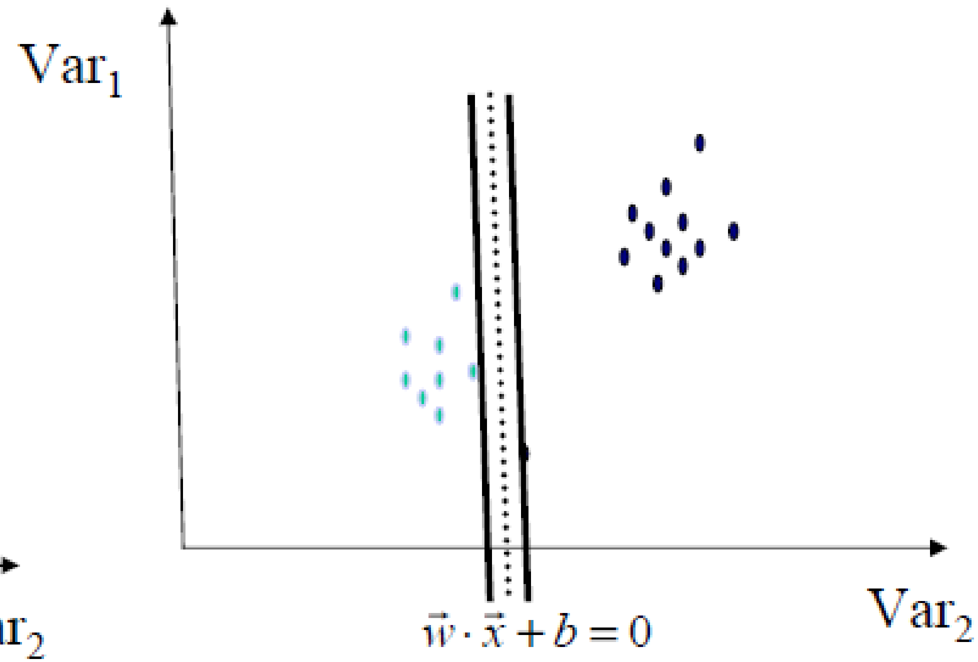
- The algorithm try to keep ξ null, maximising the margin
- The algorithm does not minimise the number of error. Instead, it minimises the sum of distances fron the hyperplane

- When C increases the number of errors tend to lower. At the limit of C tending to infinite, the solution tend to that given by the hard margin formulation, with 0 errors

Soft margin is more robust to outliers



Soft Margin SVM



Hard Margin SVM

Extension to Non-linear Decision Boundary

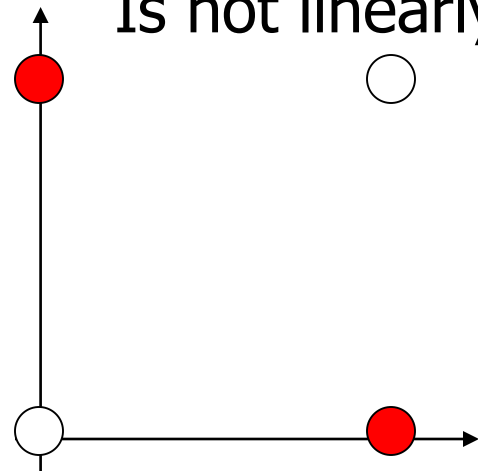
- So far, we have only considered large-margin classifier with a linear decision boundary
- How to generalize it to become nonlinear?
- Key idea: transform $\mathbf{x}_i$ to a higher dimensional space to "make life easier"
  - Input space: the space the point $\mathbf{x}_i$ are located
  - Feature space: 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
  - Classification can become easier with a proper transformation. In the XOR problem, for example, adding a new feature of x_1x_2 make the problem linearly separable

XOR

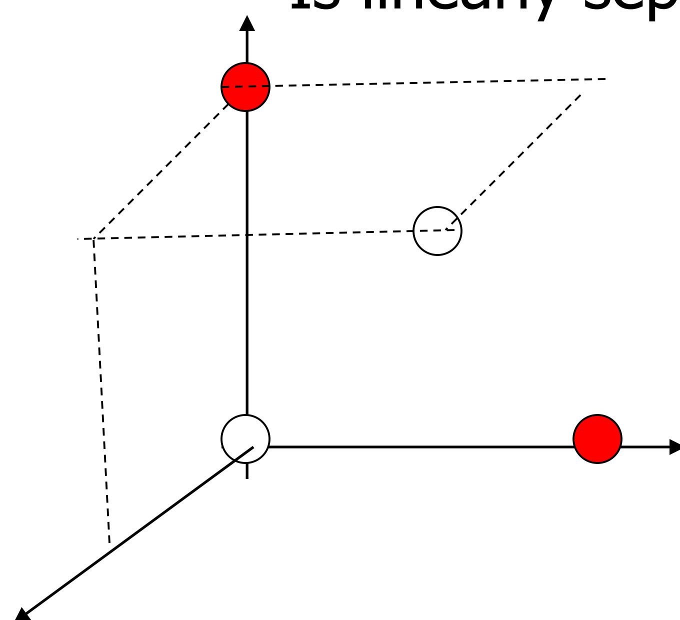
X	Y	
0	0	0
0	1	1
1	0	1
1	1	0

X	Y	XY	
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

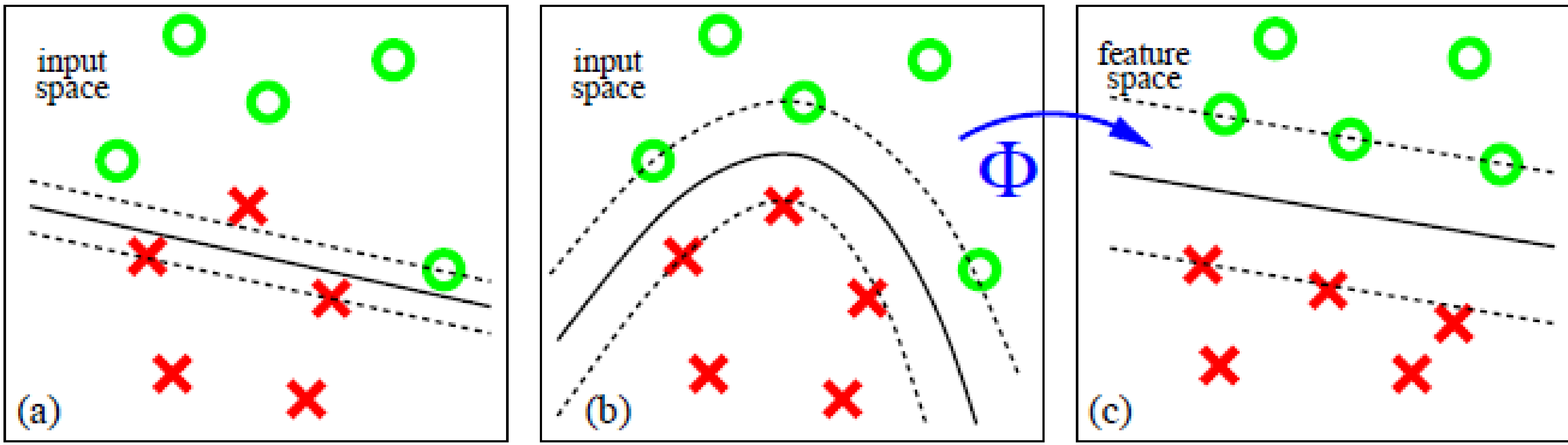
Is not linearly separable



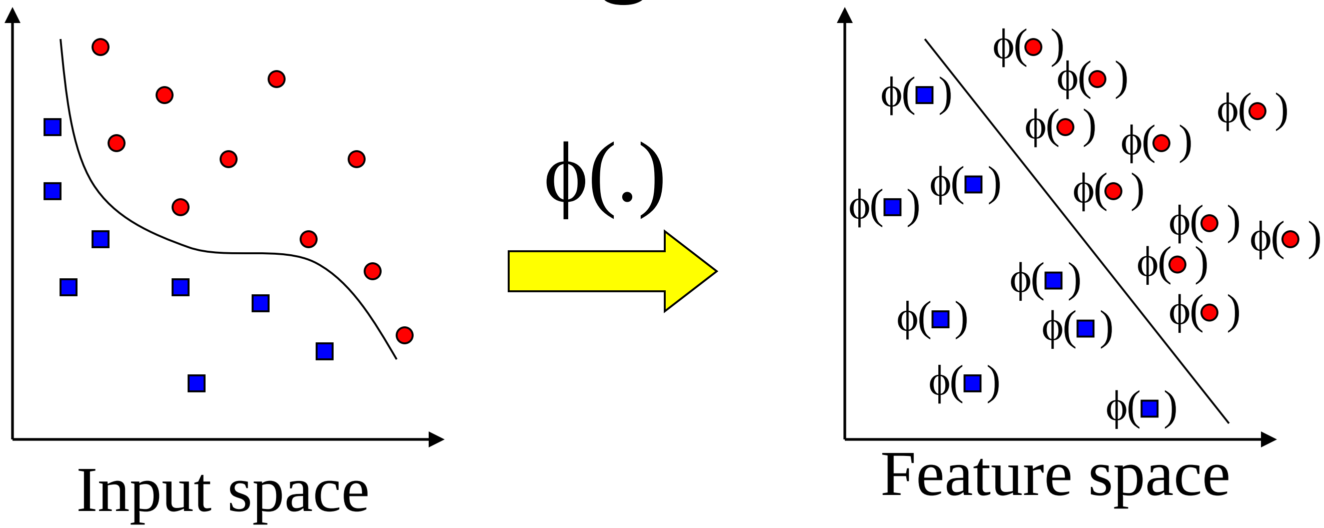
Is linearly separable



Find a feature space



Transforming the Data



Note: feature space is of higher dimension than the input space in practice

- Computation in the feature space can be costly because it is high dimensional
  - The feature space is typically infinite-dimensional!
- The kernel trick comes to rescue

The Kernel Trick

- Recall the SVM optimization problem

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

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

- The data points only appear as inner product
- As long as we can calculate the inner product in the feature space, we do not need the mapping explicitly
- Many common geometric operations (angles, distances) can be expressed by inner products
- Define the kernel function *K* by

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

An Example for ϕ(.) and K(.,.)

- Suppose ϕ(.) is given as follows

$$\phi(\left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right]) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- An inner product in the feature space is

$$\langle \phi(\left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right]), \phi(\left[\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right]) \rangle = (1 + x_1y_1 + x_2y_2)^2$$

- So, if we define the kernel function as follows, there is no need to carry out ϕ(.) explicitly

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

- This use of kernel function to avoid carrying out ϕ(.) explicitly is known as the kernel trick

Kernels

- Given a mapping: $\mathbf{x} \to \boldsymbol{\varphi}(\mathbf{x})$

a kernel is represented as the inner product

$$K(\mathbf{x},\mathbf{y}) \to \sum_i \varphi_i(\mathbf{x})\varphi_i(\mathbf{y})$$

A kernel must satisfy the Mercer's condition:

$$\forall g(\mathbf{x}) \text{ such that } \int g^2(\mathbf{x})d\mathbf{x} \geq 0 \Rightarrow \int K(\mathbf{x},\mathbf{y})g(\mathbf{x})g(\mathbf{y})d\mathbf{x}d\mathbf{y} \geq 0$$

Modification Due to Kernel Function

- Change all inner products to kernel functions
- For training,

Original

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

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

With kernel function

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

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

Modification Due to Kernel Function

- For testing, the new data **z** is classified as class 1 if $f \geq 0$, and as class 2 if $f < 0$

Original

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

$$f = \mathbf{w}^T \mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}^T \mathbf{z} + b$$

With kernel function

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \phi(\mathbf{x}_{t_j})$$

$$f = \langle \mathbf{w}, \phi(\mathbf{z}) \rangle + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} K(\mathbf{x}_{t_j}, \mathbf{z}) + b$$

More on Kernel Functions

- Since the training of SVM only requires the value of $K(\mathbf{x}_i, \mathbf{x}_j)$, there is no restriction of the form of $\mathbf{x}_i$ and $\mathbf{x}_j$
  - $\mathbf{x}_i$ can be a sequence or a tree, instead of a feature vector
- $K(\mathbf{x}_i, \mathbf{x}_j)$ is just a similarity measure comparing $\mathbf{x}_i$ and $\mathbf{x}_j$
- For a test object $\mathbf{z}$, the discriminant function essentially is a weighted sum of the similarity between z and a pre-selected set of objects (the support vectors)

$$f(\mathbf{z}) = \sum_{\mathbf{x}_i \in \mathcal{S}} \alpha_i y_i K(\mathbf{z}, \mathbf{x}_i) + b$$

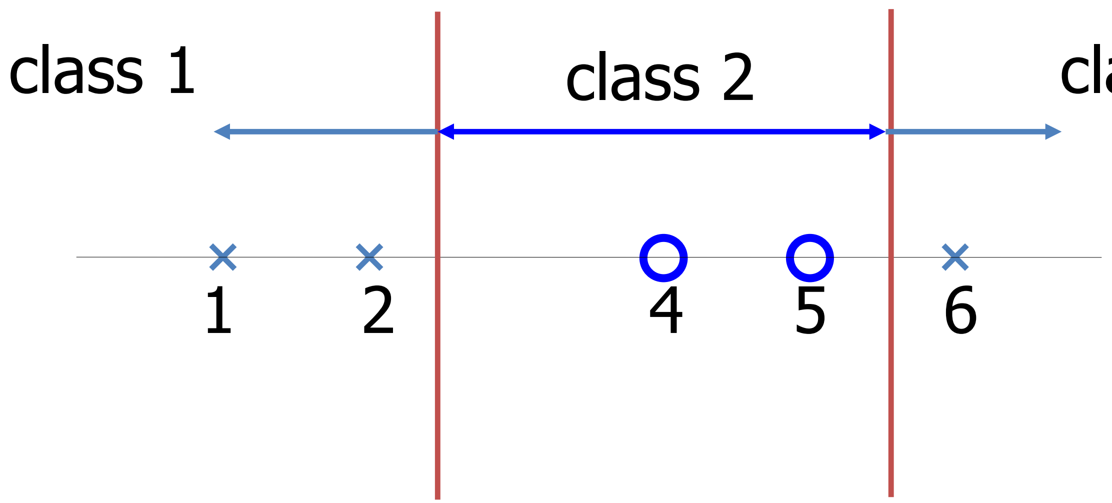
$\mathcal{S}$: the set of support vectors

Example

- Suppose we have 5 1D data points
  - x_1=1, x_2=2, x_3=4, x_4=5, x_5=6, with 1, 2, 6 as class 1 and 4, 5 as class 2 $\Rightarrow$ y_1=1, y_2=1, y_3=-1, y_4=-1, y_5=1

Example

- Suppose we have 5 1D data points
  - x_1=1, x_2=2, x_3=4, x_4=5, x_5=6, with 1, 2, 6 as class 1 and 4, 5 as class 2 $\Rightarrow$ y_1=1, y_2=1, y_3=-1, y_4=-1, y_5=1



Example

- We use the polynomial kernel of degree 2
  - $K(x,y) = (xy+1)^2$
  - C is set to 100
- We first find α_i (i=1, …, 5) by

$$\text{max.} \quad \sum_{i=1}^{5} \alpha_i - \frac{1}{2} \sum_{i=1}^{5} \sum_{j=1}^{5} \alpha_i \alpha_j y_i y_j (x_i x_j + 1)^2$$

$$\text{subject to } 100 \geq \alpha_i \geq 0, \sum_{i=1}^{5} \alpha_i y_i = 0$$

Example

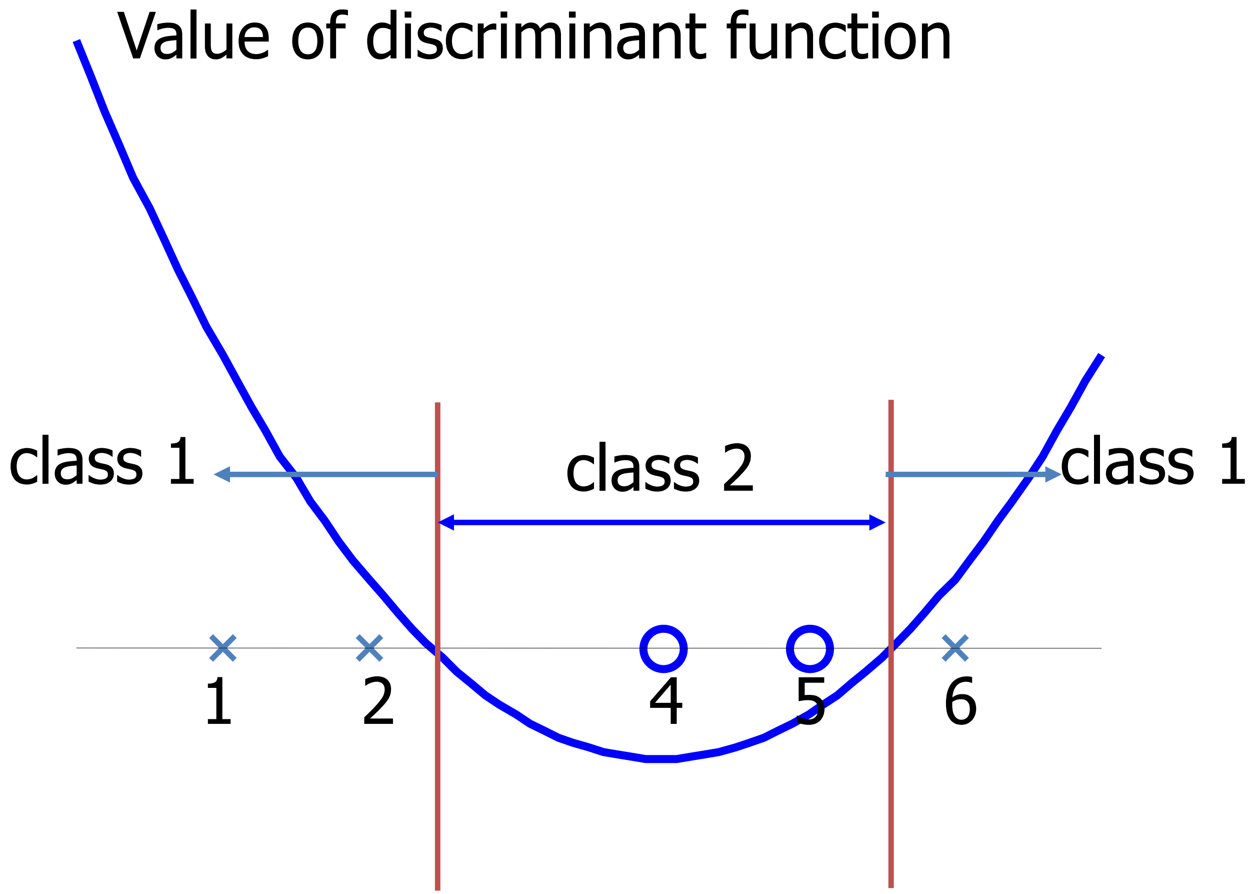
- By using a QP solver, we get
  - $\alpha_1=0$, $\alpha_2=2.5$, $\alpha_3=0$, $\alpha_4=7.333$, $\alpha_5=4.833$
  - Note that the constraints are indeed satisfied
  - The support vectors are $\{x_2=2, x_4=5, x_5=6\}$
- The discriminant function is

$$
\begin{aligned}
& f(z) \\
&= 2.5(1)(2z+1)^2 + 7.333(-1)(5z+1)^2 + \overset{\alpha_5}{4.833}\overset{y_5}{(1)}\overset{K(z,x_5)}{(6z+1)^2} + b \\
&= 0.6667z^2 - 5.333z + b
\end{aligned}
$$

- *b* is recovered by solving $f(2)=1$ or by $f(5)=-1$ or by $f(6)=1$,

- All three give $b=9$ $\Longrightarrow$ $f(z) = 0.6667z^2 - 5.333z + 9$

Example



Value of discriminant function
class 1
class 2
class 1
1
2
4
5
6

Kernel Functions

- In practical use of SVM, the user specifies the kernel function; the transformation $\phi(.)$ is not explicitly stated
- Given a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$, the transformation $\phi(.)$ is given by its eigenfunctions (a concept in functional analysis)
  - Eigenfunctions can be difficult to construct explicitly
  - This is why people only specify the kernel function without worrying about the exact transformation
- Another view: kernel function, being an inner product, is really a similarity measure between the objects

A kernel is associated to a transformation

- Given a kernel, in principle it should be recovered the transformation in the feature space that originates it.

- $K(x,y) = (xy+1)^2 = x^2y^2+2xy+1$

It corresponds the transformation $$x \to \begin{pmatrix} x^2 \\ \sqrt{2}x \\ 1 \end{pmatrix}$$

Examples of Kernel Functions

- Polynomial kernel up to degree *d*

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

- Polynomial kernel up to degree *d*

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

- Radial basis function kernel with width σ

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

  – The feature space is infinite-dimensional
- 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

Building new kernels

- If k_1(x,y) and k_2(x,y) are two valid kernels then the following kernels are valid
  - *Linear Combination*

$$k(x,y) = c_1 k_1(x,y) + c_2 k_2(x,y)$$

  - *Exponential*

$$k(x,y) = \exp\left[k_1(x,y)\right]$$

  - *Product*

$$k(x,y) = k_1(x,y) \cdot k_2(x,y)$$

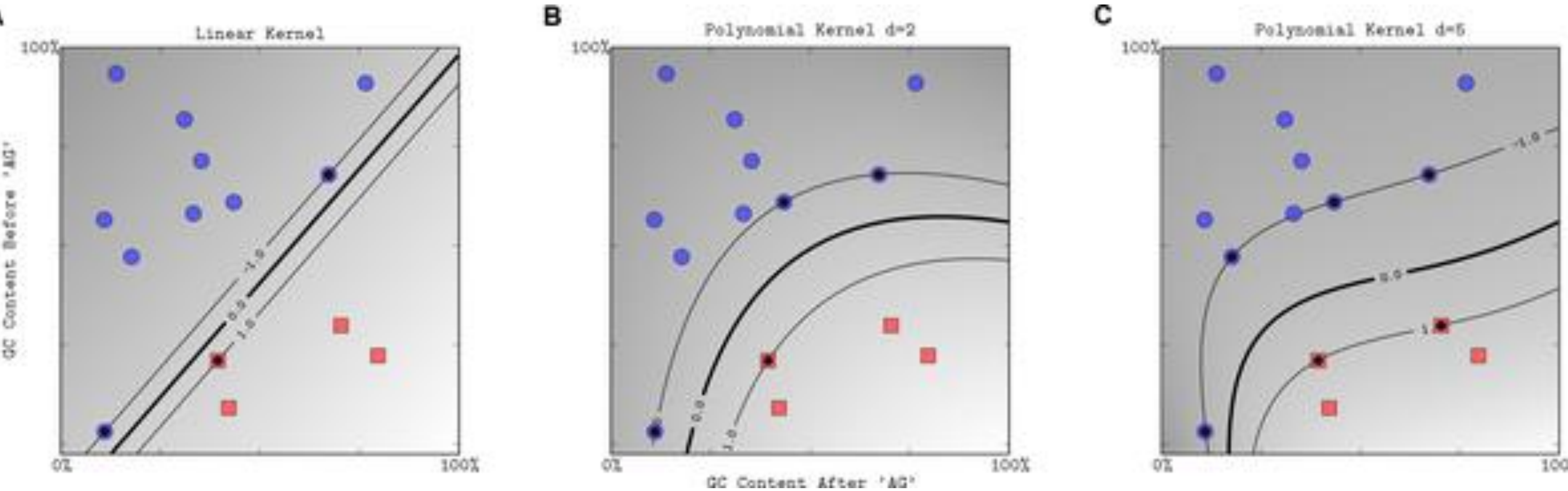
  - *Polymomial tranfsormation (Q: polymonial with non negative coeffients)*

$$k(x,y) = Q\left[k_1(x,y)\right]$$

  - *Function product (f: any function)*

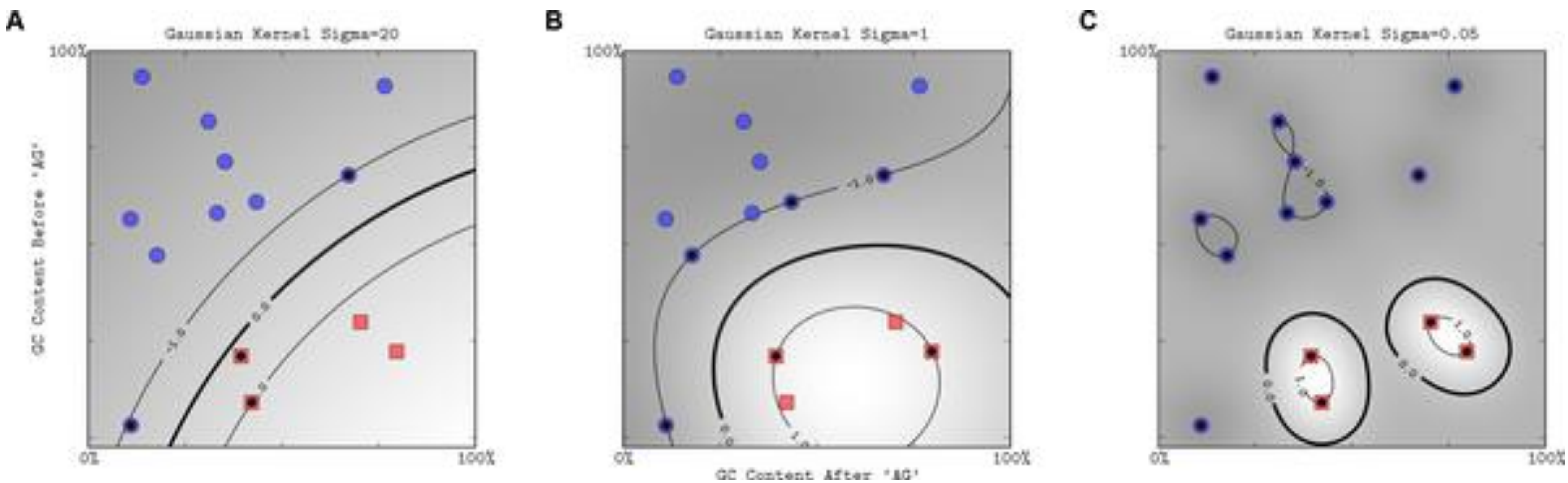
$$k(x,y) = f(x) k_1(x,y) f(y)$$

Polynomial kernel



Ben-Hur et al, PLOS computational Biology 4 (2008)

Gaussian RBF kernel



Ben-Hur et al, PLOS computational Biology 4 (2008)

Spectral kernel for sequences

- Given a DNA sequence x we can count the number of bases (4-D feature space)

$$\phi_1(x) = (n_A, n_C, n_G, n_T)$$

- Or the number of dimers (16-D space)

$$\phi_2(x) = (n_{AA}, n_{AC}, n_{AG}, n_{AT}, n_{CA}, n_{CC}, n_{CG}, n_{CT}, ..)$$

- Or l-mers (4^l –D space)

- The spectral kernel is $k_l(x, y) = \phi_l(x) \cdot \phi_l(y)$

Choosing the Kernel Function

- Probably the most tricky part of using SVM.
- The kernel function is important because it creates the kernel matrix, which summarizes all the data
- Many principles have been proposed (diffusion kernel, Fisher kernel, string kernel, …)
- There is even research to estimate the kernel matrix from available information

- In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try
- Note that SVM with RBF kernel is closely related to RBF neural networks, with the centers of the radial basis functions automatically chosen for SVM

Other Aspects of SVM

- How to use SVM for multi-class classification?
  - One can change the QP formulation to become multi-class
  - More often, multiple binary classifiers are combined
    - See DHS 5.2.2 for some discussion
  - One can train multiple one-versus-all classifiers, or combine multiple pairwise classifiers "intelligently"
- How to interpret the SVM discriminant function value as probability?
  - By performing logistic regression on the SVM output of a set of data (validation set) that is not used for training
- Some SVM software (like libsvm) have these features built-in

Software

- A list of SVM implementation can be found at http://www.kernel-machines.org/software.html
- Some implementation (such as LIBSVM) can handle multi-class classification
- SVMLight is among one of the earliest implementation of SVM
- Several Matlab toolboxes for SVM are also available

Summary: Steps for Classification

- Prepare the pattern matrix
- Select the kernel function to use
- Select the parameter of the kernel function and the value of *C*
  - You can use the values suggested by the SVM software, or you can set apart a validation set to determine the values of the parameter
- Execute the training algorithm and obtain the α_i
- Unseen data can be classified using the α_i and the support vectors

Strengths and Weaknesses of SVM

- Strengths
  - Training is relatively easy
    - No local optimal, unlike in neural networks
  - It scales relatively well to high dimensional data
  - Tradeoff between classifier complexity and error can be controlled explicitly
  - Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors
- Weaknesses
  - Need to choose a “good” kernel function.

Conclusion

- SVM is a useful alternative to neural networks
- Two key concepts of SVM: maximize the margin and the kernel trick
- Many SVM implementations are available on the web for you to try on your data set!

Resources

- http://www.kernel-machines.org/
- http://www.support-vector.net/
- http://www.support-vector.net/icml-tutorial.pdf
- http://www.kernel-machines.org/papers/tutorial-nips.ps.gz
- http://www.clopinet.com/isabelle/Projects/SVM/applist.html

Thank You!