CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya

BOOSTING

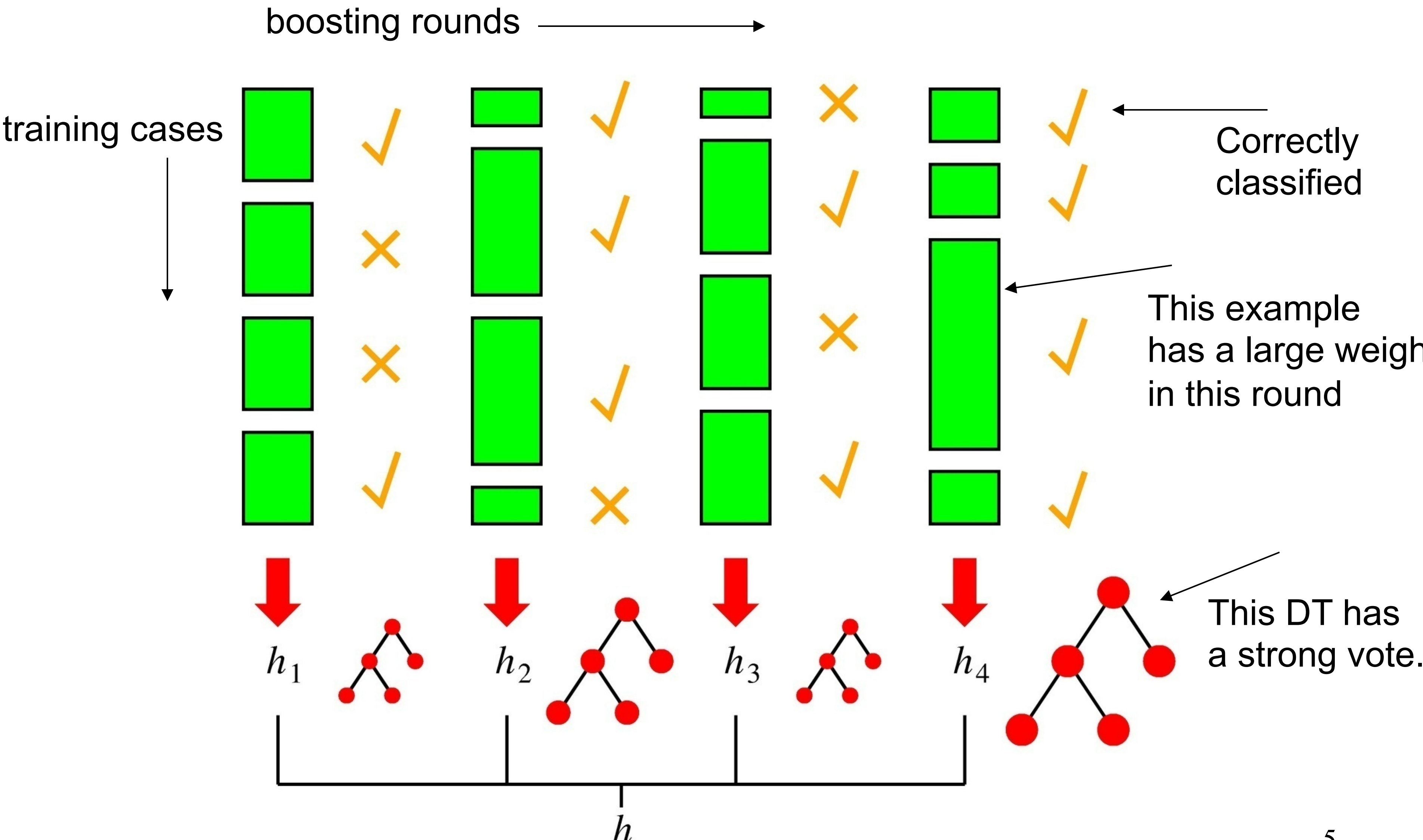
Boosting

- Train classifiers (e.g. decision trees) in a sequence.
- A new classifier should focus on those cases which were incorrectly classified in the last round.
- Combine the classifiers by letting them vote on the final prediction (like bagging).
- Each classifier is "weak" but the ensemble is "strong."
- AdaBoost is a specific boosting method.

Boosting Intuition

- We adaptively weigh each data case.
- Data cases which are wrongly classified get high weight (the algorithm will focus on them)
- Each boosting round learns a new (simple) classifier on the weighed dataset.
- These classifiers are weighed to combine them into a single powerful classifier.
- Classifiers that that obtain low training error rate have high weight.
- We stop by using monitoring a hold out set (cross-validation).

Boosting in a Picture



Boosting

- Combining multiple “base” classifiers to come up with a “good” classifier.
- Base classifiers have to be “weak learners”, accuracy > 50%
- Base classifiers are trained on a weighted training dataset.
- Boosting involves sequentially learning α_m and $y_m(x)$.

Adaboost

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)$$

   where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

   (b) Evaluate the quantities

$$\epsilon_m = \frac{\displaystyle\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\displaystyle\sum_{n=1}^{N} w_n^{(m)}}$$

   and then use these to evaluate

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}.$$

Adaboost (contd..)

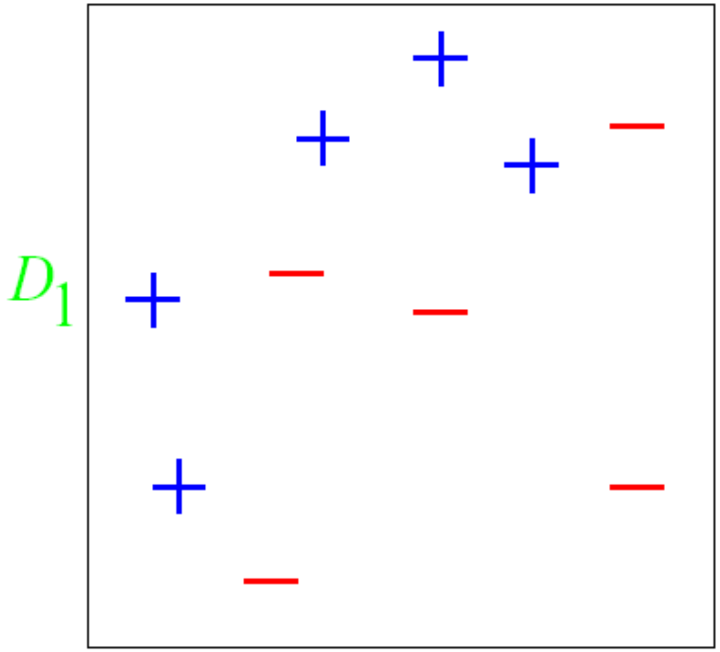
(c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp\left\{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\right\}$$

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \operatorname{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right).$$

And in animation

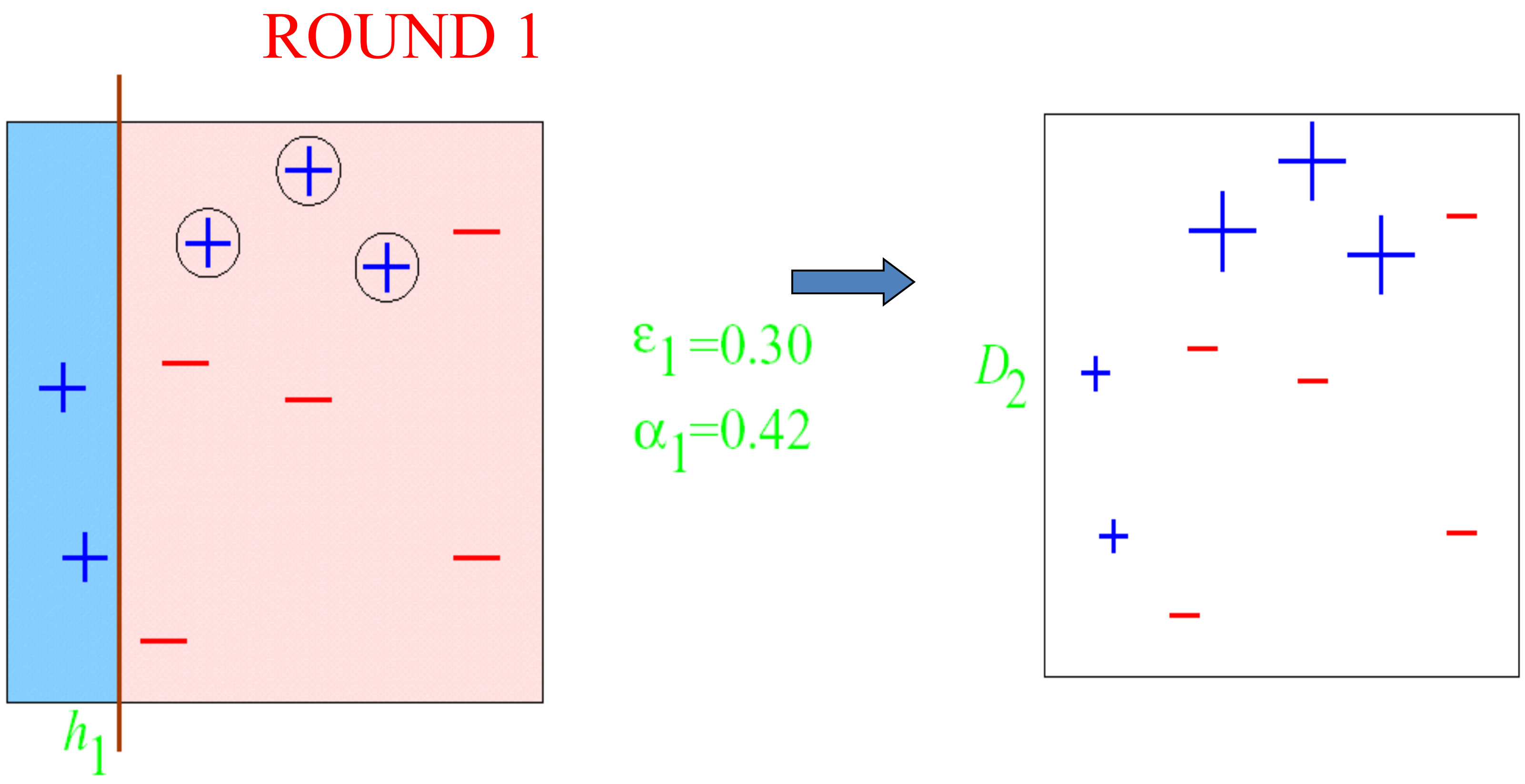


Original training set: equal weights to all training samples

Taken from "**A Tutorial on Boosting**" **by** Yoav Freund and Rob Schapire

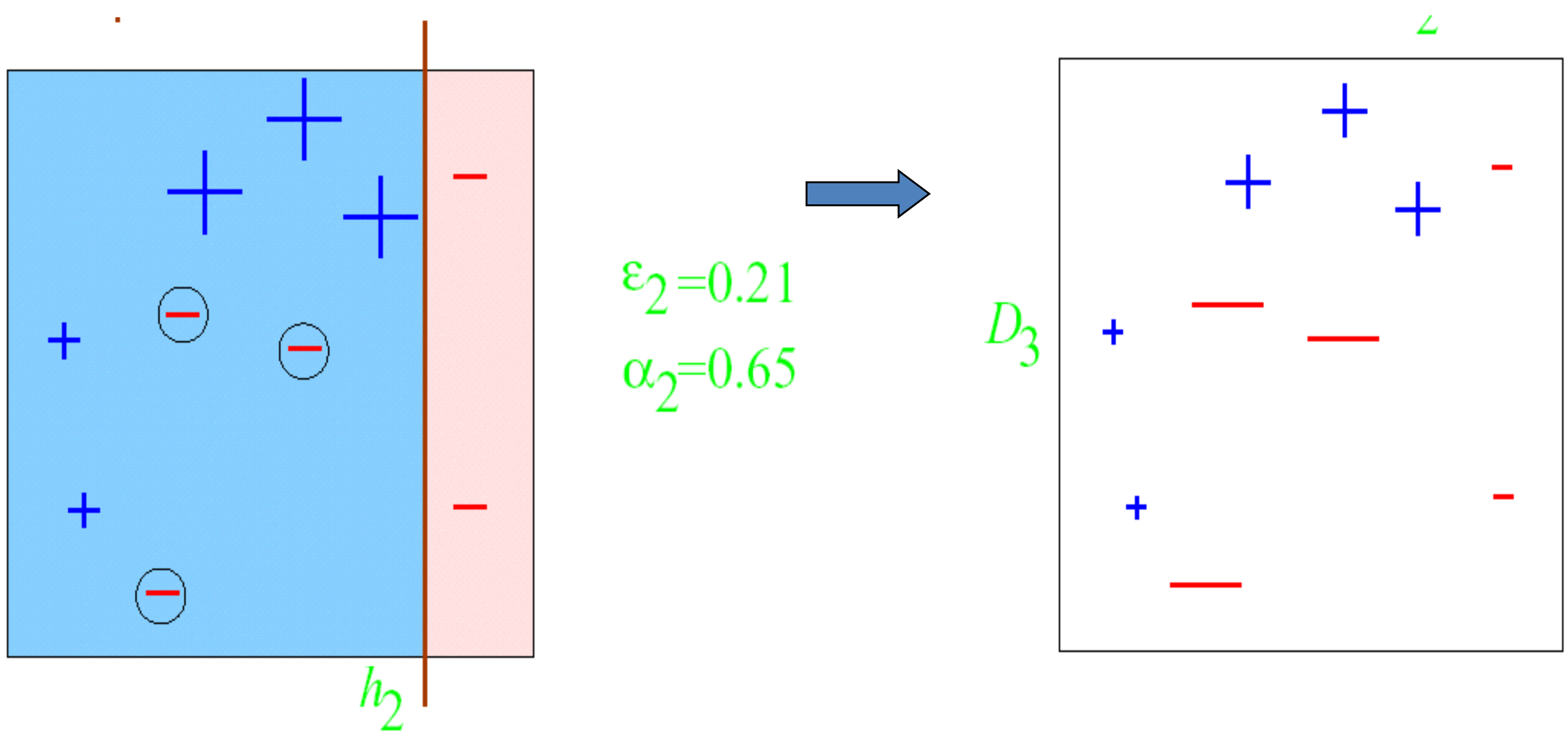
AdaBoost example

ε = error rate of classifier
α = weight of classifier



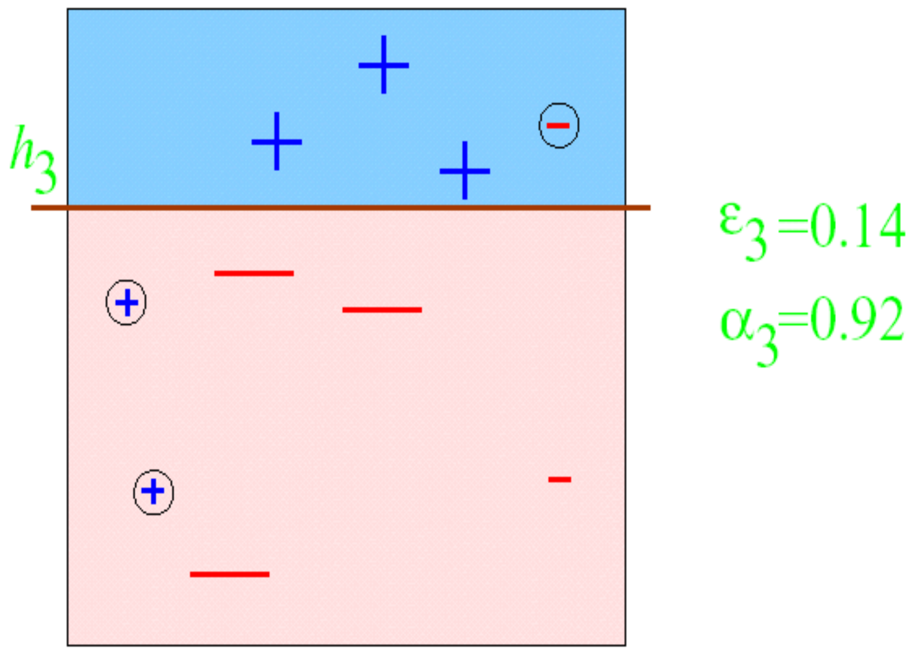
AdaBoost example

ROUND 2

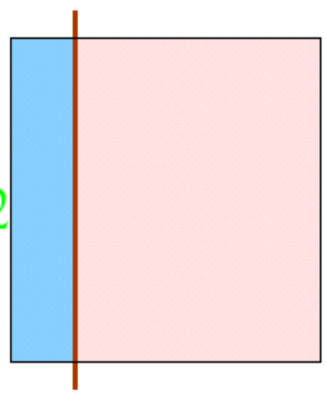
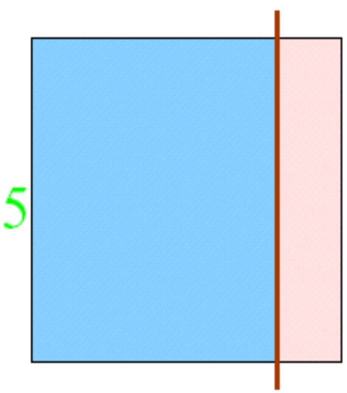
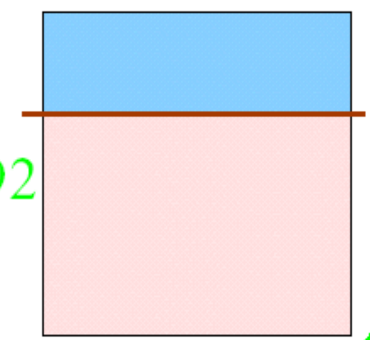


AdaBoost example

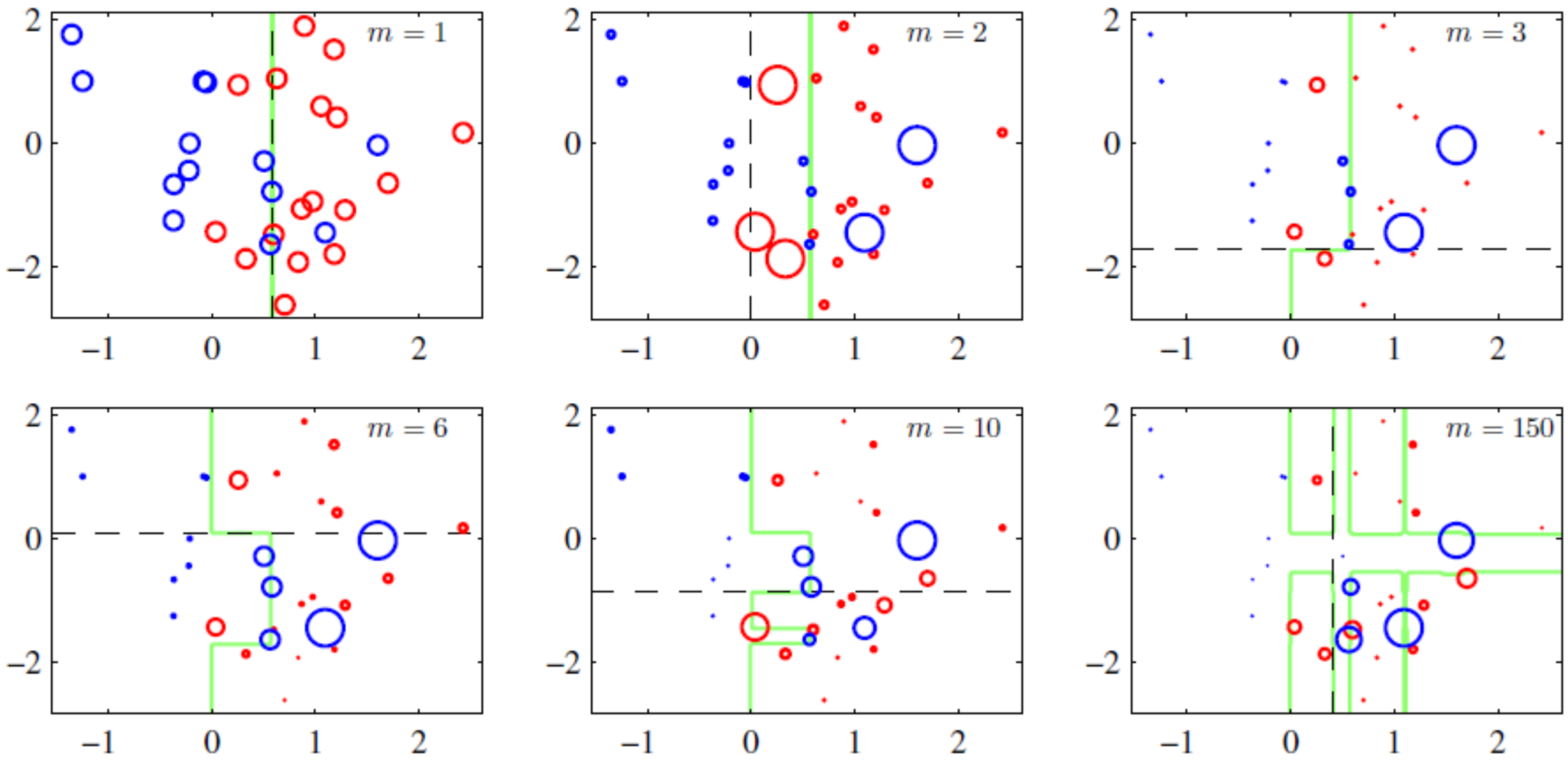
ROUND 3



AdaBoost example

$H_{\text{final}} = \text{sign}\Big(0.42$  $+ \; 0.65$  $+ \; 0.92$  $\Big)$

Adaboost illustration



Adaboost - Observations

- ϵ_m: weighted error $\in [0,0.5)$
- $\alpha_m \geq 0$
- w_i^{m+1} is higher than w_i^m by a factor $(1 - \epsilon_m)/\epsilon_m$, when *i* is misclassified.

Adaboost - derivation

- Consider the error function:

$$E = \sum_{n=1}^{N} \exp\{-t_n f_m(\mathbf{x}_n)\}$$

- Where

$$f_m(\mathbf{x}) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l y_l(\mathbf{x})$$

- Goal: Minimize E w.r.t. α_l and $y_l(x)$, sequentially.

Adaboost - derivation

- Minimize w.r.t. α_m

$$\begin{aligned} E &= \sum_{n=1}^{N} \exp\left\{-t_n f_{m-1}(\mathbf{x}_n) - \frac{1}{2} t_n \alpha_m y_m(\mathbf{x}_n)\right\} \\ &= \sum_{n=1}^{N} w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m y_m(\mathbf{x}_n)\right\} \end{aligned}$$

- Let τ_m be the set of datapoints correctly classified by y_m.

$$\begin{aligned} E &= e^{-\alpha_m/2} \sum_{n \in \mathcal{T}_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in \mathcal{M}_m} w_n^{(m)} \\ &= (e^{\alpha_m/2} - e^{-\alpha_m/2}) \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}. \end{aligned}$$

Adaboost - derivation

- Minimizing w.r.t. y_m and α_m, we get the updates 2(a) and 2(b).
- We can see that:

$$w_n^{(m+1)} = w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m y_m(\mathbf{x}_n)\right\}.$$

- Using:

$$t_n y_m(\mathbf{x}_n) = 1 - 2I(y_m(\mathbf{x}_n) \neq t_n)$$

- We get:

$$w_n^{(m+1)} = w_n^{(m)} \exp(-\alpha_m/2) \exp\left\{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\right\}.$$

Adaboost

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)$$

   where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

   (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

   and then use these to evaluate

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}.$$

Adaboost (contd..)

(c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp\left\{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\right\}$$

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \operatorname{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right).$$

SUPPORT VECTOR MACHINES

Support vector machines

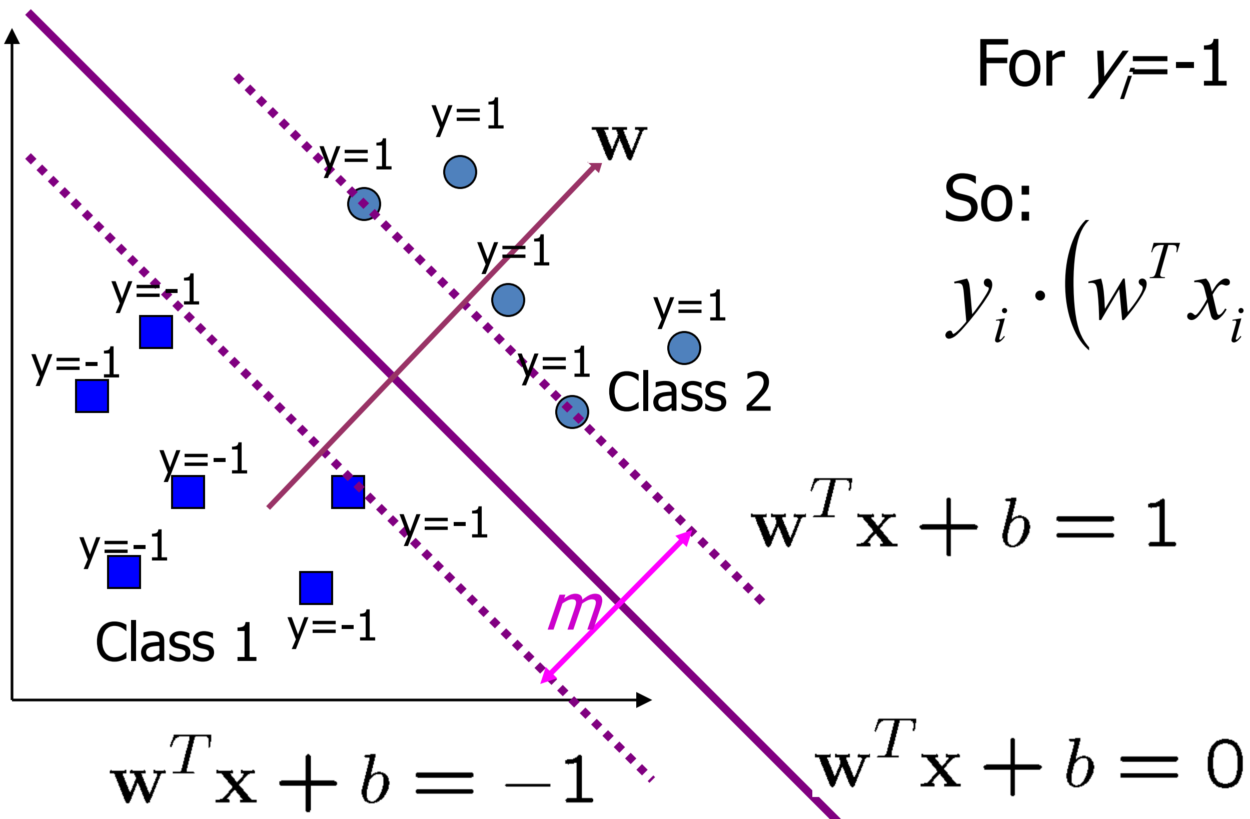
- Let $\{x_1, ..., x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i

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

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

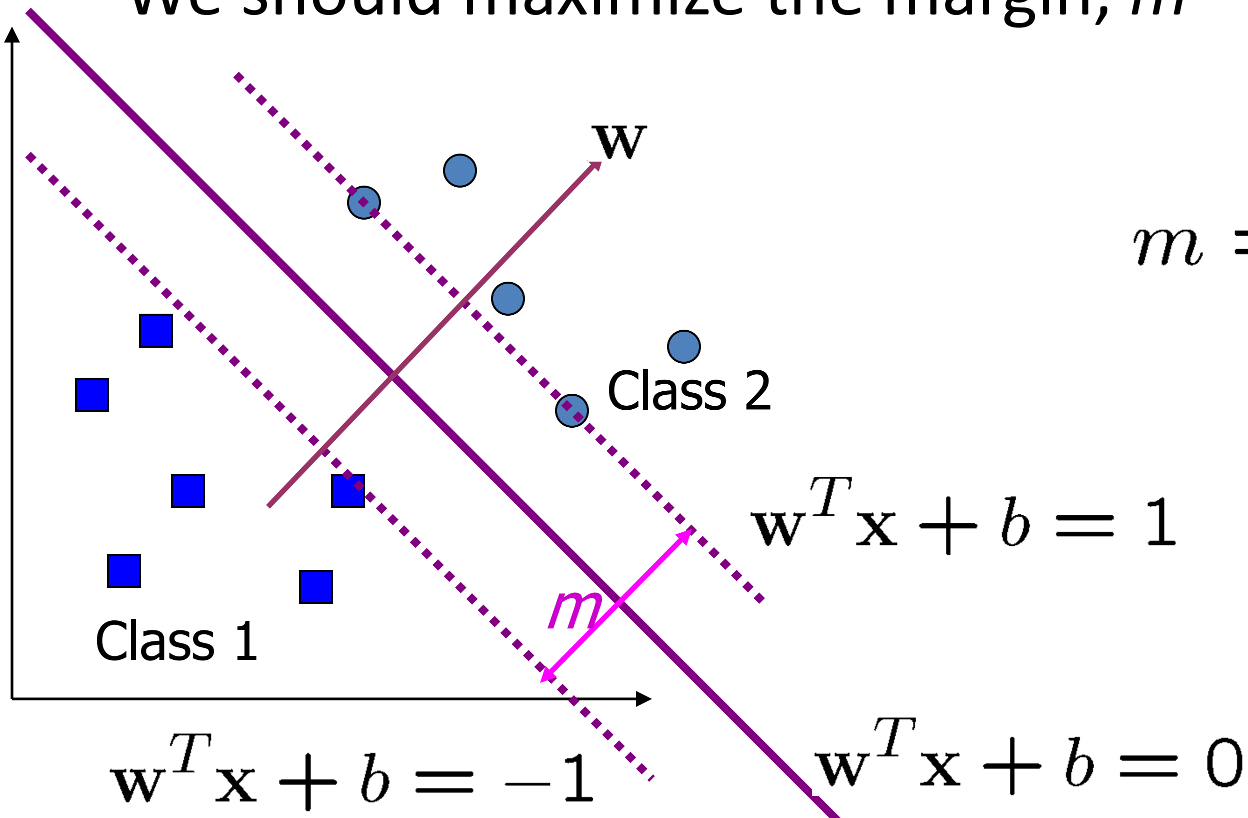
So:

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



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

KKT Conditions

- Problem:

$$\min_{x} f(x) \quad \text{sub. to:} \, \mathrm{g_i(x)} \leq 0 \ \ \forall\, i$$

- Lagrangian: $L(x, \mu) = f(x) - \sum_i \mu_i g_i(x)$
- Conditions:
  - Stationarity: $\nabla_{\mathrm{x}} \mathrm{L}(\mathrm{x}, \mu) = 0.$
  - Primal feasibility: $g_i(x) \leq 0 \ \ \forall\, i.$
  - Dual feasibility: $\mu_i \geq 0.$
  - Complementary slackness: $\mu_i g_i(x) = 0.$

Finding the Decision Boundary

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

subject to $1-y_i(\mathbf{w}^T\mathbf{x}_i+b) \leq 0$ 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 $||\mathbf{w}||^2 = \mathbf{w}^T\mathbf{w}$

The Dual Problem

- 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

$$\begin{cases} \dfrac{\partial L}{\partial w^k} = 0, \forall k \\ \dfrac{\partial L}{\partial b} = 0 \end{cases} \qquad \mathbf{w} + \sum_{i=1}^{n} \alpha_i (-y_i) \mathbf{x}_i = \mathbf{0} \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

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

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

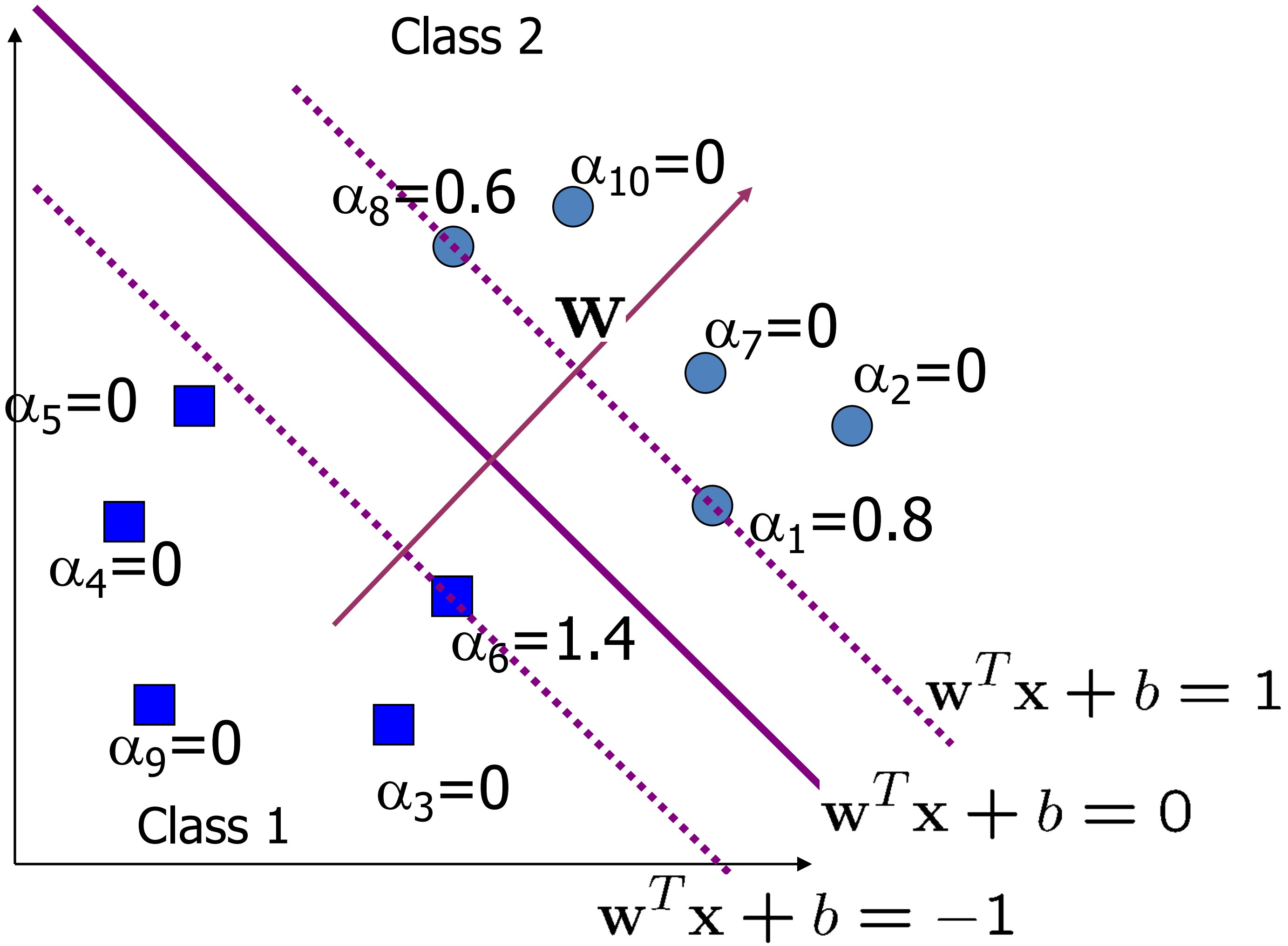
Characteristics of the Solution

- Many of the α_i are zero
  - Complementary slackness: $\alpha_i\left(1 - y_i(w^T x_i + b)\right) = 0$
  - Sparse representation: **w** is a linear combination of a small number of data points

- $\mathbf{x}_i$ 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

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

A Geometrical Interpretation

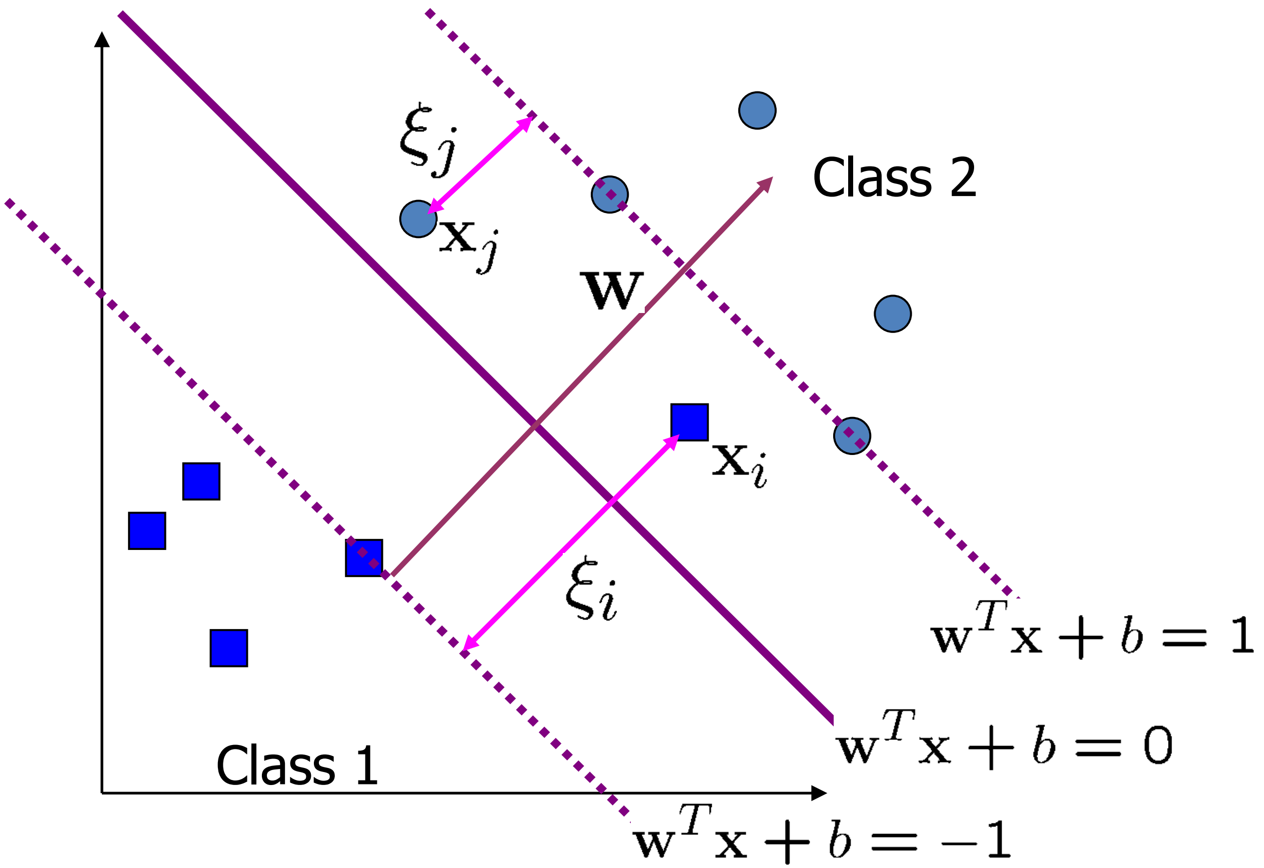


Characteristics of the Solution

- 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

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

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

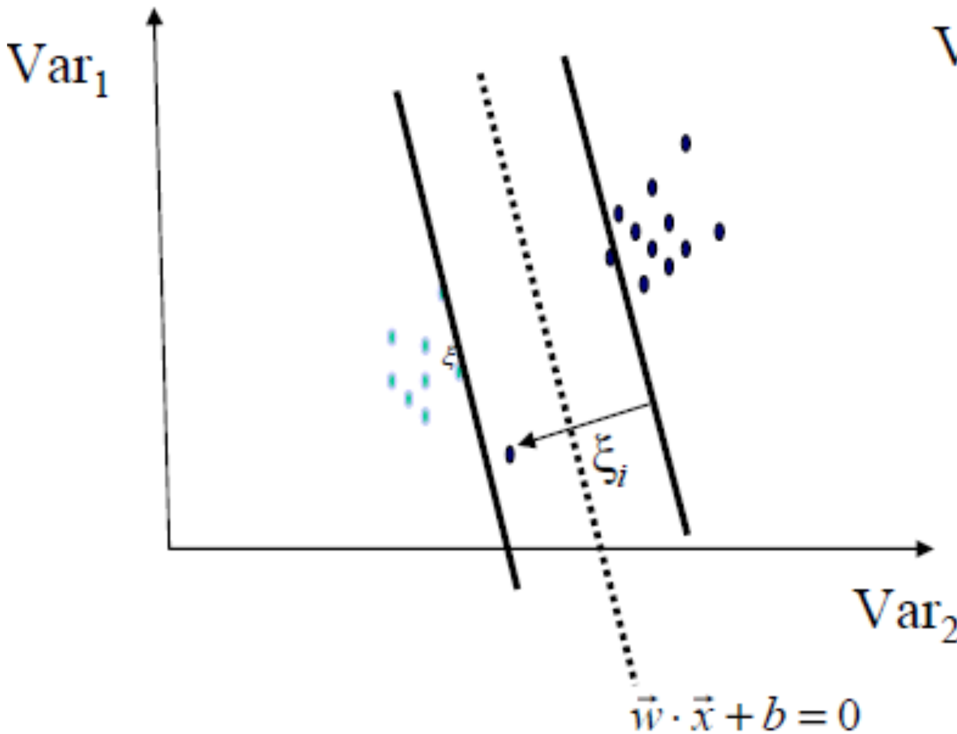
$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- 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

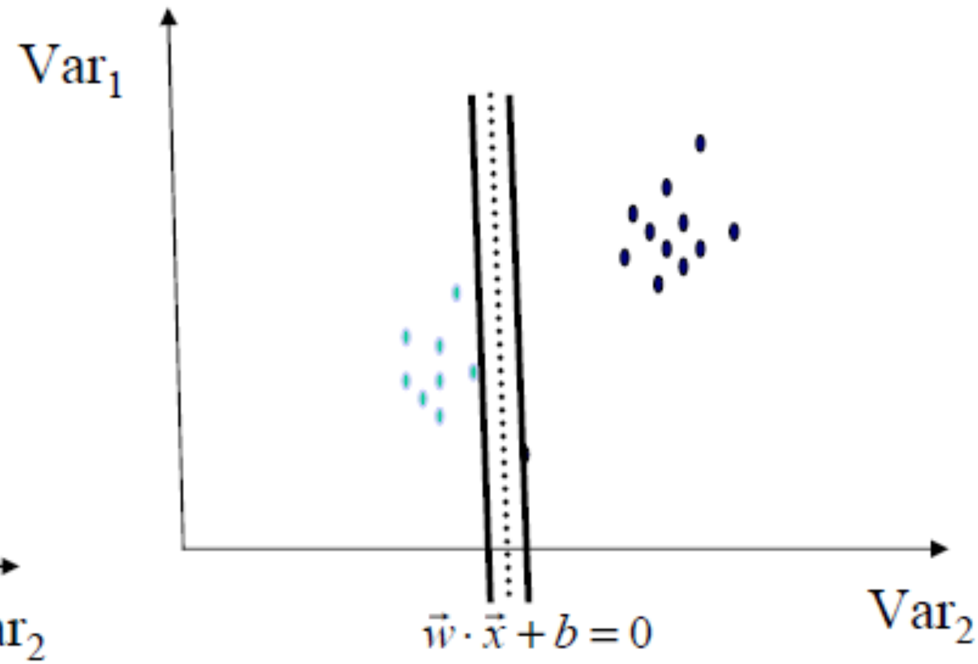
$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

- The algorithm try to keep ξ low, maximizing the margin
- The algorithm does not minimize the number of error. Instead, it minimizes the sum of distances from 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

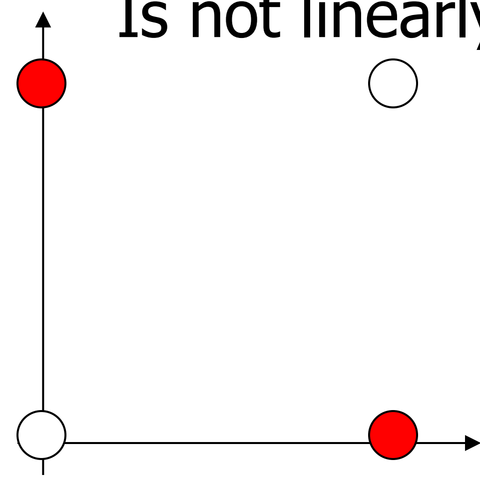
Extension to Non-linear Decision Boundary

- 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

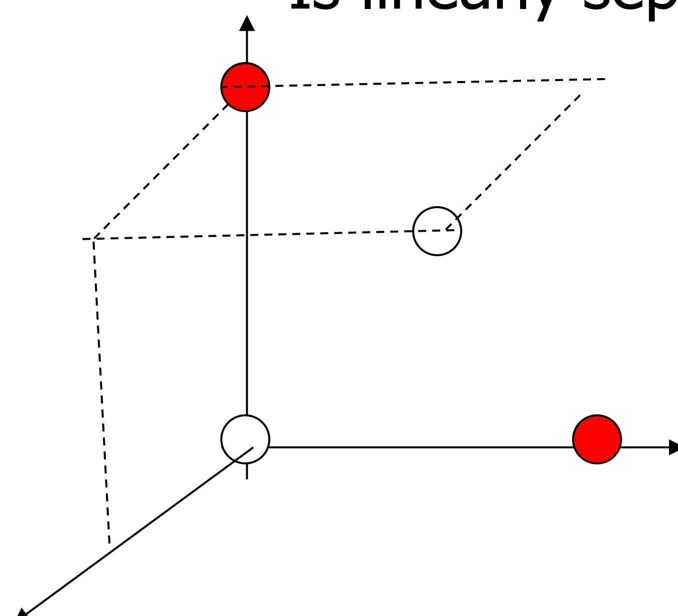
X	Y	
0	0	0
0	1	1
1	0	1
1	1	0

Is not linearly separable

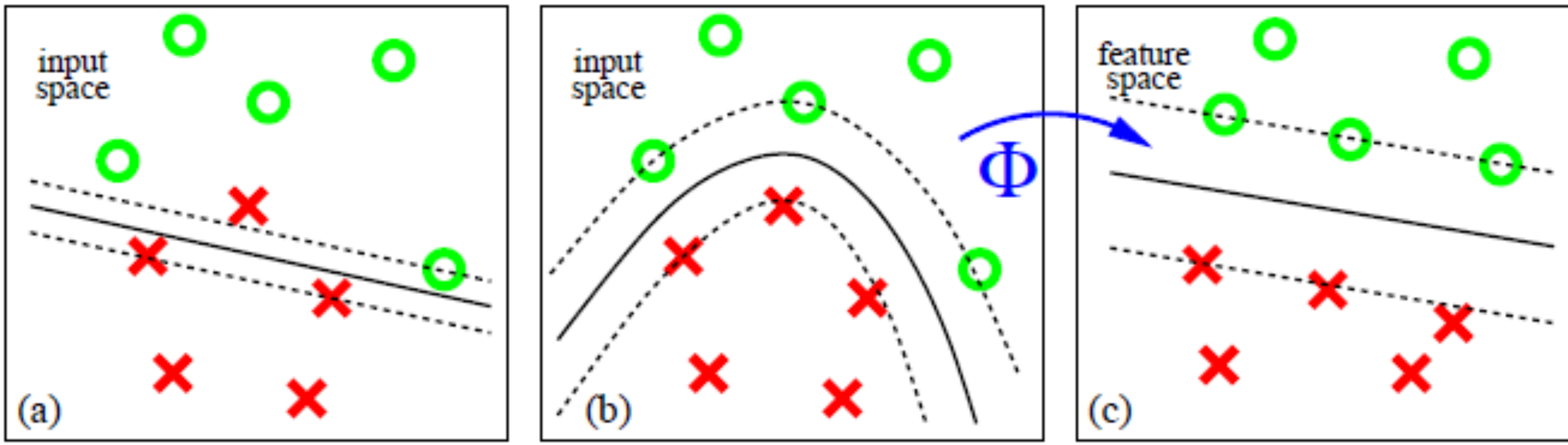


X	Y	XY	
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

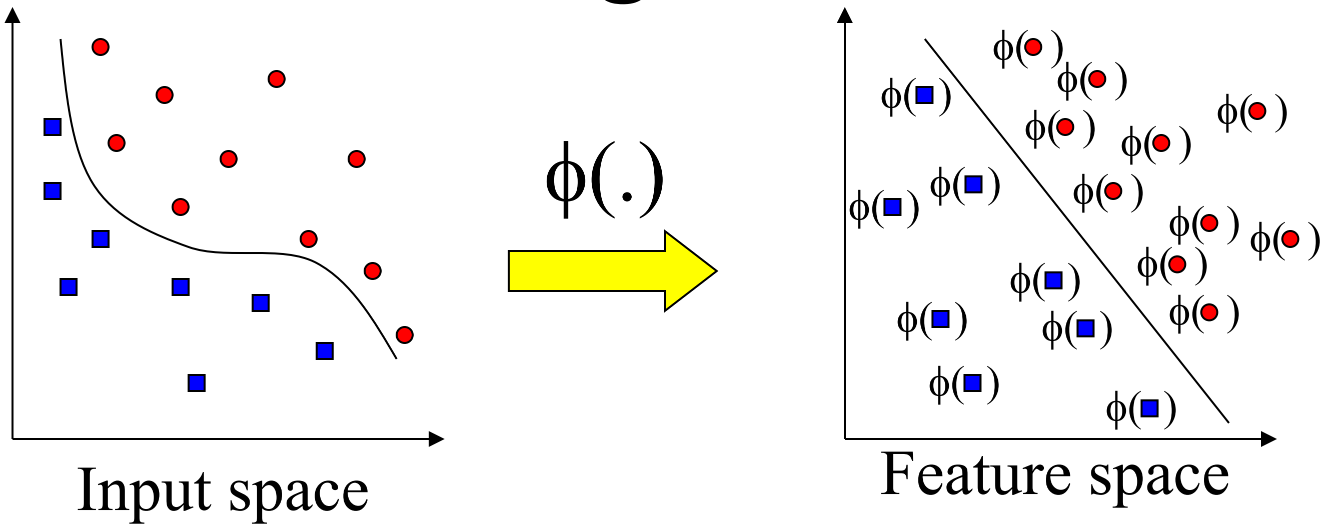
Is linearly separable



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

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

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

- 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: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$

a kernel is represented as the inner product

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

A kernel must satisfy the Mercer's condition:

$$\forall g(\mathbf{x}) \int K(\mathbf{x},\mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0$$

Modification Due to Kernel Function

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

Original

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

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

With kernel function

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

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

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 **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

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 of 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*

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

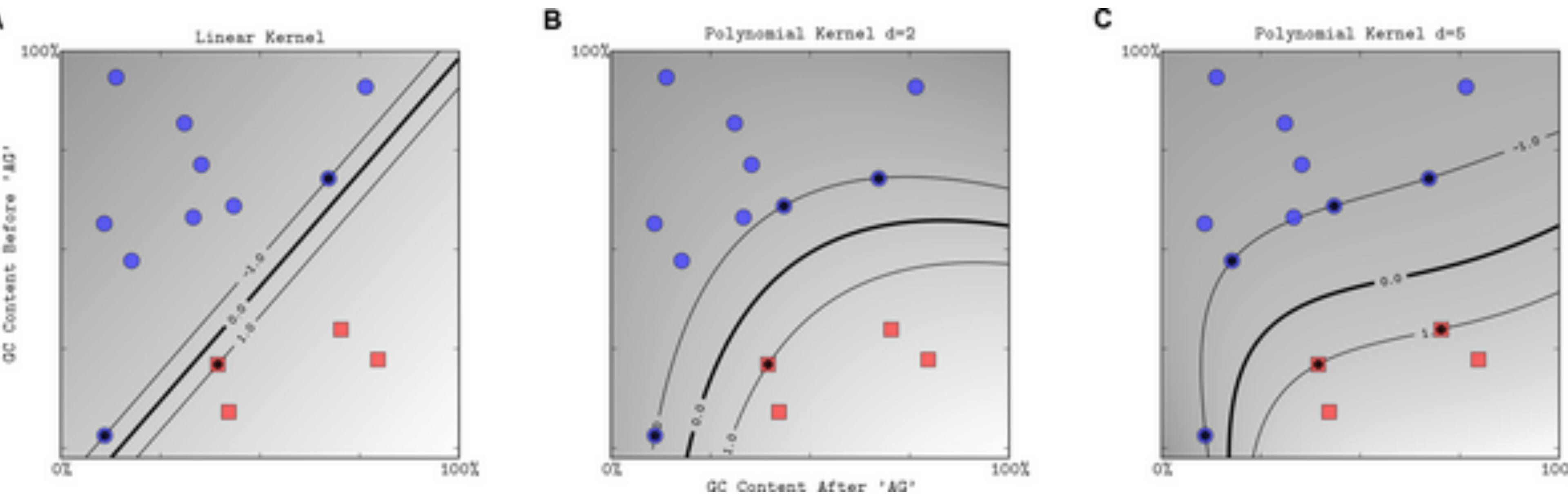
  - *Polynomial transformation (Q: polynomial with non negative coeffcients)*

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

  - *Function product (f: any function)*

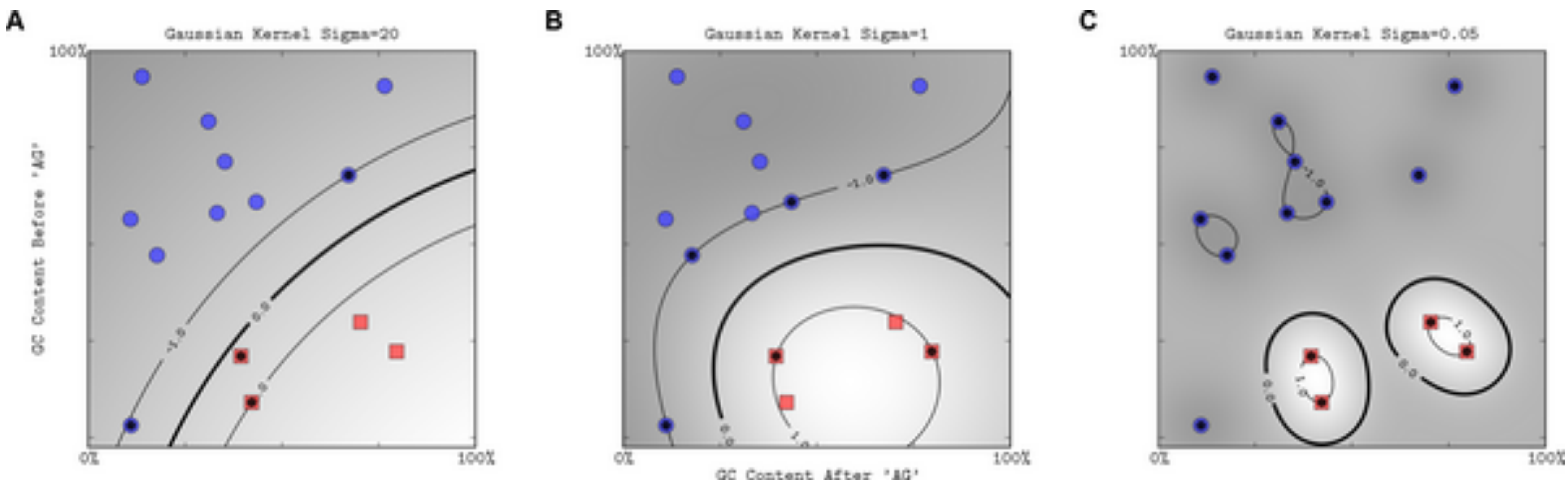
    $$k(x,y) = f(x) k_1(x,y) f(y)$$

Polynomial kernel



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

Gaussian RBF kernel



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