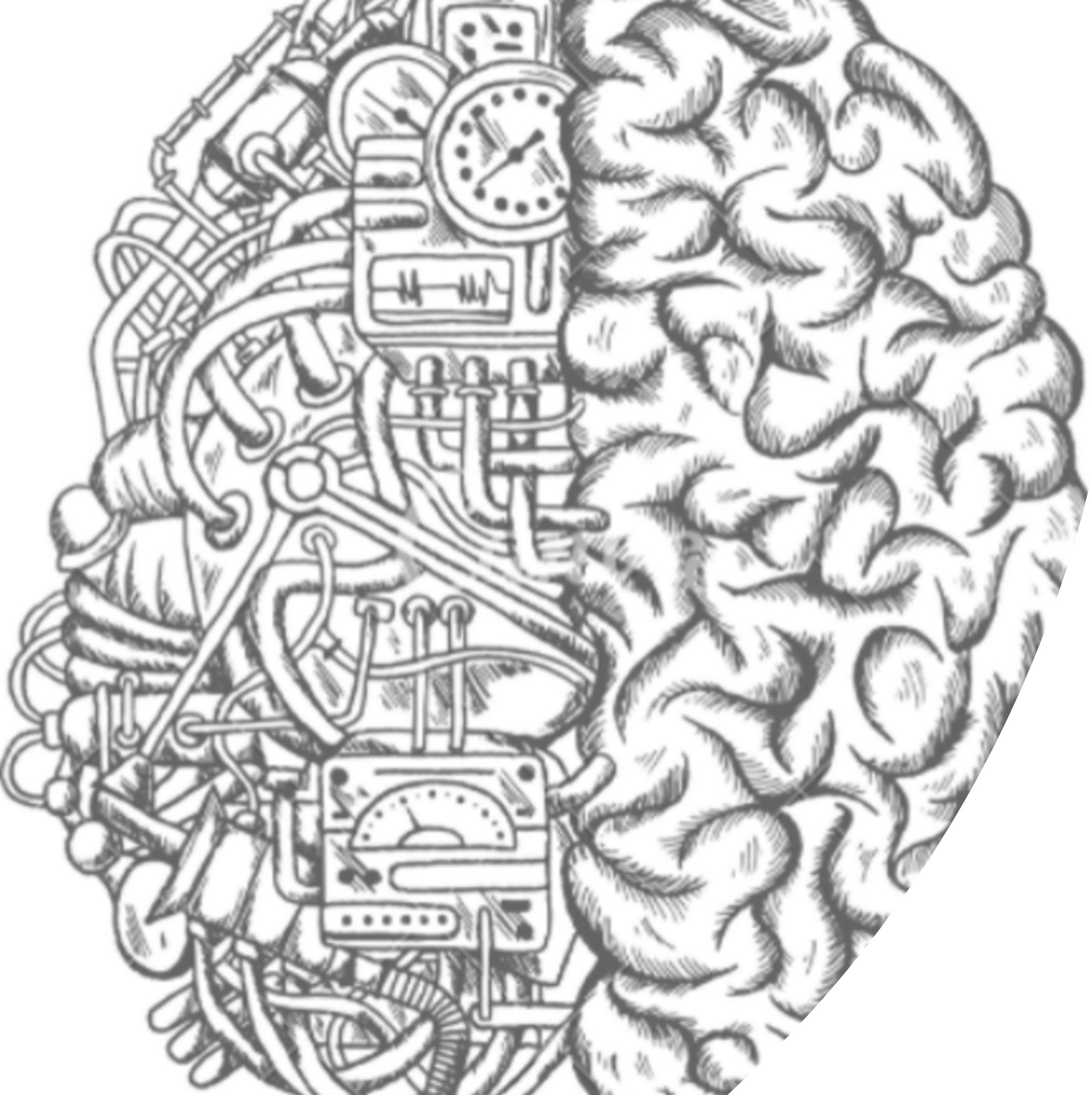


CS 559: Machine Learning Fundamentals and Applications

Lecture 8

Lecturer: Xinchao Wang

xinchao.wang@stevens.edu

Teaching Assistant: Yiding Yang

yyang99@stevens.edu

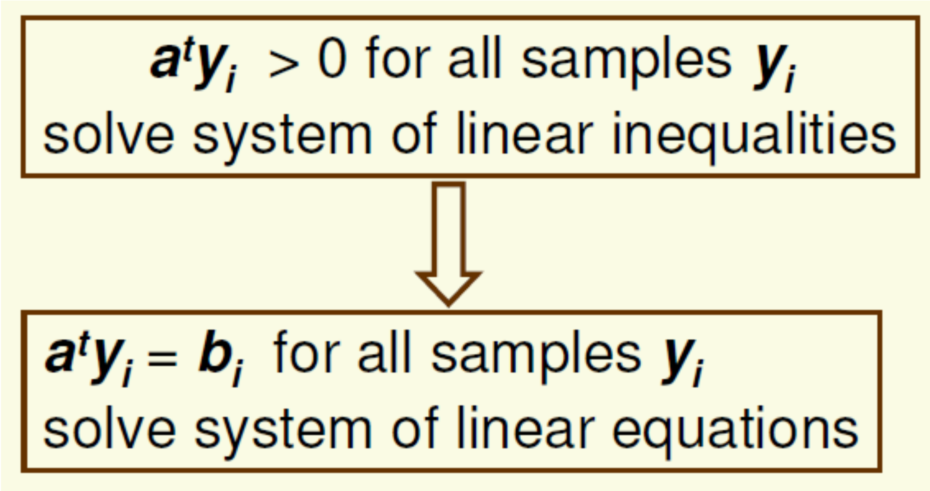
Overview

- Minimum Squared Error (MSE)
- Support Vector Machines (SVM)
  - Introduction
  - Linear Discriminant
    - Linearly Separable Case
    - Linearly Non Separable Case
  - Kernel Trick
    - Non Linear Discriminant
  - Multi-class SVMs

Minimum Squared-Error Procedures

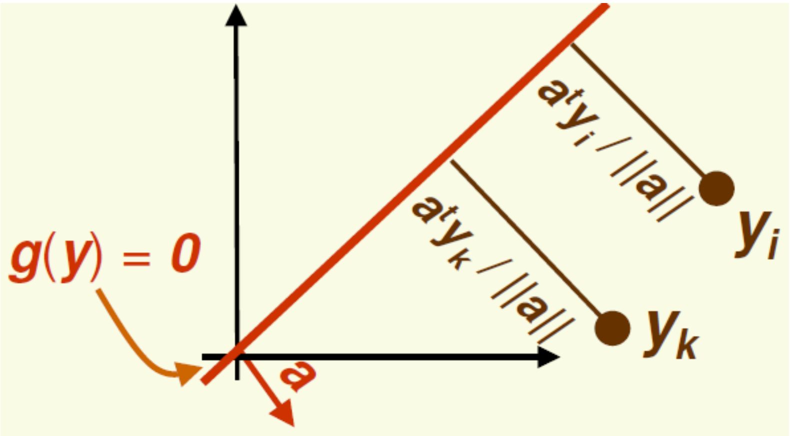
Minimum Squared-Error Procedures

- Idea: convert to easier and better understood problem



- MSE procedure
  - Choose positive constants $\mathbf{b_1, b_2, \ldots, b_n}$
  - Try to find weight vector **a** such that $\mathbf{a^t y_i = b_i}$ for all samples $\mathbf{y_i}$
  - If we can find such a vector, then **a** is a solution because the $\mathbf{b_i}$'s are positive
  - Consider all the samples (not just the misclassified ones)

MSE Margins



- If $\mathbf{a}^t\mathbf{y}_i = \mathbf{b}_i$, $\mathbf{y}_i$ must be at distance $\mathbf{b}_i$ from the separating hyperplane (normalized by ||**a**||)
- Thus $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ give relative expected distances or "***margins***" of samples from the hyperplane
- Should make $\mathbf{b}_i$ small if sample **i** is expected to be near separating hyperplane, and large otherwise
- In the absence of any additional information, set $\mathbf{b}_1 = \mathbf{b}_2 = \ldots = \mathbf{b}_n = \mathbf{1}$

MSE Matrix Notation

- Need to solve **n** equations
- In matrix form **Ya=b**

$$\begin{cases} a^t y_1 = b_1 \\ \vdots \\ a^t y_n = b_n \end{cases}$$

$$\underbrace{\begin{bmatrix} y_1^{(0)} & y_1^{(1)} & \cdots & y_1^{(d)} \\ y_2^{(0)} & y_2^{(1)} & \cdots & y_2^{(d)} \\ \vdots & & & \vdots \\ y_n^{(0)} & y_n^{(1)} & \cdots & y_n^{(d)} \end{bmatrix}}_{Y} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix}}_{a} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_{b}$$

Exact Solution is Rare

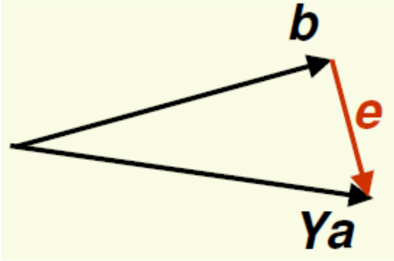
- Need to solve a linear system $\mathbf{Ya} = \mathbf{b}$
  - $\mathbf{Y}$ is an $\mathbf{n} \times (\mathbf{d} + \mathbf{1})$ matrix
- Exact solution only if $\mathbf{Y}$ is non-singular and square (the inverse $\mathbf{Y}^{-1}$ exists)
  - $a = \mathbf{Y}^{-1}\mathbf{b}$
  - (number of samples) = (number of features + 1)
  - Almost never happens in practice
  - Guaranteed to find the separating hyperplane

Approximate Solution

- Typically **Y** is overdetermined, that is it has more rows (examples) than columns (features)
  - If it has more features than examples, should reduce dimensionality
- Need **Ya = b**, but no exact solution exists for an over-determined system of equations
  - More equations than unknowns
- Find an approximate solution
  - Note that approximate solution **a** does **not** necessarily give the separating hyperplane in the separable case
  - But the hyperplane corresponding to **a** may still be a good solution, especially if there is no separating hyperplane

MSE Criterion Function

- Minimum squared error approach: find **a** which minimizes the length of the error vector **e**



$$\mathbf{e} = \mathbf{Ya} - \mathbf{b}$$

- Thus minimize the minimum squared error criterion function:

$$J_s(a) = \|Ya - b\|^2 = \sum_{i=1}^{n} \left(a^t y_i - b_i\right)^2$$

- Unlike the perceptron criterion function, we can optimize the minimum squared error criterion function analytically by setting the gradient to 0

Computing the Gradient

$$J_s(a) = \|Ya - b\|^2 = \sum_{i=1}^{n} \left(a^t y_i - b_i\right)^2$$

$$\nabla J_s(a) = \begin{bmatrix} \dfrac{\partial J_s}{\partial a_0} \\ \vdots \\ \dfrac{\partial J_s}{\partial a_d} \end{bmatrix} = \frac{dJ_s}{da} = \sum_{i=1}^{n} \frac{d}{da}\left(a^t y_i - b_i\right)^2$$

$$= \sum_{i=1}^{n} 2\left(a^t y_i - b_i\right)\frac{d}{da}\left(a^t y_i - b_i\right)$$

$$= \sum_{i=1}^{n} 2\left(a^t y_i - b_i\right) y_i$$

$$= 2Y^t(Ya - b)$$

Pseudo-Inverse Solution

$$\nabla J_s(a) = 2Y^t(Ya - b)$$

- Setting the gradient to 0:

$$2Y^t(Ya - b) = 0 \;\Rightarrow\; Y^tYa = Y^tb$$

- The matrix $\mathbf{Y^tY}$ is square (it has d +1 rows and columns) and it is often non-singular
- If $\mathbf{Y^tY}$ is non-singular, its inverse exists and we can solve for **a** uniquely:

$$a = \boxed{\left(Y^tY\right)^{-1}Y^t}\, b$$

pseudo inverse of Y

$$\left(\left(Y^tY\right)^{-1}Y^t\right)Y = \left(Y^tY\right)^{-1}\left(Y^tY\right) = I$$

MSE Procedures

- Only guaranteed separating hyperplane if **Ya > 0**
  - That is if all elements of vector **Ya** are positive

$$\boldsymbol{Ya} = \begin{bmatrix} b_1 + \varepsilon_1 \\ \vdots \\ b_n + \varepsilon_n \end{bmatrix}$$

  - where $\boldsymbol{\varepsilon}$ may be negative
- If $\boldsymbol{\varepsilon_1,\ldots,\varepsilon_n}$ are small relative to $\mathbf{b_1,\ldots,b_n}$, then each element of **Ya** is positive, and **a** gives a separating hyperplane
  - If the approximation is not good, $\boldsymbol{\varepsilon_i}$ may be large and negative, for some **i**, thus $\mathbf{b_i} + \boldsymbol{\varepsilon_i}$ will be negative and **a** is not a separating hyperplane
- In linearly separable case, least squares solution **a** does not necessarily give separating hyperplane

MSE Procedures

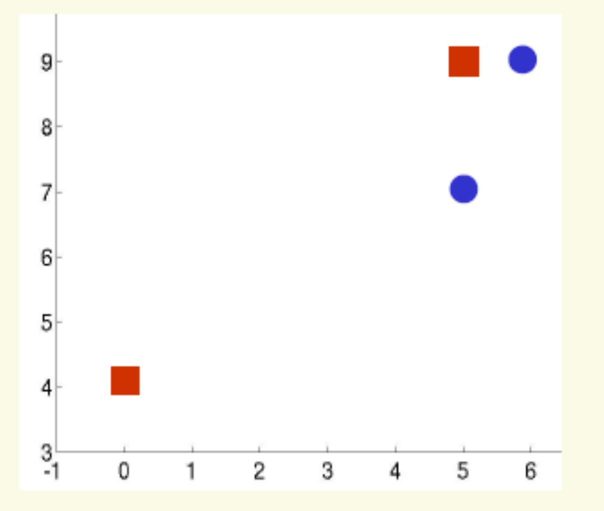
- We are free to choose **b.** We may be tempted to make **b** large as a way to ensure **Ya** =**b > 0**
  - Does not work
  - Let β be a scalar, let's try β**b** instead of **b**
- If **a*** is a least squares solution to **Ya = b,** then for any scalar β, the least squares solution to **Ya =** β**b** is β**a***

$$\arg\min_a \|Ya - \beta b\|^2 = \arg\min_a \beta^2 \|Y(a/\beta) - b\|^2 = \beta a^*$$

- Thus if the **i** th element of **Ya** is less than **0**, that is $\mathbf{y}_i^t\mathbf{a} < 0$, then $\mathbf{y}_i^t(\beta\mathbf{a}) < 0$,
  - The relative difference between components of **b** matters, but not the size of each individual component

LDF using MSE: Example 1

- Class 1: (6 9), (5 7)
- Class 2: (5 9), (0 4)
- Add extra feature and "normalize"

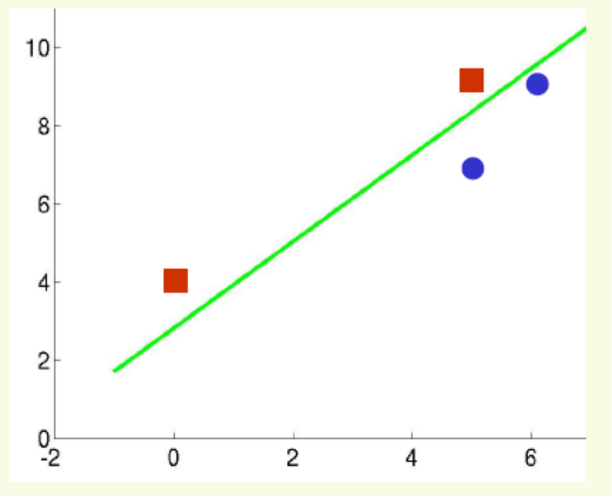


$$y_1 = \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} \quad y_3 = \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ 0 \\ -4 \end{bmatrix}$$

$$Y = \begin{bmatrix} 1 & 6 & 9 \\ 1 & 5 & 7 \\ -1 & -5 & -9 \\ -1 & 0 & -4 \end{bmatrix}$$

LDF using MSE: Example 1

- Choose **b=[1 1 1 1]^T**
- In Matlab, **a=Y\b** solves the least squares problem



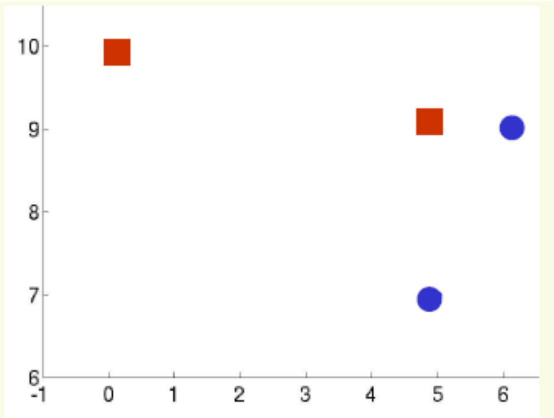
$$a = \begin{bmatrix} 2.66 \\ 1.045 \\ -0.944 \end{bmatrix}$$

- Note **a** is an approximation to **Ya = b,** since no exact solution exists
- This solution gives a separating hyperplane since **Ya >0**

$$Ya = \begin{bmatrix} 0.44 \\ 1.28 \\ 0.61 \\ 1.11 \end{bmatrix}$$

LDF using MSE: Example 2

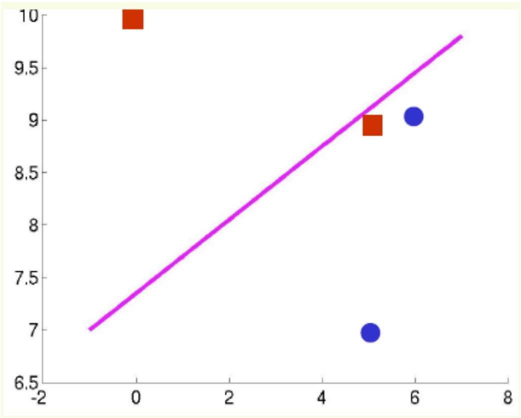
- Class 1: (6 9), (5 7)
- Class 2: (5 9), (0 10)
- The last sample is very far compared to others from the separating hyperplane



$$y_1 = \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} \quad y_3 = \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ 0 \\ -10 \end{bmatrix}$$

$$Y = \begin{bmatrix} 1 & 6 & 9 \\ 1 & 5 & 7 \\ -1 & -5 & -9 \\ -1 & 0 & -10 \end{bmatrix}$$

LDF using MSE: Example 2



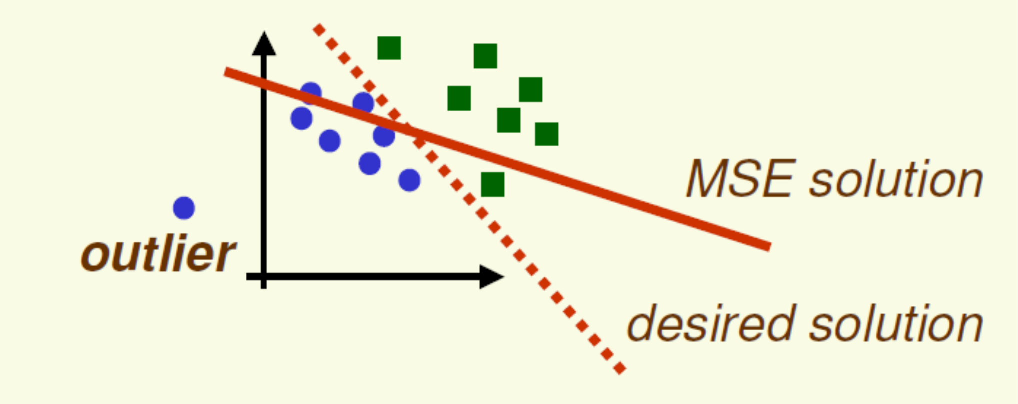
- Choose **b=[1 1 1 1]**^{T}
- In Matlab, **a=Y\b** solves the least squares problem

$$a = \begin{bmatrix} 3.2 \\ 0.2 \\ -0.4 \end{bmatrix} \qquad Ya = \begin{bmatrix} 0.2 \\ 0.9 \\ -0.04 \\ 1.16 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

- This solution does not provide a separating hyperplane since $\mathbf{a}^t\mathbf{y}_3 < 0$

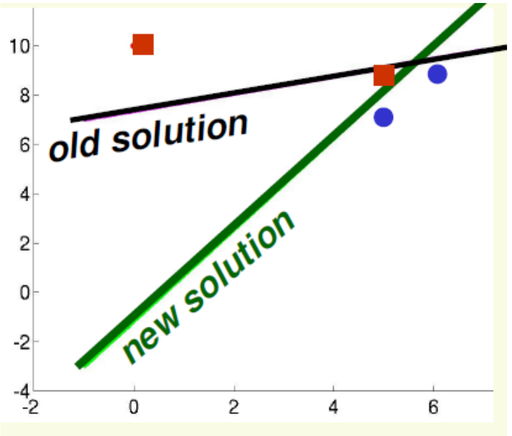
LDF using MSE: Example 2

- MSE pays too much attention to isolated “noisy” examples
  - such examples are called outliers



- No problems with convergence
- Solution ranges from reasonable to good

LDF using MSE: Example 2



- We can see that the 4th point is vary far from separating hyperplane
  - In practice we don't know this
- A more appropriate **b** could be $b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 10 \end{bmatrix}$
- In Matlab, solve **a=Y\b**

$$a = \begin{bmatrix} -1.1 \\ 1.7 \\ -0.9 \end{bmatrix} \qquad Ya = \begin{bmatrix} 0.9 \\ 1.0 \\ 0.8 \\ 10.0 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 10 \end{bmatrix}$$

- This solution gives the separating hyperplane since **Ya > 0**

Gradient Descent for MSE

$$J_s(\mathbf{a}) = \|\mathbf{Ya} - \mathbf{b}\|^2$$

- May wish to find MSE solution by gradient descent:
  1. Computing the inverse of $\mathbf{Y}^t\mathbf{Y}$ may be too costly
  2. $\mathbf{Y}^t\mathbf{Y}$ may be close to singular if samples are highly correlated (rows of **Y** are almost linear combinations of each other) computing the inverse of $\mathbf{Y}^t\mathbf{Y}$ is not numerically stable
- As shown before, the gradient is:

$$\nabla J_s(\mathbf{a}) = 2\mathbf{Y}^t(\mathbf{Ya} - \mathbf{b})$$

Widrow-Hoff Procedure

$$\nabla J_s(\mathbf{a}) = 2Y^t(Y\mathbf{a} - \mathbf{b})$$

- Thus the update rule for gradient descent is:

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} - \eta^{(k)} Y^t \left(Y\mathbf{a}^{(k)} - \mathbf{b} \right)$$

- If $\boldsymbol{\eta^{(k)}=\eta^{(1)}/k}$, then $\mathbf{a}^{(k)}$ converges to the MSE solution $\mathbf{a}$, that is $\mathbf{Y^t(Ya-b)=0}$
- The *Widrow-Hoff procedure* reduces storage requirements by considering single samples sequentially

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} - \eta^{(k)} \mathbf{y}_i \left(\mathbf{y}_i^t \mathbf{a}^{(k)} - b_i \right)$$

LDF Summary

- **Perceptron procedures**
  - Find a separating hyperplane in the linearly separable case,
  - Do not converge in the non-separable case
  - Can force convergence by using a decreasing learning rate, but are not guaranteed a reasonable stopping point
- **MSE procedures**
  - Converge in separable and not separable case
  - May not find separating hyperplane even if classes are linearly separable
  - Use pseudoinverse if $\mathbf{Y}^t\mathbf{Y}$ is not singular and not too large
  - Use gradient descent (Widrow-Hoff procedure) otherwise

Support Vector Machines

SVM Resources

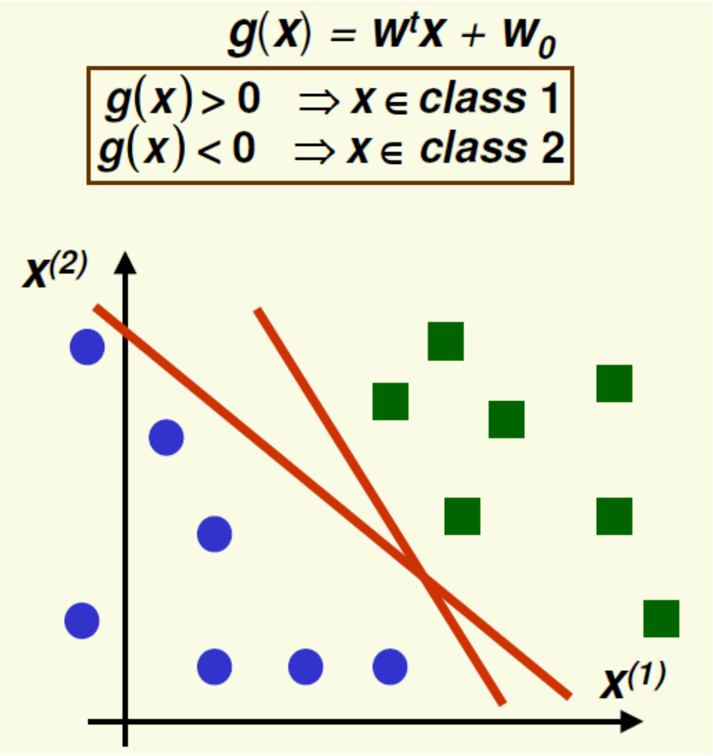
- Burges tutorial
  - http://research.microsoft.com/en-us/um/people/cburges/papers/SVMTutorial.pdf
- Shawe-Taylor and Christianini tutorial
  - http://www.support-vector.net/icml-tutorial.pdf
- Lib SVM
  - http://www.csie.ntu.edu.tw/~cjlin/libsvm/
- LibLinear
  - http://www.csie.ntu.edu.tw/~cjlin/liblinear/
- SVM Light
  - http://svmlight.joachims.org/
- Power Mean SVM (very fast for histogram features)
  - https://sites.google.com/site/wujx2001/home/power-mean-svm

SVMs

- One of the most important developments in pattern recognition in the last decades
- Elegant theory
  - Has good generalization properties
- Have been applied to diverse problems very successfully

Linear Discriminant Functions

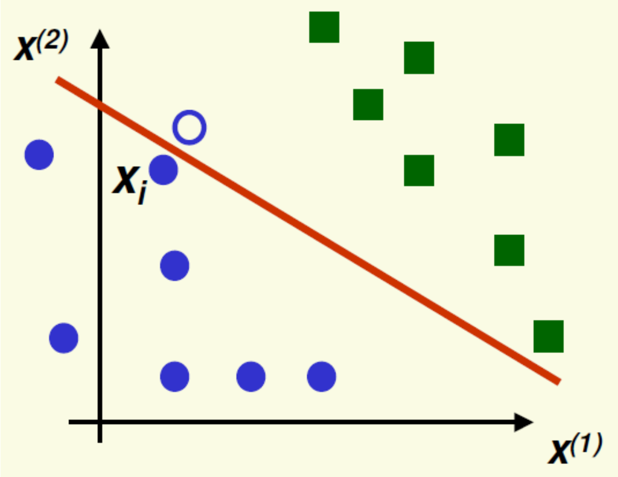
- A discriminant function is linear if it can be written as



- which separating hyperplane should we choose?

Linear Discriminant Functions

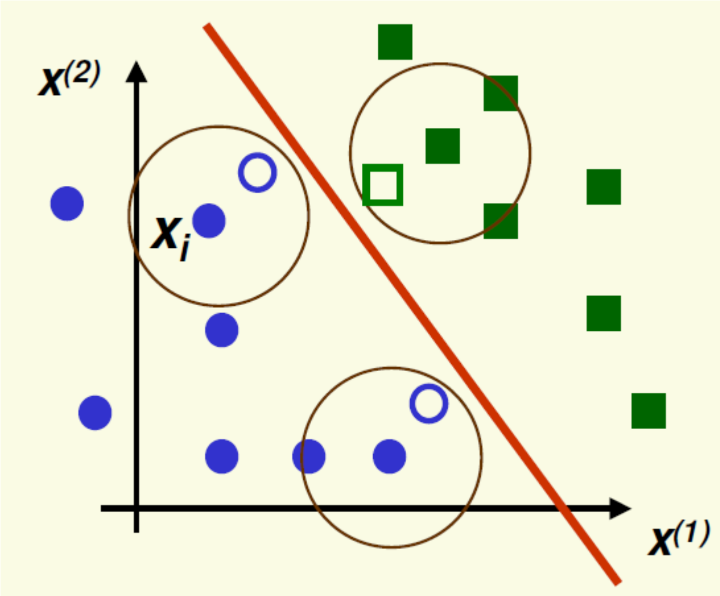
- Training data is just a subset of all possible data
  - Suppose hyperplane is close to sample $\mathbf{x}_i$
  - If we see new sample close to $\mathbf{x}_i$, it may be on the wrong side of the hyperplane



- Poor generalization (performance on unseen data)

Linear Discriminant Functions

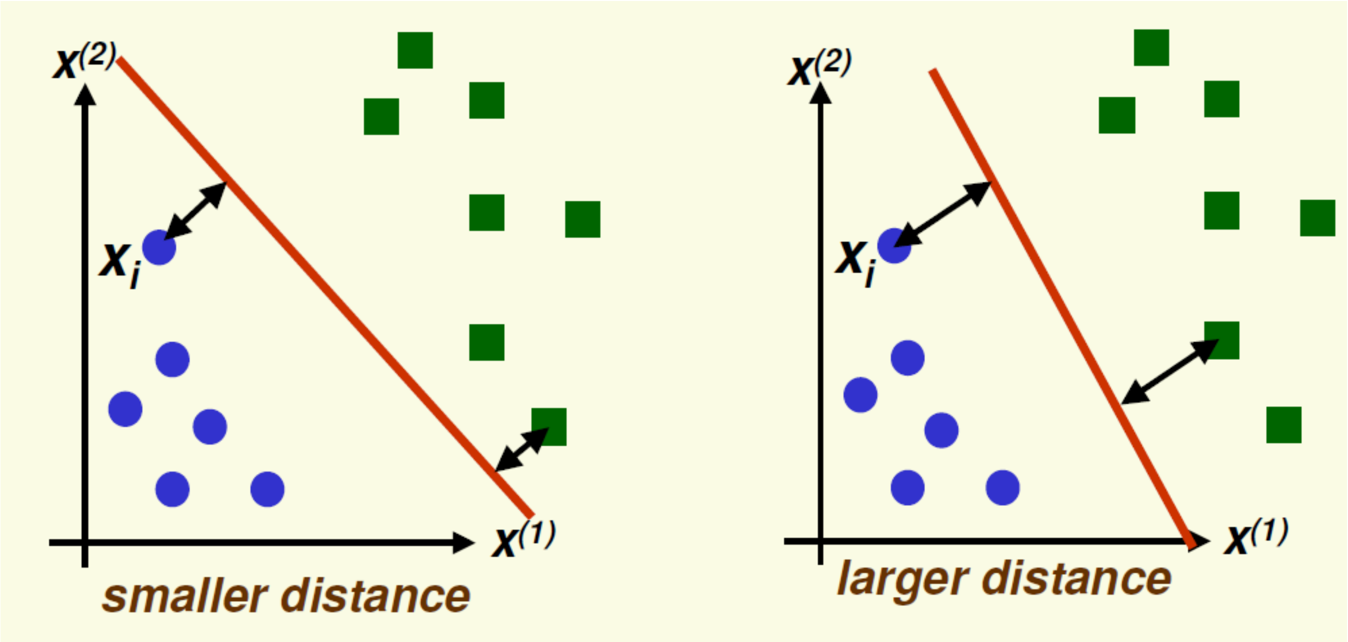
- Hyperplane as far as possible from ***<u>any</u>*** sample



- New samples close to the old samples will be classified correctly
- Good generalization

SVM

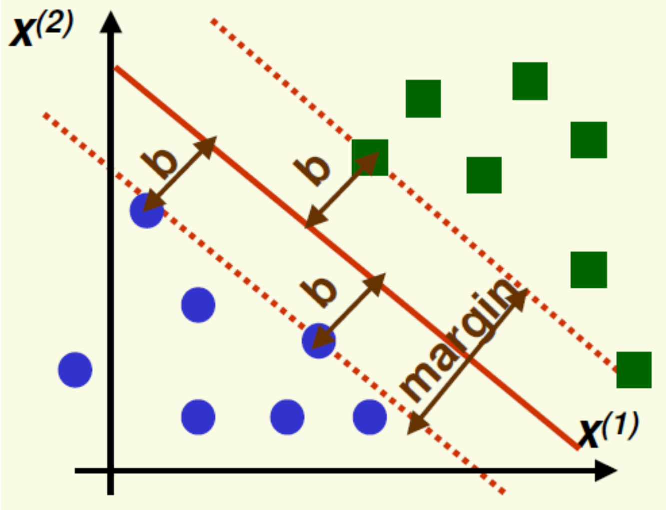
- Idea: maximize distance to the ***closest*** example



- For the optimal hyperplane
  - distance to the closest negative example = distance to the closest positive example

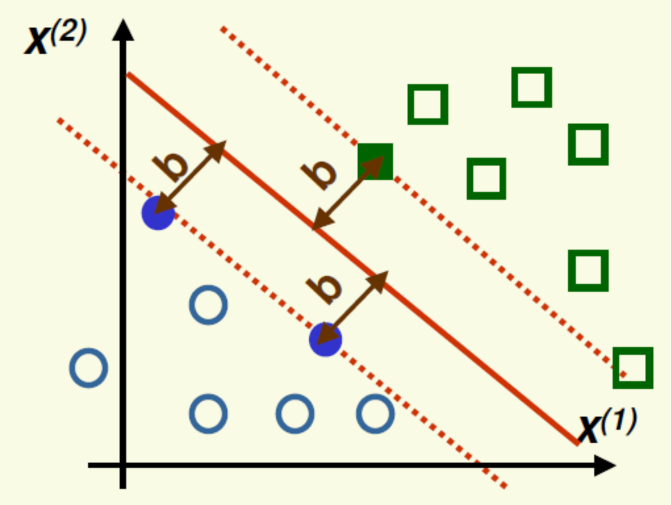
SVM: Linearly Separable Case

- SVM: maximize the margin



- The *margin* is twice the absolute value of distance ***b*** of the closest example to the separating hyperplane
- Better generalization (performance on test data)
  - in practice
  - and in theory

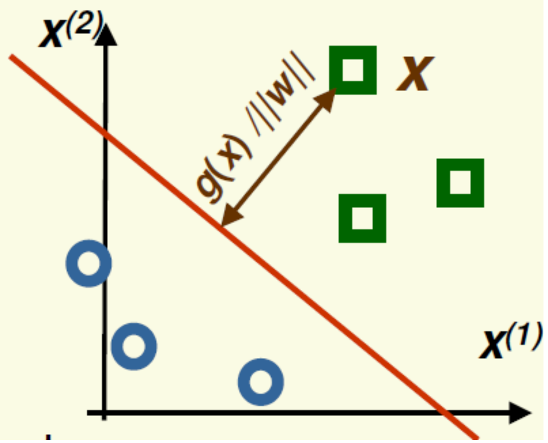
SVM: Linearly Separable Case



- ***Support vectors*** are the samples closest to the separating hyperplane
  - They are the most difficult patterns to classify
  - Recall perceptron update rule
- Optimal hyperplane is completely defined by support vectors
  - Of course, we do not know which samples are support vectors without finding the optimal hyperplane

SVM: Formula for the Margin

$$g(x) = w^t x + w_0$$



- Absolute distance between **x** and the boundary **g(x) = 0**

$$\frac{|w^t x + w_0|}{\|w\|}$$

- Distance is unchanged for hyperplane

$$g_1(x) = \alpha g(x) \qquad \frac{|\alpha w^t x + \alpha w_0|}{\|\alpha w\|} = \frac{|w^t x + w_0|}{\|w\|}$$

- Let $\mathbf{x_i}$ be an example closest to the boundary (on the positive side). Set:

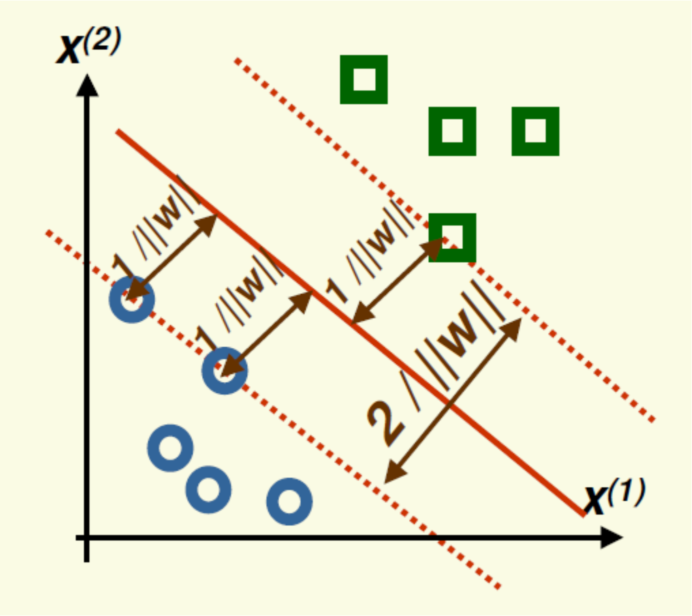
$$|w^t x_i + w_0| = 1$$

- Now the largest margin hyperplane is unique

SVM: Formula for the Margin

- For uniqueness, set $|\mathbf{w}^T\mathbf{x}_i+\mathbf{w}_0|=1$ for any sample $\mathbf{x}_i$ closest to the boundary
- The distance from closest sample $\mathbf{x}_i$ to $\mathbf{g(x) = 0}$ is

$$\frac{|w^t x_i + w_0|}{\|w\|} = \frac{1}{\|w\|}$$



- Thus the margin is

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

SVM: Optimal Hyperplane

- Maximize margin $\boxed{m = \frac{2}{\|w\|}}$
- Subject to constraints $\begin{cases} w^t x_i + w_0 \geq 1 & \text{if } x_i \text{ is positive example} \\ w^t x_i + w_0 \leq -1 & \text{if } x_i \text{ is negative example} \end{cases}$
- Let $\begin{cases} z_i = 1 & \text{if } x_i \text{ is positive example} \\ z_i = -1 & \text{if } x_i \text{ is negative example} \end{cases}$
- Can convert our problem to minimize

$$\boxed{\begin{array}{l} \text{minimize } J(w) = \frac{1}{2}\|w\|^2 \\ \text{constrained to } z_i(w^t x_i + w_o) \geq 1 \quad \forall i \end{array}}$$

- **J(w)** is a quadratic function, thus there is a single global minimum

SVM: Optimal Hyperplane

- Use Kuhn-Tucker theorem to convert our problem to:
  - Also know as the Karush–Kuhn–Tucker theorem, i.e., the KKT theorem

$$\boxed{\begin{aligned} &\text{maximize} \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j \mathbf{x}_i^t \mathbf{x}_j \\ &\text{constrained to} \quad \alpha_i \geq 0 \;\; \forall i \;\; \text{and} \;\; \sum_{i=1}^{n} \alpha_i z_i = 0 \end{aligned}}$$

- $\boldsymbol{a} = \{\boldsymbol{a_1}, \ldots, \boldsymbol{a_n}\}$ are new variables, one for each sample
- Optimized by quadratic programming

SVM: Optimal Hyperplane

- After finding the optimal $\boldsymbol{a} = \{\boldsymbol{a_1}, \ldots, \boldsymbol{a_n}\}$
- Final discriminant function:

$$g(x) = \left(\sum_{x_i \in S} \alpha_i z_i x_i \right)^t x + w_0$$

- where ***S is the set of support vectors***

$$S = \{x_i \mid \alpha_i \neq 0\}$$

SVM: Optimal Hyperplane

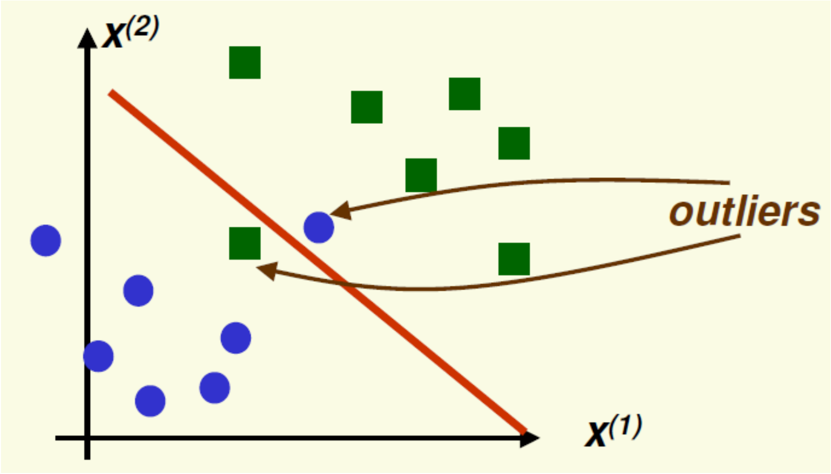
$$\text{maximize} \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j \mathbf{x}_i^t \mathbf{x}_j$$

$$\text{constrained to} \quad \alpha_i \geq 0 \quad \forall i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i z_i = 0$$

- **L_D(a)** depends on the number of samples, not on dimension
  - samples appear only through the dot products $\mathbf{x}_j^t \mathbf{x}_i$
- This will become important when looking for a nonlinear discriminant function, as we will see soon

SVM: Non-Separable Case

- Data are most likely to be not linearly separable, but linear classifier may still be appropriate



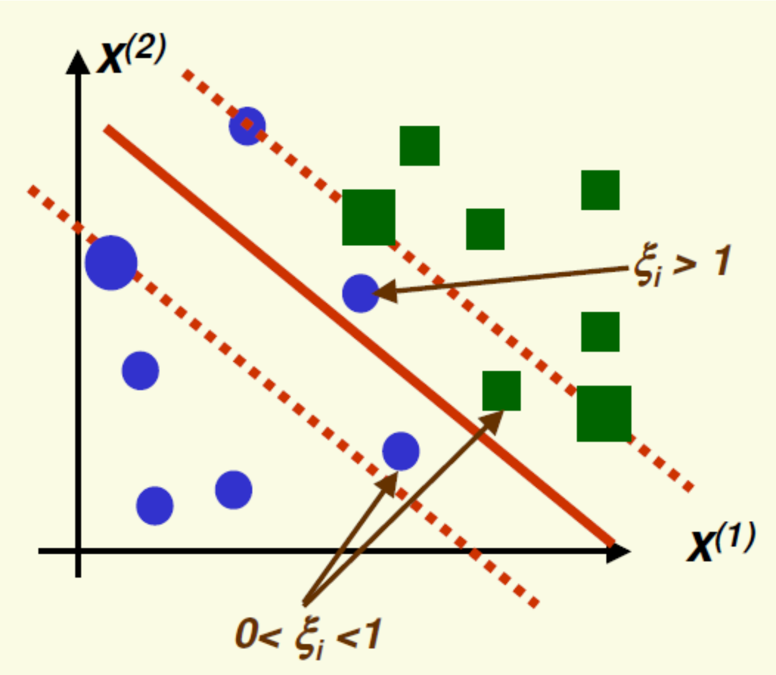
- Can apply SVM in non linearly separable case
- Data should be “almost” linearly separable for good performance

SVM: Non-Separable Case

- Use slack variables $\xi_1, \ldots, \xi_n$ (one for each sample)
- Change constraints from $z_i(w^t x_i + w_0) \geq 1 \quad \forall i$ to

$$z_i(w^t x_i + w_0) \geq 1 - \xi_i \quad \forall i$$

- ξ_i is a measure of deviation from the ideal for x_i
  - $\xi_i > 1$: x_i is on the wrong side of the separating hyperplane
  - $0 < \xi_i < 1$: x_i is on the right side of separating hyperplane but within the region of maximum margin
  - $\xi_i < 0$: is the ideal case for x_i



SVM: Non-Separable Case

- We would like to minimize

$$J(w,\xi_1,\ldots,\xi_n)=\frac{1}{2}\|w\|^2+\beta\sum_{i=1}^{n}I(\xi_i>0)$$

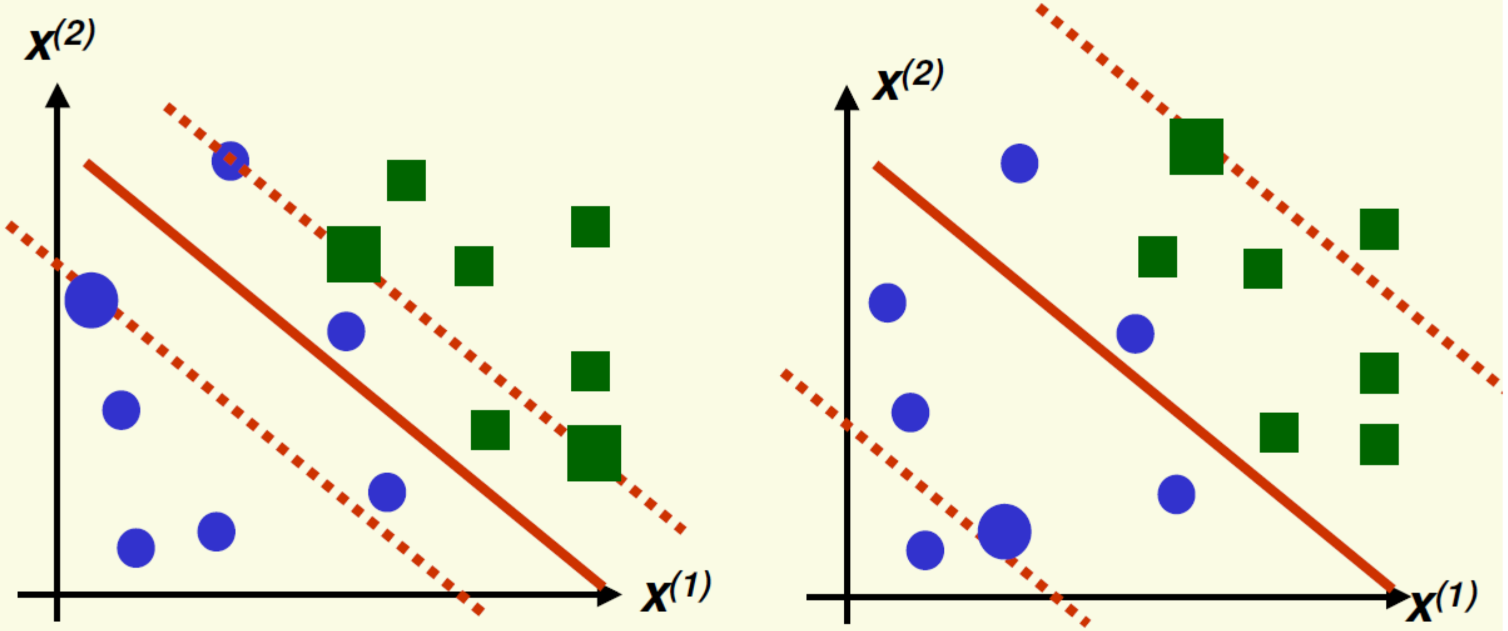
# of samples not in ideal location

- where $I(\xi_i>0)=\begin{cases}1 & \text{if } \xi_i>0\\ 0 & \text{if } \xi_i\le 0\end{cases}$
- Constrained to $z_i(w^t x_i+w_0)\ge 1-\xi_i$ and $\xi_i\ge 0 \quad \forall i$
- β is a constant that measures the relative weight of first and second term
  - If β is small, we allow a lot of samples to be in not ideal positions
  - If β is large, few samples can be in non-ideal positions

SVM: Non-Separable Case

$$J(w,\xi_1,\ldots,\xi_n)=\frac{1}{2}\|w\|^2+\beta\sum_{i=1}^{n}I(\xi_i>0)$$

\# of examples not in ideal location



large β, few samples not in ideal position

small β, a lot of samples not in ideal position

SVM: Non-Separable Case

- Unfortunately this minimization problem is NP-hard due to the discontinuity of $I(\xi_i)$
- Instead, we minimize

$$J(w,\xi_1,\dots,\xi_n)=\frac{1}{2}\|w\|^2+\beta\sum_{i=1}^{n}\xi_i$$

a measure of # of misclassified examples

- Subject to

$$\begin{cases} z_i\left(w^t x_i + w_0\right)\geq 1-\xi_i & \forall i \\ \xi_i \geq 0 & \forall i \end{cases}$$

SVM: Non-Separable Case

- Use Kuhn-Tucker theorem to convert to:

$$\text{maximize} \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j x_i^t x_j$$

$$\text{constrained to} \quad 0 \le \alpha_i \le \beta \quad \forall i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i z_i = 0$$

- **w** is computed using:

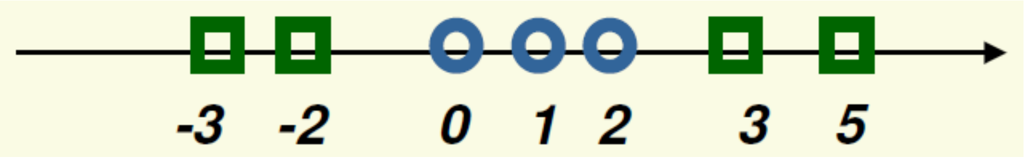
$$w = \sum_{i=1}^{n} \alpha_i z_i x_i$$

- Remember that

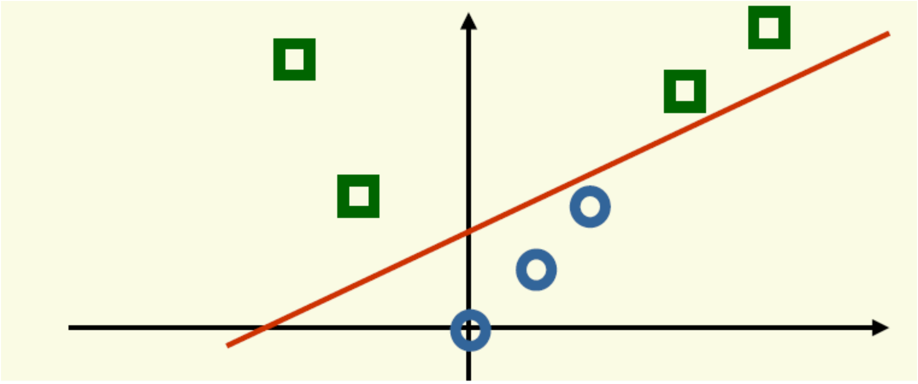
$$g(x) = \left(\sum_{x_i \in S} \alpha_i z_i x_i \right)^t x + w_0$$

Nonlinear Mapping

- Cover's theorem: *"a pattern-classification problem cast in a high dimensional space non-linearly is more likely to be linearly separable than in a low-dimensional space"*
- One dimensional space, not linearly separable



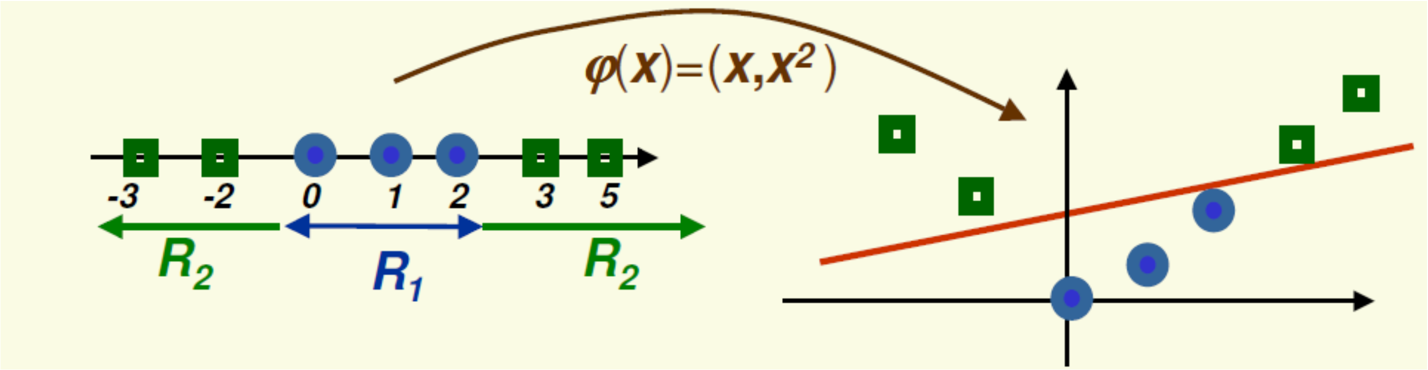
- Lift to two dimensional space with $\phi(\mathbf{x})=(\mathbf{x},\mathbf{x}^2)$



Nonlinear Mapping

- To solve a non linear classification problem with a linear classifier

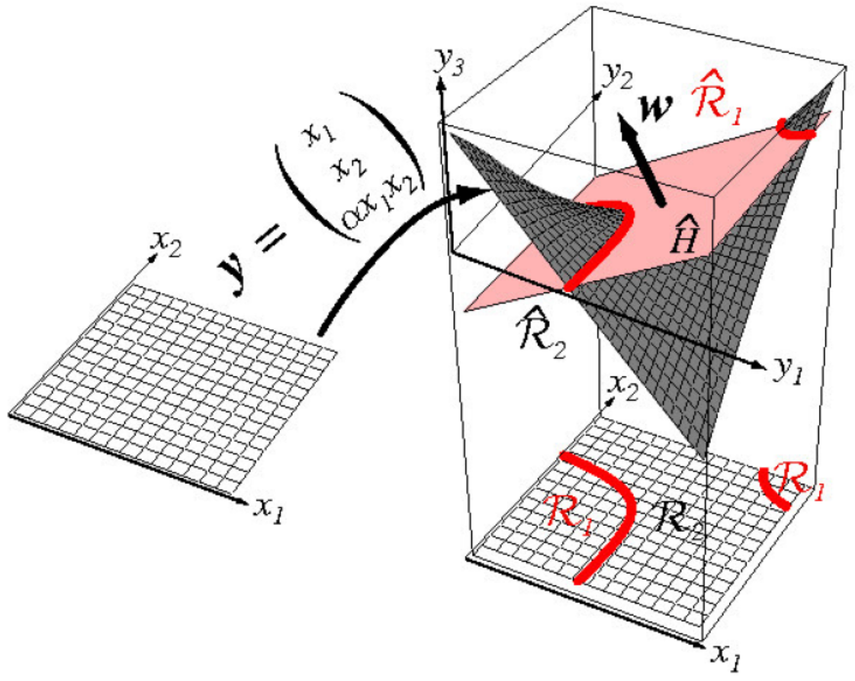
1. Project data ***x*** to high dimension using function **ϕ(x)**
2. Find a linear discriminant function for transformed data **ϕ(x)**
3. Final nonlinear discriminant function is $\mathbf{g(x) = w^t\phi(x) + w_0}$



- In 2D, the discriminant function is linear $$g\left(\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix}\right) = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} + w_0$$

- In 1D, the discriminant function is not linear $$g(x) = w_1 x + w_2 x^2 + w_0$$

Nonlinear Mapping



- However, there always exists a mapping of N samples to an N-dimensional space in which the samples are separable by hyperplanes

Nonlinear SVM

- Can use any linear classifier after lifting data to a higher dimensional space. However we will have to deal with the curse of dimensionality
  - Poor generalization to test data
  - Computationally expensive

- SVM avoids the curse of dimensionality problems
  - Enforcing largest margin permits good generalization
    - It can be shown that generalization in SVM is a function of the margin, independent of the dimensionality
  - Computation in the higher dimensional case is performed only implicitly through the use of ***kernel functions***

Kernels

- SVM optimization:

maximize
$$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j x_i^t x_j$$

- Note this optimization depends on samples $\boldsymbol{x_i}$ only through the dot product $\boldsymbol{x_i^t x_j}$
- If we lift $\boldsymbol{x_i}$ to high dimension using $\boldsymbol{\varphi(x)}$, we need to compute high dimensional product $\boldsymbol{\varphi(x_i)^t \varphi(x_j)}$

maximize
$$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j \underbrace{\varphi(x_i)^t \varphi(x_j)}_{K(x_i, x_j)}$$

- Idea: find kernel function $\mathbf{K(x_i,x_j)}$ s.t. $K(x_i, x_j) = \varphi(x_i)^t \varphi(x_j)$

Kernel Trick

- Then we only need to compute $\mathbf{K(x_i,x_j)}$ instead of $\boldsymbol{\phi}(\mathbf{x_i})^\mathbf{t}\ \boldsymbol{\phi}\mathbf{(x_j)}$
- "kernel trick": do not need to perform operations in high dimensional space explicitly

Kernel Example

- Suppose we have two features and **K(x,y) = (x^{t}y)^2**
- Which mapping **ϕ(x)** does this correspond to?

$$K(x,y) = (x^t y)^2 = \left(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix} \begin{bmatrix} y^{(1)} \\ y^{(2)} \end{bmatrix} \right)^2 = \left(x^{(1)} y^{(1)} + x^{(2)} y^{(2)} \right)^2$$

$$= \left(x^{(1)} y^{(1)} \right)^2 + 2\left(x^{(1)} y^{(1)} \right)\left(x^{(2)} y^{(2)} \right) + \left(x^{(2)} y^{(2)} \right)^2$$

$$= \left[\left(x^{(1)} \right)^2 \quad \sqrt{2} x^{(1)} x^{(2)} \quad \left(x^{(2)} \right)^2 \right] \left[\left(y^{(1)} \right)^2 \quad \sqrt{2} y^{(1)} y^{(2)} \quad \left(y^{(2)} \right)^2 \right]^t$$

$$\varphi(x) = \left[\left(x^{(1)} \right)^2 \quad \sqrt{2} x^{(1)} x^{(2)} \quad \left(x^{(2)} \right)^2 \right]$$

Choice of Kernel

- How to choose kernel function $\mathbf{K(x_i,x_j)}$?
  - $\mathbf{K(x_i,x_j)}$ should correspond to $\boldsymbol{\phi}(\mathbf{x_i})^t\ \boldsymbol{\phi}(\mathbf{x_j})$ in a higher dimensional space
  - Mercer's condition tells us which kernel function can be expressed as dot product of two vectors
  - If **K** and **K'** are kernels **aK+bK'** is a kernel
- Intuitively: Kernel should measure the similarity between $\mathbf{x_i}$ and $\mathbf{x_j}$
  - As inner product measures similarity of unit vectors
  - May be problem-specific

Choice of Kernel

- Some common choices:
  - Polynomial kernel

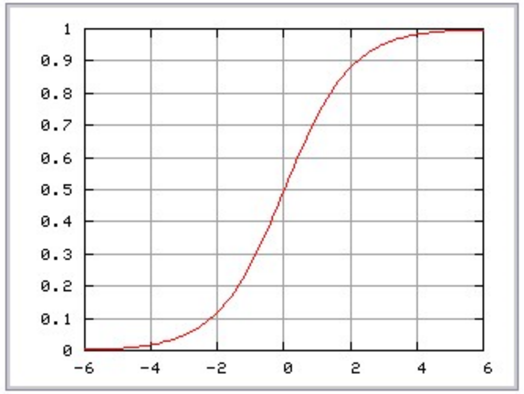
$$K(x_i, x_j) = (x_i^t x_j + 1)^p$$

  - Gaussian radial Basis kernel

$$K(x_i, x_j) = \exp\left(-\frac{1}{2\sigma^2}\|x_i - x_j\|^2\right)$$

  - Hyperbolic tangent (sigmoid) kernel

$$K(x_i,x_j) = \tanh(k\, x_i^t x_j + c)$$



- The mappings $\boldsymbol{\phi}(\mathbf{x}_i)$ never have to be computed!!

Intersection Kernel

- Feature vectors are histograms

$$K(x_i, x_j) = \sum_{k=1}^{n} \min(x_{ik}, x_{jk})$$

- When $\mathbf{K(x_i,x_j)}$ is small, $\mathbf{x_i}$ and $\mathbf{x_j}$ are dissimilar
- When $\mathbf{K(x_i,x_j)}$ is large, $\mathbf{x_i}$ and $\mathbf{x_j}$ are similar

- The mapping $\boldsymbol{\phi}(\mathbf{x})$ does not exist

More Additive Kernels

- χ^2 kernel

$$K_{\chi^2} = \sum_{k=1}^{n} \frac{2x_k y_k}{x_k + y_k}$$

- Hellinger's kernel

$$K_H = \sum_{k=1}^{n} \sqrt{x_k y_k}$$

- Designed for feature vectors that are histograms
  - Can be used for other feature vectors
- Offer very large speed-ups

The Kernel Matrix

- a.k.a the Gram matrix

K=

K(1,1)	K(1,2)	K(1,3)	…	K(1,m)
K(2,1)	K(2,2)	K(2,3)	…	K(2,m)
…	…	…	…	…
K(m,1)	K(m,2)	K(m,3)	…	K(m,m)

- Contains all necessary information for the learning algorithm
- Fuses information about the data and the kernel (similarity measure)

Bad Kernels

- The kernel matrix is mostly diagonal
  - All points are orthogonal to each other
- Bad similarity measure
- Too many irrelevant features in high dimensional space

- We need problem-specific knowledge to choose appropriate kernel

Nonlinear SVM Step-by-Step

- Start with data $\mathbf{x}_1,\ldots,\mathbf{x}_n$ which live in feature space of dimension $\mathbf{d}$
- Choose kernel $\mathbf{K}(\mathbf{x}_i,\mathbf{x}_j)$ or function $\boldsymbol{\phi}(\mathbf{x}_i)$ which lifts sample $\mathbf{x}_i$ to a higher dimensional space
- Find the maximum margin linear discriminant function in the higher dimensional space by using quadratic programming package to solve:

$$\text{maximize} \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j K(x_i, x_j)$$

$$\text{constrained to} \quad 0 \le \alpha_i \le \beta \quad \forall i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i z_i = 0$$

Nonlinear SVM Step-by-Step

- Weight vector **w** in the high dimensional space:

$$\mathbf{w} = \sum_{x_i \in S} \alpha_i z_i \varphi(\mathbf{x}_i)$$

  - where **S** is the set of support vectors

- Linear discriminant function of maximum margin in the high dimensional space:

$$g(\varphi(\mathbf{x})) = \mathbf{w}^t \varphi(\mathbf{x}) = \left(\sum_{x_i \in S} \alpha_i z_i \varphi(\mathbf{x}_i) \right)^t \varphi(\mathbf{x})$$

- Non linear discriminant function in the original space:

$$g(\mathbf{x}) = \left(\sum_{x_i \in S} \alpha_i z_i \varphi(\mathbf{x}_i) \right)^t \varphi(\mathbf{x}) = \sum_{x_i \in S} \alpha_i z_i \varphi^t(\mathbf{x}_i) \varphi(\mathbf{x}) = \sum_{x_i \in S} \alpha_i z_i K(\mathbf{x}_i, \mathbf{x})$$

- decide class 1 if **g(x) > 0**, otherwise decide class 2

Nonlinear SVM

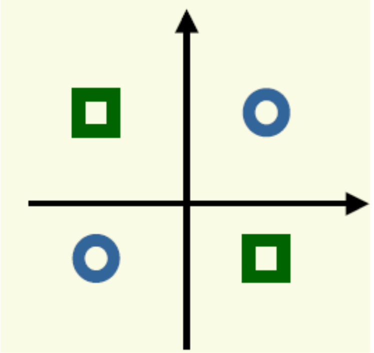
- Nonlinear discriminant function

$$g(\mathbf{x}) = \sum_{\mathbf{x}_i \in S} \boxed{\alpha_i}\,\boxed{z_i}\,\boxed{K(\mathbf{x}_i, \mathbf{x})}$$

$$g(\mathbf{x}) = \sum_{\substack{\text{most important}\\ \text{training samples,}\\ \text{i.e. support vectors}}} \boxed{\begin{array}{c}\text{weight of support}\\ \text{vector } \boldsymbol{x}_i\end{array}}\ \boxed{\mp 1}\ \boxed{\begin{array}{c}\text{“inverse distance”}\\ \text{from } \boldsymbol{x} \text{ to}\\ \text{support vector } \boldsymbol{x}_i\end{array}}$$

$$K(\mathbf{x}_i, \mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_i - \mathbf{x}\right\|^2\right)$$

SVM Example: XOR Problem



- Class 1: $\mathbf{x_1} = [1,-1]$, $\mathbf{x_2} = [-1,1]$
- Class 2: $\mathbf{x_3} = [1,1]$, $\mathbf{x_4} = [-1,-1]$
- Use polynomial kernel of degree 2:

$$K(x_i, x_j) = (x_i^{\,t} x_j + 1)^2$$

  - This kernel corresponds to the mapping

$$\varphi(x) = \begin{bmatrix} 1 & \sqrt{2}x^{(1)} & \sqrt{2}x^{(2)} & \sqrt{2}x^{(1)}x^{(2)} & \left(x^{(1)}\right)^2 & \left(x^{(2)}\right)^2 \end{bmatrix}^t$$

- Need to maximize

$$L_D(\alpha) = \sum_{i=1}^{4} \alpha_i - \frac{1}{2} \sum_{i=1}^{4} \sum_{j=1}^{4} \alpha_i \alpha_j z_i z_j \left(x_i^t x_j + 1\right)^2$$

constrained to

$$0 \le \alpha_i \quad \forall i \quad and \quad \alpha_1 + \alpha_2 - \alpha_3 - \alpha_4 = 0$$

SVM Example: XOR Problem

- After some manipulation …
- The solution is **$a_1 = a_2 = a_3 = a_4 = 0.25$**
  - satisfies the constraints

$$\forall i,\ \ 0 \le \alpha_i \ \ \textit{and} \ \ \alpha_1 + \alpha_2 - \alpha_3 - \alpha_4 = 0$$

- All samples are support vectors

SVM Example: XOR Problem

$$\varphi(x) = \begin{bmatrix} 1 & \sqrt{2}x^{(1)} & \sqrt{2}x^{(2)} & \sqrt{2}x^{(1)}x^{(2)} & \left(x^{(1)}\right)^2 & \left(x^{(2)}\right)^2 \end{bmatrix}^t$$

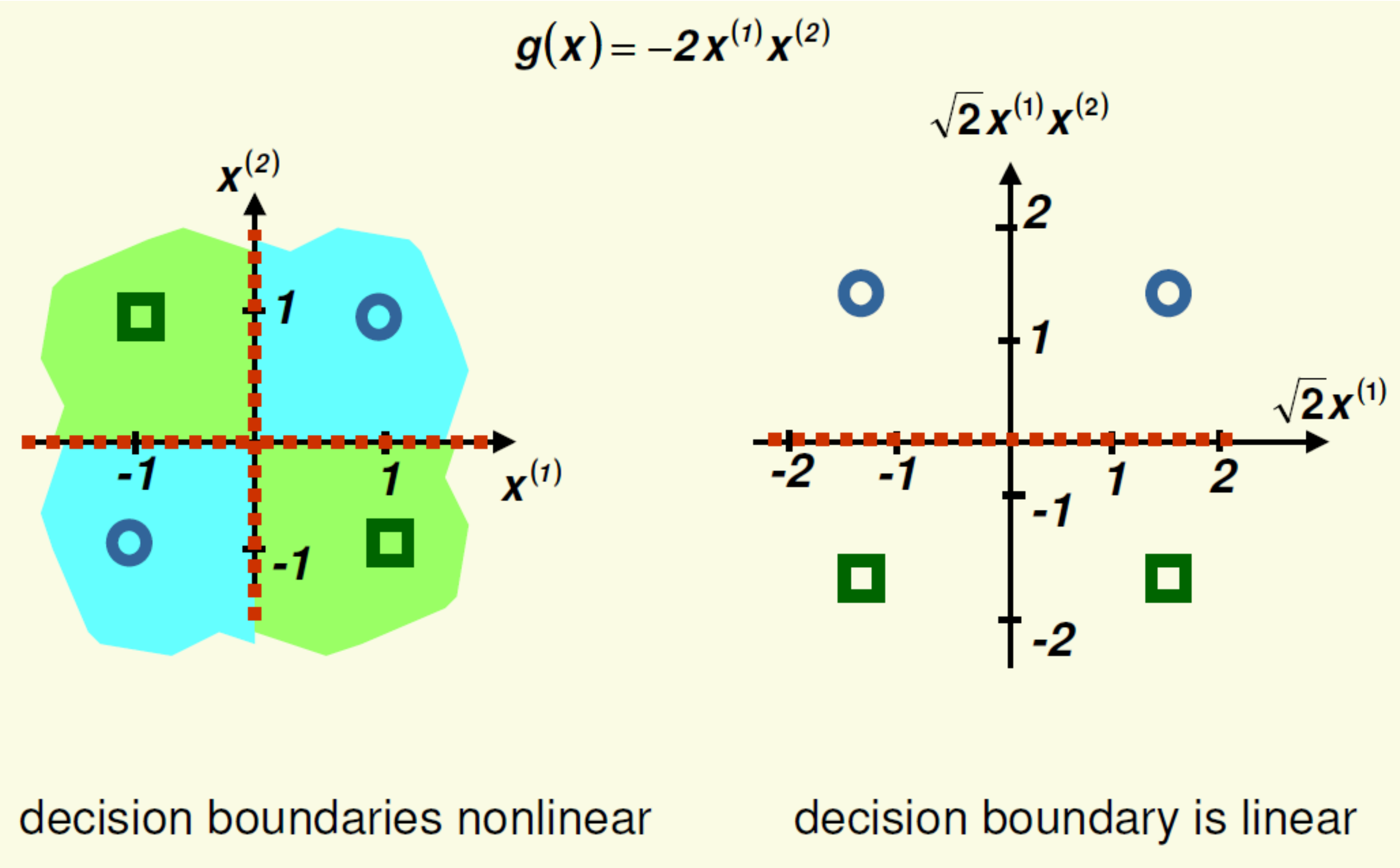
- The weight vector **w** is:

$$w = \sum_{i=1}^{4} \alpha_i z_i \varphi(x_i) = 0.25(\varphi(x_1) + \varphi(x_2) - \varphi(x_3) - \varphi(x_4))$$
$$= \begin{bmatrix} 0 & 0 & 0 & -\sqrt{2} & 0 & 0 \end{bmatrix}$$

- Thus the nonlinear discriminant function is:

$$g(x) = w\varphi(x) = \sum_{i=1}^{6} w_i \varphi_i(x) = -\sqrt{2}\left(\sqrt{2}x^{(1)}x^{(2)}\right) = -2x^{(1)}x^{(2)}$$

SVM Example: XOR Problem



SVM Summary

- Advantages:
  - Based on very strong theory
  - Excellent generalization properties
  - Objective function has no local minima
  - Can be used to find non linear discriminant functions
  - Complexity of the classifier is characterized by the number of support vectors rather than the dimensionality of the transformed space
- Disadvantages:
  - Directly applicable to two-class problems
  - Quadratic programming is computationally expensive
  - Need to choose kernel

Multi-Class SVMs

- One against all
- Pairwise

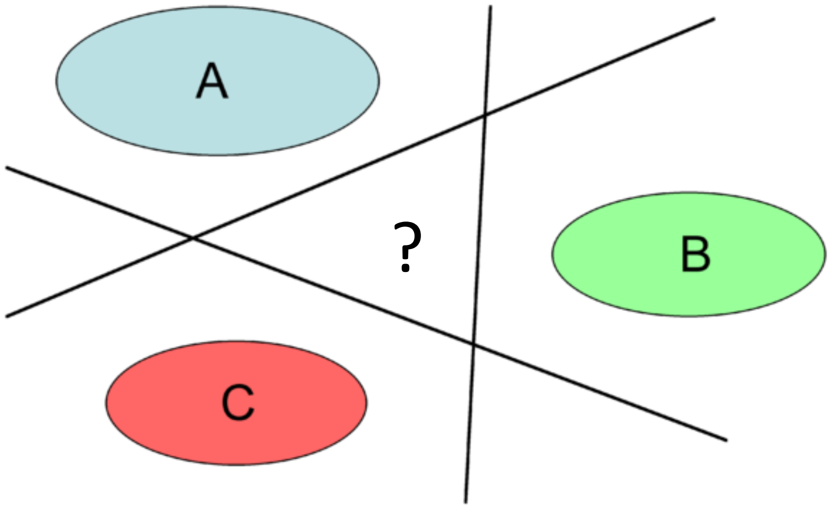
- These ideas apply to all binary classifiers when faced with multi-class problems

One-Against-All

- SVMs can only handle two-class outputs
- What can be done?
- Answer: learn N SVM’s
  - SVM 1 learns “Output==1” vs “Output != 1”
  - SVM 2 learns “Output==2” vs “Output != 2”
  - …
  - SVM N learns “Output==N” vs “Output != N”

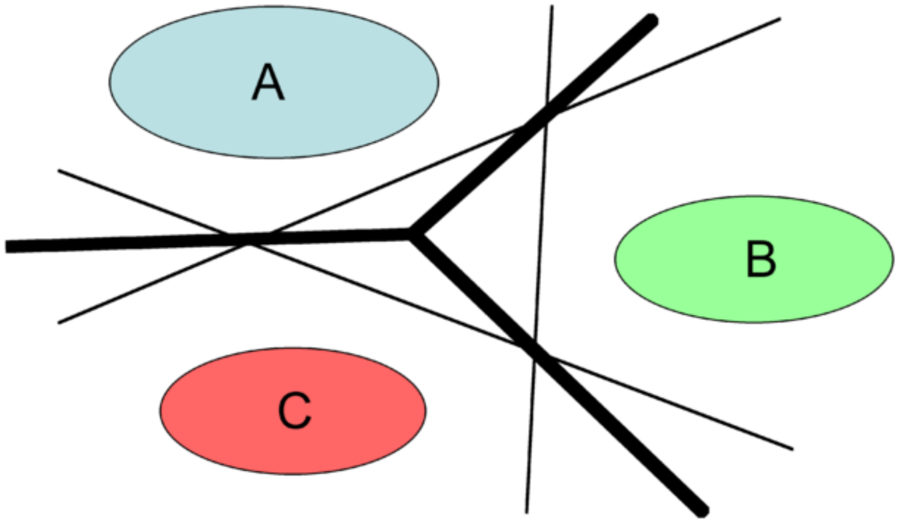
One-Against-All

- Original idea (Vapnik, 1995): classify x as ω_i if and only if the corresponding SVM accepts x and all other SVMs reject it



One-Against-All

- Modified idea (Vapnik, 1998): classify x according to the SVM that produces the highest value (use more than sign of decision function)



Pairwise SVMs

- Learn N(N-1)/2 SVM's
  - SVM 1 learns "Output==1" vs "Output == 2"
  - SVM 2 learns "Output==1" vs "Output == 3"
  - …
  - SVM M learns "Output==N-1" vs "Output == N"

Pairwise SVMs

- To classify a new input, apply each SVM and choose the label that "wins" most often