Machine Learning: Models and Applications

Lecture 8

Lecturer: Xinchao Wang

xinchao@nus.edu.sg

National University of Singapore
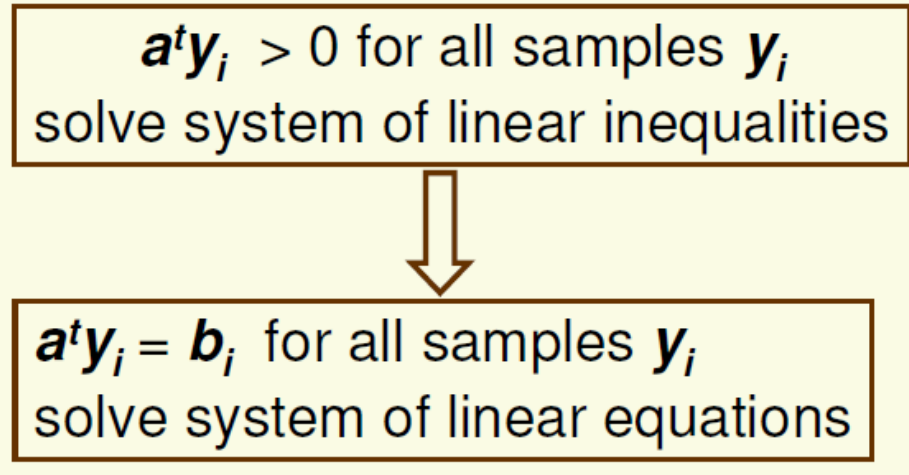
Overview

- 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

$$a^t y_i > 0 \text{ for all samples } y_i$$
solve system of linear inequalities

⬇

$$a^t y_i = b_i \text{ for all samples } y_i$$
solve system of linear equations

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

MSE Margins
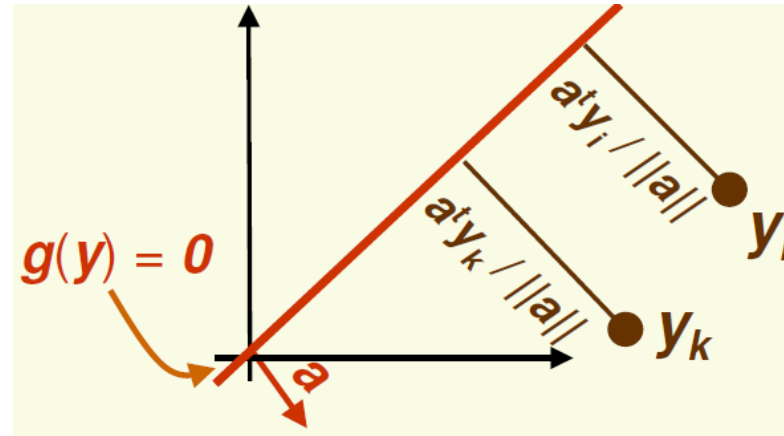
- If $\mathbf{a}^t \mathbf{y}_i = b_i$, $\mathbf{y}_i$ must be at distance b_i from the separating hyperplane (normalized by $\|\mathbf{a}\|$)
- Thus $b_1, b_2, ..., b_n$ give relative expected distances or "*margins*" of samples from the hyperplane
- Should make b_i small if sample i is expected to be near separating hyperplane, and large otherwise
- In the absence of any additional information, set $b_1 = b_2 = ... = b_n = 1$

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 **Ya = b**
  - **Y** is an **n×(d +1)** matrix
- Exact solution only if **Y** is non-singular and square (the inverse **Y⁻¹** exists)
  - a =**Y⁻¹ 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**
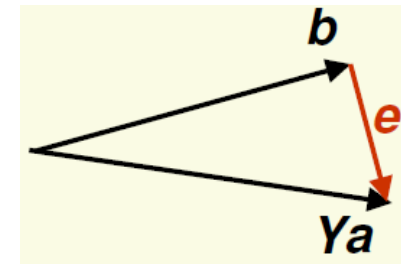
$$\mathbf{e} = \mathbf{Ya} - \mathbf{b}$$

- Thus minimize the minimum squared error criterion function:

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

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

Computing the Gradient

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

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

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

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

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

Pseudo-Inverse Solution

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

- Setting the gradient to 0:

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

- The matrix Y^tY is square (it has d +1 rows and columns) and it is often non-singular

- If Y^tY is non-singular, its inverse exists and we can solve for **a** uniquely:

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

pseudo inverse of **Y**

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

MSE Procedures

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

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

  - where ε may be negative

- If $\varepsilon_1, \dots, \varepsilon_n$ are small relative to $b_1, \dots, b_n$, then each element of **Ya** is positive, and **a** gives a separating hyperplane
  - If the approximation is not good, ε_i may be large and negative, for some **i**, thus $b_i + \varepsilon_i$ 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 = βb** is β**a***

$$\underset{a}{arg\ min}\|Ya - \beta b\|^2 = \underset{a}{arg\ min}\ \beta^2\|Y(a/\beta) - b\|^2 = \beta a *$$

- Thus if the **i** th element of **Ya** is less than **0**, that is $\mathbf{y_i^t a < 0,}$ then $\mathbf{y_i^t (\beta a) < 0,}$
    - 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)
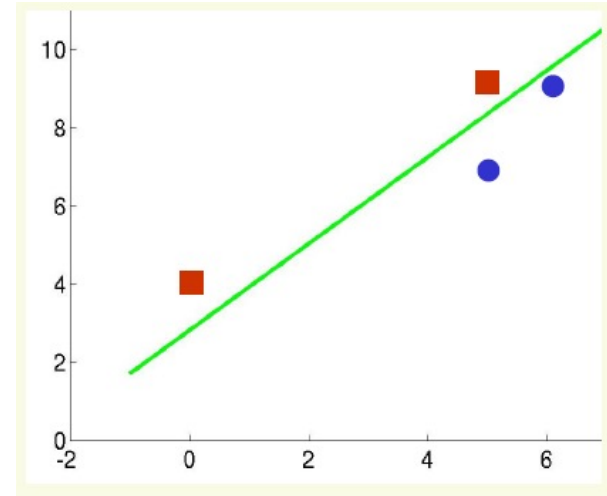- Add extra feature and "normalize"
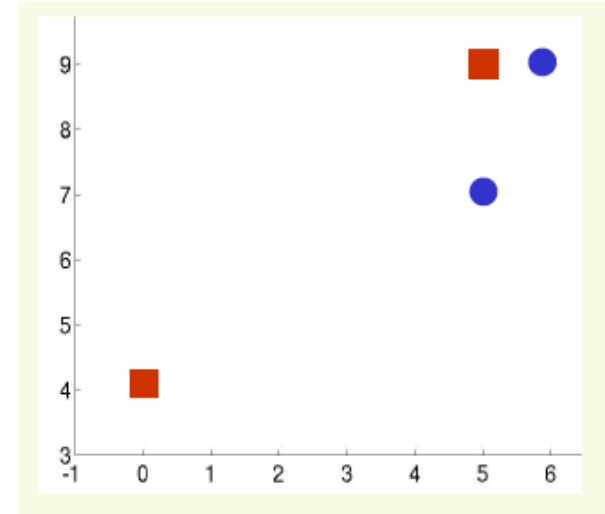
$$y_1 = \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} \quad y_3 = \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ 0 \\ -4 \end{bmatrix}$$

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

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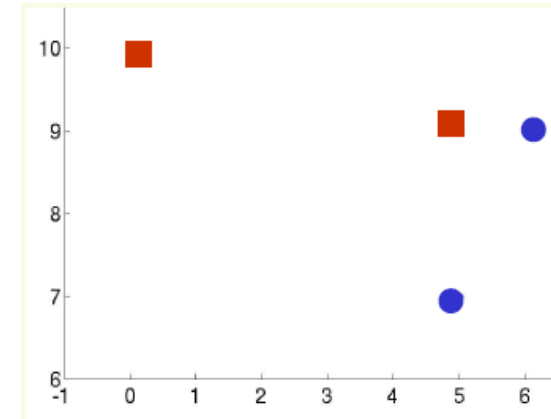
$$y_1 = \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} \quad y_3 = \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ 0 \\ -10 \end{bmatrix}$$
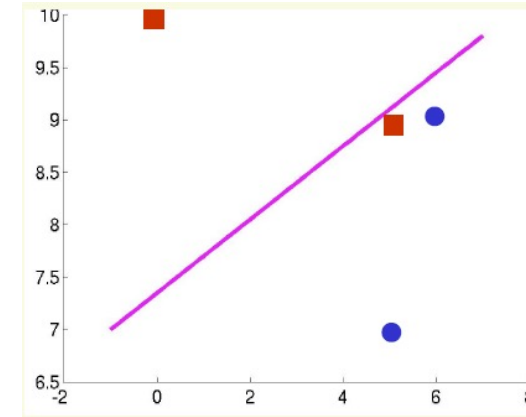
$$Y = \begin{bmatrix} 1 & 6 & 9 \\ 1 & 5 & 7 \\ -1 & -5 & -9 \\ -1 & 0 & -10 \end{bmatrix}$$

LDF using MSE: Example 2

- Choose **b=[1 1 1 1]$^\mathsf{T}$**

- In Matlab, **a=Y\b** solves the least squares problem
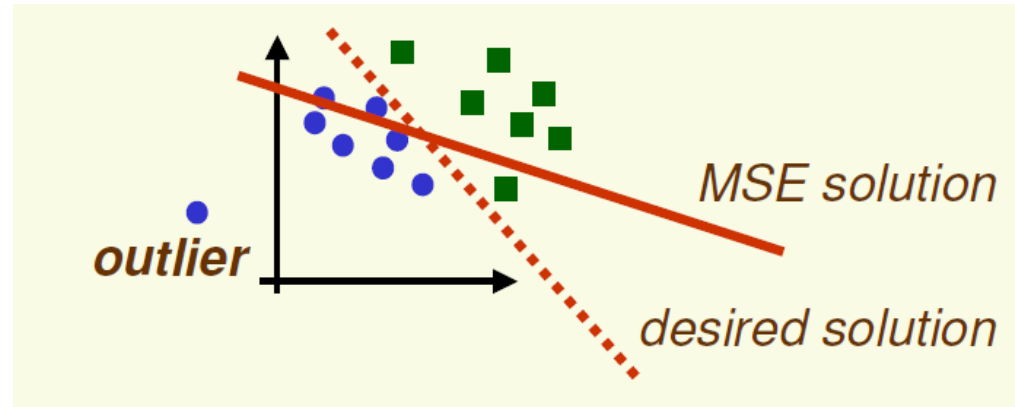

$$a = \begin{bmatrix} 3.2 \\ 0.2 \\ -0.4 \end{bmatrix}$$

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

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

16

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
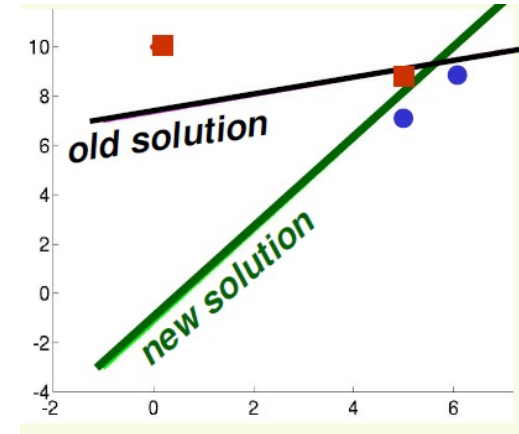
LDF using MSE: Example 2

- We can see that the 4th point is vary far from separating hyperplane
  - In practice we don't know this

$$b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 10 \end{bmatrix}$$

- A more appropriate **b** could be

- In Matlab, solve **a=Y\b**

$$a = \begin{bmatrix} -1.1 \\ 1.7 \\ -0.9 \end{bmatrix}$$

$$Ya = \begin{bmatrix} 0.9 \\ 1.0 \\ 0.8 \\ 10.0 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 10 \end{bmatrix}$$

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

Gradient Descent for MSE

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

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

- As shown before, the gradient is:

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

Widrow-Hoff Procedure

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

- Thus the update rule for gradient descent is:

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

- If $\eta^{(k)} = \eta^{(1)}/k$, then $a^{(k)}$ converges to the MSE solution **a**, that is $Y^t(Ya-b)=0$

- The *Widrow-Hoff procedure* reduces storage requirements by considering single samples sequentially

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

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 $\mathbf{Y^tY}$ 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
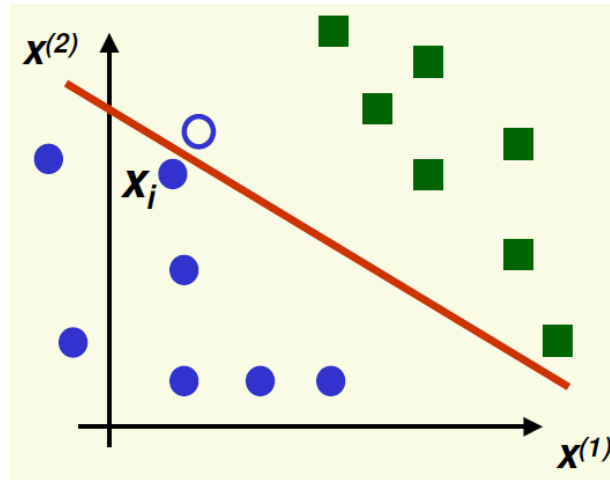
Linear Discriminant Functions

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

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

$$g(x) > 0 \Rightarrow x \in class\ 1$$
$$g(x) < 0 \Rightarrow x \in class\ 2$$
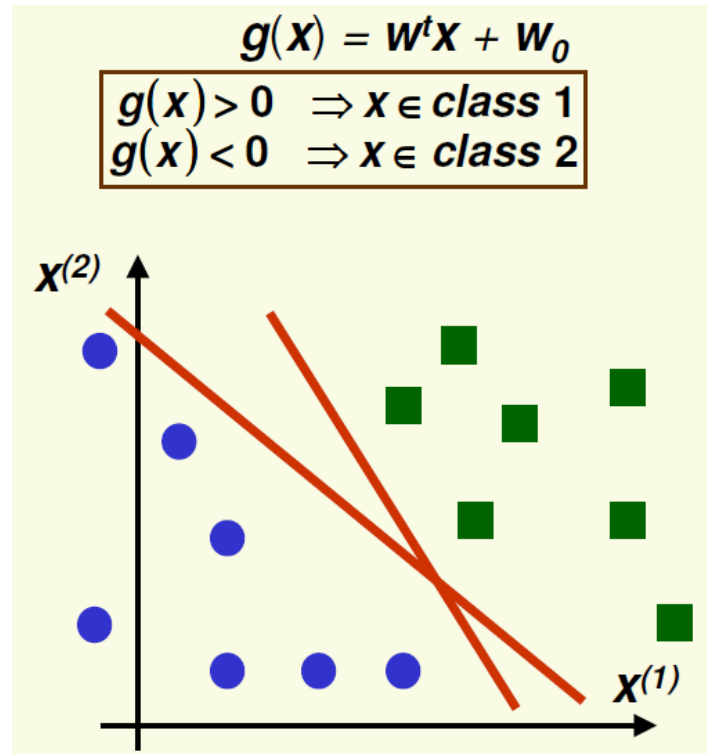
- which separating hyperplane should we choose?

Linear Discriminant Functions

- Training data is just a subset of all possible data
  - Suppose hyperplane is close to sample x_i
  - If we see new sample close to x_i, it may be on the wrong side of the hyperplane

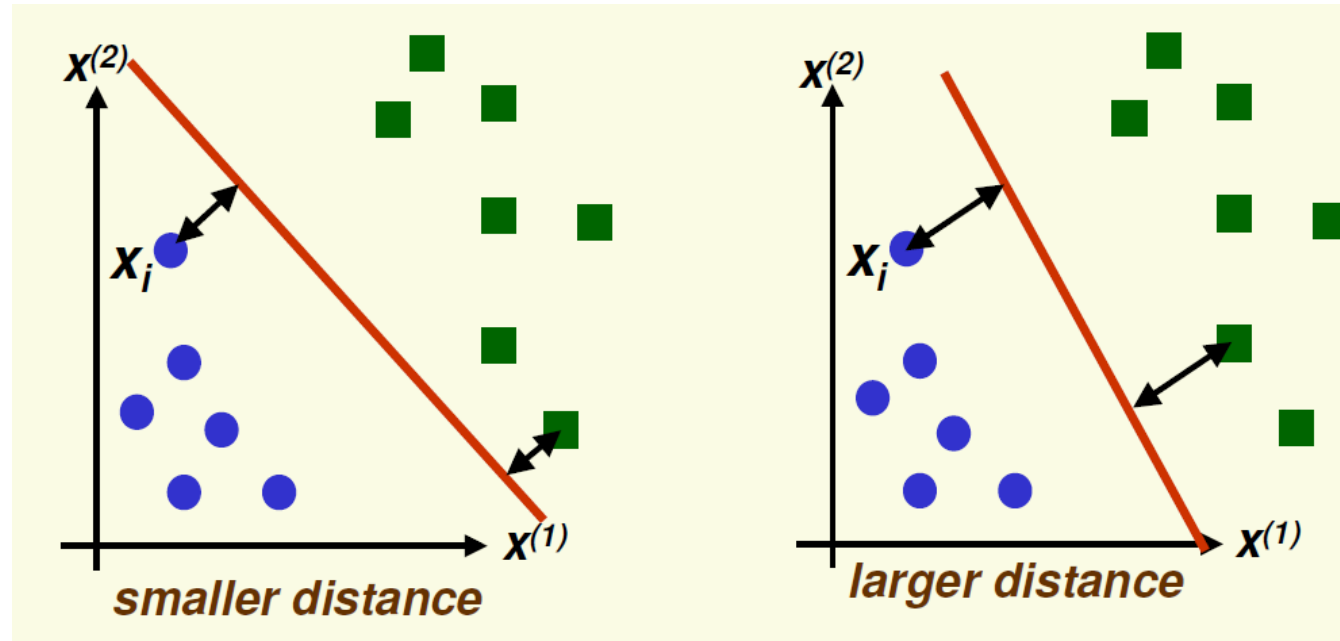- Poor generalization (performance on unseen data)

Linear Discriminant Functions

- Hyperplane as far as possible from **_any_** sample
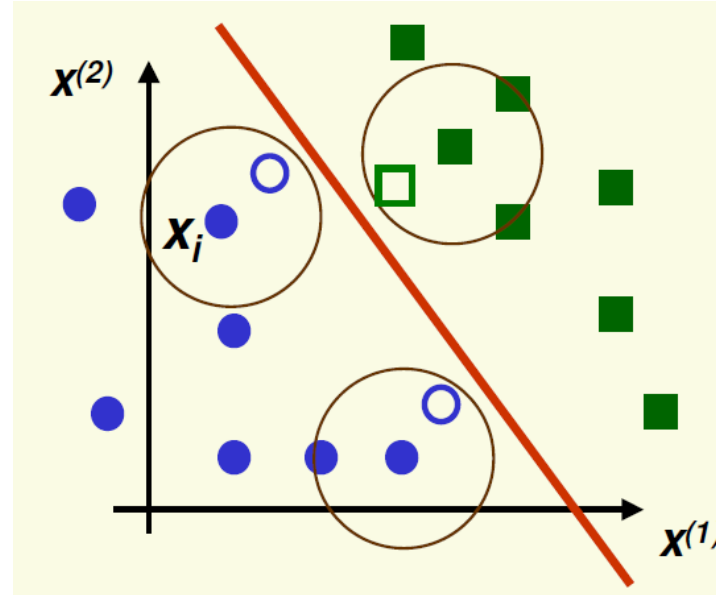
- New samples close to the old samples will be classified correctly
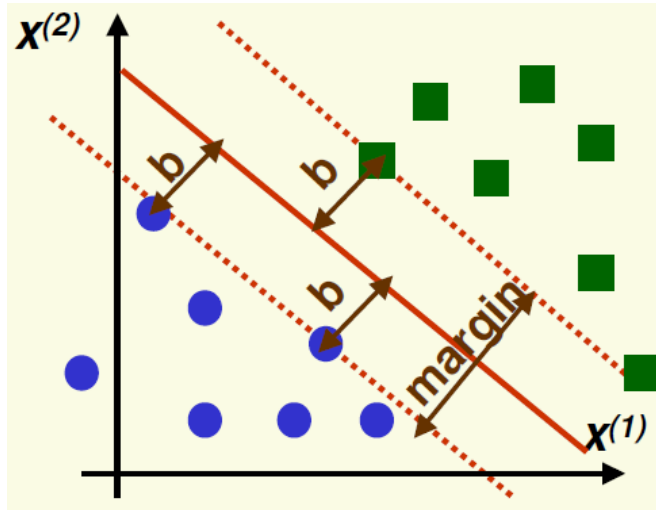
- Good generalization

SVM

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

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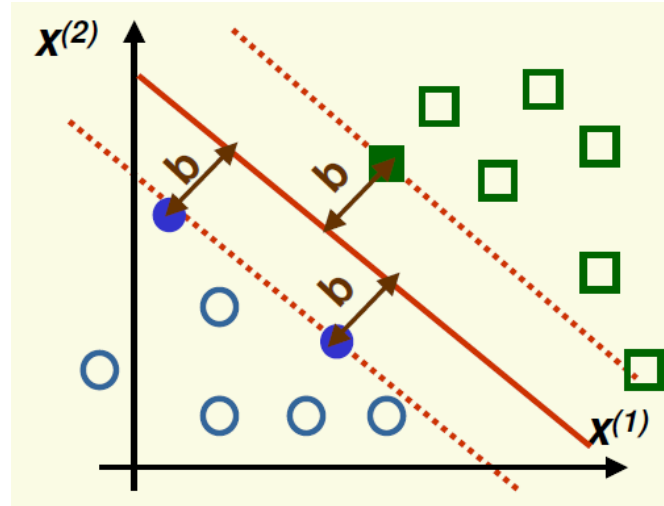 - distance to the closest negative example = distance to the closest positive example

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**

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

- Distance is unchanged for hyperplane

$$g_1(x) = \alpha g(x)$$

$$\frac{\left| \alpha w^t x + \alpha w_0 \right|}{\| \alpha w \|} = \frac{\left| w^t x + w_0 \right|}{\| w \|}$$

- Let **x$_i$** be an example closest to the boundary (on the positive side). Set:
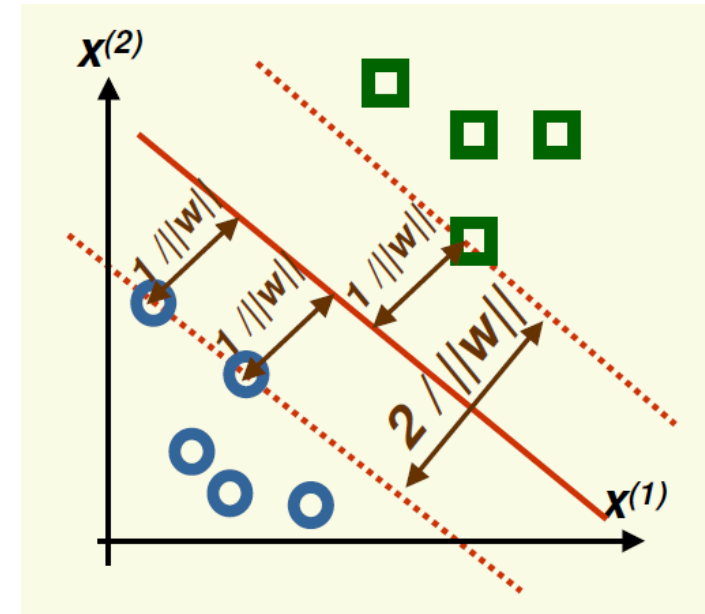
$$\left| w^t x_i + w_0 \right| = 1$$

- Now the largest margin hyperplane is unique
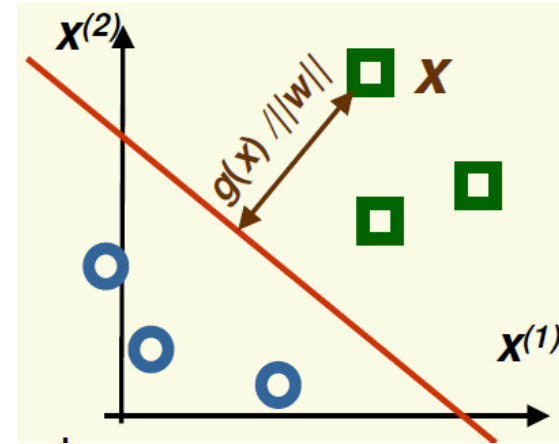
SVM: Formula for the Margin

- For uniqueness, set $|w^T x_i + w_0| = 1$ for any sample x_i closest to the boundary

- The distance from closest sample x_i to $g(x) = 0$ is

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

- Thus the margin is

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

SVM: Optimal Hyperplane

- Maximize margin $m = \dfrac{2}{\|w\|}$

- 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

$$\text{minimize } J(w) = \frac{1}{2}\|w\|^2$$
$$\text{constrained to } \quad z_i(w^t x_i + w_0) \geq 1 \quad \forall i$$

- **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

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

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

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

SVM: Optimal Hyperplane

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

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

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

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

SVM: Optimal Hyperplane

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

constrained to $\quad \alpha_i \geq 0 \quad \forall i \quad$ 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 $x_j^t x_i$
- This will become important when looking for a nonlinear discriminant function, as we will see soon
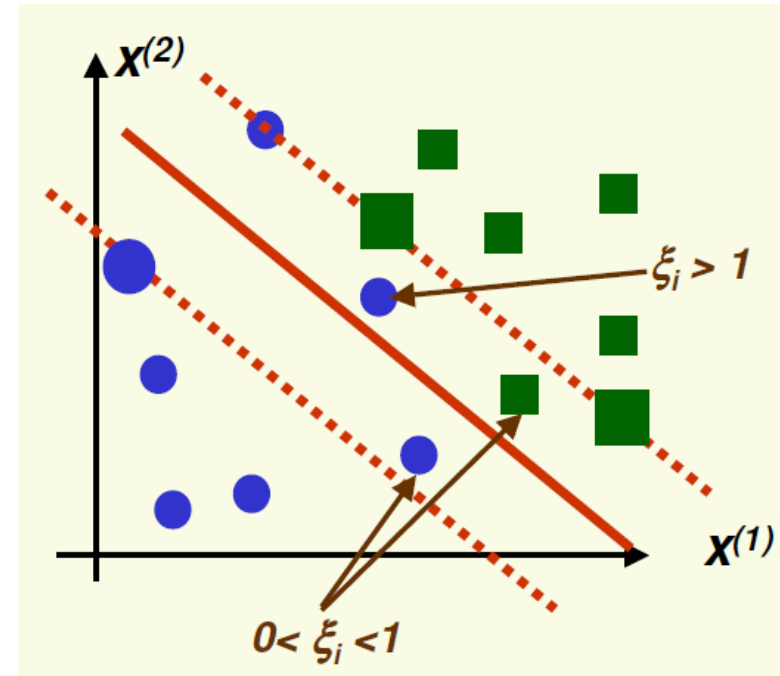
SVM: Non-Separable Case

- Data are most likely to be not linearly separable, but linear classifier may still be appropriate
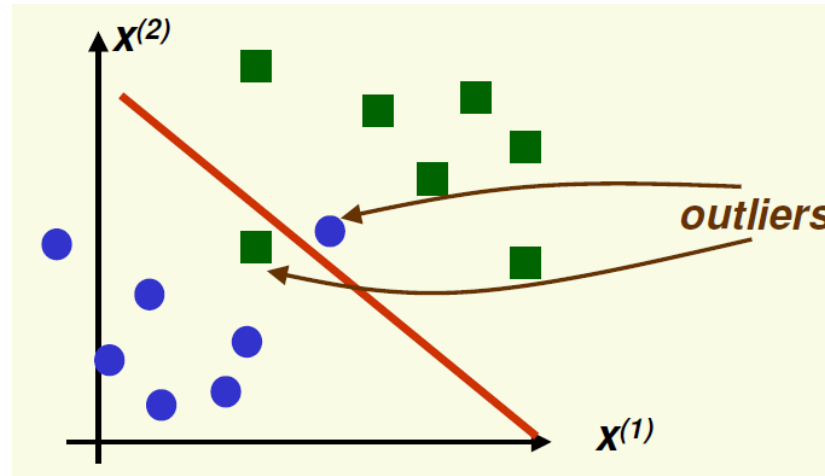
- Can apply SVM in non linearly separable case
- Data should be "almost" linearly separable for good performance

SVM: Non-Separable Case

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

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

- 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

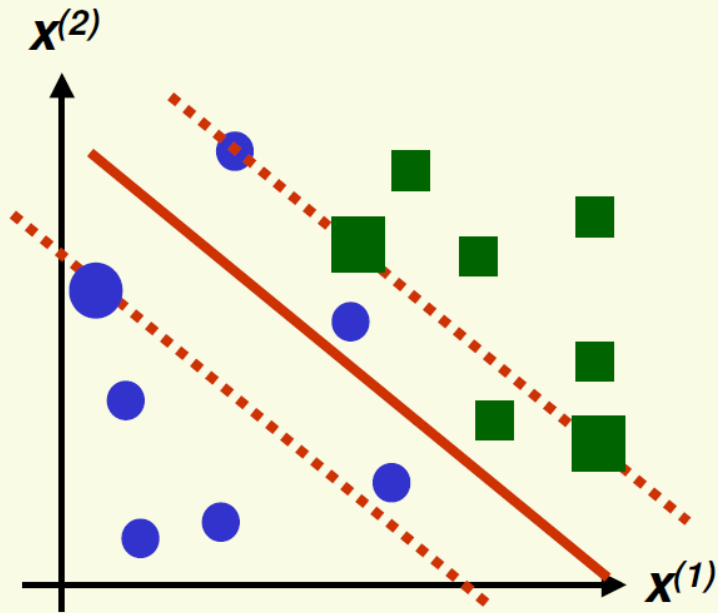$$J(w, \xi_1, \ldots, \xi_n) = \frac{1}{2}\|w\|^2 + \beta \sum_{i=1}^{n} I(\xi_i > 0)$$

# of samples not in ideal location

- where $I(\xi_i > 0) = \begin{cases} 1 & if\ \xi_i > 0 \\ 0 & if\ \xi_i \le 0 \end{cases}$

- 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
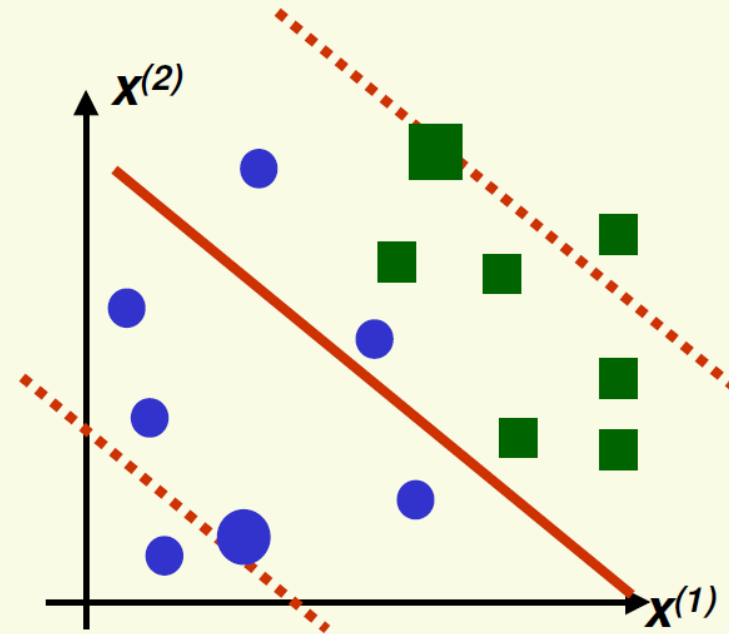 - If β is large, few samples can be in non-ideal positions

SVM: Non-Separable Case

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

# 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

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

a measure of # of misclassified examples

- Subject to

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

SVM: Non-Separable Case

- Use Kuhn-Tucker theorem to convert to:

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

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

- **w** is computed using:

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

- Remember that

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

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"*

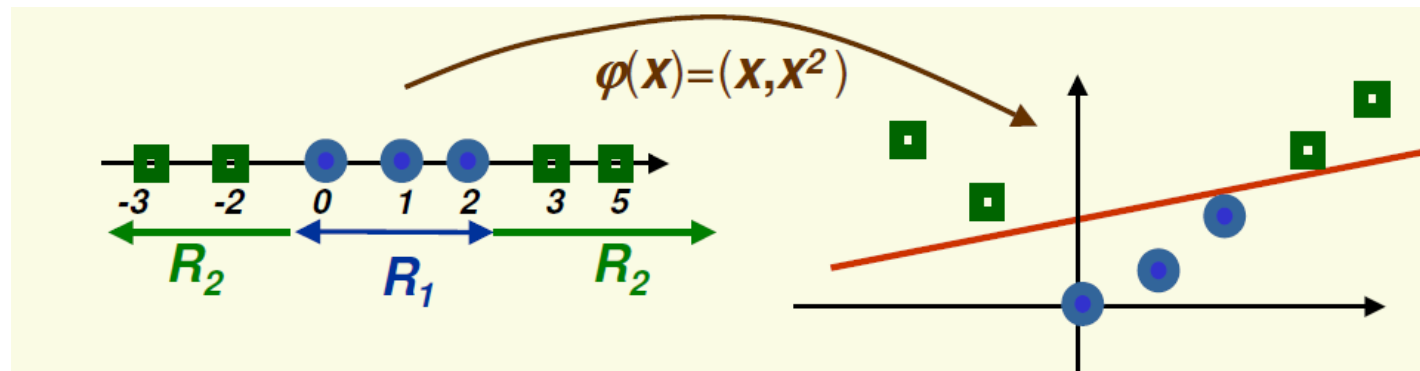- One dimensional space, not linearly separable
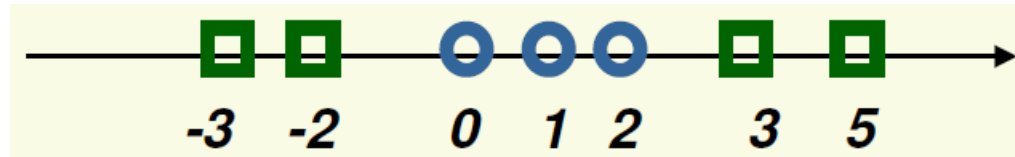
- Lift to two dimensional space with ɸ(**x**)=(**x**,**x**^{2})
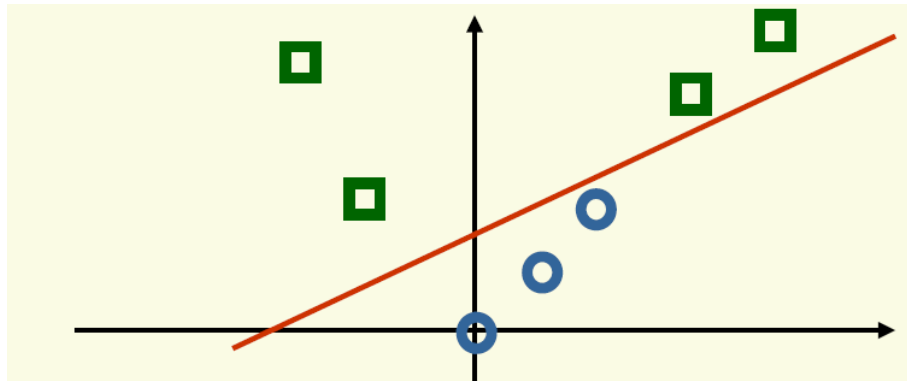
Nonlinear Mapping

- To solve a non linear classification problem with a linear classifier
1. Project data x to high dimension using function $\phi(x)$
2. Find a linear discriminant function for transformed data $\phi(x)$
3. Final nonlinear discriminant function is $g(x) = w^t\phi(x) + w_0$

$$\varphi(x) = (x, x^2)$$

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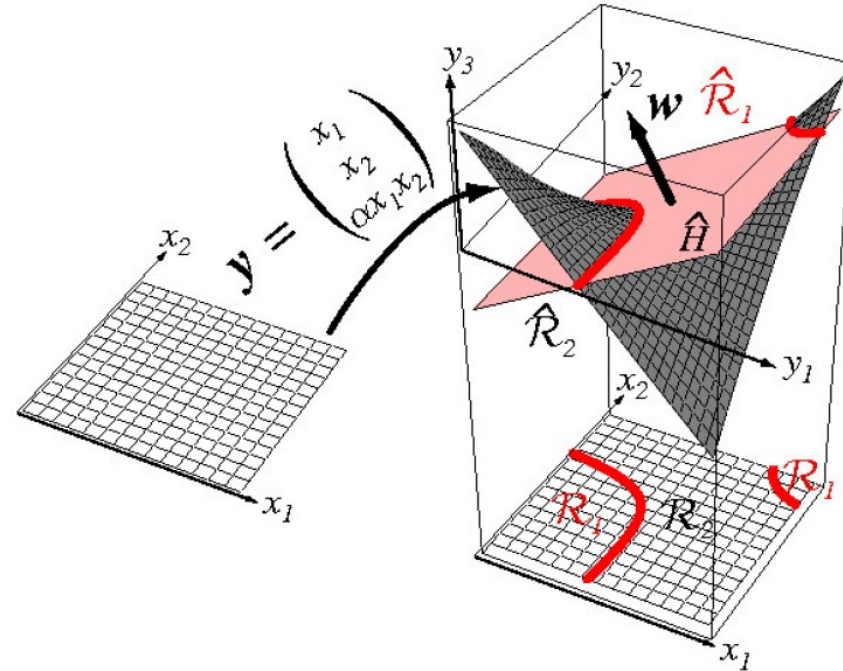
- However, there always exists a mapping of N samples to an N-dimensional space in which the samples are separable by hyperplanes

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 x_i only through the dot product $x_i^t x_j$

- If we lift x_i to high dimension using $\varphi(x)$, we need to compute high dimensional product $\varphi(x_i)^t \varphi(x_j)$

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 $K(x_i,x_j)$ 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 $\boldsymbol{\phi}(\mathbf{x_i})^t\ \boldsymbol{\phi}(\mathbf{x_j})$

- "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 $\phi(x)$ does this correspond to?

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

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

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

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

Choice of Kernel

- How to choose kernel function $K(x_i, x_j)$?
  - $K(x_i, x_j)$ should correspond to $\phi(x_i)^t \phi(x_j)$ 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 x_i and x_j
  - As inner product measures similarity of unit vectors
  - May be problem-specific

Choice of Kernel
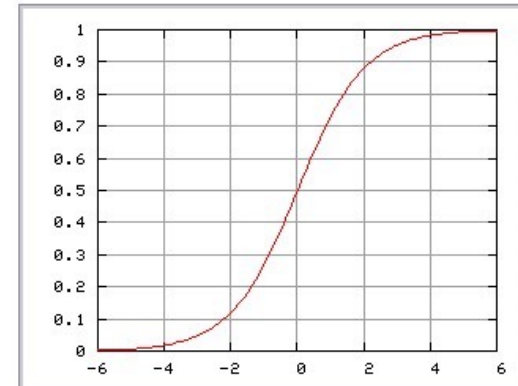
- Some common choices:
  - Polynomial kernel

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

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

- The mapping $\phi(x)$ 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 $x_1,...,x_n$ which live in feature space of dimension **d**

- Choose kernel $K(x_i,x_j)$ or function $\phi(x_i)$ which lifts sample 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:

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

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

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

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

- Non linear discriminant function in the original space:

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

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

Nonlinear SVM

- Nonlinear discriminant function

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

$$g(x) = \sum$$

most important
training samples,
i.e. support vectors

$\boxed{\text{weight of support vector } x_i}$

$\boxed{\mp 1}$

$\boxed{\text{"inverse distance" from } x \text{ to support vector } x_i}$

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

SVM Example: XOR Problem

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

$$K(x_i, x_j) = (x_i{}^t x_j + 1)^2$$
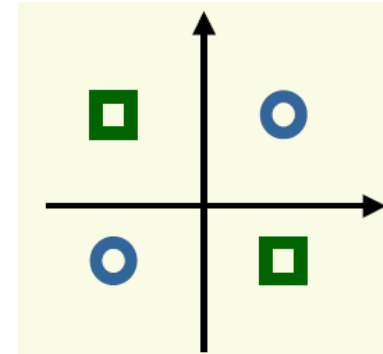
  - This kernel corresponds to the mapping

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

- 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

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

SVM Example: XOR Problem

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

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

- All samples are support vectors

SVM Example: XOR Problem

$$\varphi(x) = \begin{bmatrix} 1 & \sqrt{2}x^{(1)} & \sqrt{2}x^{(2)} & \sqrt{2}x^{(1)}x^{(2)} & (x^{(1)})^2 & (x^{(2)})^2 \end{bmatrix}^t$$
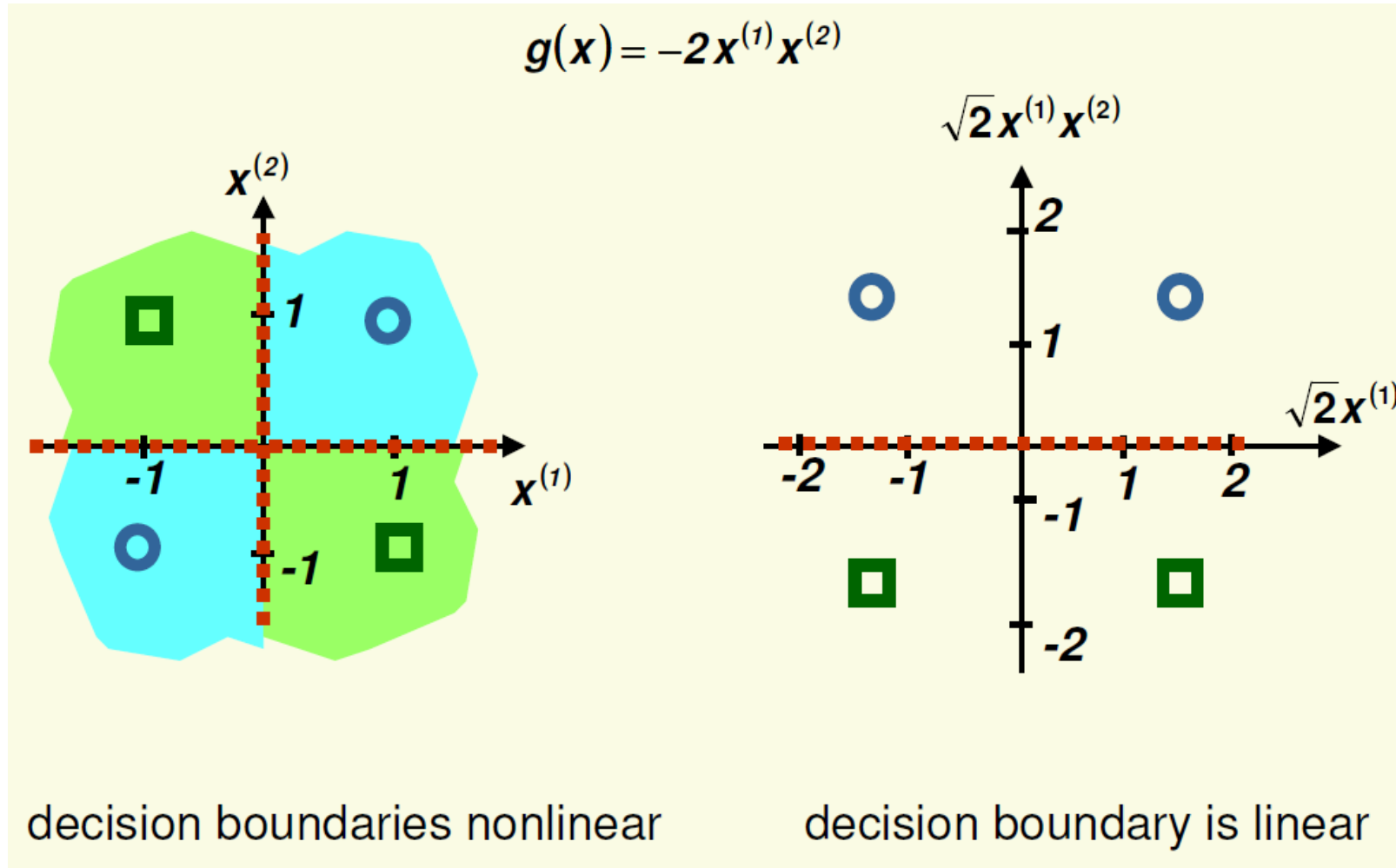
- The weight vector **w** is:

$$w = \sum_{i=1}^{4} \alpha_i z_i \varphi(x_i) = 0.25(\varphi(x_1) + \varphi(x_2) - \varphi(x_3) - \varphi(x_4))$$

$$= \begin{bmatrix} 0 & 0 & 0 & -\sqrt{2} & 0 & 0 \end{bmatrix}$$

- 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

$$g(x) = -2x^{(1)}x^{(2)}$$

decision boundaries nonlinear   decision boundary is linear

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"
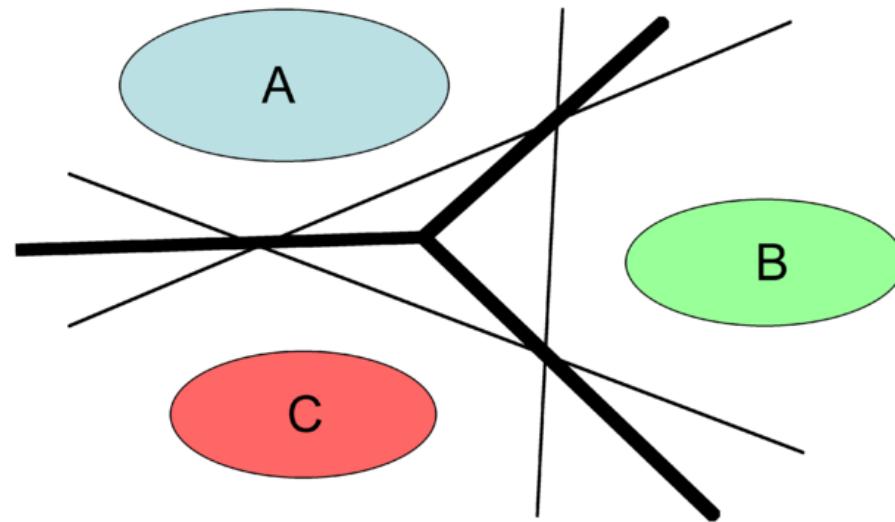
One-Against-All

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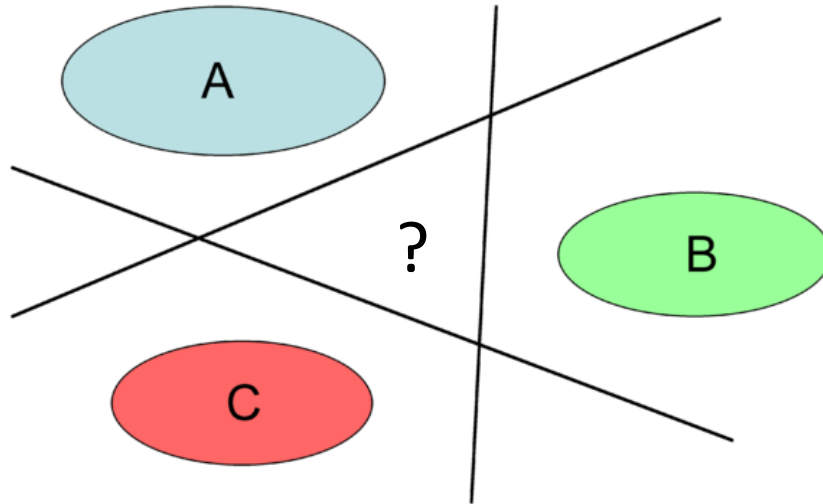
One-Against-All

- 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