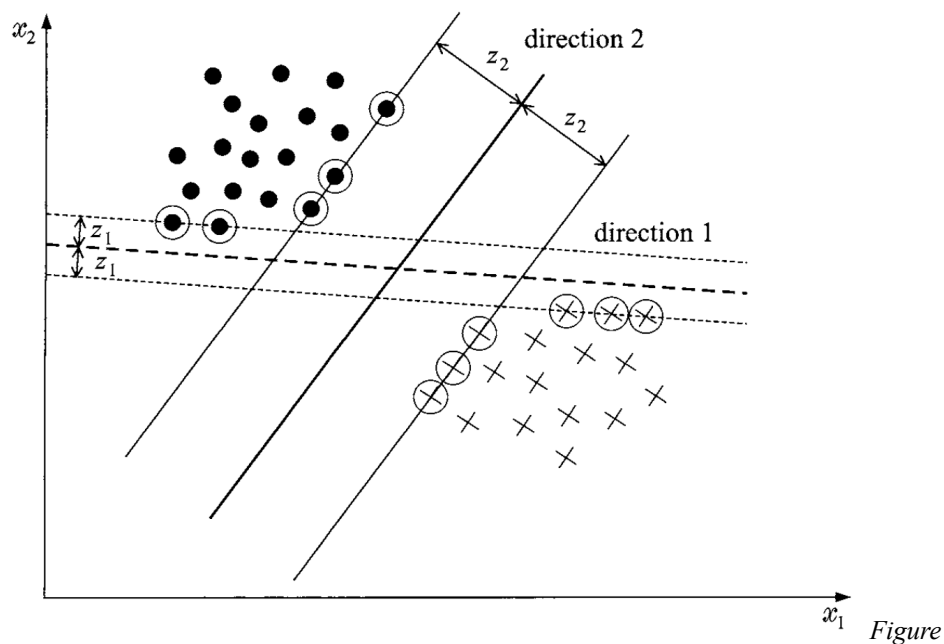


ML4 – Support Vector Machines

Introduction

The Perceptron algorithm calculates the decision line between two linearly separable classes clustered in feature space. We note that the solution found by the Perceptron is not unique. Running the algorithm with a different set of initial weights will give you a different but equally valid solution.

What if we want to find the best decision line between the two classes? By “best” we mean that the decision line should be the one that offers the best generalization when tested with novel data. Visually, the best decision line should have the largest margin between the two classes as shown in Figure 1. For linearly separable classes the solution to the best decision line is unique.



1. Decision line along direction 2 has a larger margin (expressed as perpendicular distance z) than that of direction 1. The feature vectors that are closest to the decision lines are known as the support vectors. Figure from Chapter 3, Theodoridis 'Pattern Recognition'.

If the best decision line is give by

$$g(\mathbf{x}) = w_o + \mathbf{x}^T \mathbf{w} = 0 \quad (1)$$

the perpendicular distance z is given by

$$z = \frac{|g(\mathbf{x})|}{\|\mathbf{w}\|} \quad (2)$$

We can scale \mathbf{w} and w_o such that the value of $g(\mathbf{x})$ is 1 for ω_1 and -1 for ω_2 . This is equivalent to imposing that

1. The margin given by $\frac{1}{\|\mathbf{w}\|} + \frac{1}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$ is maximized;
2. And that
$$\begin{aligned} w_o + \mathbf{x}^T \mathbf{w} &\geq 1, \forall \mathbf{x} \in \omega_1 \\ w_o + \mathbf{x}^T \mathbf{w} &\leq -1, \forall \mathbf{x} \in \omega_2 \end{aligned}$$

Recall that for each data point \mathbf{x}_i we indicate the class by y_i which is +1 for ω_1 and -1 for ω_2 . Our two conditions above can therefore be efficiently rewritten as follows:

minimize :

$$J(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2 \quad (3)$$

subject to:

$$y_i (w_o + \mathbf{x}_i^T \mathbf{w}) \geq 1, i = 1, 2, \dots, N \quad (4)$$

Using Lagrange multipliers λ_i we can convert the margin maximization problem given by:

$$\begin{aligned} \min L_p &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \lambda_i y_i (w_o + \mathbf{x}_i^T \mathbf{w}) + \sum_{i=1}^N \lambda_i \\ \text{subject to} \quad & \mathbf{w} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i \\ & \sum_{i=1}^N \lambda_i y_i = 0 \\ & \lambda_i \geq 0 \end{aligned} \quad (5)$$

where N is the number of training points, to a problem involving only dot products given by

$$\begin{aligned}
 \max L_D(\lambda_i) &= \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\
 \text{subject to} \quad & \sum_{i=1}^l \lambda_i y_i = 0 \\
 & \lambda_i \geq 0.
 \end{aligned} \tag{6}$$

Finding the decision line with the widest margin from the nearest feature vectors using Equation 6 is an optimization problem. Using quadratic problem solver packages we can solve for λ_i and use the expression for \mathbf{w} in Equation 5 to solve for the weights \mathbf{w} and bias w_0 .

Procedure

1. Similar to the Perceptron, SVM is applied to pairs of classes. Read through Reference 2 slides and for each pair of fruits set up the matrices \mathbf{H} , \mathbf{B} and \mathbf{A} , and vectors \mathbf{f} , \mathbf{a} and \mathbf{b} found in slides 19 to 22.
2. Use quadratic programming packages available from Python (numpy and cvxopt) or Matlab (quadprog in Optimization toolbox) using the vectors and matrices setup in Step 1 to solve for the Lagrange multipliers λ_i .
3. Solve for the weights \mathbf{w} and bias w_0 and plot the decision lines in feature space with your fruit data.

Reference

1. Theodoridis, Chapter 3, Pattern Recognition
2. Veksler, O., CS 434a/541a: Pattern Recognition Lecture 11 slides.

CS434a/541a: Pattern Recognition
Prof. Olga Veksler

Lecture 11

Today

- Support Vector Machines (SVM)
 - Introduction
 - Linear Discriminant
 - Linearly Separable Case
 - Linearly Non Separable Case
 - Kernel Trick
 - Non Linear Discriminant

SVM

- Said to start in 1979 with Vladimir Vapnik's paper
- Major developments throughout 1990's
- Elegant theory
 - Has good generalization properties
- Have been applied to diverse problems very successfully in the last 10-15 years
- One of the most important developments in pattern recognition in the last 10 years

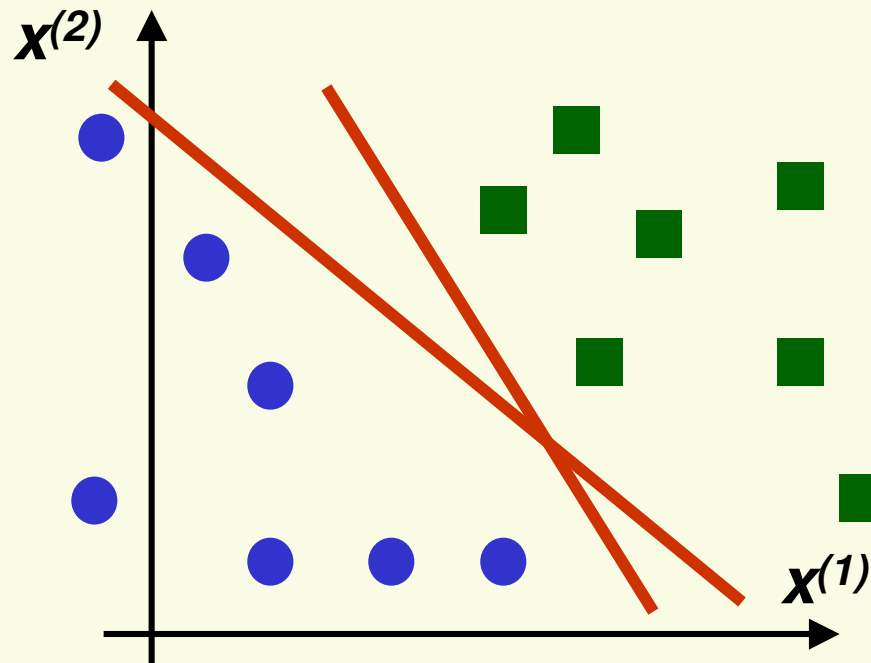


Linear Discriminant Functions

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

$$g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$$

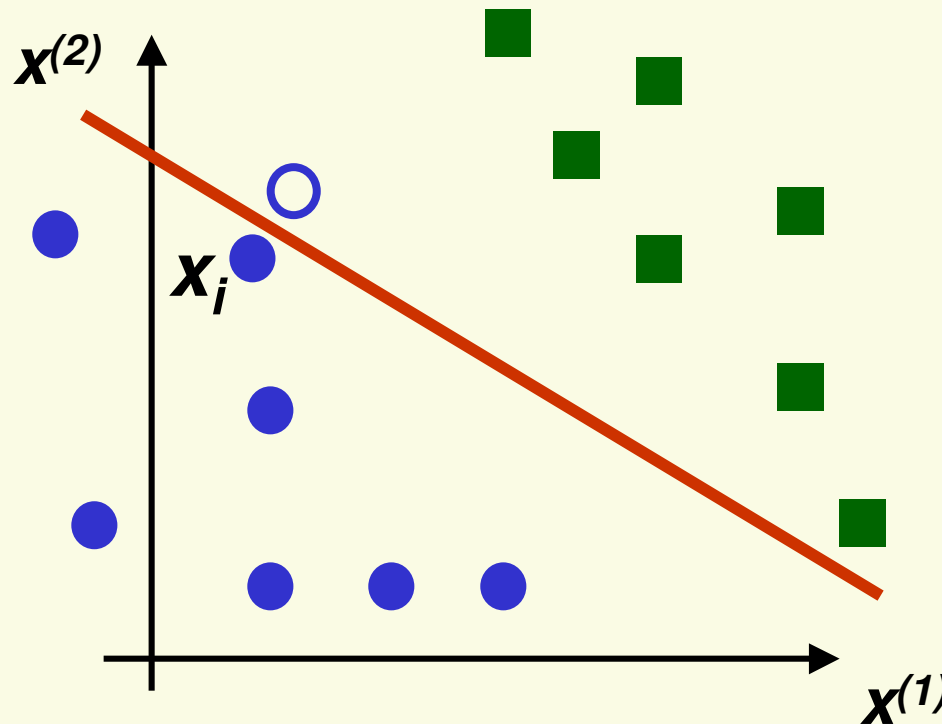
$$\begin{array}{l} g(\mathbf{x}) > 0 \Rightarrow \mathbf{x} \in \text{class 1} \\ g(\mathbf{x}) < 0 \Rightarrow \mathbf{x} \in \text{class 2} \end{array}$$



- which separating hyperplane should we choose?

Linear Discriminant Functions

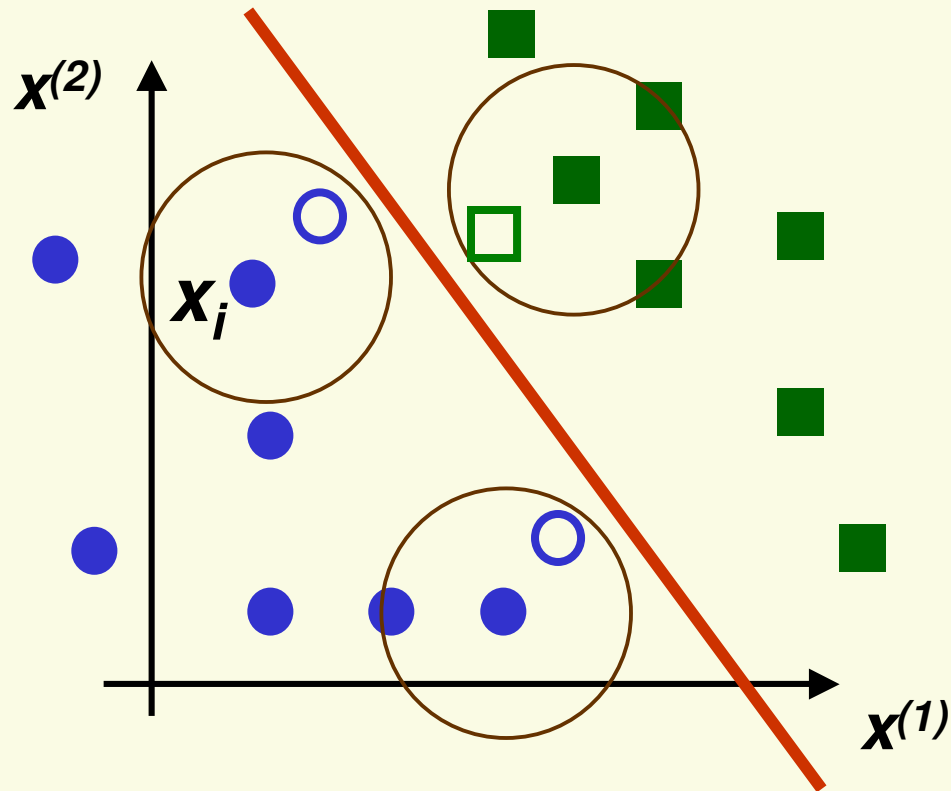
- Training data is just a subset of all possible data
- Suppose hyperplane is close to sample \mathbf{x}_i
- If we see new sample close to sample i , it is likely to be on the wrong side of the hyperplane



- Poor generalization (performance on unseen data)

Linear Discriminant Functions

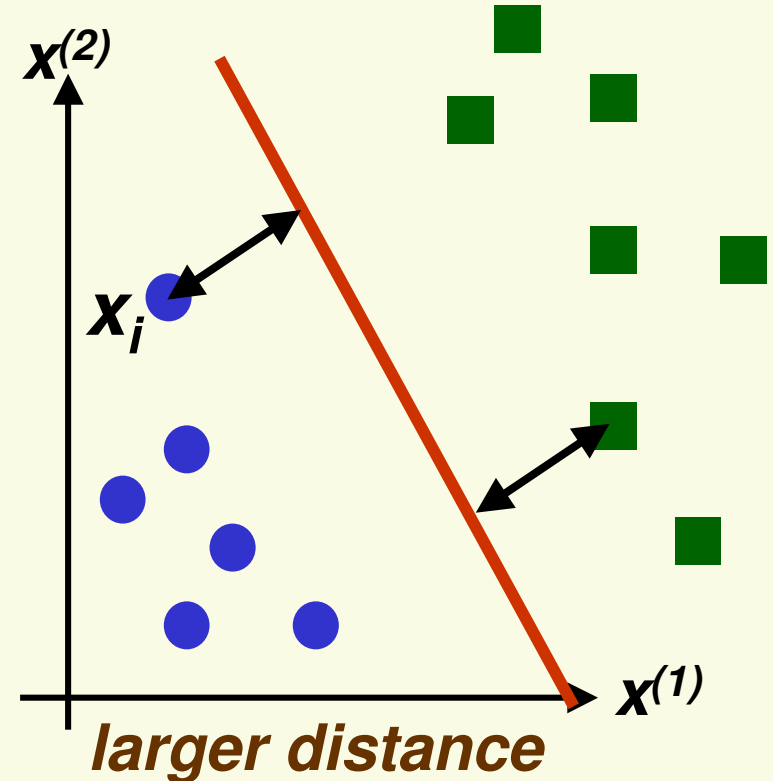
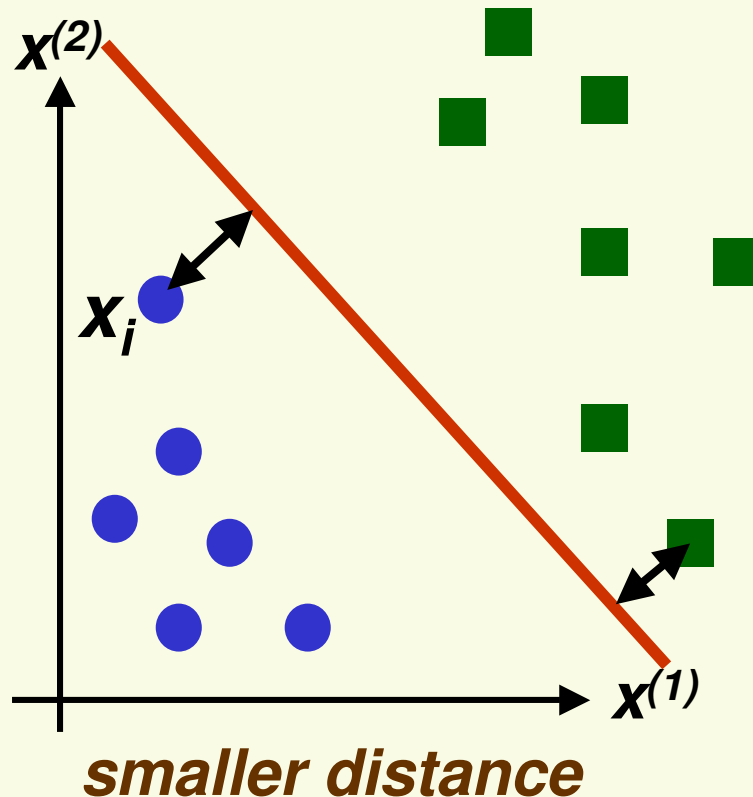
- Hyperplane as far as possible from any sample



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

SVM

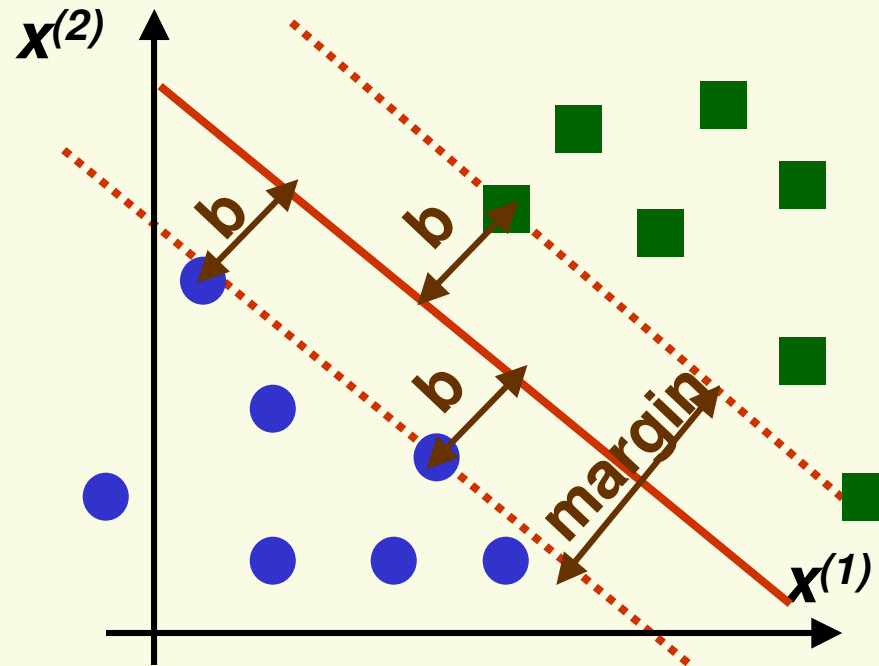
- Idea: maximize distance to the closest example



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

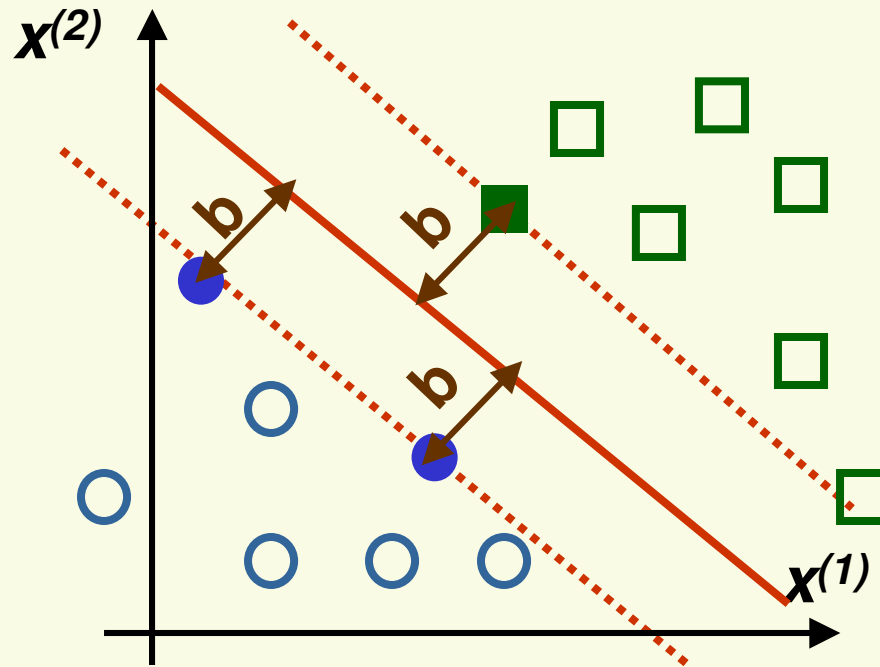
SVM: Linearly Separable Case

- SVM: maximize the *margin*



- *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

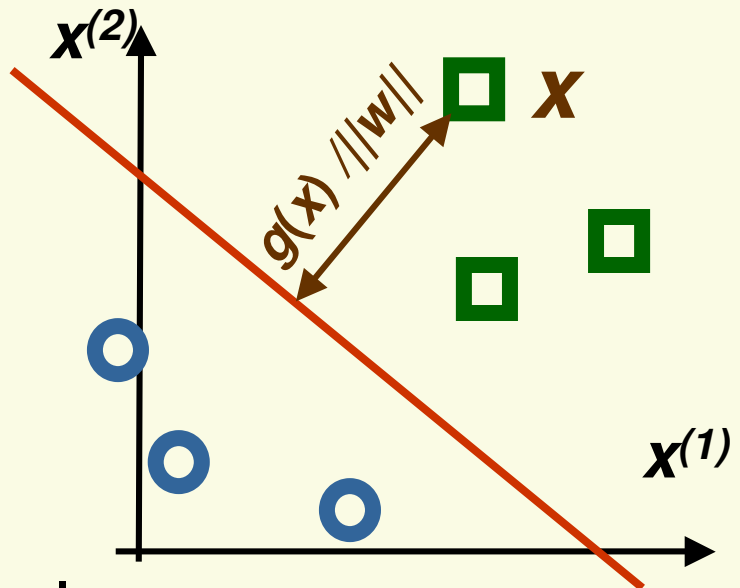


- **Support vectors** are the samples closest to the separating hyperplane
 - they are the most difficult patterns to classify
 - Optimal hyperplane is completely defined by support vectors
 - of course, we do not know which samples are support vectors without finding the optimal hyperplane

SVM: Formula for the Margin

- $g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$
- absolute distance between \mathbf{x} and the boundary $g(\mathbf{x}) = 0$

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



- distance is unchanged for hyperplane
 $g_1(\mathbf{x}) = \alpha g(\mathbf{x})$

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

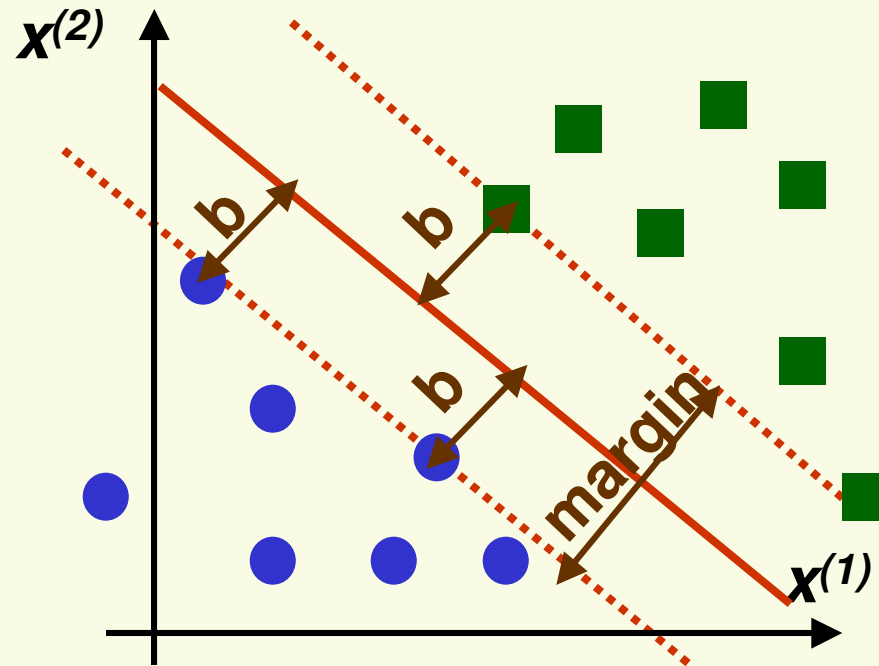
- Let \mathbf{x}_i be an example closest to the boundary. Set
 $|\mathbf{w}^t \mathbf{x}_i + w_0| = 1$
- Now the largest margin hyperplane is unique

Today

- Continue Support Vector Machines (SVM)
 - Linear Discriminant
 - Linearly Separable Case
 - Linearly Non Separable Case
 - Kernel Trick
 - Non Linear Discriminant

SVM: Linearly Separable Case

- SVM: maximize the *margin*



- *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

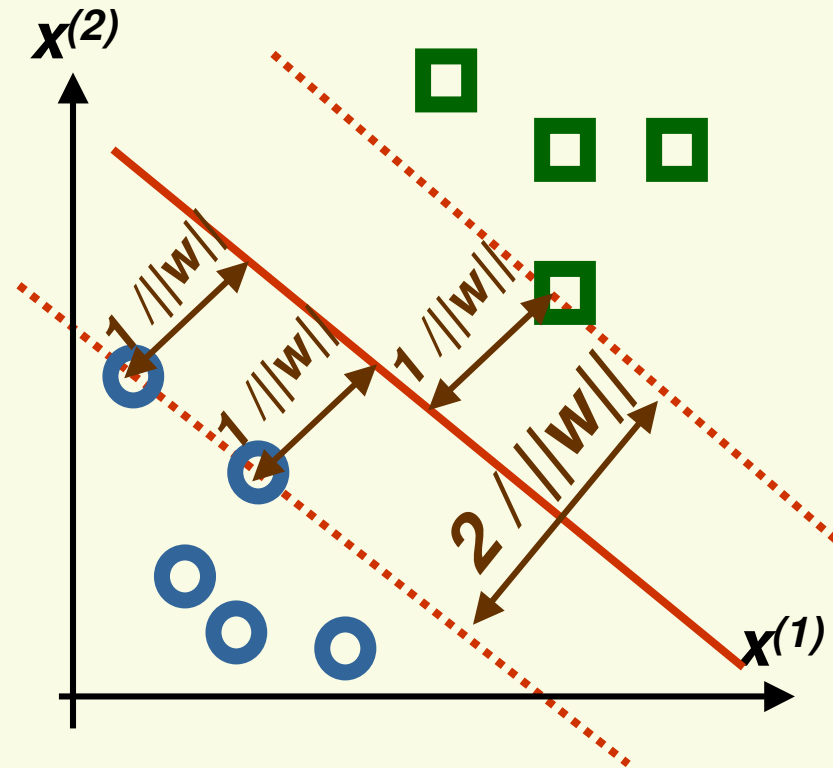
SVM: Formula for the Margin

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

$$\frac{|\mathbf{w}^t \mathbf{x}_i + \mathbf{w}_0|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

- Thus the margin is

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



SVM: Optimal Hyperplane

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

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

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

SVM: Optimal Hyperplane

- Use Kuhn-Tucker theorem to convert our problem to:

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

- $\alpha = \{\alpha_1, \dots, \alpha_n\}$ are new variables, one for each sample
- Can rewrite $L_D(\alpha)$ using n by n matrix \mathbf{H} :

$$L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}^t \mathbf{H} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}$$

- where the value in the i th row and j th column of \mathbf{H} is
$$\mathbf{H}_{ij} = \mathbf{z}_i \mathbf{z}_j \mathbf{x}_i^t \mathbf{x}_j$$

SVM: Optimal Hyperplane

- Use Kuhn-Tucker theorem to convert our problem to:

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

- $\alpha = \{\alpha_1, \dots, \alpha_n\}$ are new variables, one for each sample
- $L_D(\alpha)$ can be optimized by quadratic programming
- $L_D(\alpha)$ formulated in terms of α
 - it depends on \mathbf{w} and \mathbf{w}_0 indirectly

SVM: Optimal Hyperplane

- After finding the optimal $\alpha = \{\alpha_1, \dots, \alpha_n\}$
 - For every sample i , one of the following must hold
 - $\alpha_i = 0$ (sample i is not a support vector)
 - $\alpha_i \neq 0$ and $\mathbf{z}_i(\mathbf{w}^t \mathbf{x}_i + \mathbf{w}_0 - 1) = 0$ (sample i is support vector)
 - can find \mathbf{w} using $\mathbf{w} = \sum_{i=1}^n \alpha_i \mathbf{z}_i \mathbf{x}_i$
 - can solve for \mathbf{w}_0 using any $\alpha_i > 0$ and $\alpha_i [\mathbf{z}_i(\mathbf{w}^t \mathbf{x}_i + \mathbf{w}_0) - 1] = 0$
$$\mathbf{w}_0 = \frac{1}{\mathbf{z}_i} - \mathbf{w}^t \mathbf{x}_i$$
- Final discriminant function:
$$g(\mathbf{x}) = \left(\sum_{\mathbf{x}_i \in \mathbf{S}} \alpha_i \mathbf{z}_i \mathbf{x}_i \right)^t \mathbf{x} + \mathbf{w}_0$$
 - where \mathbf{S} is the set of support vectors
$$\mathbf{S} = \{\mathbf{x}_i \mid \alpha_i \neq 0\}$$

SVM: Optimal Hyperplane

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

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

SVM: Example using Matlab

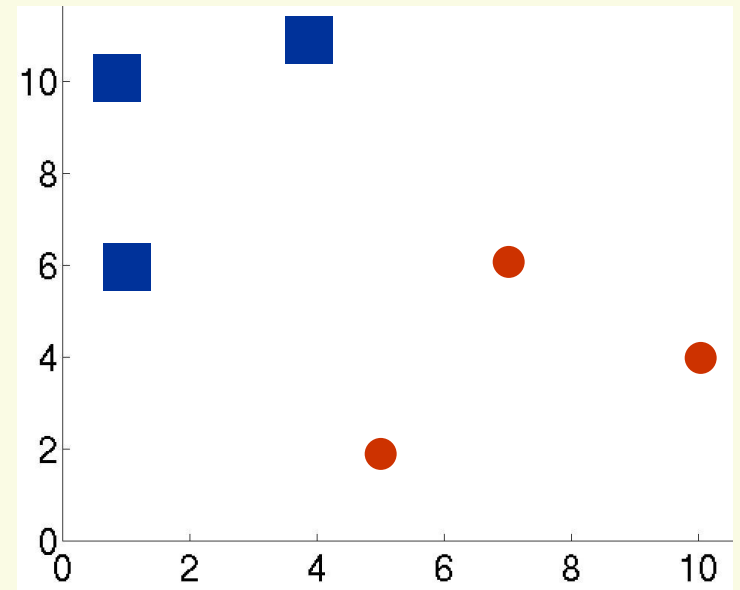
- Class 1: [1,6], [1,10], [4,11]
- Class 2: [5,2], [7,6], [10,4]
- Let's pile all data into array X

$$X = \begin{bmatrix} 1 & 6 \\ 1 & 10 \\ 4 & 11 \\ 5 & 2 \\ 7 & 6 \\ 10 & 4 \end{bmatrix}$$

- Pile z_i 's into vector $z = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$

- Matrix H with $H_{ij} = z_i z_j x_i^t x_j$, in matlab use $H = (x * x'). * (z * z')$

$$H = \begin{bmatrix} 37 & 61 & 70 & -17 & -43 & -34 \\ 61 & 101 & 114 & -25 & -67 & -50 \\ 70 & 114 & 137 & -42 & -94 & -84 \\ -17 & -25 & -42 & 29 & 47 & 58 \\ -43 & -67 & -94 & 47 & 85 & 94 \\ -34 & -50 & -84 & 58 & 94 & 116 \end{bmatrix}$$



SVM: Example using Matlab

- Matlab expects quadratic programming to be stated in the *canonical* (standard) form which is

$$\begin{array}{ll} \text{minimize} & L_D(\alpha) = \mathbf{0.5}\alpha^t \mathbf{H}\alpha + \mathbf{f}^t \alpha \\ \text{constrained to} & \mathbf{A}\alpha \leq \mathbf{a} \text{ and } \mathbf{B}\alpha = \mathbf{b} \end{array}$$

- where $\mathbf{A}, \mathbf{B}, \mathbf{H}$ are n by n matrices and $\mathbf{f}, \mathbf{a}, \mathbf{b}$ are vectors
- Need to convert our optimization problem to canonical form

$$\begin{array}{ll} \text{maximize} & L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}^t \mathbf{H} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \\ \text{constrained to} & \alpha_i \geq \mathbf{0} \quad \forall i \quad \text{and} \quad \sum_{i=1}^n \alpha_i \mathbf{z}_i = \mathbf{0} \end{array}$$

SVM: Example using Matlab

- Multiply by -1 to convert to minimization:

$$\text{minimize } L_D(\alpha) = -\sum_{i=1}^n \alpha_i + \frac{1}{2} \alpha^t H \alpha$$

- Let $f = \begin{bmatrix} -1 \\ \vdots \\ -1 \end{bmatrix} = -\mathbf{ones}(6,1)$, then can write

$$\text{minimize } L_D(\alpha) = f^t \alpha + \frac{1}{2} \alpha^t H \alpha$$

- First constraint is $\alpha_i \geq 0 \quad \forall i$

- Let $A = \begin{bmatrix} -1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & -1 \end{bmatrix} = -\mathbf{eye}(6)$, $a = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} = \mathbf{zeros}(6,1)$

- Rewrite the first constraint in canonical form:

$$\mathbf{A} \alpha \leq \mathbf{a}$$

SVM: Example using Matlab

- Our second constraint is $\sum_{i=1}^n \alpha_i \mathbf{z}_i = \mathbf{0}$

- Let $\mathbf{B} = \begin{bmatrix} \mathbf{z}_1 & \cdots & \cdots & \mathbf{z}_6 \\ \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \vdots & \cdots & \cdots & \vdots \\ \mathbf{0} & \cdots & \cdots & \mathbf{0} \end{bmatrix} = [[\mathbf{z}]; [\mathbf{zeros}(5,6)]]$

and $\mathbf{b} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} = \mathbf{zeros}(6,1)$

- Second constraint in canonical form is:

$$\mathbf{B}\alpha = \mathbf{b}$$

- Thus our problem is in canonical form and can be solved by matlab:

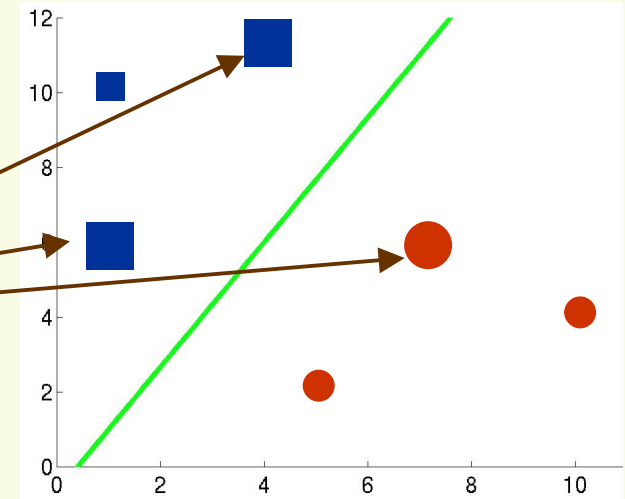
$$\begin{array}{ll} \text{minimize} & L_D(\alpha) = 0.5\alpha^t H \alpha + \mathbf{f}^t \alpha \\ \text{constrained to} & \mathbf{A}\alpha \leq \mathbf{a} \text{ and } \mathbf{B}\alpha = \mathbf{b} \end{array}$$

SVM: Example using Matlab

- $\alpha = \text{quadprog}(\mathbf{H} + \text{eye}(6) * 0.001, \mathbf{f}, \mathbf{A}, \mathbf{a}, \mathbf{B}, \mathbf{b})$
for stability

- Solution $\alpha = \begin{bmatrix} 0.036 \\ 0 \\ 0.039 \\ 0 \\ 0.076 \\ 0 \end{bmatrix}$

support vectors

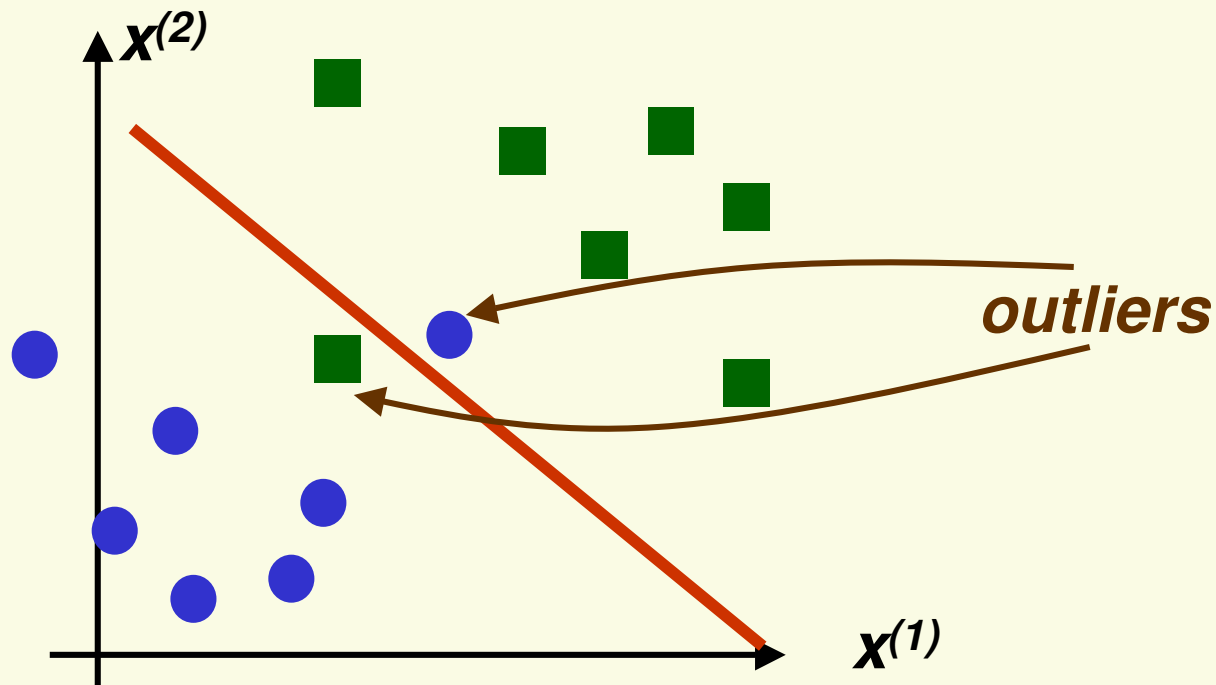


- find \mathbf{w} using $\mathbf{w} = \sum_{i=1}^n \alpha_i \mathbf{z}_i \mathbf{x}_i = (\alpha .* \mathbf{z})^t \mathbf{x} = \begin{bmatrix} -0.33 \\ 0.20 \end{bmatrix}$
- since $\alpha_1 > 0$, can find \mathbf{w}_0 using

$$\mathbf{w}_0 = \frac{1}{\mathbf{z}_1} - \mathbf{w}^t \mathbf{x}_1 = 0.13$$

SVM: Non Separable Case

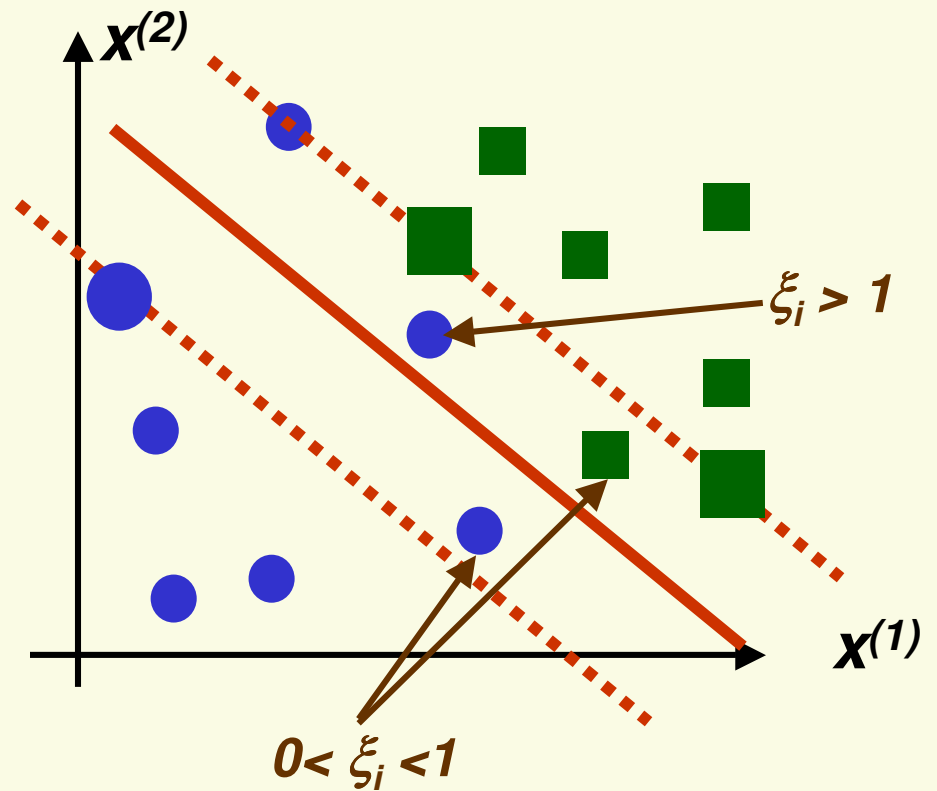
- Data is most likely to be not linearly separable, but linear classifier may still be appropriate



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

SVM: Non Separable Case

- Use slack variables ξ_1, \dots, ξ_n (one for each sample)
- Change constraints from $\mathbf{z}_i(\mathbf{w}^t \mathbf{x}_i + \mathbf{w}_o) \geq 1 \quad \forall i$ to
$$\mathbf{z}_i(\mathbf{w}^t \mathbf{x}_i + \mathbf{w}_o) \geq 1 - \xi_i \quad \forall i$$
- ξ_i is a measure of deviation from the ideal for sample i
 - $\xi_i > 1$ sample i is on the wrong side of the separating hyperplane
 - $0 < \xi_i < 1$ sample i is on the right side of separating hyperplane but within the region of maximum margin
 - $\xi_i < 0$ is the ideal case for sample i



SVM: Non Separable Case

- Would like to minimize

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

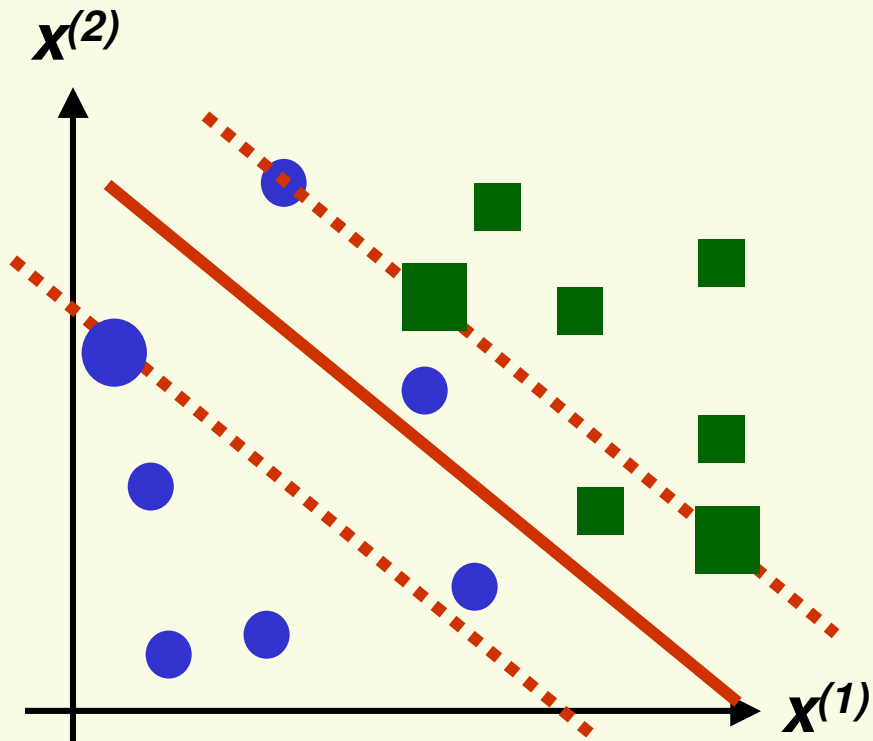
*# of samples
not in ideal location*

- where $I(\xi_i > 0) = \begin{cases} 1 & \text{if } \xi_i > 0 \\ 0 & \text{if } \xi_i \leq 0 \end{cases}$
- constrained to $\mathbf{z}_i(\mathbf{w}^t \mathbf{x}_i + \mathbf{w}_0) \geq 1 - \xi_i$ and $\xi_i \geq 0 \quad \forall i$
- β is a constant which measures relative weight of the first and second terms
 - if β is small, we allow a lot of samples not in ideal position
 - if β is large, we want to have very few samples not in ideal position

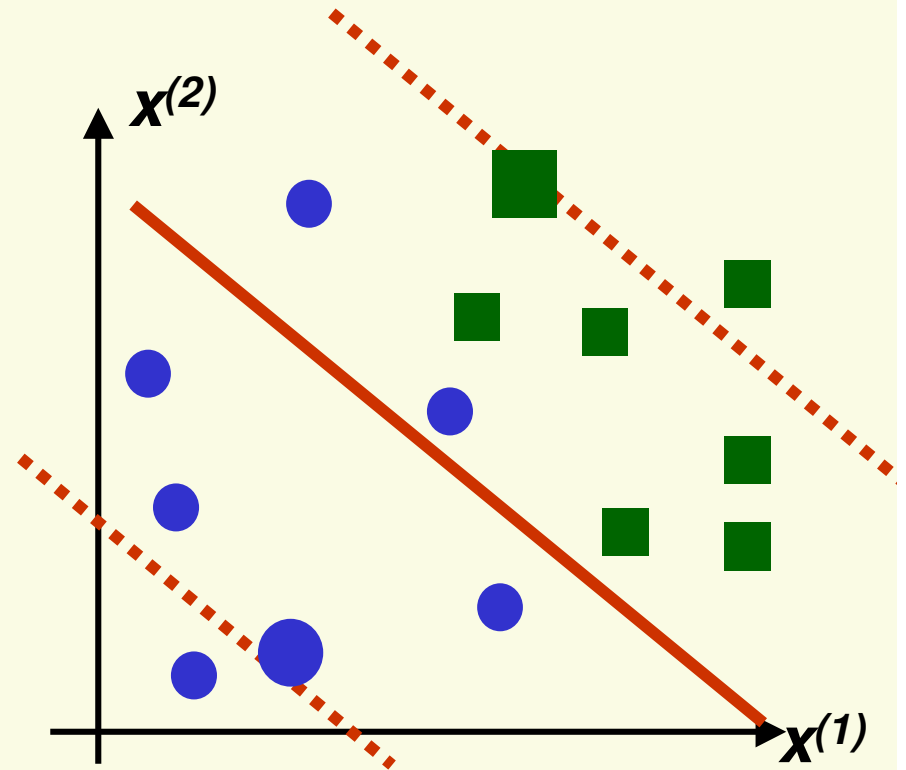
SVM: Non Separable Case

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

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 discontinuity of functions $I(\xi_i)$

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

*# of examples
not in ideal location*

- where $I(\xi_i > 0) = \begin{cases} 1 & \text{if } \xi_i > 0 \\ 0 & \text{if } \xi_i \leq 0 \end{cases}$
- constrained to $\mathbf{z}_i(\mathbf{w}^t \mathbf{x}_i + \mathbf{w}_0) \geq 1 - \xi_i$ and $\xi_i \geq 0 \quad \forall i$

SVM: Non Separable Case

- Instead we minimize

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

a measure of # of misclassified examples

- constrained to
$$\begin{cases} \mathbf{z}_i (\mathbf{w}^t \mathbf{x}_i + w_0) \geq 1 - \xi_i & \forall i \\ \xi_i \geq 0 & \forall i \end{cases}$$

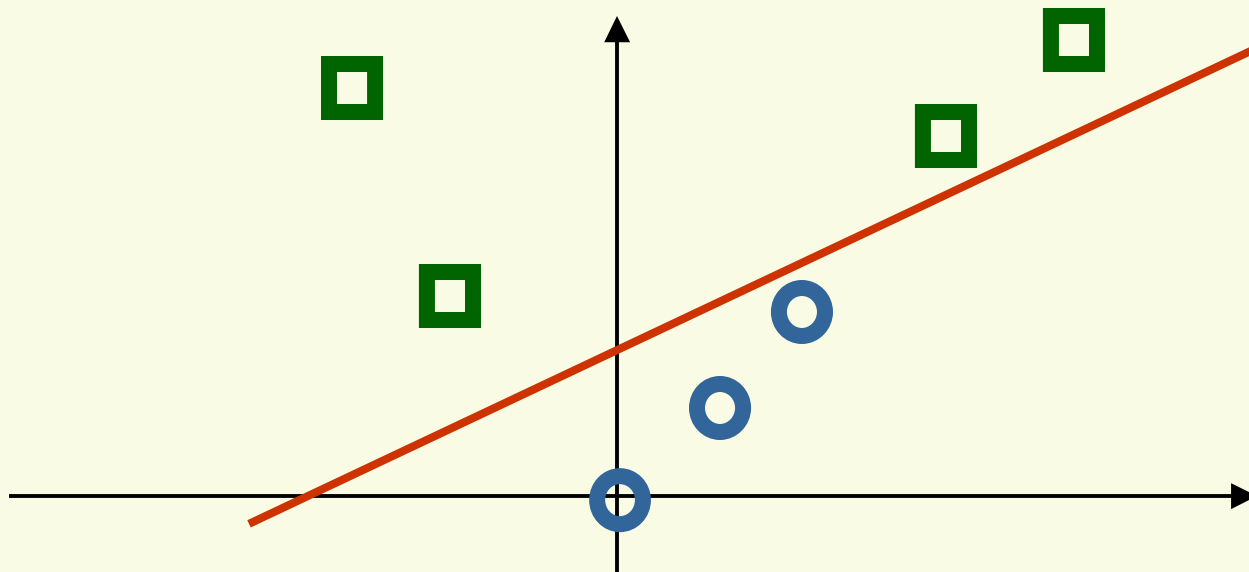
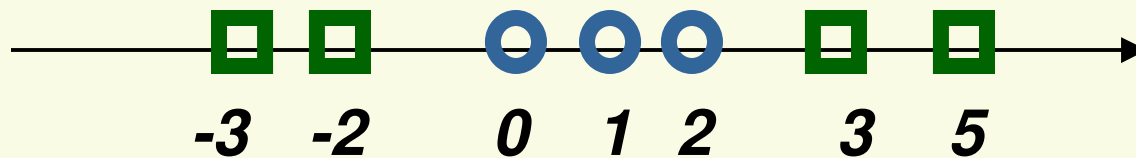
- Can use Kuhn-Tucker theorem to converted to

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

- find \mathbf{w} using
$$\mathbf{w} = \sum_{i=1}^n \alpha_i \mathbf{z}_i \mathbf{x}_i$$
- solve for w_0 using any $0 < \alpha_i < \beta$ and
$$\alpha_i [\mathbf{z}_i (\mathbf{w}^t \mathbf{x}_i + w_0) - 1] = 0$$

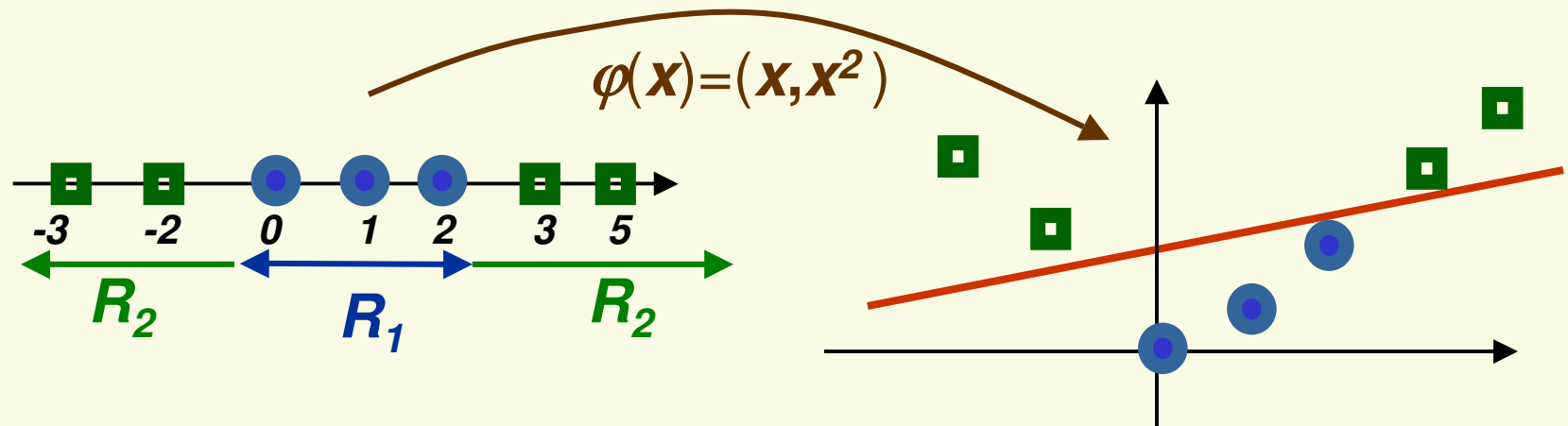
Non Linear Mapping

- Cover's theorem:
 - “*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 $\phi(\mathbf{x})=(\mathbf{x},\mathbf{x}^2)$



Non Linear Mapping

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

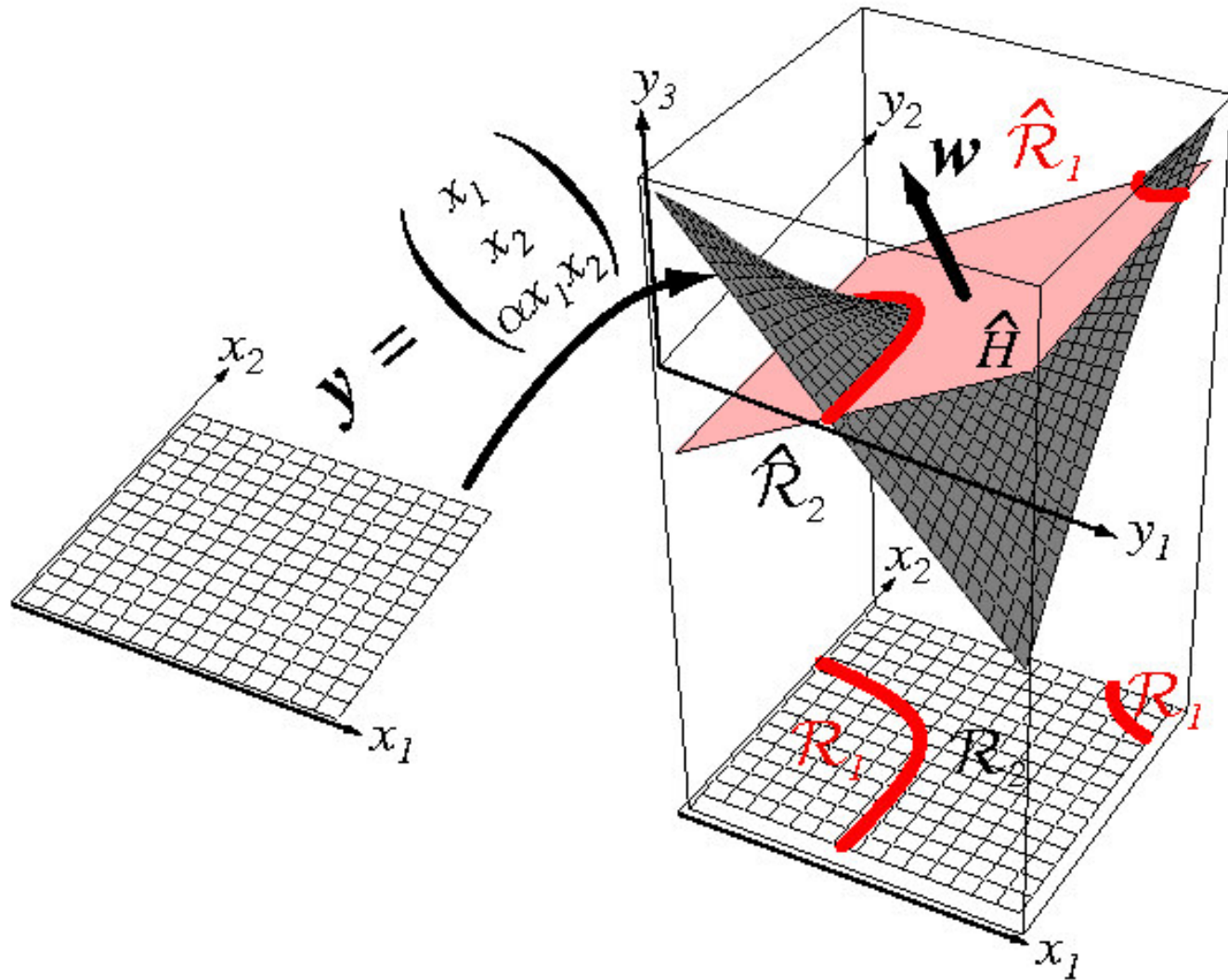


- In 2D, discriminant function is linear

$$g\left(\begin{bmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \end{bmatrix}\right) = [\mathbf{w}_1 \quad \mathbf{w}_2] \begin{bmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \end{bmatrix} + w_0$$

- In 1D, discriminant function is not linear $g(\mathbf{x}) = \mathbf{w}_1 \mathbf{x} + \mathbf{w}_2 \mathbf{x}^2 + w_0$

Non Linear Mapping: Another Example



Non Linear SVM

- Can use any linear classifier after lifting data into a higher dimensional space. However we will have to deal with the “curse of dimensionality”
 1. poor generalization to test data
 2. computationally expensive

- SVM avoids the “curse of dimensionality” problems by
 1. enforcing largest margin permits good generalization
 - It can be shown that generalization in SVM is a function of the margin, independent of the dimensionality
 2. computation in the higher dimensional case is performed only implicitly through the use of **kernel** functions

Non Linear SVM: Kernels

- Recall SVM optimization

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

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

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

$K(\mathbf{x}_i, \mathbf{x}_j)$

- Idea: find **kernel** function $K(\mathbf{x}_i, \mathbf{x}_j)$ s.t.
$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^t \phi(\mathbf{x}_j)$$

Non Linear SVM: Kernels

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

- Then we only need to compute $K(\mathbf{x}_i, \mathbf{x}_j)$ instead of $\varphi(\mathbf{x}_i)^t \varphi(\mathbf{x}_j)$
 - “kernel trick”: do not need to perform operations in high dimensional space explicitly

Non Linear SVM: Kernels

- Suppose we have 2 features and $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^t \mathbf{y})^2$
- Which mapping $\phi(\mathbf{x})$ does it correspond to?

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

- Thus $\phi(\mathbf{x}) = \begin{bmatrix} (\mathbf{x}^{(1)})^2 & \sqrt{2} \mathbf{x}^{(1)} \mathbf{x}^{(2)} & (\mathbf{x}^{(2)})^2 \end{bmatrix}$

Non Linear SVM: Kernels

- How to choose kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$?
 - $K(\mathbf{x}_i, \mathbf{x}_j)$ should correspond to product $\phi(\mathbf{x}_i)^t \phi(\mathbf{x}_j)$ in a higher dimensional space
 - Mercer's condition tells us which kernel function can be expressed as dot product of two vectors

- Some common choices:

- Polynomial kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^t \mathbf{x}_j + 1)^p$$

- Gaussian radial Basis kernel (data is lifted in infinite dimension)

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

Non Linear SVM

- search for separating hyperplane in high dimension

$$\mathbf{w}\phi(\mathbf{x}) + \mathbf{w}_0 = 0$$

- Choose $\phi(\mathbf{x})$ so that the first (“0”th) dimension is the augmented dimension with feature value fixed to 1

$$\phi(\mathbf{x}) = [1 \quad \mathbf{x}^{(1)} \quad \mathbf{x}^{(2)} \quad \mathbf{x}^{(1)}\mathbf{x}^{(2)}]^t$$

- Threshold parameter \mathbf{w}_0 gets folded into the weight vector \mathbf{w}

$$[\mathbf{w}_0 \quad \mathbf{w}] \underbrace{\begin{bmatrix} 1 \\ * \\ \phi(\mathbf{x}) \end{bmatrix}} = 0$$

Non Linear SVM

- Will not use notation $\mathbf{a} = [\mathbf{w}_0 \ \mathbf{w}]$, we'll use old notation \mathbf{w} and seek hyperplane through the origin

$$\mathbf{w}\phi(\mathbf{x}) = 0$$

- If the first component of $\phi(\mathbf{x})$ is not $\mathbf{1}$, the above is equivalent to saying that the hyperplane has to go through the origin in high dimension
 - removes only one degree of freedom
 - But we have introduced many new degrees when we lifted the data in high dimension

Non Linear SVM Recipe

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

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

Non Linear SVM Recipe

- Weight vector \mathbf{w} in the high dimensional space:

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

- where \mathbf{S} is the set of support vectors $\mathbf{S} = \{\mathbf{x}_i \mid \alpha_i \neq 0\}$
- Linear discriminant function of largest margin in the high dimensional space:

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

- Non linear discriminant function in the original space:

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

- decide class 1 if $\mathbf{g}(\mathbf{x}) > 0$, otherwise decide class 2

Non Linear SVM

- Nonlinear discriminant function

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

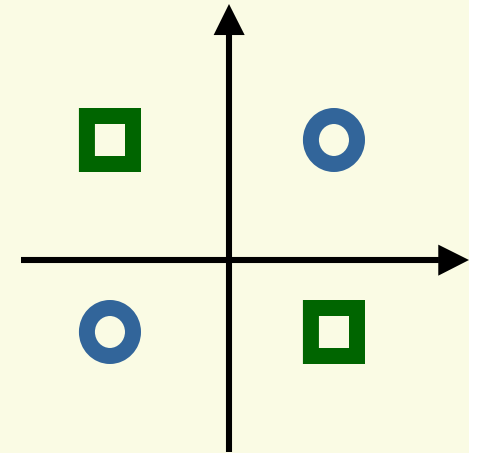
$$g(\mathbf{x}) = \sum \text{weight of support vector } \mathbf{x}_i \quad \mp 1 \quad \text{"inverse distance" from } \mathbf{x} \text{ to support vector } \mathbf{x}_i$$

most important training samples, i.e. support vectors

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

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:
 - $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^t \mathbf{x}_j + 1)^2$
 - This kernel corresponds to mapping



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

- Need to maximize

$$L_D(\alpha) = \sum_{i=1}^4 \alpha_i - \frac{1}{2} \sum_{i=1}^4 \sum_{j=1}^4 \alpha_i \alpha_j \mathbf{z}_i \mathbf{z}_j (\mathbf{x}_i^t \mathbf{x}_j + 1)^2$$

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

SVM Example: XOR Problem

- Can rewrite $L_D(\alpha) = \sum_{i=1}^4 \alpha_i - \frac{1}{2} \alpha^t H \alpha$
 - where $\alpha = [\alpha_1 \ \alpha_2 \ \alpha_3 \ \alpha_4]^t$ and $H = \begin{bmatrix} 9 & 1 & -1 & -1 \\ 1 & 9 & -1 & -1 \\ -1 & -1 & 9 & 1 \\ -1 & -1 & 1 & 9 \end{bmatrix}$
- Take derivative with respect to α and set it to $\mathbf{0}$
$$\frac{d}{d\alpha} L_D(\alpha) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 9 & 1 & -1 & -1 \\ 1 & 9 & -1 & -1 \\ -1 & -1 & 9 & 1 \\ -1 & -1 & 1 & 9 \end{bmatrix} \alpha = \mathbf{0}$$
- Solution to the above is $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \mathbf{0.25}$
 - satisfies the constraints $\forall i, \mathbf{0} \leq \alpha_i$ **and** $\alpha_1 + \alpha_2 - \alpha_3 - \alpha_4 = \mathbf{0}$
 - all samples are support vectors

SVM Example: XOR Problem

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

- Weight vector \mathbf{w} is:

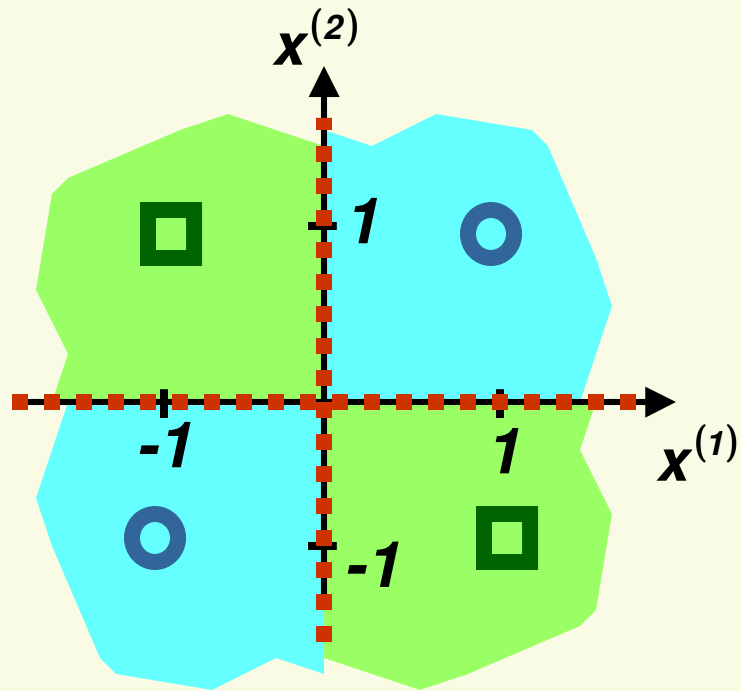
$$\begin{aligned}\mathbf{w} &= \sum_{i=1}^4 \alpha_i \mathbf{z}_i \varphi(\mathbf{x}_i) = 0.25(\varphi(\mathbf{x}_1) + \varphi(\mathbf{x}_2) - \varphi(\mathbf{x}_3) - \varphi(\mathbf{x}_4)) \\ &= \begin{bmatrix} 0 & 0 & 0 & -\sqrt{2} & 0 & 0 \end{bmatrix}\end{aligned}$$

- Thus the nonlinear discriminant function is:

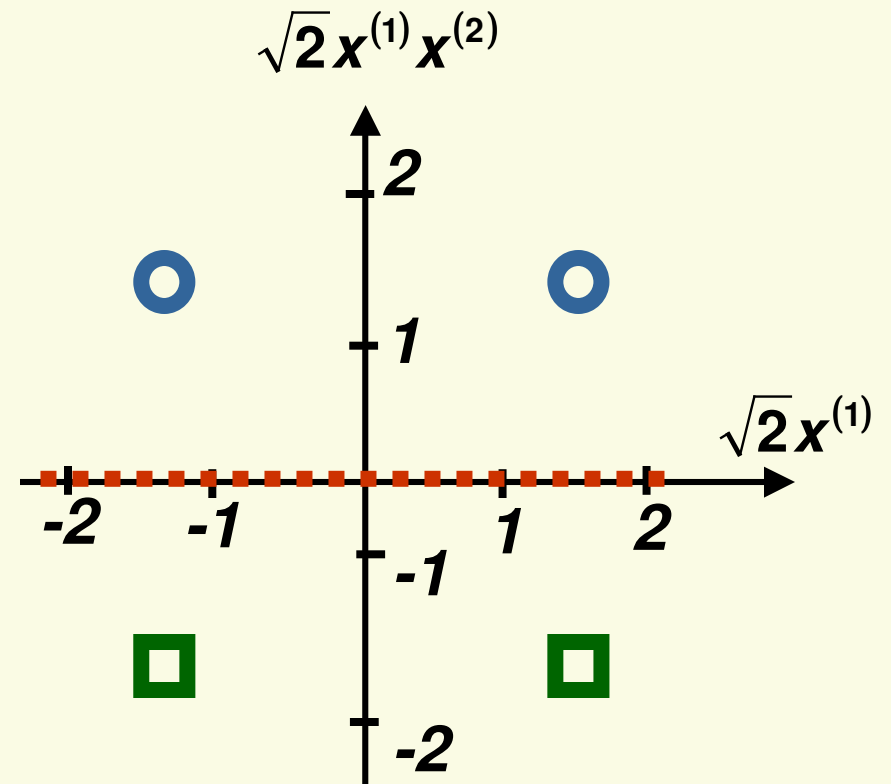
$$g(\mathbf{x}) = \mathbf{w}\varphi(\mathbf{x}) = \sum_{i=1}^6 \mathbf{w}_i \varphi_i(\mathbf{x}) = -\sqrt{2}(\sqrt{2}\mathbf{x}^{(1)}\mathbf{x}^{(2)}) = -2\mathbf{x}^{(1)}\mathbf{x}^{(2)}$$

SVM Example: XOR Problem

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



decision boundaries nonlinear



decision boundary is linear

SVM Summary

- Advantages:
 - Based on nice 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:
 - tends to be slower than other methods
 - quadratic programming is computationally expensive