
Machine Learning

Support Vector Machines

(contains material adapted from talks by Constantin F. Aliferis & Ioannis Tsamardinos, and Martin Law)

Support Vector Machines

- Are binary classifiers
- Are very popular because
 - They are very effective classifiers in many domains
 - They can train fairly quickly on large data sets
 - They can be used without understanding their underlying math
 - ...yet those who want to can geek out on the formulae.

Support Vector Machines

Main ideas:

1. Find an “optimal” hyperplane to split the data into two sets: maximize margin
2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

Defining a hyperplane

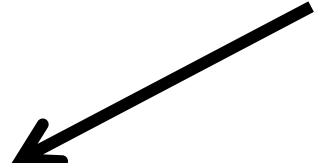
- Any hyperplane can be written as the set of points \mathbf{x} satisfying the equation below, where \mathbf{w} and \mathbf{x} are vectors in \mathbf{R}^d

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

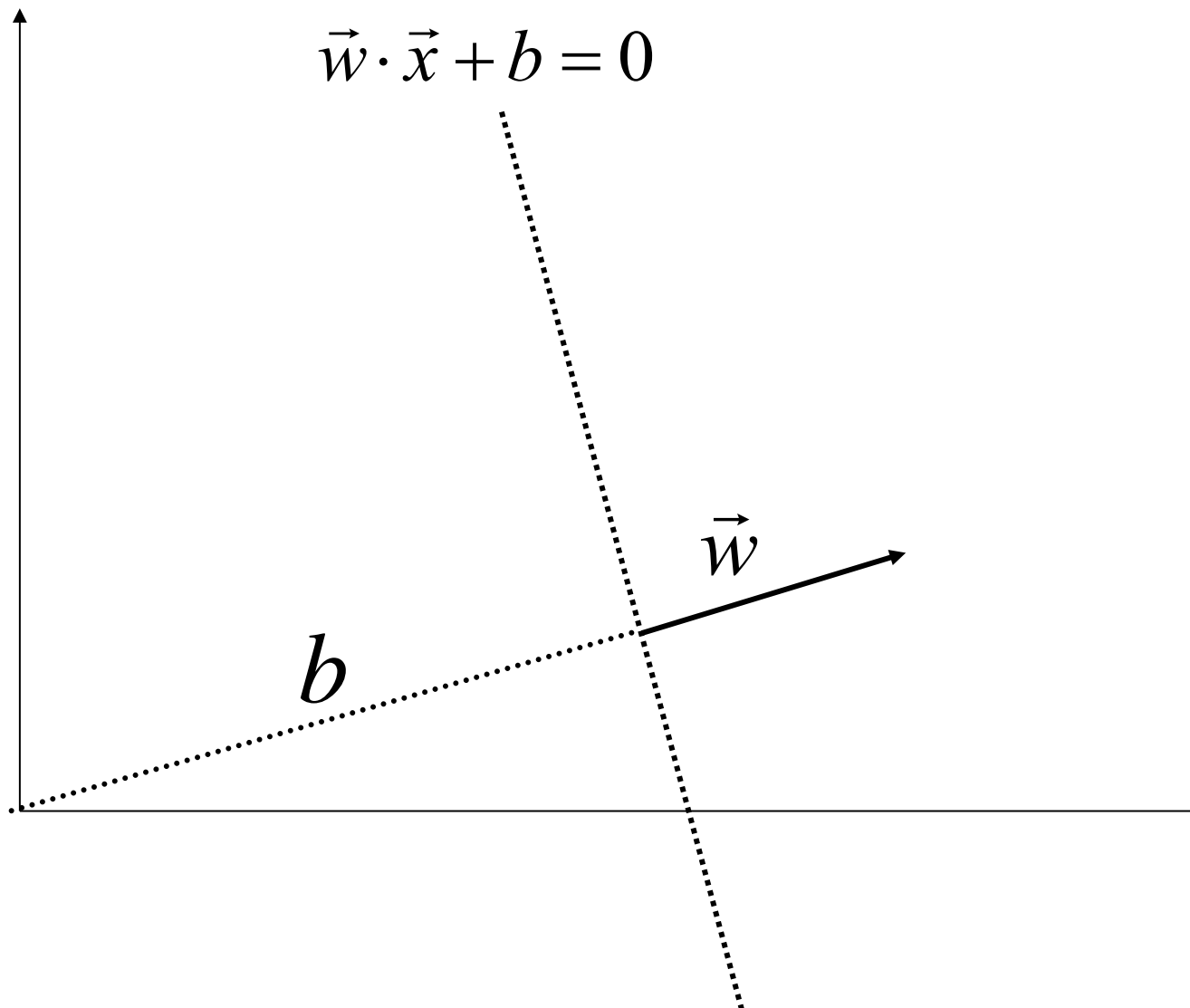
- The vector \mathbf{w} is a normal vector: it is perpendicular to the hyperplane. The parameter b determines the offset of the hyperplane from the origin along the normal vector.

$$\text{dist to origin} = \frac{|b|}{\|\mathbf{w}\|}$$

Euclidean norm



A hyperplane in 2 dimensions



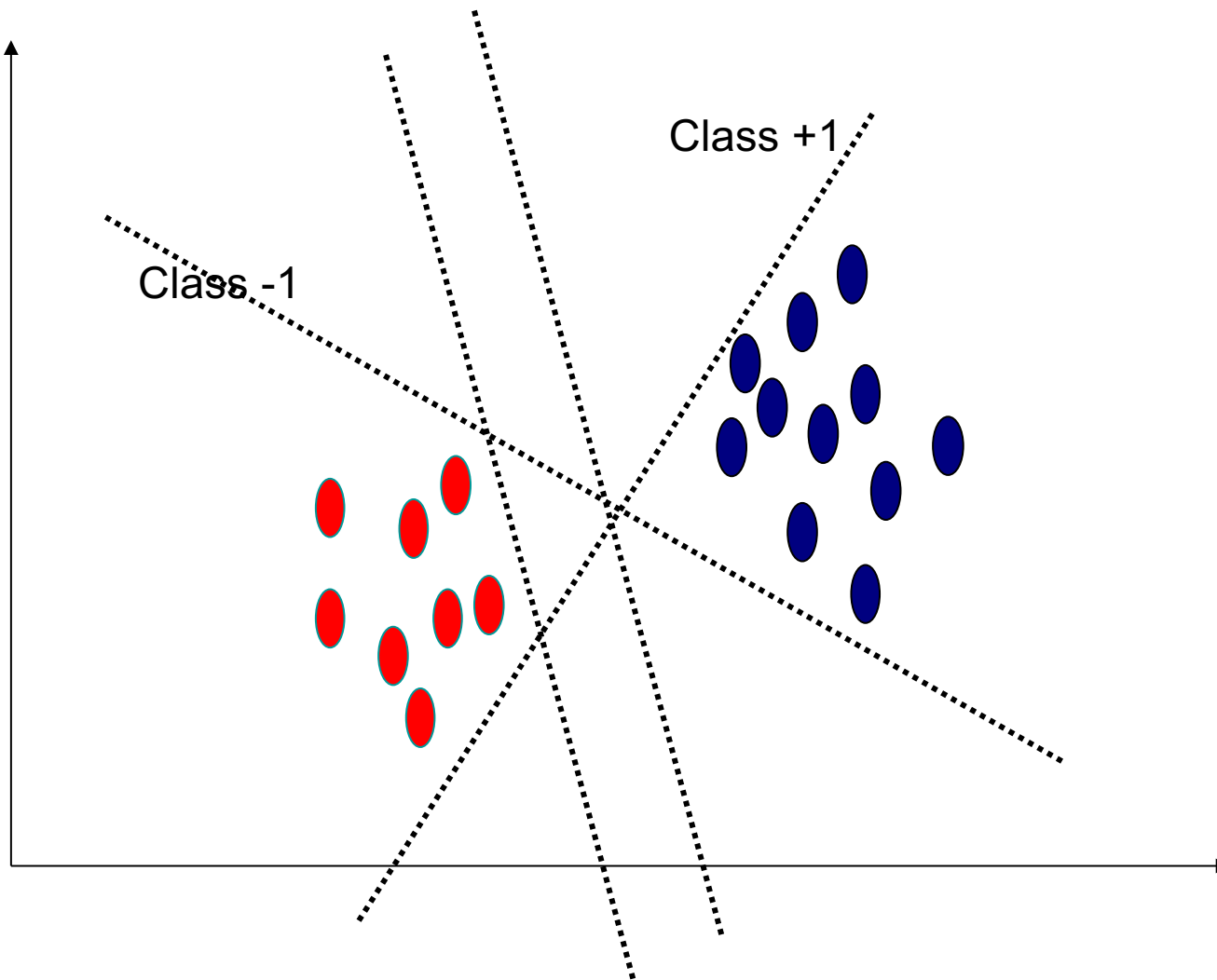
Some definitions

- Assume training data consisting of vectors of d real values that belong to class 1 or class -1

$$D = \{(x_i, y_i) \mid x_i \in \mathfrak{R}^d, y_i \in \{-1, 1\}\}$$

- Find a hyperplane to separate the data into the two classes (assume this is possible, for the moment).

Which hyperplane is best?



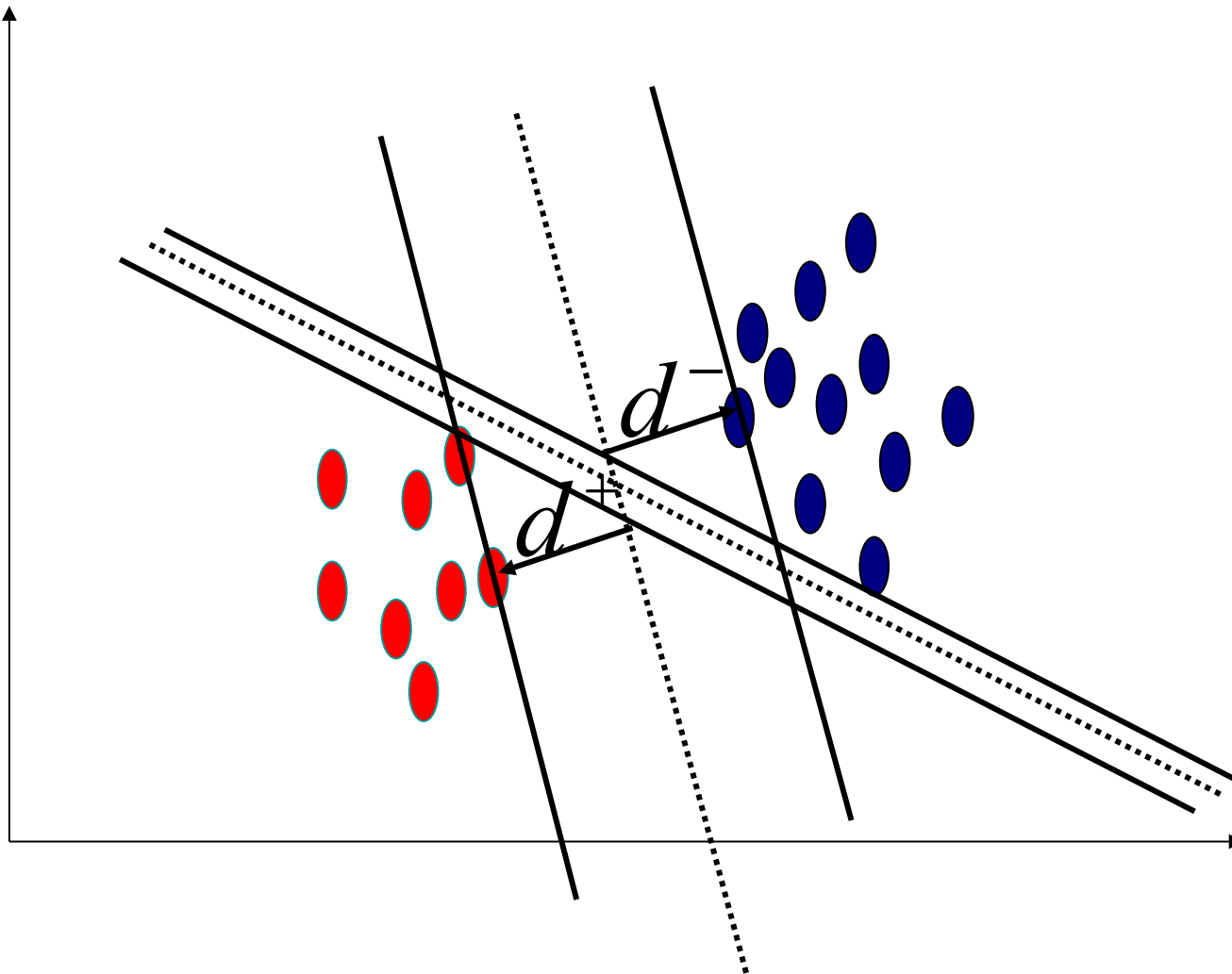
The margin

- Define d^+ as the distance from a hyperplane to the closest positive example.
- Define d^- as the distance to the closest negative example
- Define the “margin”, m as...

$$m = d^+ + d^-$$

- Look for the largest margin!

An example



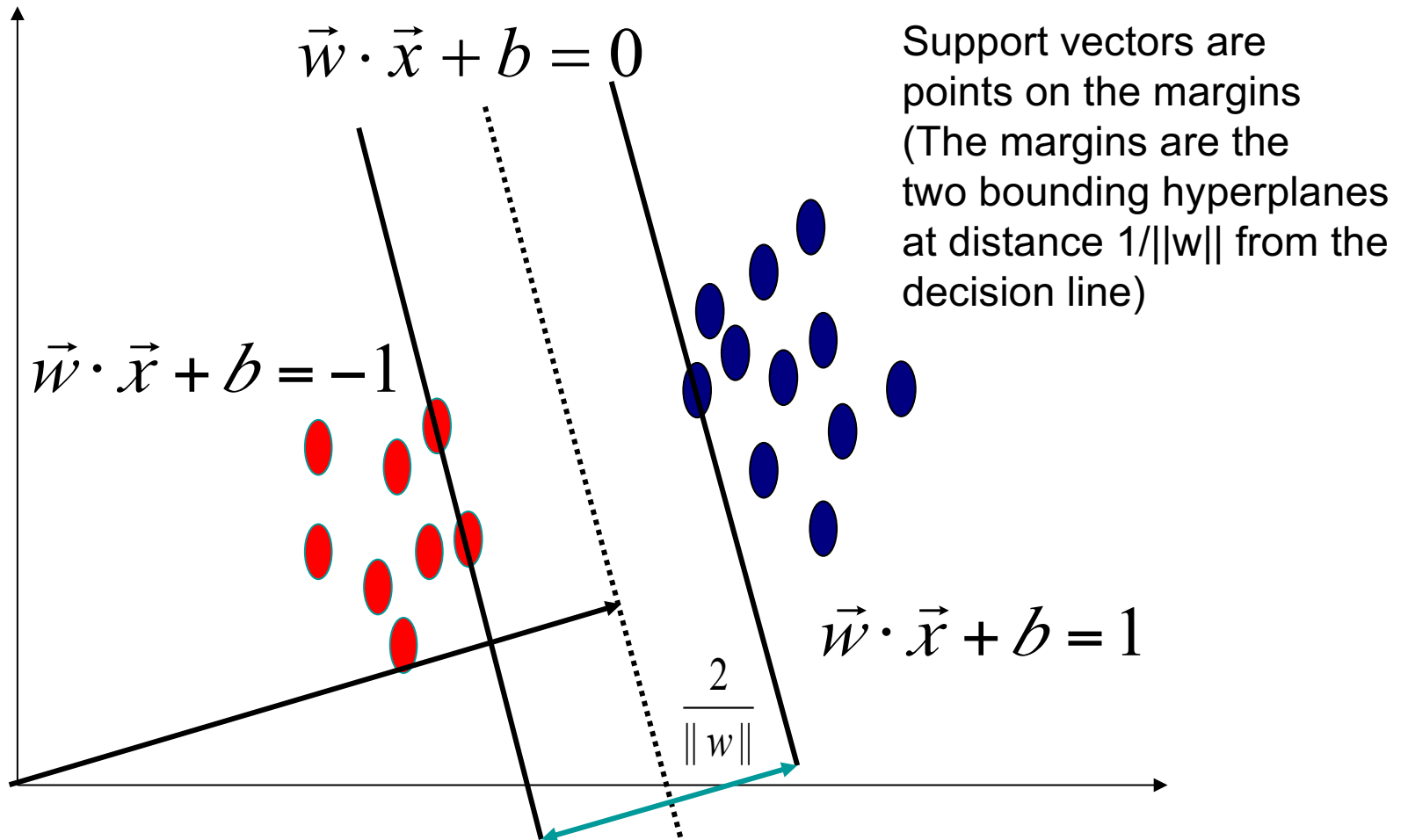
The margin

- There is some scale for \mathbf{w} and for b where the following equation holds.

$$d^+ = d^- = \frac{1}{\|\mathbf{w}\|}$$

- Those data points that lie within distance $1/\|\mathbf{w}\|$ of the hyperplane are called the **support vectors**.
- The support vectors define two planes parallel to the hyperplane separator

Three hyperplanes to consider



Optimization = maximize margin

- We write the optimization problem as...

$$\text{maximize} \left(\frac{2}{\|w\|} \right)$$

such that $y_i(w \cdot x_i + b) \geq 1, \forall \{x_i, y_i\} \in D$

remember y_i is the class label, and $y_i \in \{-1, 1\}$

Maximizing margins (not mathy)

- Given a guess of \mathbf{w} and b we can...
 - See if all data points are correctly classified
 - Compute the width of the margin
- Now, we just search the space of \mathbf{w} 's and b 's to find the widest margin hyperplane that correctly classifies the data.
- How?
Gradient descent? Simulated Annealing?
Matrix Inversion?

A little more mathy

Find \mathbf{w} and b to maximize

$$\text{margin} = \frac{2}{\|\mathbf{w}\|}$$

such that $y_i (w \cdot x_i + b) \geq 1, \forall \{x_i, y_i\} \in D$

- This is a quadratic function with linear constraints.
- Quadratic optimization problems have known algorithmic solutions.
- Naively, this quadratic optimization can be solved in $O(n^3)$ time, where n = the number of data points.

Still mathy-er

- Maximizing $\left(\frac{2}{\|w\|}\right)$ equivalent to minimizing $\|w\|$

- For math convenience we ACTUALLY minimize

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

- Now, we associate a Lagrange multiplier α_i with each point x_i in the data. This gives...

$$L_p \equiv \frac{1}{2} \|w\|^2 - \sum_{i=1}^{|D|} \alpha_i y_i (x_i \cdot w + b) + \sum_{i=1}^{|D|} \alpha_i$$

(psst: What's a Lagrange Multiplier?)

- A way of optimizing a function subject to one (or more) constraint(s) from another function(s).
- You incorporate the original function and the constraint equations into one new equation and then solve.
- For more, check out the wikipedia.

Why use Lagrangian Multipliers?

- We want to put a line between the sets that maximize the margin between classes subject to the constraint all points are correctly classified.
- So...
 - The margin maximization is a classic maximization problem
 - Classifying each point correctly is a constraint
- This is exactly the setup needed for Lagrangians

The Lagrangian “dual”

- In practice, people don't minimize this formula from the previous slide.

$$L_p \equiv \frac{1}{2} \|w\|^2 - \sum_{i=1}^{|D|} \alpha_i y_i (x_i \cdot w + b) + \sum_{i=1}^{|D|} \alpha_i$$

- Instead they maximize its “dual,” which will also give us what we need and is in a more convenient format for later work.

$$L_D \equiv \sum_{i=1}^{|D|} \alpha_i - \frac{1}{2} \sum_{i=1}^{|D|} \sum_{j=1}^{|D|} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

The Lagrangian “dual”

- When we maximize this, a cool thing happens. Only those points defining the margin have non-zero values for α

$$L_D \equiv \sum_{i=1}^{|D|} \alpha_i - \frac{1}{2} \sum_{i=1}^{|D|} \sum_{j=1}^{|D|} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

- Note also this formulation relates two examples to each other via a dot product.
- This will become meaningful later...

Classification with SVM

- For testing, the new data \mathbf{z} is classified as class 1 if $f \geq 0$ and as class -1 if $f < 0$
- SO...our weights are determined by this (where S is the number of support vectors...aka important points from our training data)

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

- And our decision function is a normal linear discriminant.

$$f = \mathbf{w}^T \mathbf{z} + b$$

Classification with SVM: PART 2

While our decision function is a normal linear discriminant....

$$f = \mathbf{w}^T \mathbf{z} + b$$

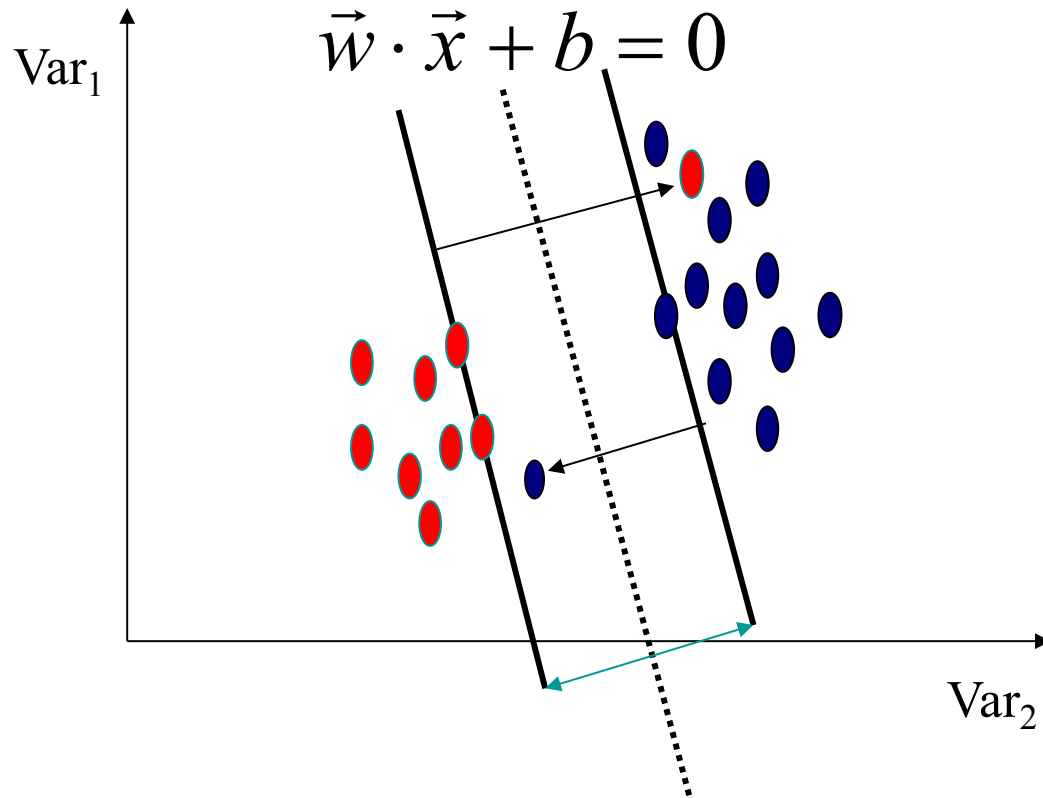
...people usually calculate the class using the support vectors (those data points with non-0 alpha values that lie on the +1 and -1 margins).

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

The new element \mathbf{z} is compared to all the support vectors and its value is determined by where it lies in comparison to them.

What if we have noisy training data?

- Can we combine minimizing misclassifications (with some forgiveness) with maximizing the margin?

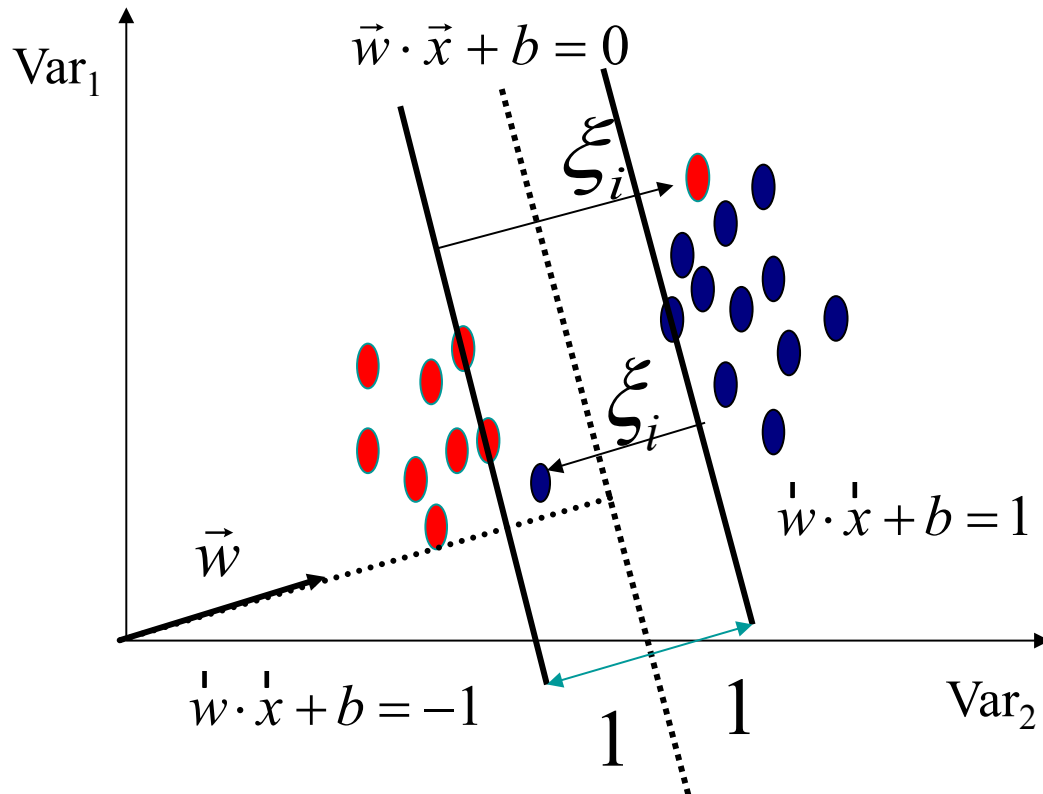


Support Vector Machines

Main ideas:

1. Find an “optimal” hyperplane to split the data into two sets: maximize margin
2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

Non-Linearly Separable Data



Allow some instances to fall within the margin, but penalize them

Introduce slack variables ξ_i (one per data point)

The constraints then become...

$$y_i(w \cdot x + b) \geq 1 - \xi_i \quad \forall (x_i, y_i) \in D, \xi_i \geq 0$$

Formulating the Problem

- We are now minimizing this formula

$$\frac{1}{2} w \cdot w + C \sum_i \xi_i$$

- Subject to the constraints

$$y_i(w \cdot x + b) \geq 1 - \xi_i \quad \forall (x_i, y_i) \in D, \xi_i \geq 0$$

- Where C determines the weight to give misclassification error.

Linear, Soft-Margin SVMs

$$\min \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \quad \begin{array}{l} y_i (w \cdot x_i + b) \geq 1 - \xi_i, \quad \forall x_i \\ \xi_i \geq 0 \end{array}$$

- Algorithm tries to minimize ξ_i while maximizing margin
- Notice: algorithm does not minimize the *number* of misclassifications, but the sum of distances from the margin hyperplanes
- As $C \rightarrow \infty$, we get closer to the hard-margin solution

The Lagrange Dual

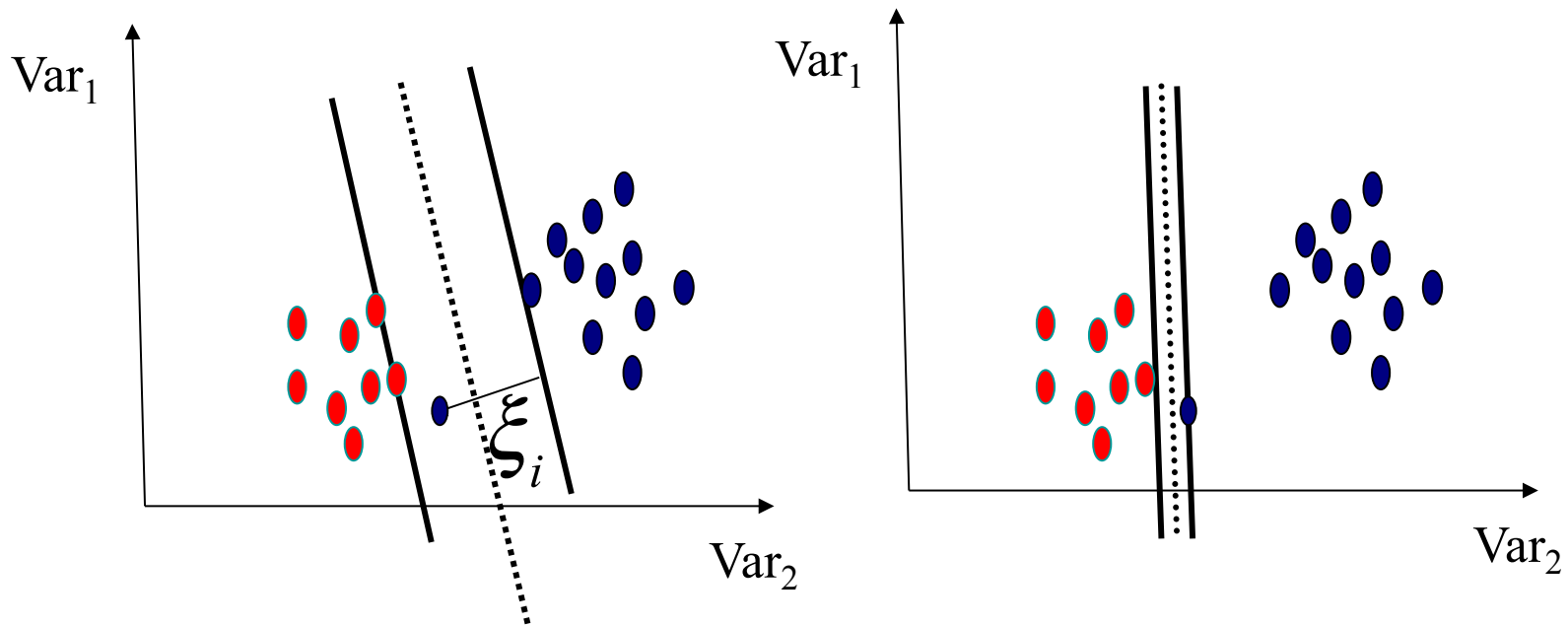
- The dual of the “soft” problem is

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

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

- \mathbf{w} is also recovered as $\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- The only difference with the linearly separable case is that there is an upper bound C on α_i
- Once again, a QP solver can be used to find α_i

Robustness of Soft vs Hard Margin



Soft Margin SVM

Hard Margin SVM

Soft vs Hard Margin SVM

- Soft-Margin always have a solution
- Soft-Margin is more robust to outliers
 - Smoother surfaces (in the non-linear case)
- Hard-Margin requires no parameters at all

Linear SVMs: Overview

- The classifier is a *separating hyperplane*.
- Most “important” training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points \mathbf{x}_i are support vectors with non-zero Lagrangian multipliers.
- Both in the dual formulation of the problem and in the solution, training points appear only inside inner products:

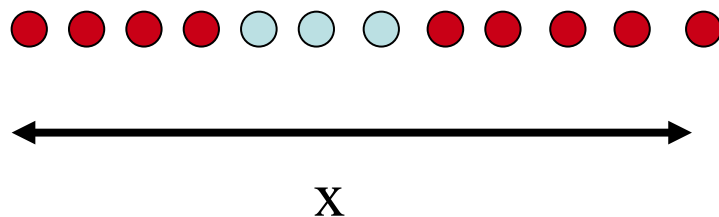
$$L_D \equiv \sum_{i=1}^{|D|} \alpha_i - \frac{1}{2} \sum_{i=1}^{|D|} \sum_{j=1}^{|D|} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

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

Extension to Non-linear Decision Boundary

- So far, we have only considered large-margin classifier with a linear decision boundary
- What if the decision boundary is non-linear?

A one-dimensional decision problem



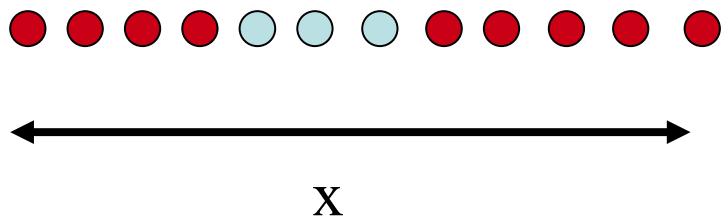
Support Vector Machines

Main ideas:

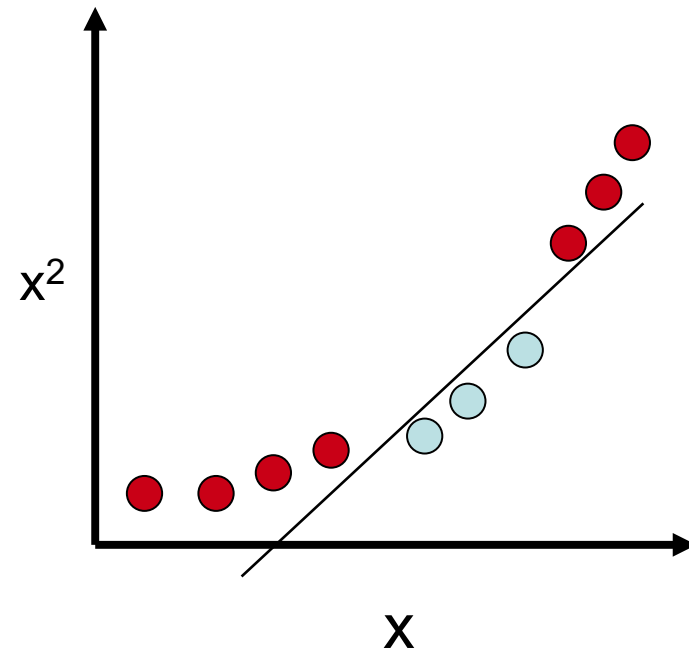
1. Find an “optimal” hyperplane to split the data into two sets: maximize margin
2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

Higher dimensional mapping

- The idea is to map the vectors to a higher dimensional space to gain linear separation.

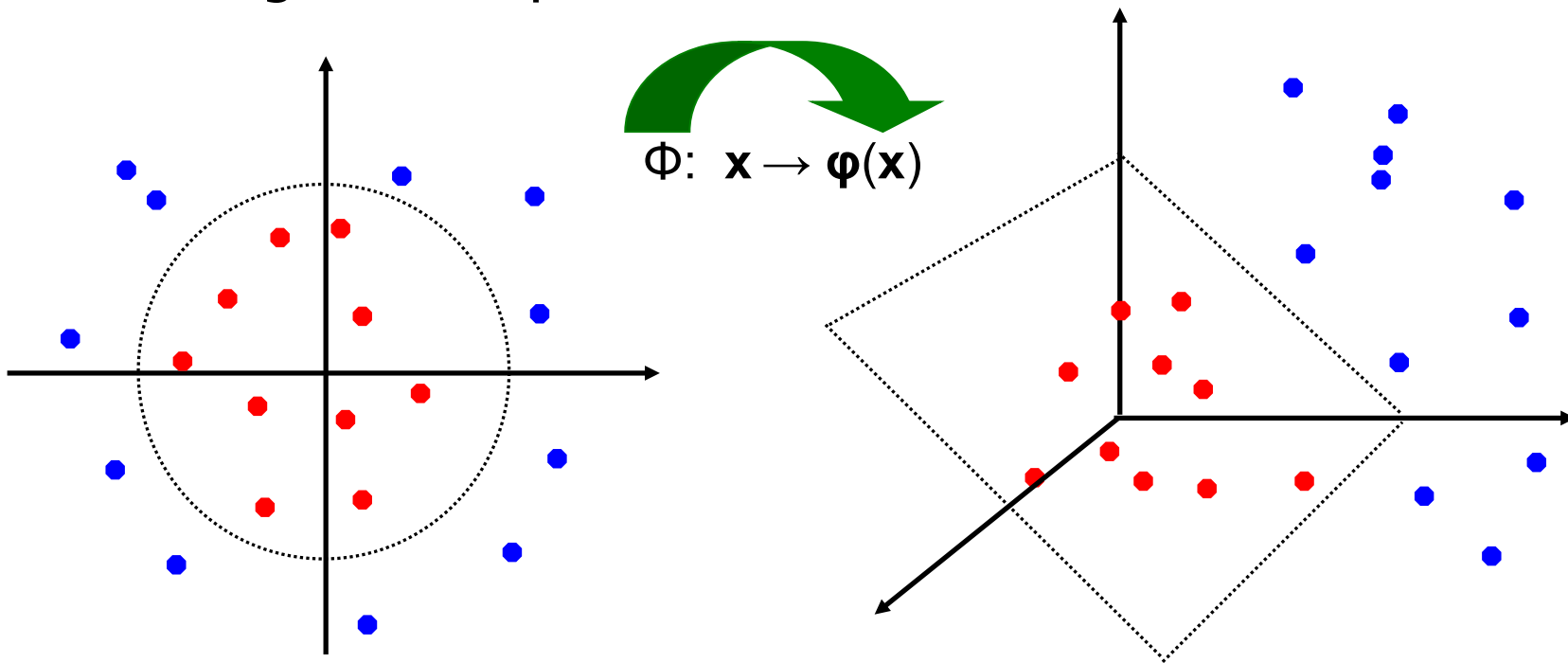


Mapping x onto $\{x^2, x\}$ gives linear separation.



Non-linear SVMs: Feature spaces

- General idea: the original feature space can be mapped to some higher-dimensional feature space where the training set is separable:



What this does to the math

- Recall the SVM optimization problem with its inner product between data points

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

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

- If we transform the data into a new vector space, this changes the math ever so slightly to be...

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

What are Kernel Functions?

$$K(x_1, x_2) = \langle \Phi(x_1), \Phi(x_2) \rangle$$

- Kernels (in this context) are functions that return inner products between the images of data points
- Since they are inner products, they can be thought of as similarity functions

Why Use Kernel Functions

$$K(x_1, x_2) = \langle \Phi(x_1), \Phi(x_2) \rangle$$

- You can define kernels for many things that aren't vectors of real values in Euclidean space (e.g. text documents)
- As long as we can calculate the inner product in the original feature space, we do not need to explicitly calculate the mapping Φ
- It can often be more efficient to compute K directly, without going through the step of computing Φ

That math again...

This original formula...

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

...becomes this, when we apply a data transformation...

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

...and if we can define a kernel...

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

...it becomes this...

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

An Example Kernel

- Suppose ϕ is given as follows

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

- An inner product in the feature space is

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

**** NOTE: On this slide y is not the label, it is a feature vector, just like x ****

- So, if we define the kernel function as follows, there is no need to carry out $\phi(\cdot)$ explicitly

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

- This use of kernel function to avoid calculating ϕ explicitly is known as the **kernel trick**

Examples of Kernel Functions

** NOTE: On this slide y is not the label, it is a feature vector, just like x **

- Polynomial kernel with degree d

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

- Radial basis function kernel with width σ

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

- Closely related to radial basis function neural networks
- The feature space is infinite-dimensional

- Sigmoid with parameter κ and θ

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x}^T \mathbf{y} + \theta)$$

- It does not satisfy the Mercer condition on all κ and θ

Building Kernels from Other Kernels

- Kernels can be composed from other kernels
- The following 2 slides give some basic rules for building complex kernels from simple kernels

Definitions for the following slide

$k_1(x, x')$ and $k_2(x, x')$ are valid kernels on $\{x, x'\} \in S$

S is some set (of anything: emails, images, integers)

$c > 0$ is a constant

$f(\cdot)$ is any function

$q(\cdot)$ is a polynomial with non-negative coefficients

$\phi(x)$ is a function from the $\rightarrow \mathbb{R}^m$

$k_3(\cdot, \cdot)$ is a valid kernel in \mathbb{R}^m

A is a symmetric positive semidefinite matrix

$x = (x_a, x_b)$...essentially, x can be decomposed into subparts

...like scalars in a vector

$k_a(\cdot, \cdot), k_b(\cdot, \cdot)$ are valid kernels over their respective spaces

Techniques for Kernel Construction

Given valid kernels $k_1(x, x')$ and $k_2(x, x')$,

the following are also valid kernels

$$k(x, x') = ck_1(x, x')$$

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

$$k(x, x') = q(k_1(x, x'))$$

$$k(x, x') = \exp(k_1(x, x'))$$

$$k(x, x') = k_1(x, x') + k_2(x, x')$$

$$k(x, x') = k_3(\phi(x), \phi(x'))$$

$$k(x, x') = x^T Ax' \quad \text{This one assumes } x, x' \text{ are vectors}$$

$$k(x, x') = k_a(x_a, x'_a) + k_b(x_b, x'_b)$$

$$k(x, x') = k_a(x_a, x'_a)k_b(x_b, x'_b)$$

Classifying with a Kernel

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

Original

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

With kernel function

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

NOTE: y is once again a label from the set $\{-1, 1\}$ **

Strengths and Weaknesses of SVM

- Strengths
 - Training is relatively easy
 - It scales relatively well to high dimensional data
 - Tradeoff between classifier complexity and error can be controlled explicitly
 - Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors
- Weaknesses
 - Tuning SVMs remains a black art: selecting a specific kernel and parameters is usually done in a try-and-see manner.

You as the SVM user

- You have two main choices to make:
 - 1) What kernel will you use?
 - Polynomial?
 - Radial Basis Function?
 - Something else?
 - 2) How much “slack” will you allow?
 - Depends on how much you trust data collection and labeling.

SVM applications

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- SVM techniques have been extended to a number of tasks such as regression [Vapnik *et al.* '97], principal component analysis [Schölkopf *et al.* '99], etc.
- Most popular optimization algorithms for SVMs use *decomposition* to hill-climb over a subset of a_i 's at a time, e.g. SMO [Platt '99] and [Joachims '99]