



Never Stand Still

# Kernel Methods

COMP9417 Machine Learning & Data Mining

Term 3, 2019

Adapted from slides by Dr Michael Bain

# Aims

This lecture will develop your understanding of kernel methods in machine learning. Following it you should be able to:

- describe perceptron learning
- describe learning with the dual perceptron
- outline the idea of learning in a dual space
- describe the concept of maximizing the margin in linear classification
- outline the typical loss function for maximizing the margin
- describe the method of support vector machines (SVMs)
- describe the concept of kernel functions
- outline the idea of using a kernel in a learning algorithm
- outline non-linear classification with kernel methods

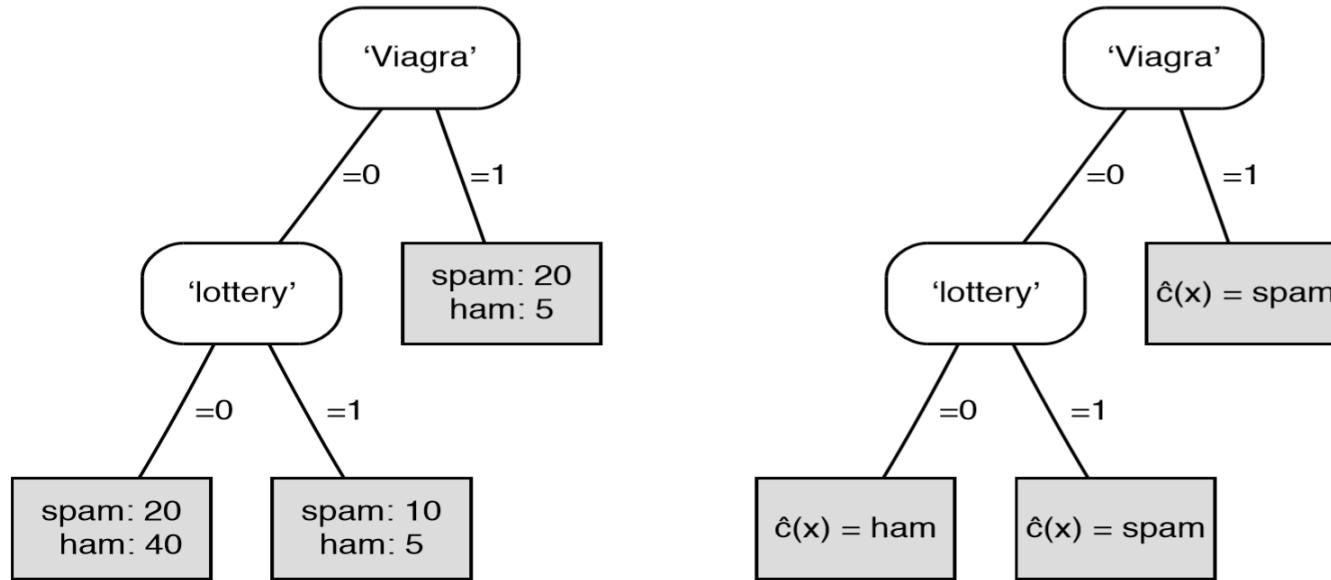
# Predictive machine learning scenarios

<i>Task</i>	<i>Label space</i>	<i>Output space</i>	<i>Learning problem</i>
Classification	$\mathcal{L} = \mathcal{C}$	$\mathcal{Y} = \mathcal{C}$	learn an approximation $\hat{c} : \mathcal{X} \rightarrow \mathcal{C}$ to the true labelling function $c$
Scoring and ranking	$\mathcal{L} = \mathcal{C}$	$\mathcal{Y} = \mathbb{R}^{ \mathcal{C} }$	learn a model that outputs a score vector over classes
Probability estimation	$\mathcal{L} = \mathcal{C}$	$\mathcal{Y} = [0, 1]^{ \mathcal{C} }$	learn a model that outputs a probability vector over classes
Regression	$\mathcal{L} = \mathbb{R}$	$\mathcal{Y} = \mathbb{R}$	learn an approximation $\hat{f} : \mathcal{X} \rightarrow \mathbb{R}$ to the true labelling function $f$

# Classification

- A classifier is a mapping  $\hat{c}: \mathcal{X} \rightarrow C$ , where  $C = \{C_1, C_2, \dots, C_k\}$  is a finite and usually small set of class labels. We will sometimes also use  $C_i$  to indicate the set of examples of that class.
- We use the ‘hat’ to indicate that  $\hat{c}(x)$  is an estimate of the true but unknown function  $c(x)$ . Examples for a classifier take the form  $(x, c(x))$ , where  $x \in \mathcal{X}$  is an instance and  $c(x)$  is the true class of the instance (sometimes contaminated by noise).
- Learning a classifier involves constructing the function such that it matches  $c$  as closely as possible (and not just on the training set, but ideally on the entire instance space  $\mathcal{X}$ ).

# A decision tree



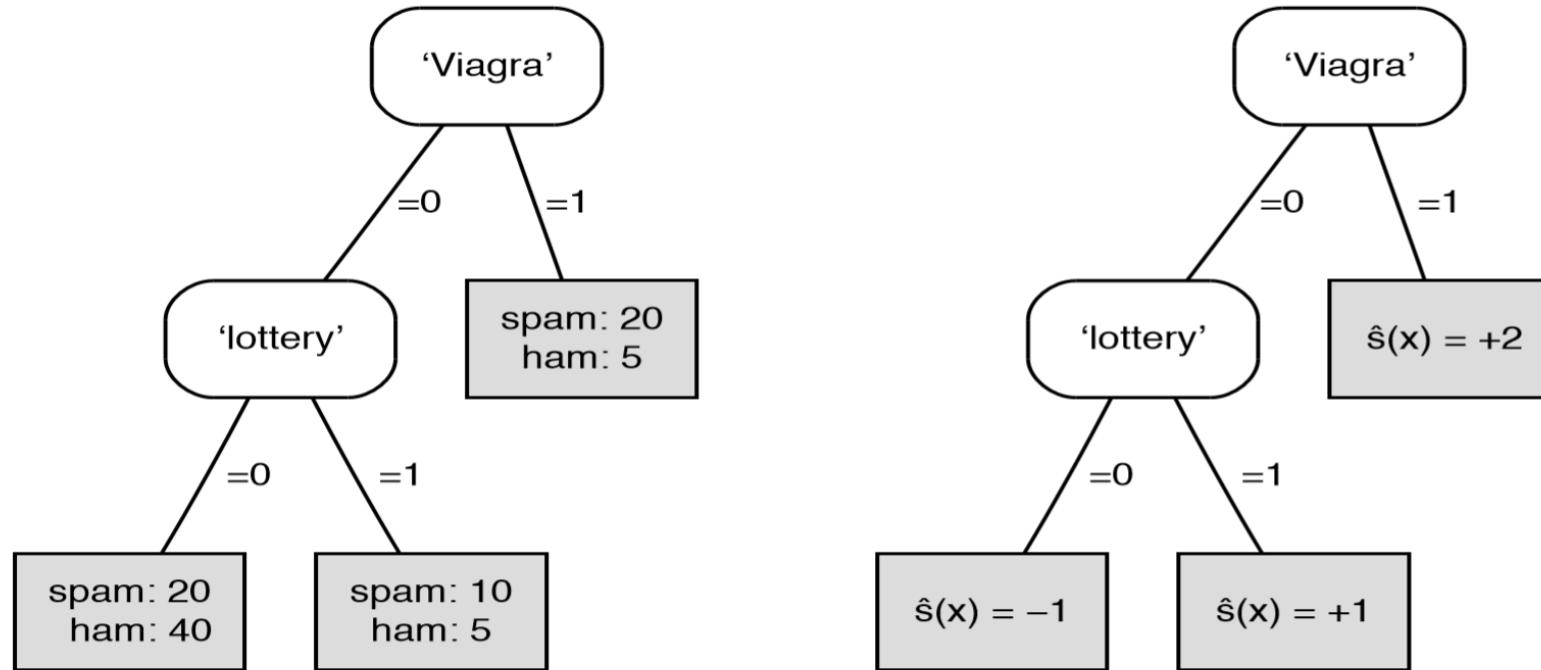
(left) A tree with the training set class distribution in the leaves.

(right) A tree with the majority class prediction rule in the leaves

# Scoring classifier

- A scoring classifier is a mapping  $\hat{S}: \mathcal{X} \rightarrow \mathbb{R}^k$  i.e., a mapping from the instance space to a  $k$  –vector of real numbers.
- The boldface notation indicates that a scoring classifier outputs a vector  $\hat{S}(x) = (\hat{s}_1(x), \dots, \hat{s}_k(x))$  rather than a single number;  $\hat{s}_i(x)$  is the score assigned to class  $C_i$  for instance  $x$ .
- This score indicates how likely it is that class label  $c_i$  applies.
- If we only have two classes, it usually suffices to consider the score for only one of the classes; in that case, we use  $\hat{s}(x)$  to denote the score of the positive class for instance  $x$ .

# A scoring tree



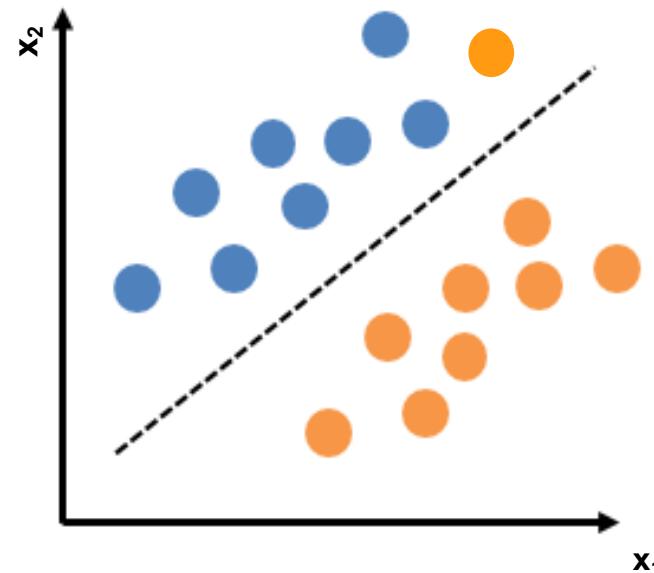
(left) A tree with the training set class distribution in the leaves.

(right) A tree using the logarithm of the class ratio as scores; spam is taken as the positive class.

# Margins and loss functions

If we take the true class  $c(x)$  as  $+1$  for positive examples and  $-1$  for negative examples, then the quantity  $z(x) = c(x)\hat{s}(x)$  is positive for correct predictions and negative for incorrect predictions: this quantity is called the margin assigned by the scoring classifier to the example.

For example in a linear classifier, we can define the score to be the distance between the examples and the line



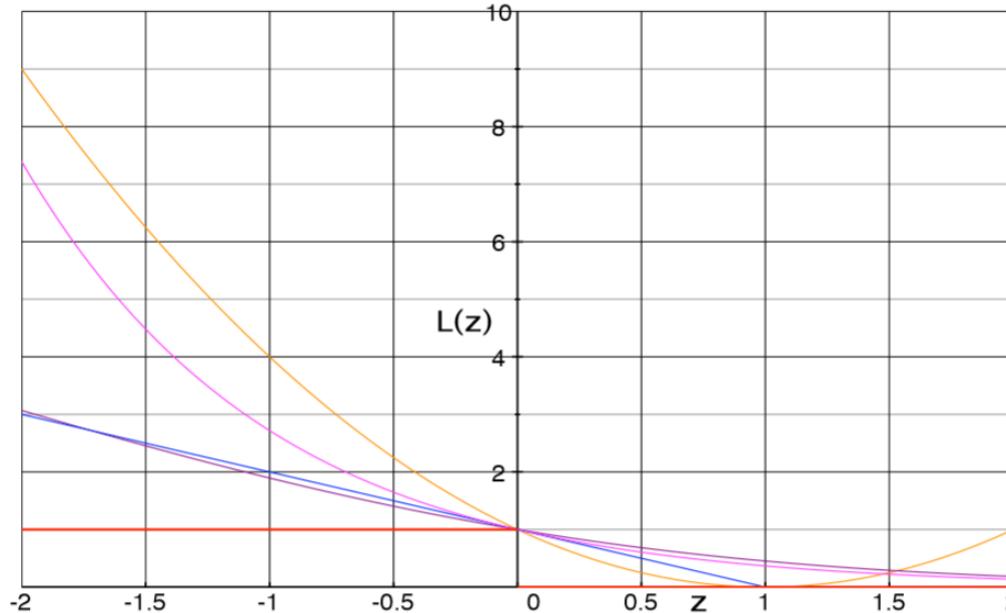
# Margins and loss functions

We would like to reward large positive margins, and penalize large negative values. This is achieved by means of a so-called **loss function**  $L : \mathbb{R} \rightarrow [0, \infty)$  which maps each example's margin  $z(x)$  to an associated loss  $L(z(x))$ .

We will assume that  $L(0) = 1$ , which is the loss incurred by having an example on the decision boundary. We furthermore have  $L(z) \geq 1$  for  $z < 0$ , and usually also  $0 \leq L(z) < 1$  for  $z > 0$ . The average loss over a test set  $T_e$  is

$$\frac{1}{|T_e|} \sum_{x \in T_e} L(z(x))$$

# Loss functions

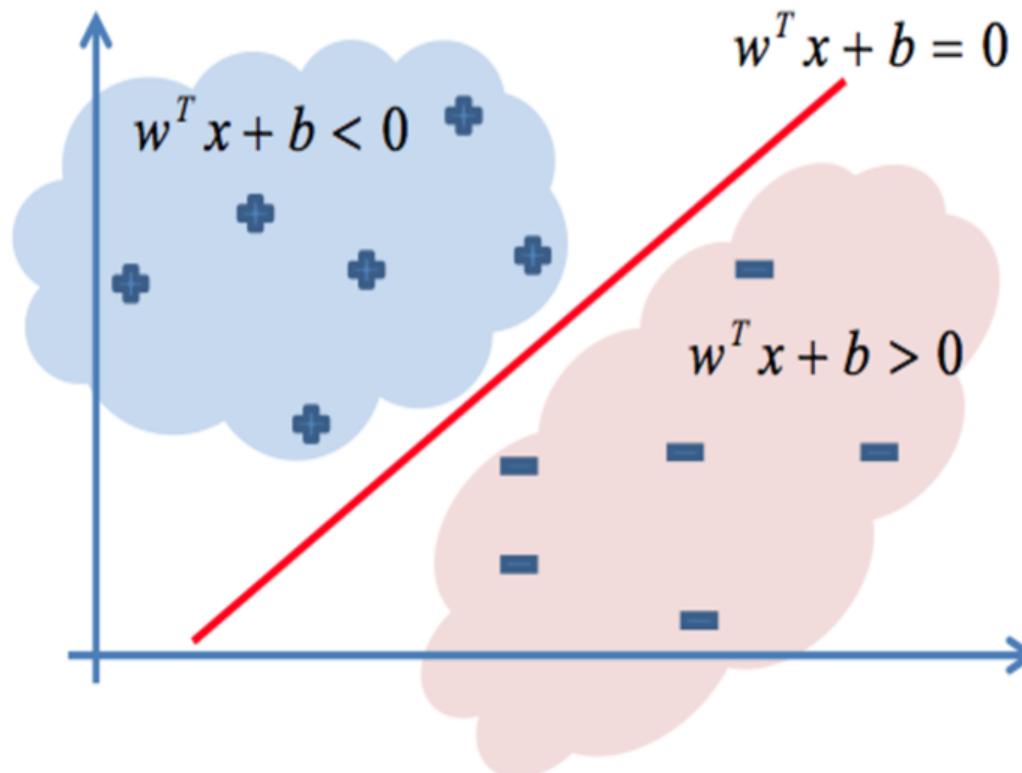


From bottom-left: (i) 0–1 loss  $L_{01}(z) = 1$  if  $z \leq 0$ , and  $L_{01}(z) = 0$  if  $z > 0$ ; (ii) hinge loss  $L_h(z) = (1 - z)$  if  $z \leq 1$ , and  $L_h(z) = 0$  if  $z > 1$ ; (iii) logistic loss  $L_{log}(z) = \log_2(1 + \exp(-z))$ ; (iv) exponential loss  $L_{exp}(z) = \exp(-z)$ ; (v) squared loss  $L_{sq}(z) = (1 - z)^2$  (can be set to 0 for  $z > 1$ , just like hinge loss).

# Review: Linear classification

- Two-class classifier “separates” instances in feature space:

$$f(x) = \text{sign}(w^T x + b)$$

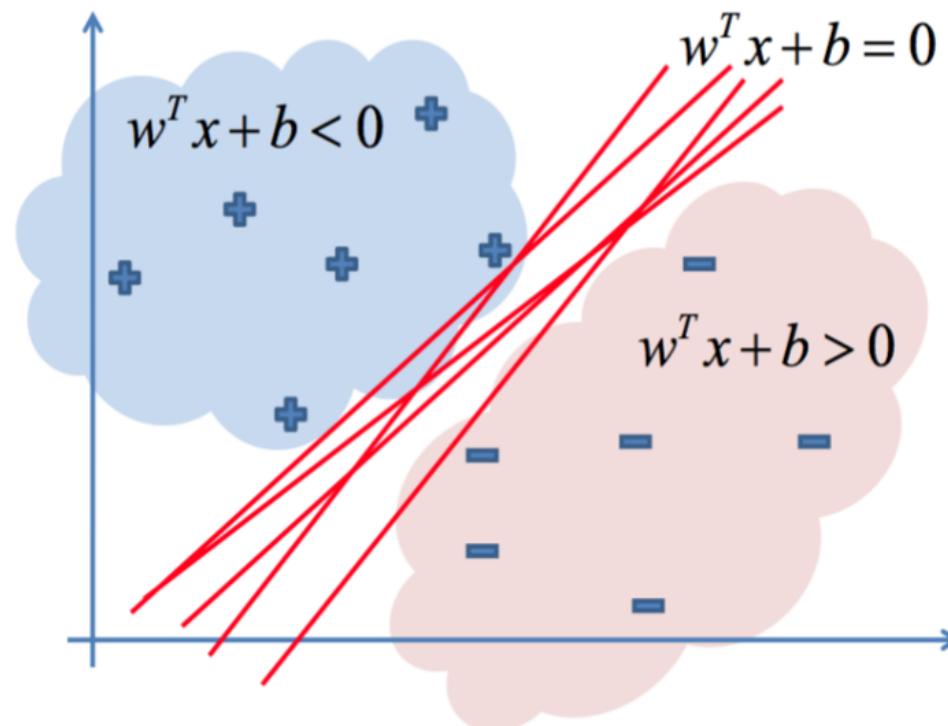


# Issues in linear classification

- Define a decision boundary by a hyperplane in feature space
- A linear model can be used for classification

# Issues in linear classification

- Many possible linear decision boundaries: which one to choose?

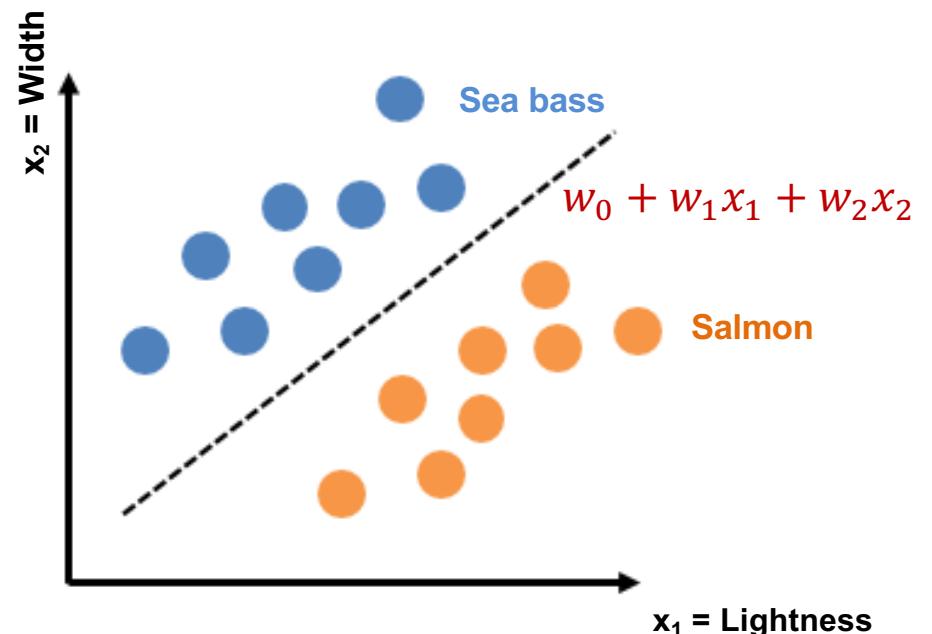


# Linear classification: Perceptron

**Perceptron**: is an algorithm for binary classification that uses a linear prediction function

If we have two attributes/features of  $x_1$  and  $x_2$  then we can predict the target function  $f(x)$  with:

$$f(x) = \begin{cases} +1 & \text{if } w_0 + w_1x_1 + w_2x_2 > 0 \\ -1 & \text{otherwise.} \end{cases}$$



# Linear classification: Perceptron

For a general case with  $n$  attributes:

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } w_0 + w_1x_1 + \cdots + w_nx_n \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

If we add  $x_0 = 1$  to the feature vector:

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } \sum_{i=0}^n w_i x_i \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

$$\sum_{i=0}^n w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

Dot product:  $\mathbf{a} \cdot \mathbf{b} = a_1b_1 + a_2b_2 + \cdots + a_nb_n$

# Perceptron learning

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

$$\hat{y} = f(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x})$$

Now, we have to find a good set of weights using our training set  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$  with labels  $y_1, y_2, \dots, y_m$

Please note that, here, the subscripts correspond to the sample number and not features.  $\mathbf{x} = [x_1, \dots, x_n]$

# Perceptron learning

The perceptron algorithm initializes all weights  $w_i$  to zero, and learns the weights using the following update rule:

$$w := w + \frac{1}{2} (y_j - f(x_j)) x_j$$

There are 4 cases:

$$y = +1, f(x) = +1 \Rightarrow (y - f(x)) = 0$$

$$y = +1, f(x) = -1 \Rightarrow (y - f(x)) = +2$$

$$y = -1, f(x) = +1 \Rightarrow (y - f(x)) = -2$$

$$y = -1, f(x) = -1 \Rightarrow (y - f(x)) = 0$$

# Perceptron learning

$w$  gets updated only if the prediction mismatches the actual class label (misclassification) and otherwise remains the same. Therefore, for misclassified instances we can write:

$$w := w + y_j x_j$$

Perceptron algorithm:

1. Initialize all weights  $w$  to zero
2. Iterate through the training data. For each training sample, classify the sample:
  - a) If the prediction was correct, don't do anything
  - b) If the prediction was wrong modify the weights by using the update rules
3. Repeat step 2 some number of times

# Perceptron training algorithm

**Algorithm** Perceptron( $D$ ) / perceptron training for linear classification

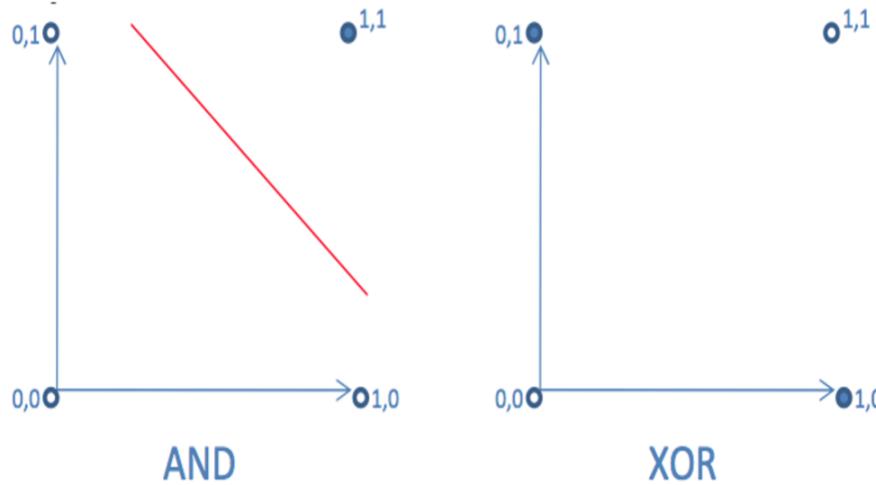
**Input:** labelled training data  $D$  in homogeneous coordinates

**Output:** weight vector  $w$

```
w ← 0
converged ← false
while converged = false do
    converged ← true
    for i = 1 to |D| do
        if  $y_i w \cdot x_i \leq 0$  then
            w ← w +  $y_i x_i$ 
            converged ← false
        end
    end
end
```

# Issues in linear classification

- May not be possible to find a linear separating hyperplane



- Filled / empty circles are in / out of the target concept
- AND is linearly separable – but not XOR

# Extending linear classification

- Linear classifiers can't model nonlinear class boundaries
- Simple trick to allow them to do that:
  - Nonlinear mapping: map attributes into new space consisting of combinations of attribute values
  - For example: all products with  $n$  factors that can be constructed from the attributes (feature construction or basis expansion)
- e.g., for 2 attributes, all products with  $n = 3$  factors
$$y = w_1 x_1^3 + w_2 x_1^2 x_2 + w_3 x_1 x_2^2 + w_4 x_2^3$$
- $y$  is predicted output for instances with two attributes  $x_1$  and  $x_2$

# Two main problems

- Efficiency:
  - With 10 attributes and  $n = 5$  have to learn more than 2000 coefficients (weights)
  - Linear regression (with attribute selection) running time is cubic in the number of attributes
- Overfitting:
  - “Too nonlinear” – number of coefficients large relative to number of training instances
  - Curse of dimensionality applies ...

# Duality (optimization)

- Essentially, an optimization approach has to be used to find the discriminative line
- Duality concept in optimization can help to view the problem from another perspective and make it simpler
- Sometimes solving the dual form is much easier. (Computational advantage)
- We will see, how using *dual form* and *kernel trick* simplify the computations.

# Duality (optimization)

One way of thinking about duality is that when you have an optimization problem, we can construct another optimization problem which is called *dual problem* and is related to our original problem (*primal problem*) and can be useful in solving the primal problem.

In convex optimization problems, the optimal values of the primal and dual problems are equal under a constraint qualification condition.

# Perceptron classifiers in dual form

Every time an example  $x_i$  is misclassified, add  $y_i x_i$  to the weight vector.

- After training has completed, each example has been misclassified zero or more times. Denoting this number as  $\alpha_i$  for example  $x_i$ , the weight vector can be expressed as

$$w = \sum_{i=1}^m \alpha_i y_i x_i$$

- In the dual instance-based view of linear classification we are learning instance weights  $\alpha_i$  rather than feature weights  $w_j$ . An instance  $x$  is classified as

$$\hat{y} = \text{sign} \left( \sum_{i=1}^m \alpha_i y_i (x_i \cdot x) \right)$$

# Perceptron training in dual form

**Algorithm** Dual-Perceptron( $D$ ) / perceptron training in dual form

**Input:** labelled training data  $D$  in homogeneous coordinates

**Output:** coefficients  $\alpha_i$  defining weight vector  $W = \sum_{i=1}^{|D|} \alpha_i y_i x_i$

```
 $\alpha_i \leftarrow 0$ 
converged  $\leftarrow$  false
while converged = false do
    converged  $\leftarrow$  true
    for  $i = 1$  to  $|D|$  do
        if  $y_i \sum_{j=1}^{|D|} \alpha_j y_j x_j \cdot x_i \leq 0$  then
             $\alpha_i \leftarrow \alpha_i + 1$ 
            converged  $\leftarrow$  false
        end
    end
end
```

# Perceptron classifiers in dual form

- Using the dual form of perceptron, we estimate values of  $\alpha_i$  instead of  $w$
- During training, the only information needed about the training data is all pairwise dot products: the  $n$ -by- $n$  matrix  $G = XX^T$  containing these dot products is called the **Gram matrix**.

$$G(x_1, \dots, x_n) = \begin{bmatrix} x_1 \cdot x_1, & x_1 \cdot x_2, & \dots, & x_1 \cdot x_n \\ x_2 \cdot x_1, & x_2 \cdot x_2, & \dots, & x_2 \cdot x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n \cdot x_1, & x_n \cdot x_2, & \dots, & x_n \cdot x_n \end{bmatrix}$$

# Nonlinear dual perceptron

- We can use nonlinear mapping to map attributes into new space consisting of combinations of attribute values

$$x \rightarrow \varphi(x)$$

- Then our perceptron will be:

$$\hat{y} = \text{sign}\left(\sum_{i=1}^m \alpha_i y_i (\varphi(x_i) \cdot \varphi(x))\right)$$

- So the only thing we need is the dot product in the new feature space ( $(\varphi(x_i) \cdot \varphi(x))$  or  $\langle \varphi(x_i), \varphi(x) \rangle$ )
- Why this matters?

# The kernel trick

Let  $x = (x_1, x_2)$  and  $x' = (x'_1, x'_2)$  be two data points, and consider the following mapping to a three-dimensional feature space:

$$(x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

(original feature space)  $\mathcal{X} \rightarrow \mathcal{Z}$  (new feature space)

The points in feature space corresponding to  $x$  and  $x'$  are

$$z = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \text{ and } z' = (x_1'^2, x_2'^2, \sqrt{2}x_1'x_2')$$

The dot product of these two feature vectors is

$$z \cdot z' = x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2x_1 x_1' x_2 x_2' = (x_1 x_1' + x_2 x_2')^2 = (x_1 \cdot x_2)^2$$

# The kernel trick

- By squaring the dot product in the original space we obtain the dot product in the new space without actually constructing the feature vectors! A function that calculates the dot product in feature space directly from the vectors in the original space is called a kernel – here the kernel is  $K(x_1, x_2) = (x_1 \cdot x_2)^2$ .
- In this example order is 2, so the computational gain may not be very obvious, but if we aim for higher orders, for example 20, then we see more clearly the computational advantage

# The kernel trick

- A **valid kernel** function is equivalent to a **dot product in some space**.
- The kernel trick helps to go to a high dimensional space without paying the price (!!), because if we find the right kernel, we do not need to explicitly map the features into the other high dimensional feature space.

# Kernel function

**Example:** if  $x = (x_1, x_2)$  and  $x' = (x'_1, x'_2)$ , is the following function a valid kernel? If so, what is the feature map function?

$$K(x, x') = (1 + x \cdot x')^2$$

$$\begin{aligned} x \cdot x' &= \\ (1 + x \cdot x')^2 &= 1 + x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2x_1 x'_1 + 2x_2 x'_2 + 2x_1 x'_1 x_2 x'_2 \end{aligned}$$

This is the dot product of:

$$(1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2) \quad \text{and} \quad (1, x'^2_1, x'^2_2, \sqrt{2}x'_1, \sqrt{2}x'_2, \sqrt{2}x'_1 x'_2)$$

So this is a valid kernel.

# The kernel trick

- The **kernel trick** avoids the explicit mapping from original feature space ( $\mathcal{X}$ ) to another space ( $\mathcal{Z}$ ).
- For  $\{x, x'\} \in \mathcal{X}$ , certain function  $K(x, x')$  can be expressed as an inner product in another space and  $K$  is called **kernel** or **kernel function**.
- If  $\varphi(x)$  is the input  $x$  in the new (higher dimensional) feature space, then computation becomes much simpler, if we can have kernel function that:

$$K(x, x') = \varphi(x) \cdot \varphi(x')$$

# The kernel trick

- A kernel function is a *similarity* function that corresponds to a dot product in some expanded feature space
- Some very useful kernels in machine learning are **polynomial kernel** and **radial basis function kernel (RBF kernel)**
- Polynomial kernel is defined as:

$$K(x, x') = (x \cdot x' + c)^q$$

- RBF kernel is defined as:

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

Using Taylor expansion, it can be shown that RBF kernel is equivalent of mapping features into infinite dimensions

# Nonlinear dual perceptron

Using kernel trick, the nonlinear perceptron:

$$\hat{y} = \text{sign} \left( \sum_{i=1}^m \alpha_i y_i (\varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x})) \right)$$

can be solved using the dual form and the kernel as follow:

$$\hat{y} = \text{sign} \left( \sum_{i=1}^m \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) \right)$$

The same algorithm as in linear perceptron can be used, but the  $\mathbf{x}_j \cdot \mathbf{x}_i$  has to be replaced with  $K(\mathbf{x}_j, \mathbf{x}_i)$

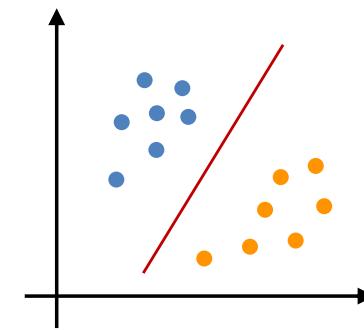
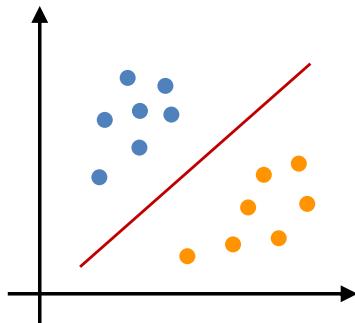
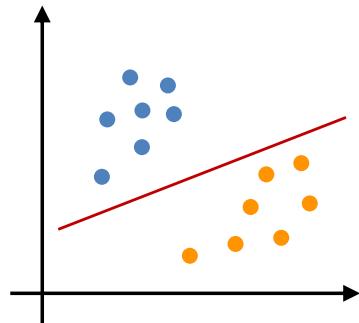
# Issues in linear classification

Different classification learning algorithms use different criteria:

- Basic linear classifier finds class means (centroids), joins them by a straight line, and its perpendicular bisector is the separating hyperplane
- Perceptron training uses iterative reweighting (gradient descent)
- Perceptron may find different models depending on starting conditions

# Issues in linear classification

Which line is a better classifier?

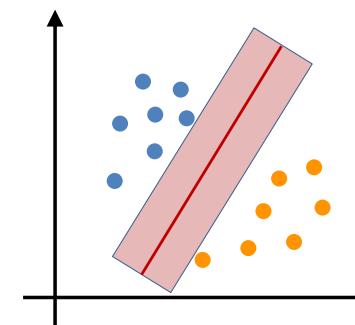
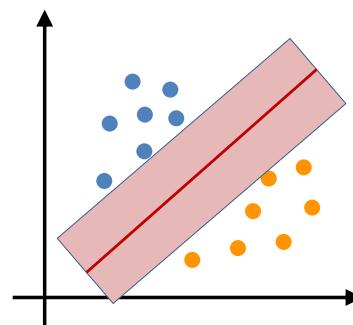
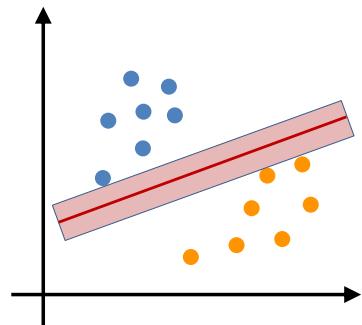


What is the advantage?

# Issues in linear classification

Which line is a better classifier?

- The line with bigger margin



- Why bigger margin is better?
- Can I find a  $w$  that maximizes the margin?

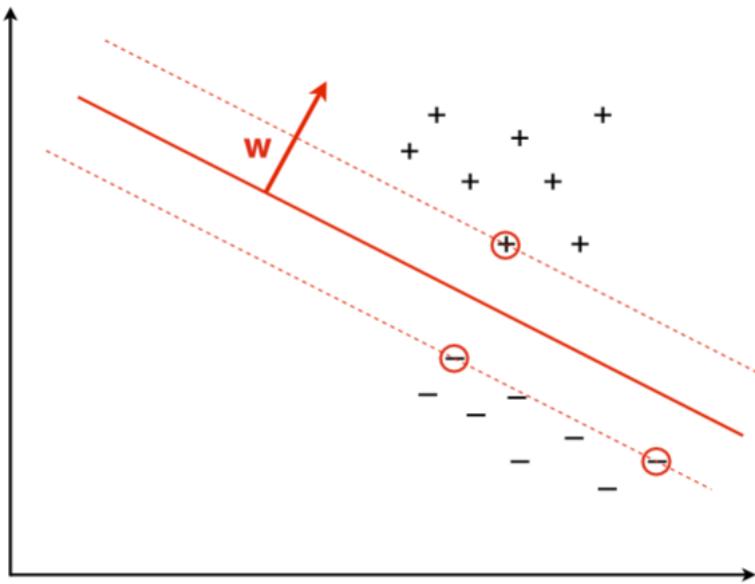
# Issues in linear classification

Is there an optimal linear classification learning method ?

- one approach is to define the empirical risk, or error on the data
- the maximum margin separating hyperplane
- unique solution
- minimizes empirical risk
- is there a way to trade-off minimizing risk with model complexity?
- answer: yes, under Vapnik's framework for statistical learning
  - structural risk minimization

# Support Vector Machine

# Support vector machine

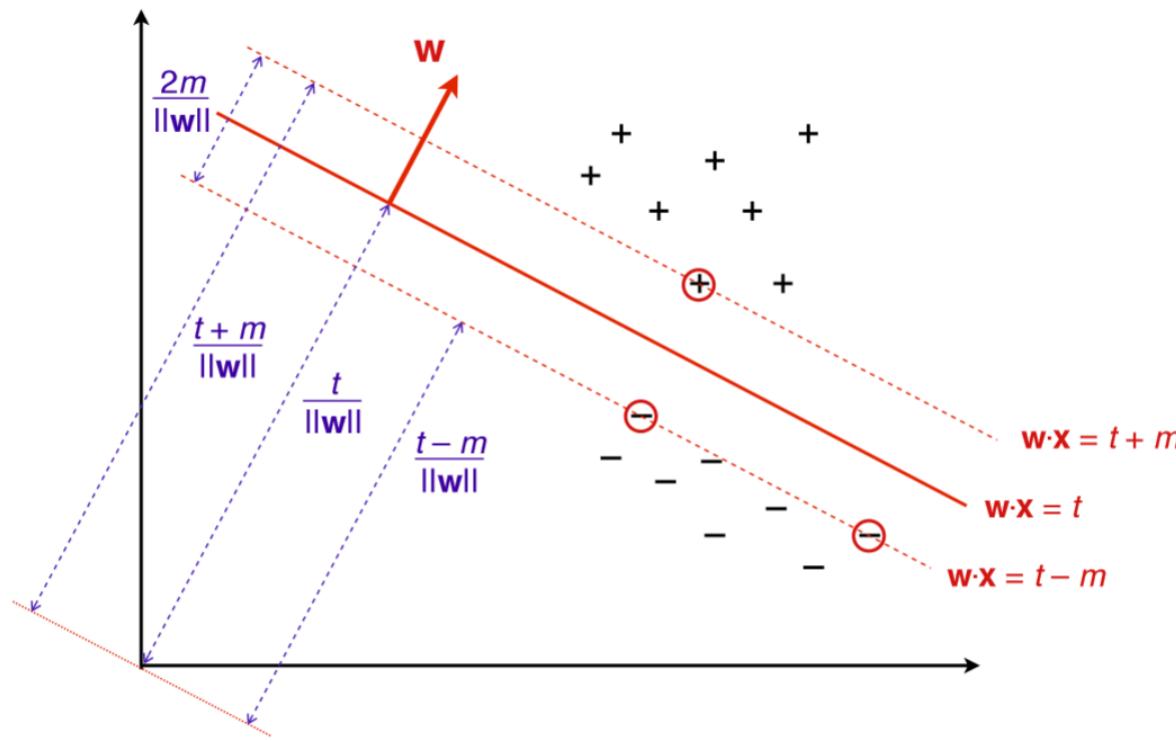


The decision boundary learned by a support vector machine maximizes the margin, which is indicated by the dotted lines. The circled data points are the support vectors.

# Support vector machine

- Support vector machines (machine  $\equiv$  algorithm) learn linear classifiers
- Can avoid overfitting – learn a form of decision boundary called the maximum margin hyperplane
- Fast for mappings to nonlinear spaces
  - employ a mathematical trick (kernel) to avoid the actual creation of new “pseudo-attributes” in transformed instance space
  - i.e. the nonlinear space is created implicitly

# Support vector machine



The geometry of a support vector classifier. The circled data points are the support vectors, which are the training examples nearest to the decision boundary. The support vector machine finds the decision boundary that maximizes the margin  $m/\|w\|$ .

# Training a support vector machine

- Learning problem: fit maximum margin hyperplane, i.e. a kind of linear model
- For a linearly separable two-class data set the maximum margin hyperplane is the classification surface which
  - correctly classifies all examples in the data set
  - has the greatest separation between classes

# Support vector machine

Let's  $x_s$  be the closest point to the separating hyperplane (line in 2D) with the following equation:

$$w \cdot x = t$$

Let's have 2 minor technicalities to simplify the math later:

1. Pull out  $w_0$ :  $w = [w_1, \dots, w_n]$  and  $w_0 = -t$ , therefore we will have:

2. Normalize  $w$ :

We know that  $|w \cdot x_s - t| > 0$  and we know that we can scale  $w$  and  $t$  together without having any effect on the hyperplane, so we choose the scale such that:

$$|w \cdot x_s - t| = 1$$

This means  $m = 1$

# Support vector machine

- We have the line equation which is:  $w \cdot x - t = 0$
- And we have the constraint  $|w \cdot x_s - t| = 1$  where  $x_s$  is the closest point to the separating line (hyperplane)
- $w$  is perpendicular to the line (hyperplane). How?

For every two points  $x'$  and  $x''$  on the line (hyperplane), we can write:

$$\begin{aligned} w \cdot x' - t &= 0 \quad \text{and} \quad w \cdot x'' - t = 0 \\ \Rightarrow \quad w \cdot (x' - x'') &= 0 \end{aligned}$$

When the dot product of two vectors is zero, it means they are perpendicular

# Support vector machine

From geometry, we know that the distance between point  $x_s$  and line (hyperplane)  $w \cdot x - t = 0$  is:

$$\text{distance} = \frac{|w \cdot x_s - t|}{\|w\|}$$

And we have  $|w \cdot x_s - t| = 1$ , so:

$$\text{distance} = \frac{1}{\|w\|}$$

This *distance* is the *margin* of our classifier which we want to maximize:

$$\max \frac{1}{\|w\|} \quad \text{subject to} \quad \min_{i=1,\dots,m} |w \cdot x_i - t| = 1$$

*This is not a friendly optimisation as it has “min” in the constraint!*

# Support vector machine

Our focus is on the points which the line (hyperplane) can predict them correctly. So we can have:

$$|w \cdot x_i - t| = y_i(w \cdot x_i - t)$$

And we can change the “min” in constraint as follow:

$$y_i(w \cdot x_i - t) \geq 1 \quad \text{for } i = 1, \dots, m$$

We can also transform the maximization problem into the following minimization problem:

$$\min \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i(w \cdot x_i - t) \geq 1 \quad \text{for } i = 1, \dots, m$$

$$w \in \mathbb{R}^n, t \in \mathbb{R}$$

How to solve this? **Largangian multipliers**

# Lagrangian multipliers

- Constrained optimization problems are generally expressed as:

$$\min_{x_1, \dots, x_n} f(x_1, \dots, x_n)$$

Subject to:

$$g_1(x_1, \dots, x_n) \leq 0, g_2(x_1, \dots, x_n) \leq 0, \dots, g_k(x_1, \dots, x_n) \leq 0$$

- Lagrange multiplier methods involve the modification of the objective function through the addition of terms that describe the constraints. The objective function  $f(x)$  is augmented by the constraint equations through a set of non-negative multiplicative Lagrange multipliers,  $\alpha_j \geq 0$  and it is called the dual Lagrangian:

$$\mathcal{L}(x_1, \dots, x_n, \alpha_1, \dots, \alpha_m) = f(x_1, \dots, x_n) + \sum_{j=1}^k \alpha_j g_j(x_1, \dots, x_n)$$

# Lagrangian multipliers

In Lagrangian form, the optimization problem becomes:

$$\max_{\alpha_1, \dots, \alpha_m} \min_{x_1, \dots, x_n} \mathcal{L}(x_1, \dots, x_n, \alpha_1, \dots, \alpha_m) \text{ such that } \alpha_j \geq 0 \forall j$$

- first minimizing with respect to  $x_1, \dots, x_n$
- then maximize with respect to  $\alpha_1, \dots, \alpha_m$

# Deriving the dual problem

Adding the constraints with multipliers  $\alpha_i$  for each training example gives the Lagrange function:

$$\begin{aligned}\mathcal{L}(w, t, \alpha_1, \dots, \alpha_m) &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^m \alpha_i (y_i(w \cdot x_i - t) - 1) \\ &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^m \alpha_i y_i (w \cdot x_i) + \sum_{i=1}^m \alpha_i y_i t + \sum_{i=1}^m \alpha_i \\ &= \frac{1}{2} w \cdot w - w \cdot \left( \sum_{i=1}^m \alpha_i y_i x_i \right) + t \left( \sum_{i=1}^m \alpha_i y_i \right) + \sum_{i=1}^m \alpha_i\end{aligned}$$

# Deriving the dual problem

- First we have to minimize  $\mathcal{L}$  with respect to  $w$  and  $t$
- By taking the partial derivative of the Lagrange function with respect to  $t$  and setting it to 0 we find:

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

- Similarly, by taking the partial derivative of the Lagrange function with respect to  $w$  and setting to 0 we obtain:

$$w = \sum_{i=1}^m \alpha_i y_i x_i$$

- the same expression as we derived for the perceptron.

# Deriving the dual problem

- These expressions allow us to eliminate  $w$  and  $t$  and lead to the dual Lagrangian

$$\begin{aligned}\mathcal{L}(\alpha_1, \dots, \alpha_n) &= -\frac{1}{2} \left( \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \right) \cdot \left( \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \right) + \sum_{i=1}^m \alpha_i \\ &= -\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^m \alpha_i\end{aligned}$$

# Deriving the dual problem

- For the perceptron, the instance weights  $\alpha_i$  are non-negative integers denoting the number of times an example has been misclassified in training. For a support vector machine, the  $\alpha_i$  are non-negative reals.
- What they have in common is that, if  $\alpha_i = 0$  for a particular example  $x_i$ , that example could be removed from the training set without affecting the learned decision boundary. In the case of support vector machines this means that  $\alpha_i > 0$  only for the support vectors: the training examples nearest to the decision boundary.

# SVM in dual form

- The dual optimization problem for support vector machines is to maximize the dual Lagrangian under positivity constraints and one equality constraint:

$$\alpha_1^*, \dots, \alpha_m^* = \underset{\alpha_1, \dots, \alpha_n}{\operatorname{argmax}} -\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^m \alpha_i$$

subject to  $\alpha_i > 0, 1 \leq i \leq m, \sum_{i=1}^m \alpha_i y_i = 0$

# Finding support vectors

$$\alpha_1^*, \dots, \alpha_m^* = \underset{\alpha_1, \dots, \alpha_n}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j - \sum_{i=1}^m \alpha_i$$

subject to  $\alpha_i > 0, 1 \leq i \leq m, \sum_{i=1}^m \alpha_i y_i = 0$

- Determining parameters is a constrained quadratic optimization problem
- standard algorithms, or special-purpose algorithms (usually faster, e.g. Platt's sequential minimal optimization (SMO), or LibSVM)

# Training a support vector machine

When you solve this optimization problem and get  $\alpha_1^*, \dots, \alpha_m^*$  back, you will see that it will be mostly zero and only for the points which are the closest to the separating line,  $\alpha_i$  will be non-zero and positive. Those points are called the support vectors.

$$w = \sum_{x_i \in \{support\,vectors\}} \alpha_i y_i x_i$$

Solve for  $t$  using any of support vectors:

$$y_i(w \cdot x_i - t) = 1$$

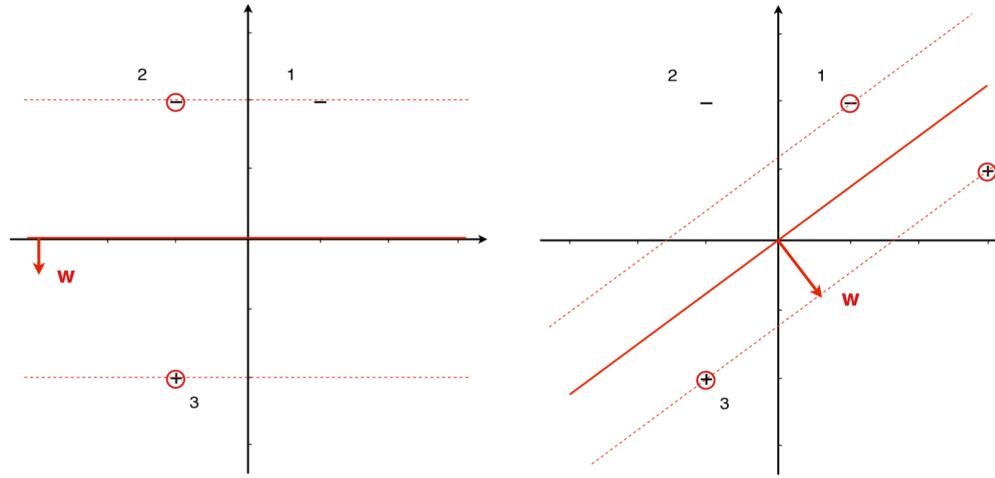
It can be shown that error is bounded by number of support vectors, irrespective of dimensionality

Note: all this assumes separable data!

# Training a support vector machine

- The more “separated” the classes, the larger the margin, the better the generalization
- Instances closest to maximum margin hyperplane are support vectors
- Important observation: support vectors define maximum margin hyperplane!
  - All other instances can be deleted without changing position and orientation of the hyperplane!

# Two maximum-margin classifiers



(left) A maximum-margin classifier built from three examples, with  $w = (0, -1/2)$  and margin 2. The circled examples are the support vectors: they receive non-zero Lagrange multipliers and define the decision boundary. (right) By adding a second positive the decision boundary is rotated to  $w = (3/5, -4/5)$  and the margin decreases to 1.

# Two maximum-margin classifiers

$$X = \begin{pmatrix} 1 & 2 \\ -1 & 2 \\ -1 & -2 \end{pmatrix} \quad y = \begin{pmatrix} -1 \\ -1 \\ +1 \end{pmatrix} \quad X' = \begin{pmatrix} -1 & -2 \\ 1 & -2 \\ -1 & -2 \end{pmatrix}$$

The matrix  $X'$  on the right incorporates the class labels; i.e., the rows are  $y_i x_i$ . The Gram matrix is (without and with class labels):

$$XX^T = \begin{pmatrix} 5 & 3 & -5 \\ 3 & 5 & -3 \\ -5 & -3 & 5 \end{pmatrix} \quad X'X'^T = \begin{pmatrix} 5 & 3 & 5 \\ 3 & 5 & 3 \\ 5 & 3 & 5 \end{pmatrix}$$

# Two maximum-margin classifiers

The dual optimization problem is thus

$$\begin{aligned} & \underset{\alpha_1, \alpha_2, \alpha_3}{\operatorname{argmax}} -\frac{1}{2}(5\alpha_1^2 + 3\alpha_1\alpha_2 + 5\alpha_1\alpha_3 + 3\alpha_2\alpha_1 + 5\alpha_2^2 + 3\alpha_2\alpha_3 + 5\alpha_3\alpha_1 + 3\alpha_3\alpha_2 \\ & + 5\alpha_3^2) + \alpha_1 + \alpha_2 + \alpha_3 \end{aligned}$$

$$= \underset{\alpha_1, \alpha_2, \alpha_3}{\operatorname{argmax}} -\frac{1}{2}(5\alpha_1^2 + 6\alpha_1\alpha_2 + 10\alpha_1\alpha_3 + 5\alpha_2^2 + 36\alpha_3 + 5\alpha_3^2) + \alpha_1 + \alpha_2 + \alpha_3$$

subject to  $\alpha_1 \geq 0, \alpha_2 \geq 0, \alpha_3 \geq 0$  and  $-\alpha_1 - \alpha_2 + \alpha_3 = 0$ .

# Two maximum-margin classifiers

- Using the equality constraint we can eliminate one of the variables, say  $\alpha_3$ , and simplify the objective function to

$$\underset{\alpha_1, \alpha_2}{\operatorname{argmax}} -\frac{1}{2}(20\alpha_1^2 + 32\alpha_1\alpha_2 + 16\alpha_2^2) + 2\alpha_1 + 2\alpha_2$$

- Setting partial derivatives to 0 we obtain  $-20\alpha_1 - 16\alpha_2 + 2 = 0$  and  $-16\alpha_1 - 16\alpha_2 + 2 = 0$  (notice that, because the objective function is quadratic, these equations are guaranteed to be linear).
- We therefore obtain the solution  $\alpha_1 = 0$  and  $\alpha_2 = \alpha_3 = 1/8$ . We then have  $w = 1/8(x_3 - x_2) = \begin{pmatrix} 0 \\ -1/2 \end{pmatrix}$ , resulting in a margin of  $1/\|w\| = 2$ .
- Finally,  $t$  can be obtained from any support vector, say  $x_2$ , since  $y_2(w \cdot x_2 - t) = 1$ ; this gives  $-1 \cdot (-1 - t) = 1$ , hence  $t = 0$ .

# Two maximum-margin classifiers

- We now add an additional positive at (3,1). This gives the following data matrices:

$$X' = \begin{pmatrix} -1 & -2 \\ 1 & -2 \\ -1 & -2 \\ 3 & 1 \end{pmatrix} \quad X'X'^T = \begin{pmatrix} 5 & 3 & 5 & -5 \\ 3 & 5 & 3 & 1 \\ 5 & 3 & 5 & -5 \\ -5 & 1 & -5 & 10 \end{pmatrix}$$

- It can be verified by similar calculations to those above that the margin decreases to 1 and the decision boundary rotates to

$$w = \begin{pmatrix} 3/5 \\ -4/5 \end{pmatrix}.$$

- The Lagrange multipliers now are  $\alpha_1 = 1/2$ ,  $\alpha_2 = 0$ ,  $\alpha_3 = 1/10$  and  $\alpha_4 = 2/5$ . Thus, only  $x_3$  is a support vector in both the original and the extended data set.

# Nonlinear SVMs

Similar to before, we can transform our input feature  $x \in \mathcal{X}$  into a new feature space  $z \in \mathcal{Z}$

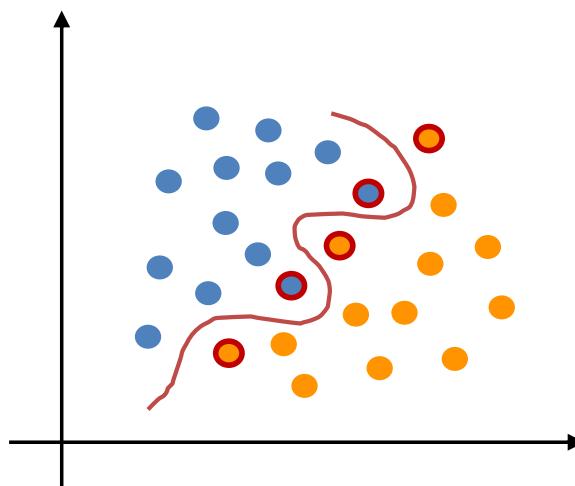
We can use the same Lagrangian function by replacing  $x$ :

$$\mathcal{L}(\alpha) = -\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j z_i \cdot z_j + \sum_{i=1}^m \alpha_i$$

- The number of  $\alpha_i$  is still the same even if your new feature space has much higher dimensionality
- As long as we provide the  $z_i \cdot z_j$ , the complexity of the optimization problem is going to be the same

# Nonlinear SVMs

- Note that your support vectors are defined in the new feature space ( $\mathcal{Z}$ )
- The margin is maintained in the  $\mathcal{Z}$  space
- They can be recognized by finding  $\alpha_i$ 's which are not zero
- However, they correspond to some of the points in the original space.



# Nonlinear SVMs with Kernel trick

If  $z = \varphi(x)$ , Lagrangian becomes:

$$\mathcal{L}(\alpha) = -\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \varphi(x_i) \cdot \varphi(x_j) + \sum_{i=1}^m \alpha_i$$

If we can find a kernel function, such that:

$$K(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j)$$

(Kernel corresponds to a map into a new feature space)

Then:

$$\mathcal{L}(\alpha) = -\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j K(x_i, x_j) + \sum_{i=1}^m \alpha_i$$

# Nonlinear SVMs

The output of your support vector machine for  $x$  is:

$$\hat{y} = \text{sign}(w \cdot z - t)$$

$$w = \sum_{z_i \in SV} \alpha_i y_i z_i$$

$$\Rightarrow \hat{y} = \text{sign}\left(\sum_{\alpha_i > 0} \alpha_i y_i K(x_i, x) - t\right)$$

Where:

$$t = \sum_{\alpha_i > 0} \alpha_i y_i K(x_i, x_j) - y_j \text{ for any support vector } x_j$$

So this is your model. As you see it is not just one model and it can change depending on the kernel that you choose.

# Nonlinear SVMs

$$\hat{y} = \text{sign}\left(\sum_{\alpha_i > 0} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) - t\right)$$

- This will let you to compare the results from different Kernels
- You can test any valid kernel and compare the results
- Using the Kernel trick we do not need to do the transformation from original feature space to the new feature space
- Possibility of finding the support vector in high dimensional space without paying the computational price of going to that new space
- Support vector machines are a popular choice as a kernel method, since they naturally promote sparsity in the support vectors.

# Kernel trick

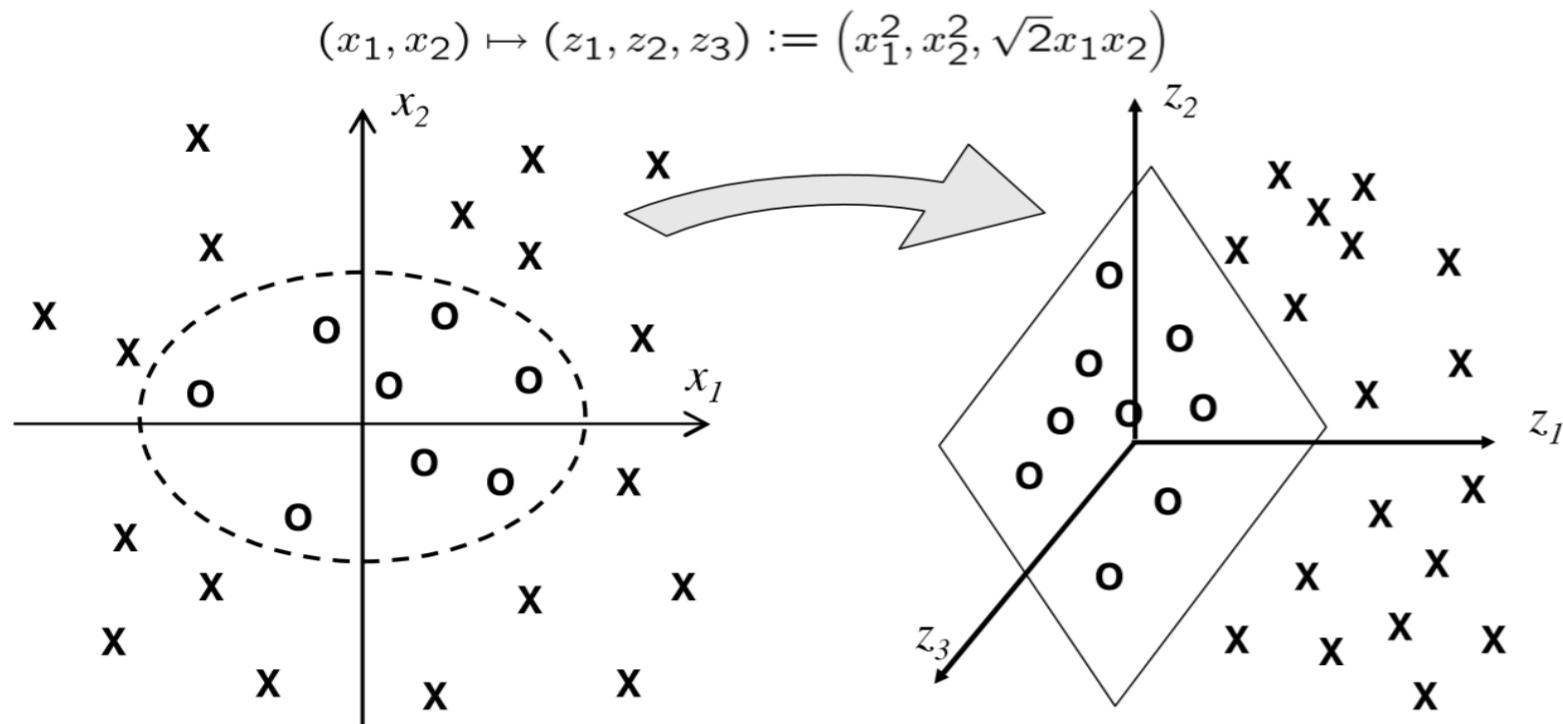
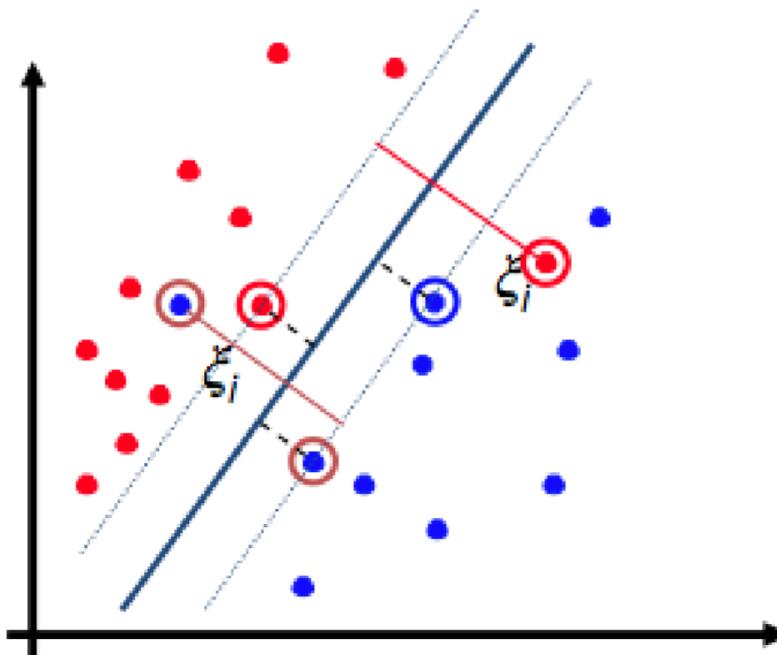


Figure by Avrim Blum, CS Dept, CMU.

# Noise

- Misclassified examples may break the separability assumption



- Introduce “slack” variables  $\xi_i$  to allow misclassification of instances

# Soft margin SVM

When classes were linearly separable, we had:

$$y_i(w \cdot x_i - t) \geq 1$$

But if we get some data that violate this slack value:

$$y_i(w \cdot x_i - t) \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

So, the total violation:

$$\text{total violation} = \sum_{i=1}^m \xi_i$$

This is a measure of violation of margin and now, we optimize for:

$$\min \frac{1}{2} \|w\|^2 + C \sum \xi_i$$

# Soft margin SVM

- “soft margin” SVMs to handle noisy data
- Parameter  $C$  bounds influence of any one training instance on decision boundary
- If  $C$  goes to infinity you go back to the hard margin SVM (doesn’t tolerate error)
- Still a quadratic optimization problem

# Allowing margin errors

- The idea is to introduce slack variables  $\xi_i$ , one for each example, which allow some of them to be inside the margin or even at the wrong side of the decision boundary. We will call these margin errors.

$$w^*, t^*, \xi^* = \underset{w, t, \xi_i}{\operatorname{argmin}} \frac{1}{2} \| w \|^2 + C \sum_{i=1}^m \xi_i$$

subject to  $y_i(w \cdot x_i - t) \geq 1 - \xi_i$  and  $\xi_i \geq 0, 1 \leq i \leq m$ .

# Allowing margin errors

- $C$  is a user-defined parameter trading off margin maximization against slack variable minimization: a high value of  $C$  means that margin errors incur a high penalty, while a low value permits more margin errors (possibly including misclassifications) in order to achieve a large margin.
- If we allow more margin errors we need fewer support vectors, hence  $C$  controls to some extent the ‘complexity’ of the SVM and hence is often referred to as the *complexity* parameter.

# Allowing margin errors

The Lagrange function is then as follows:

$$\begin{aligned}\mathcal{L}(w, t, \xi_i, \alpha_i, \beta_i) &= \frac{1}{2} \| w \|^2 + C \sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i (y_i(w \cdot x_i - t) - (1 - \xi_i)) - \sum_{i=1}^m \beta_i \xi_i \\ &= \mathcal{L}(w, t, \alpha_i) + \sum_{i=1}^m (C - \alpha_i - \beta_i) \xi_i\end{aligned}$$

Similar to before, we have to first minimize this with respect to  $w$ ,  $t$  and  $\xi_i$

# Allowing margin errors

$$\nabla_w \mathcal{L} = w - \sum_{i=1}^m \alpha_i y_i x_i = 0$$

$$\frac{\partial \mathcal{L}}{\partial t} = \sum_{i=1}^m \alpha_i y_i = 0$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \Rightarrow C - \alpha_i = \beta_i \geq 0$$

$$0 \leq \alpha_i \leq C$$

The only difference that we get compared to hard margin SVM is that  $\alpha_i$  is upper bounded by the value of  $C$

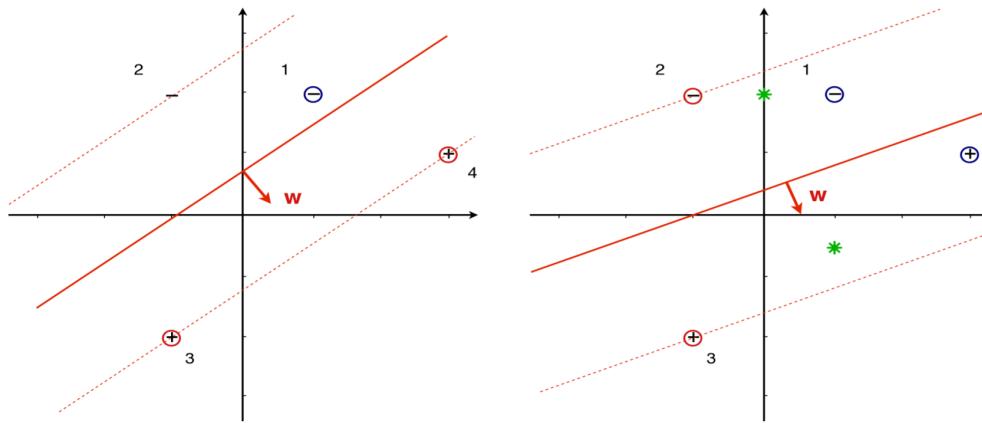
# Allowing margin errors

- For an optimal solution every partial derivative with respect to  $\xi_i$  should be 0, from which it follows that the added term vanishes from the dual problem.
- Furthermore, since both  $\alpha_i$  and  $\beta_i$  are positive, this means that  $\alpha_i$  cannot be larger than  $C$
- The next step is to optimize the Lagrangian using quadratic programming similar to before and get estimates for  $\alpha_i$

# Three cases for the training instances

- a solution to the soft margin optimization problem in dual form divides the training examples into three cases:
  - $\alpha_i = 0$  these are outside or on the margin;
  - $0 < \alpha_i < C$  these are the support vectors on the margin;
  - $\alpha_i = C$  these are on or inside the margin (called non-margin support vectors). All the points that violate the margin are non-margin support vectors and contribute to  $w$ .
- Notice that we still have  $w = \sum_{i=1}^n \alpha_i y_i x_i$ , and so both second and third case examples participate in spanning the decision boundary.
- The value of  $C$  controls the amount of violation and has to be found by, e.g., cross-validation

# Soft margins



(left) The soft margin classifier learned with  $C = 5/16$ , at which point  $x_2$  is about to become a support vector. (right) The soft margin classifier learned with  $C = 1/10$ : all examples contribute equally to the weight vector. The asterisks denote the class means, and the decision boundary is parallel to the one learned by the basic linear classifier.

# Soft margins

- Recall that the Lagrange multipliers for the classifier above are  $\alpha_1 = 1/2$ ,  $\alpha_2 = 0$ ,  $\alpha_3 = 1/10$ ,  $\alpha_4 = 2/5$ . So  $\alpha_1$  is the largest multiplier, and as long as  $C > \alpha_1 = 1/2$  no margin errors are tolerated.
- For  $C = 1/2$  we have  $\alpha_1 = C$ , and hence for  $C < 1/2$  we have that  $x_1$  becomes a margin error and the optimal classifier is a soft margin classifier. Effectively, with decreasing  $C$  the decision boundary and the upper margin move upward, while the lower margin stays the same.
- The upper margin reaches  $x_2$  or  $C = 5/16$ , at which point we have  $w = \begin{pmatrix} 3/8 \\ -1/2 \end{pmatrix}$  and the margin has increased to 1.6. Furthermore, we have  $\xi_1 = 6/8$ ,  $\alpha_1 = C = 5/16$ ,  $\alpha_2 = 0$ ,  $\alpha_3 = 1/16$  and  $\alpha_4 = 1/4$ .

# Soft margins

- If we now decrease  $C$  further, the decision boundary starts to rotate clockwise, so that  $x_4$  becomes a margin error as well, and only  $x_2$  and  $x_3$  are support vectors. The boundary rotates until  $C = 1/10$ , at which point we have  $w = \begin{pmatrix} 1/5 \\ -1/2 \end{pmatrix}$ ,  $t = 1/5$  and the margin has increased to 1.86. Furthermore, we have  $\xi_1 = 4/10$  and  $\xi_4 = 7/10$ , and all multipliers have become equal to  $C$ .
- Finally, when  $C$  decreases further the decision boundary stays where it is, but the norm of the weight vector gradually decreases and all points become margin errors.

# Soft margins

NOTE: a minimal-complexity soft margin classifier summarizes the classes by their class means in a way very similar to the basic linear classifier.

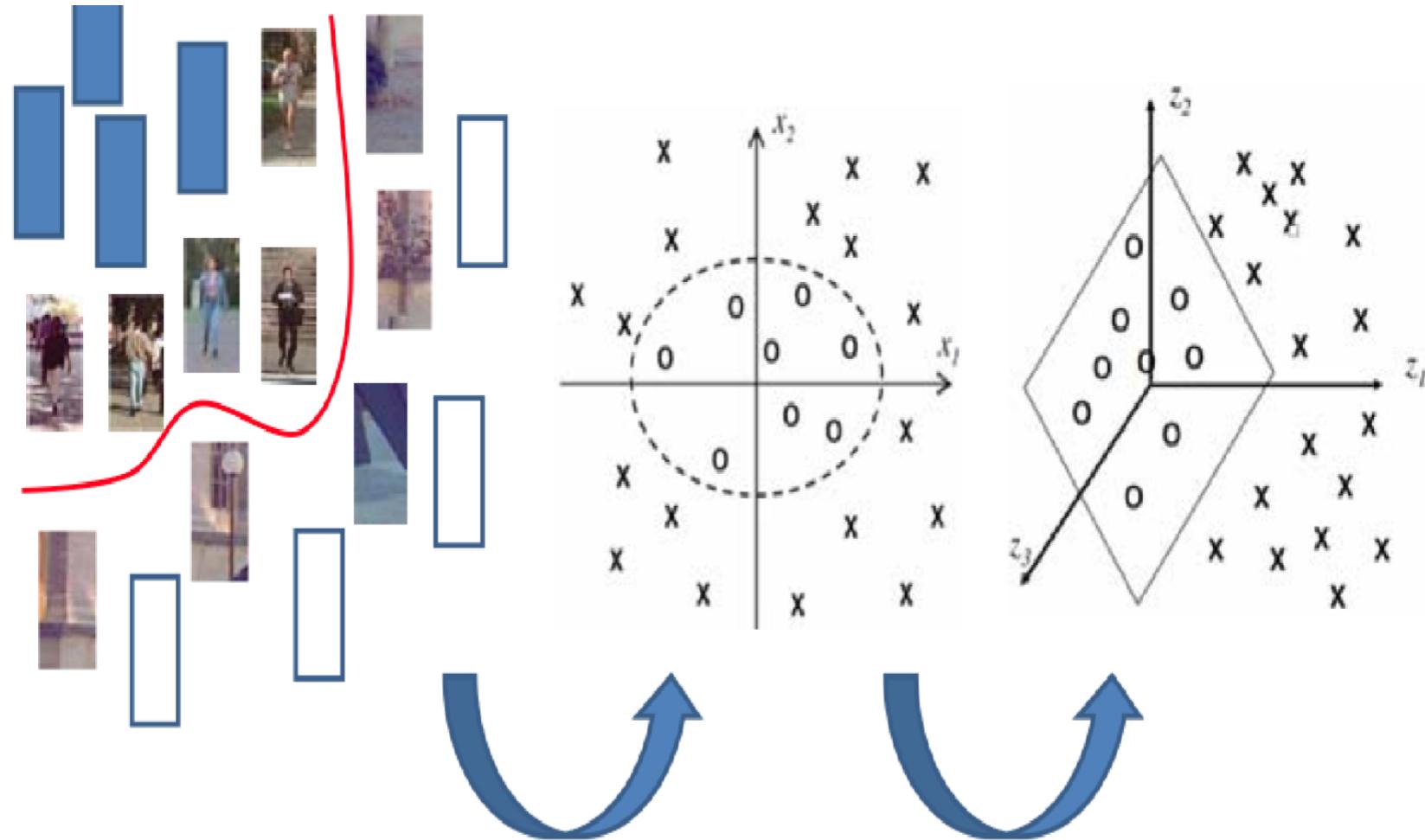
# Sparse data

- SVM algorithms can be sped up dramatically if the data is sparse (i.e. many values are 0)
- Why? Because they compute lots and lots of dot products
- With sparse data dot products can be computed very efficiently • Just need to iterate over the values that are non-zero
- SVMs can process sparse datasets with tens of thousands of attributes

# SVM Applications

- Machine vision: e.g face identification
  - Prior to deep learning, achieves lowest error
- Handwritten digit recognition:
  - Comparable to best alternative
- Bioinformatics: e.g. prediction of protein secondary structure, microarray classification
- Text classification
- Algorithm can be modified to deal with numeric prediction problems
  - support vector regression

# Example - pedestrian detection



# Summary: Learning with Kernel Methods

- Kernel methods around for a long time in statistics
- Kernelization a “modular” approach to machine learning
- Algorithms that can be kernelized can learn different model classes simply by changing the kernel
- SVMs exemplify this – mostly for classification (but also regression, “one-class” classification, etc.)
- SVMs one of the most widely used “off-the-shelf” classifier learning methods, especially for “small  $m$  (examples), large  $n$  (dimensionality)” classification problems

# Acknowledgements

- “Elements of Statistical Learning (2nd Ed.)” by T. Hastie, R. Tibshirani & J. Friedman. Springer (2009) <http://statweb.stanford.edu/~tibs/ElemStatLearn/>
- Material derived from slides for the book “Machine Learning: A Probabilistic Perspective” by P. Murphy MIT Press (2012) <http://www.cs.ubc.ca/~murphyk/MLbook>
- Material derived from slides for the book “Machine Learning” by P. Flach Cambridge University Press (2012) <http://cs.bris.ac.uk/~flach/mlbook>
- Material derived from slides for the book “Bayesian Reasoning and Machine Learning” by D. Barber Cambridge University Press (2012) <http://www.cs.ucl.ac.uk/staff/d.barber/brml>
- Material derived from slides for the book “Machine Learning” by T. Mitchell McGraw-Hill (1997) <http://www- 2.cs.cmu.edu/~tom/mlbook.html>
- Material derived from slides for the course “Machine Learning” by A. Srinivasan BITS Pilani Goa Campus, India (2016)