Lecture 11
SVM cont.

10-22 2008

What we have done so far

- We have established that we want to find a linear decision boundary whose margin is the largest
- We know how to measure the margin of a linear decision boundary
  - That is: the minimum geometric margin of all training examples

    Geometric margin of a training example = functional margin normalized by the magnitude of **w**

Functional margin

$$\gamma^i = \frac{y^i(\mathbf{w} \cdot \mathbf{x}^i + b)}{\|\mathbf{w}\|}$$

- How do we find such a linear decision boundary that has the largest margin?

Maximum Margin Classifier

- This can be formulated as a constrained optimization problem.

$$\max_{\mathbf{w},b} \gamma$$

$$\text{subject to: } \; y^{(i)} \frac{(\mathbf{w}\cdot\mathbf{x}^{(i)}+b)}{\|\mathbf{w}\|} \geq \gamma, \quad i=1,\cdots,N$$

- This optimization problem is in a nasty form (quadratic constraints), so we need to do some rewriting
- Eventually we will get the following:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{subject to: } \; y^{i}(\mathbf{w}\cdot\mathbf{x}^{i}+b) \geq 1, \quad i=1,\cdots,N$$

Maximizing the geometric margin is equivalent to minimizing the magnitude of **w** subject to maintaining a functional margin of at least 1

Solving the Optimization Problem

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$
$$\text{subject to: } y^i(\mathbf{w}\cdot\mathbf{x}^i + b) \geq 1, \quad i = 1,\cdots,N$$

- This is a **quadratic programming problem**, i.e., optimizing a quadratic fnt with linear inequality constraints.
- This is a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.
  - In practice, we can just regard the QP solver as a “black-box” without bothering how it works
- You will be spared of the excruciating details and jump to …

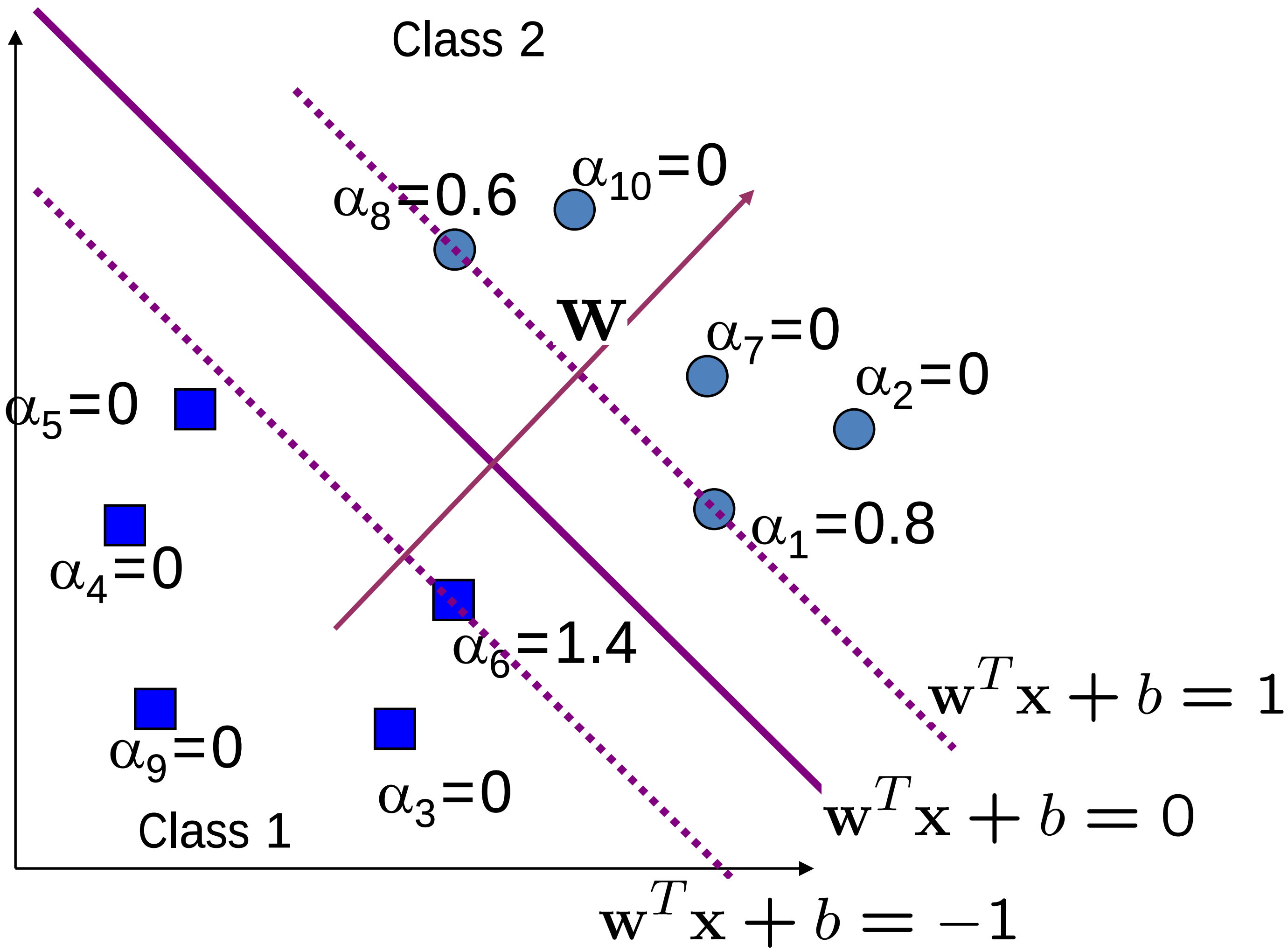
The solution

- Hold on a sec, we can not really give you a close form solution that you can directly plug in the numbers and compute for an arbitrary data sets
- But, the crystal ball tells us that the solution can always be written in the following form:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y^i x^i, \text{ s.t.} \sum_{i=1}^{N} \alpha_i y^i = 0$$

- This is the form of the solution for **w**, b can be calculated accordingly using some additional steps
- The weight vector is a linear combination of all the training examples
- Importantly, many of the α_i's are zeros
- These that have non-zero α_i's are called the **support vectors**

An example


Class 2
α_{10}=0
α_8=0.6
$\mathbf{w}$
α_7=0
α_2=0
α_5=0
α_1=0.8
α_4=0
α_6=1.4
$\mathbf{w}^T\mathbf{x} + b = 1$
α_9=0
α_3=0
$\mathbf{w}^T\mathbf{x} + b = 0$
Class 1
$\mathbf{w}^T\mathbf{x} + b = -1$

A few important notes regarding the geometric interpretation

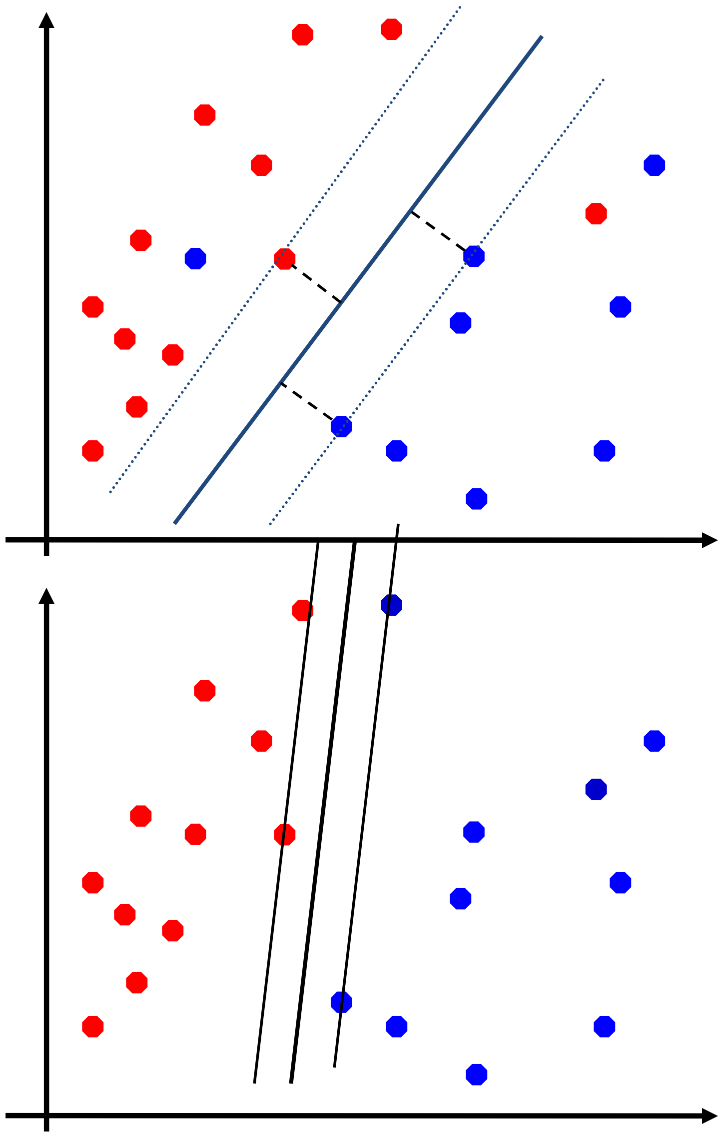
- $\mathbf{w}^T\mathbf{x} + b = 0$ gives the decision boundary
- $\mathbf{w}^T\mathbf{x} + b = 1$ positive support vectors lie on this line
- $\mathbf{w}^T\mathbf{x} + b = -1$ negative support vectors lie on this line
- All support vectors have functional margin of 1
- We can think of a decision boundary now as a tube of certain width, no points can be inside the tube
  - Learning involves adjusting the location and orientation of the tube to find the largest fitting tube for the given training set

Summarization So Far

- We defined margin (functional, geometric)
- We demonstrated that we prefer to have linear classifiers with large geometric margin.
- We formulated the problem of finding the maximum margin linear classifier as a quadratic optimization problem
- This problem can be solved using efficient QP algorithms that are available.
- The solutions are very nicely formed
- Do we have our perfect classifier yet?

Non-separable Data and Noise

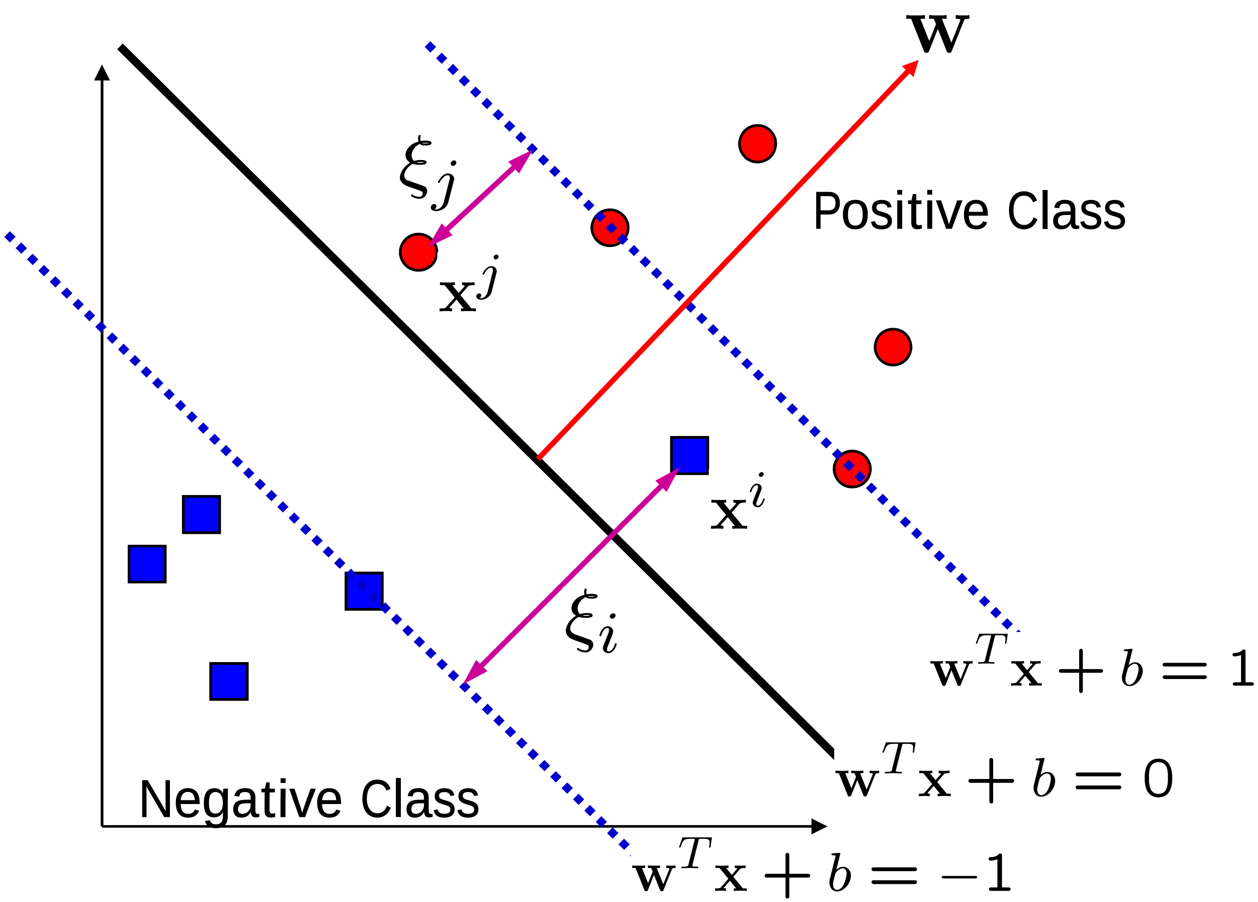
- What if the data is not linearly separable?



- We may have noise in data, and maximum margin classifier is not robust to noise!

Soft Margin

- Allow functional margins to be less than 1



Originally functional margins need to satisfy:

$$y^i(w \cdot x^i + b) \geq 1$$

Now we allow it to be less than 1:

$$y^i(w \cdot x^i + b) \geq 1 - \xi_i$$

The objective ftn also change to:

$$\min_{\mathbf{w},b} \|\mathbf{w}\|^2 + c\sum_{i=1}^{N} \xi_i$$

Soft-Margin Maximization

$$\min_{\mathbf{w},b} \|\mathbf{w}\|^2$$
$$\text{subject to}: \ y^i(\mathbf{w}\cdot\mathbf{x}^i + b) \geq 1, \quad i = 1,\cdots,N$$

$$\min_{\mathbf{w},b} \|\mathbf{w}\|^2 + c\sum_{i=1}^{N}\xi_i$$
$$\text{subject to}: \ y^i(\mathbf{w}\cdot\mathbf{x}^i + b) \geq 1-\xi_i, \quad i = 1,\cdots,N$$
$$\xi_i \geq 0, \quad i = 1,\cdots,N$$

- Introduce **slack variables** ξ_i to allow some examples to have functional margins smaller than 1
- Effect of parameter ***c***
  - Controls the tradeoff between maximizing the margin and fitting the training examples
  - Large c: slack variables incur large penalty, so the optimal solution will try to avoid them
  - Small c: small cost for slack variables, we can sacrifice a few training examples to ensure that the classifier margin is large

Solutions to SVM

$$\boxed{w = \sum_{i=1}^{N} \alpha_i y^i x^i, \quad \textbf{s.t.} \sum_{i=1}^{N} \alpha_i y^i = 0}$$

No soft margin

$$\boxed{w = \sum_{i=1}^{N} \alpha_i y^i x^i, \quad \textbf{s.t.} \sum_{i=1}^{N} \alpha_i y^i = 0 \textbf{ and } 0 \le \alpha_i \le c}$$

With soft margin

- *c* controls the tradeoff between maximizing margin and fitting training data
- It's effect is to put a **box constraint** on α, the weights of the support vectors
- It limits the influence of individual support vectors (maybe outliers)
- In practice, c can be set by cross-validation

How to make predictions?

For classifying with a new input **z**

Compute

$$w \cdot z + b = (\sum_{j=1}^{s} \alpha_{t_j} y^{t_j} x^{t_j}) \cdot z + b = \sum_{j=1}^{s} \alpha_{t_j} y^{t_j} (x^{t_j} \cdot z) + b$$

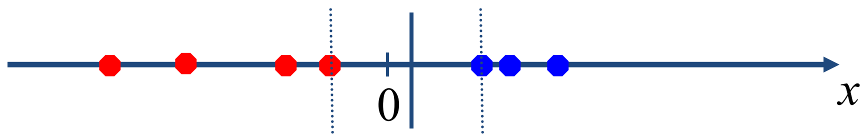
classify **z** as **+** if positive, and **-** otherwise

Note: **w** need not be formed explicitly, we can classify z by taking inner products with the support vectors

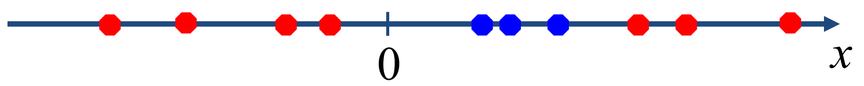
Further, the learning of w and the prediction using w both can be achieved using inner product between pair of input points – this lends itself naturally to handling cases that are not linearly separable by replacing the inner product with something that is called kernel function.

Non-linear SVMs

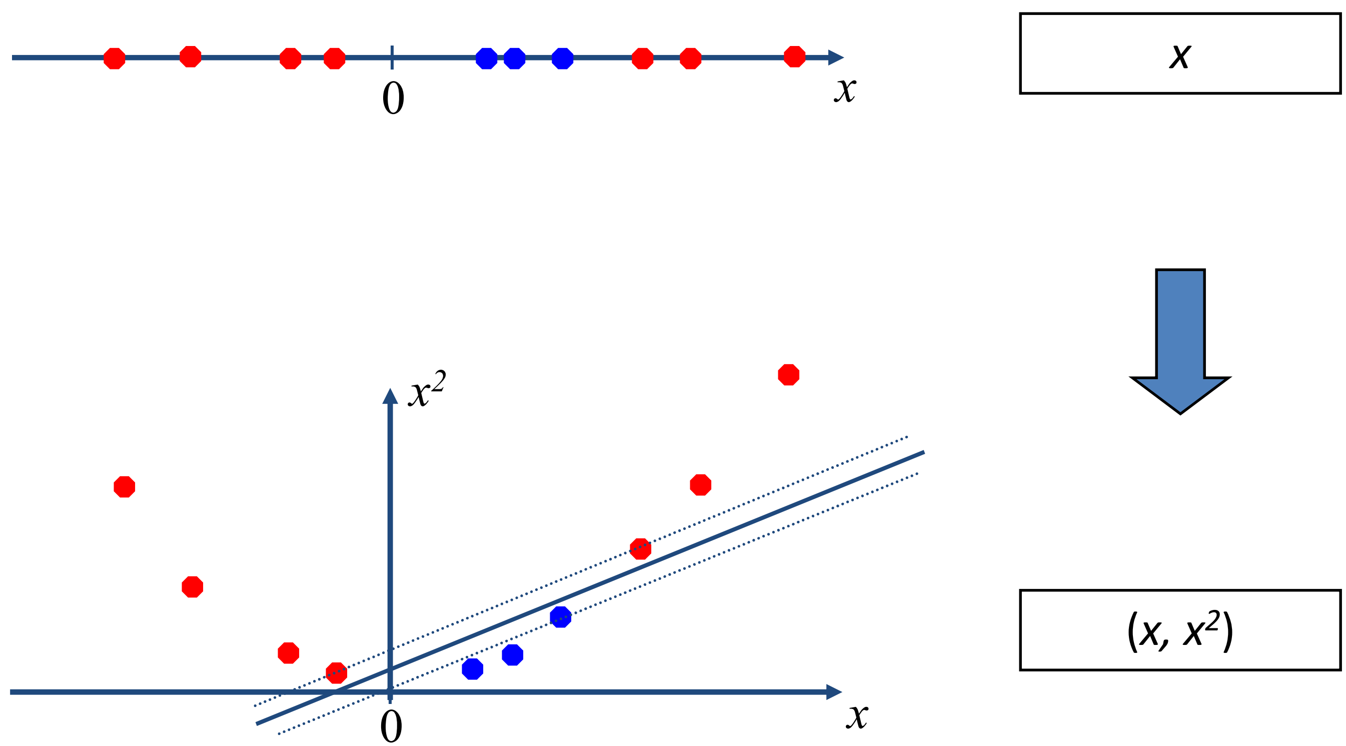
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?

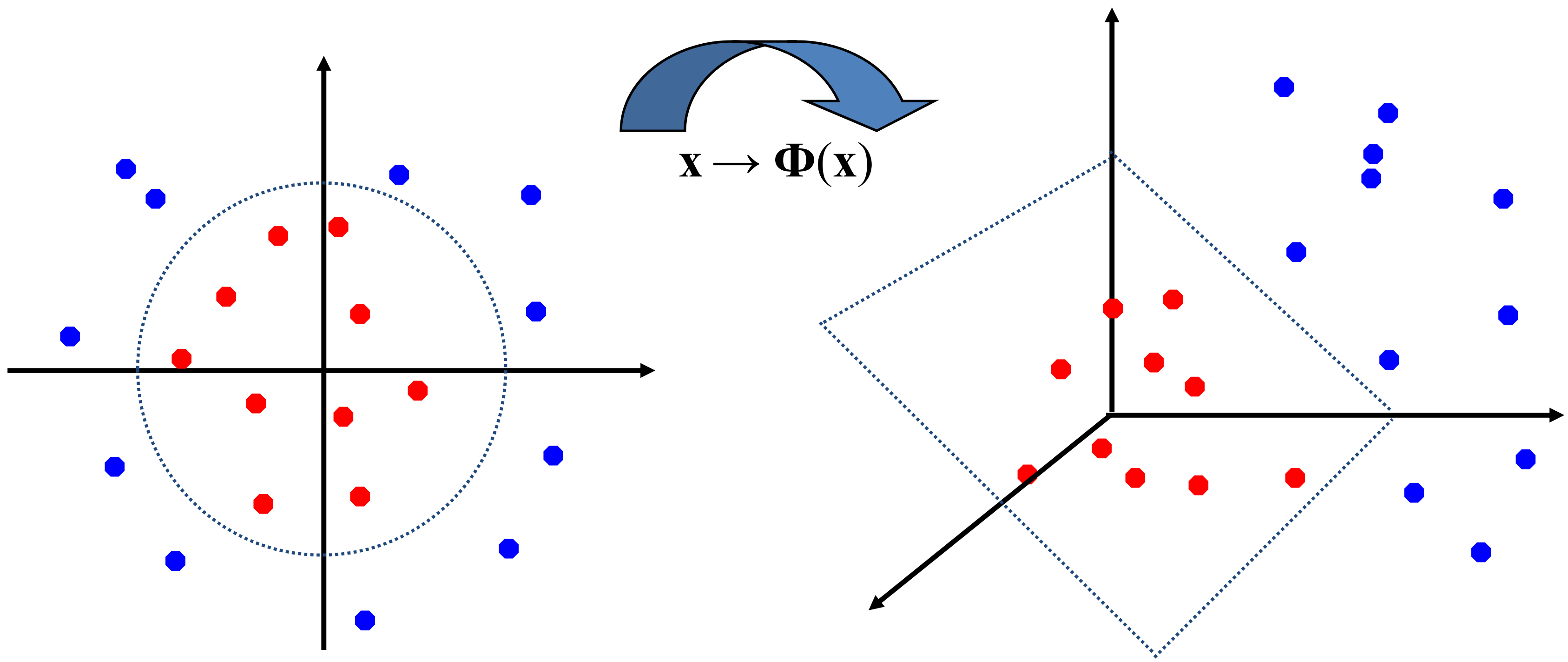


Mapping the input to a higher dimensional space can solve the linearly inseparable cases



Non-linear SVMs: Feature Spaces

- General idea: For <u>any</u> data set, the original input space can always be mapped to some higher-dimensional feature space such that the data is linearly separable:



Example: Quadratic Feature Space

- Assume *m* input dimensions

  $\mathbf{x} = (x_1, x_2, \cdots, x_m)$

- Number of quadratic terms:

  $1 + m + m + m(m-1)/2 \approx m^2$

- The number of dimensions increase rapidly!

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \vdots \\ \sqrt{2}x_m \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_m^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \sqrt{2}x_2x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \vdots \\ \sqrt{2}x_{m-1}x_m \end{pmatrix} \begin{array}{l} \}\ \text{Constant Term} \\ \}\ \text{Linear Terms} \\ \}\ \text{Pure Quadratic Terms} \\ \}\ \text{Quadratic Cross-Terms} \end{array}$$

You may be wondering about the $\sqrt{2}$
At least they won't hurt anything!

You will find out why they are there soon!

Dot product in quadratic feature space

$$\Phi(\mathrm{a})\cdot\Phi(\mathrm{b}) = \begin{pmatrix} 1 \\ \sqrt{2}a_1 \\ \sqrt{2}a_2 \\ : \\ \sqrt{2}a_m \\ a_1^2 \\ a_2^2 \\ : \\ a_m^2 \\ \sqrt{2}a_1a_2 \\ \sqrt{2}a_1a_3 \\ : \\ \sqrt{2}a_1a_m \\ \sqrt{2}a_2a_3 \\ : \\ \sqrt{2}a_1a_m \\ : \\ \sqrt{2}a_{m-1}a_m \end{pmatrix} \bullet \begin{pmatrix} 1 \\ \sqrt{2}b_1 \\ \sqrt{2}b_2 \\ : \\ \sqrt{2}b_m \\ b_1^2 \\ b_2^2 \\ : \\ b_m^2 \\ \sqrt{2}b_1b_2 \\ \sqrt{2}b_1b_3 \\ : \\ \sqrt{2}b_1b_m \\ \sqrt{2}b_2b_3 \\ : \\ \sqrt{2}b_1b_m \\ : \\ \sqrt{2}b_{m-1}b_m \end{pmatrix}$$

$$1$$

$$+$$

$$\sum_{i=1}^{m} 2a_ib_i$$

$$+$$

$$\sum_{i=1}^{m} a_i^2 b_i^2$$

$$+$$

$$\sum_{i=1}^{m}\sum_{j=i+1}^{m} 2a_ia_jb_ib_j$$

$$\mathbf{\Phi}(\mathbf{a}) \bullet \mathbf{\Phi}(\mathbf{b}) =$$
$$1 + 2\sum_{i=1}^{m} a_ib_i + \sum_{i=1}^{m} a_i^2b_i^2 + \sum_{i=1}^{m}\sum_{j=i+1}^{m} 2a_ia_jb_ib_j$$

Now let's just look at another interesting function of (a·b):

$$
\begin{aligned}
& (a \cdot b + 1)^2 \\
&= (a \cdot b)^2 + 2(a \cdot b) + 1 \\
&= (\sum_{i=1}^{m} a_ib_i)^2 + 2\sum_{i=1}^{m} a_ib_i + 1 \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m} a_ia_jb_ib_j + 2\sum_{i=1}^{m} a_ib_i + 1 \\
&= \sum_{i=1}^{m} a_i^{\,2}b_i^{\,2} + 2\sum_{i=1}^{m}\sum_{j=i+1}^{m} a_ia_jb_ib_j + 2\sum_{i=1}^{m} a_ib_i + 1
\end{aligned}
$$

They are the same! And the later only takes O(m) to compute!

Kernel Functions

- If every data point is mapped into high-dimensional space via some transformation $\mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product that we need to compute for classifying a point x becomes:

$$<\phi(\mathbf{x}^i) \cdot \phi(\mathbf{x})> \text{ for all support vectors } \boldsymbol{x}^i$$

- A ***kernel function*** is a function that is equivalent to an inner product in some feature space.

$$\boldsymbol{k}(a,b) = <\phi(\mathbf{a}) \cdot \phi(\mathbf{b})>$$

- We have seen the example:

$$\boldsymbol{k}(a,b) = (a \cdot b+1)^2$$

This is equivalent to mapping to the quadratic space!

More kernel functions

- Linear kernel: ***k***(a,b) = (a·b)
- Polynomial kernel: ***k***(a,b) = (a·b+1)^{d}
- Radial-Basis-Function kernel:

$$K(\mathbf{a},\mathbf{b}) = \exp\left(-\frac{(\mathbf{a}-\mathbf{b})^2}{2\sigma^2}\right)$$

In this case, the corresponding mapping ϕ(**x**) is *infinite-dimensional*! Lucky that we don't have to compute the mapping explicitly!

$$\boxed{w \cdot \Phi(z) + b = \sum_{j=1}^{s} \alpha_{t_j} y^{t_j} (\Phi(x^{t_j}) \cdot \Phi(z)) + b = \sum_{j=1}^{s} \alpha_{t_j} y^{t_j} K(x^{t_j} \cdot z) + b}$$

Note: We will not get into the details but the learning of ***w*** can be achieved by using kernel functions as well!

Nonlinear SVM summary

- Map the input space to a high dimensional feature space and learn a linear decision boundary in the feature space
- The decision boundary will be nonlinear in the original input space
- Many possible choices of kernel functions
  - How to choose? Most frequently used method: cross-validation