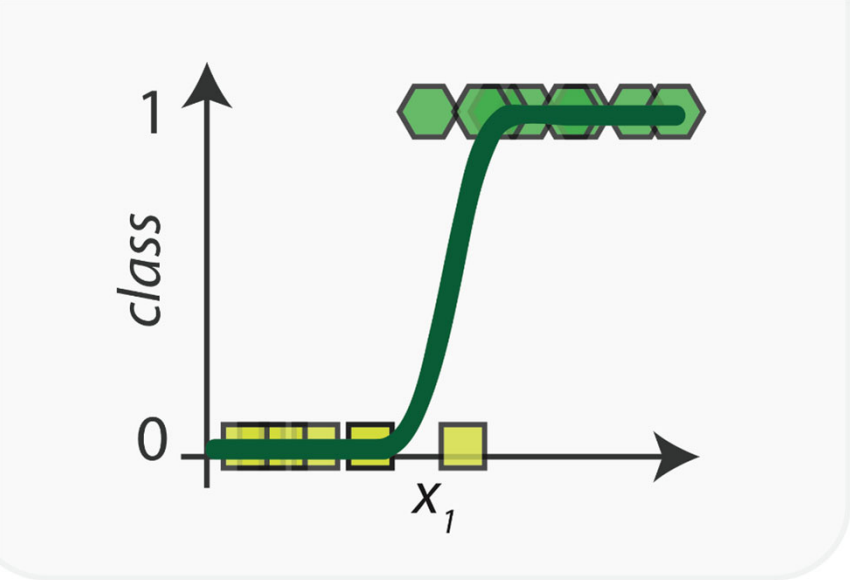
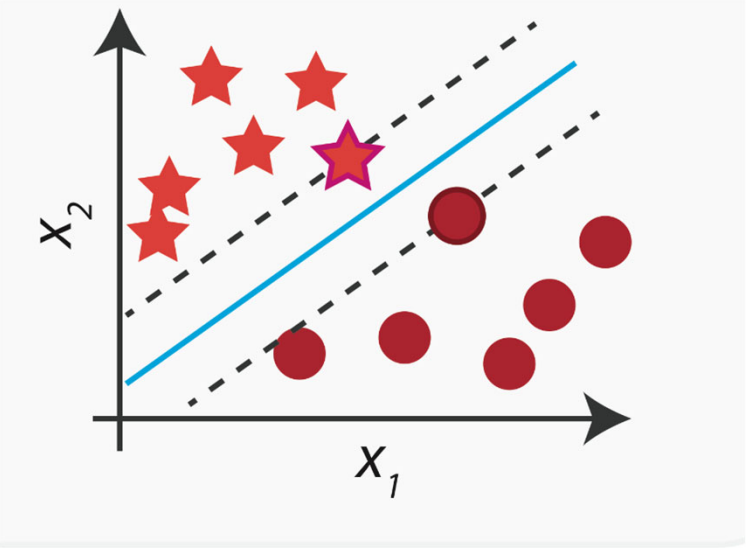
Machine Learning
Classification

Jian Liu

Part 1: Logistic Regression



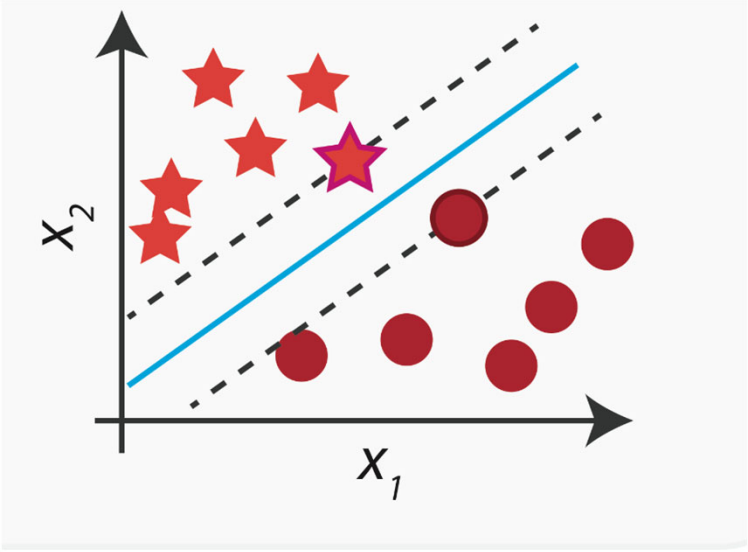
Part 2: Support Vector Machines



Machine Learning
Classification

Jian Liu

Part 2: Support Vector Machines



Logistic regression is a linear model

- We already have seen that some datasets are not linearly separable
- Logistic regression will not do well on this
- Logistic regression can be seen as two-layer networks (see MNIST)
- We can add more layers: increased computational capability
- More sophisticated training method required Backpropagation

Limitations of Linear Classifiers

- Linear Learning Machines cannot deal with
  - Non-linearly separable data
  - Noisy data

Overcome Limitations

- **Neural networks solution:** multiple layers of thresholded linear functions – multi-layer neural networks. Learning algorithms – back-propagation. Deep Learning …

- **SVM solution:** kernel representation.

  Approximation-theoretic issues are **independent** of the learning-theoretic ones.

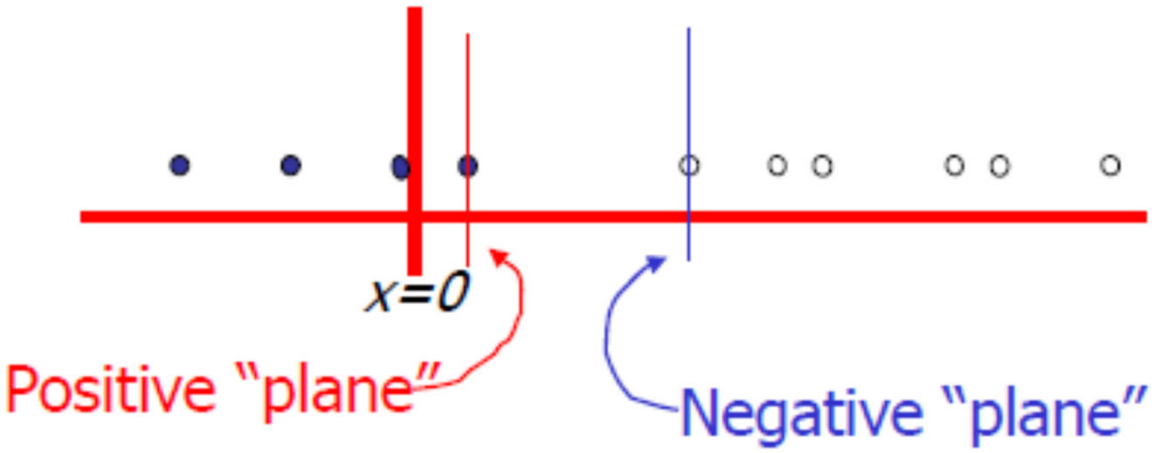
  Learning algorithms are decoupled from the specifics of the application area, which is encoded into design of kernel.

Overcome Limitations

SVM solution: kernel representation.

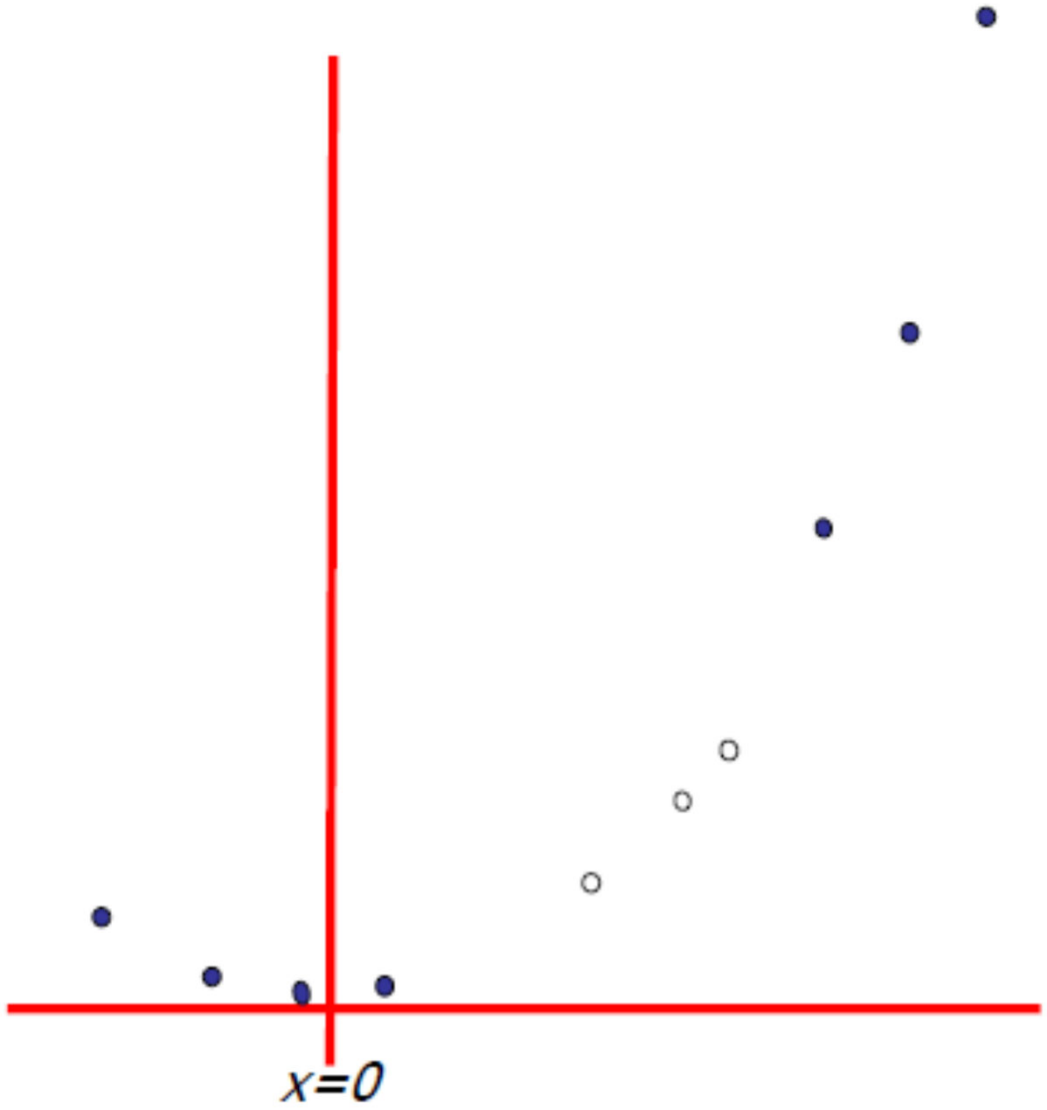
- Approximation-theoretic issues are **independent** of the learning-theoretic ones.
- Learning algorithms are decoupled from the specifics of the application area, which is encoded into design of kernel.
- SVM was/is popular before DL
- The original SVM algorithm was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963. Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes in 1992.

Easy



Difficult

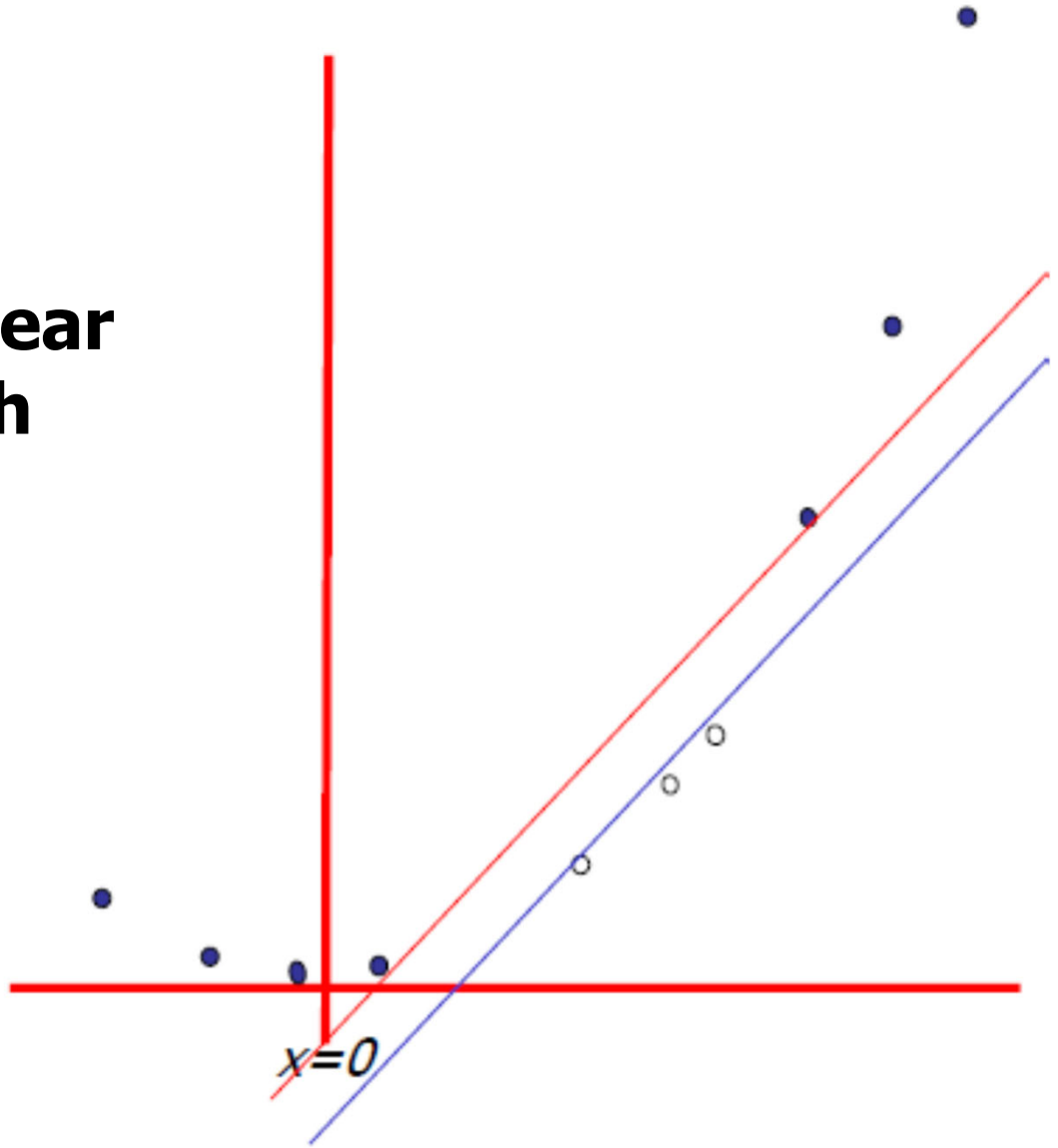
Easy?



Easy again …

Nonlinear basis functions made linear regression so much nicer

$$\mathbf{z}_k = (x_k, x_k^2)$$



Preprocessing the input vectors

Instead of trying to predict the answer directly from the raw inputs we could start by extracting a layer of "features".

- Sensible if we already know that certain combinations of input values would be useful
- The features are equivalent to a layer of hand-coded non-linear neurons.

So far as the learning algorithm is concerned, the hand-coded features are the input.

Is preprocessing cheating?

It seems like cheating if the aim to show how powerful learning is.

The really hard bit is done by the preprocessing.

Its not cheating if we learn the non-linear preprocessing.

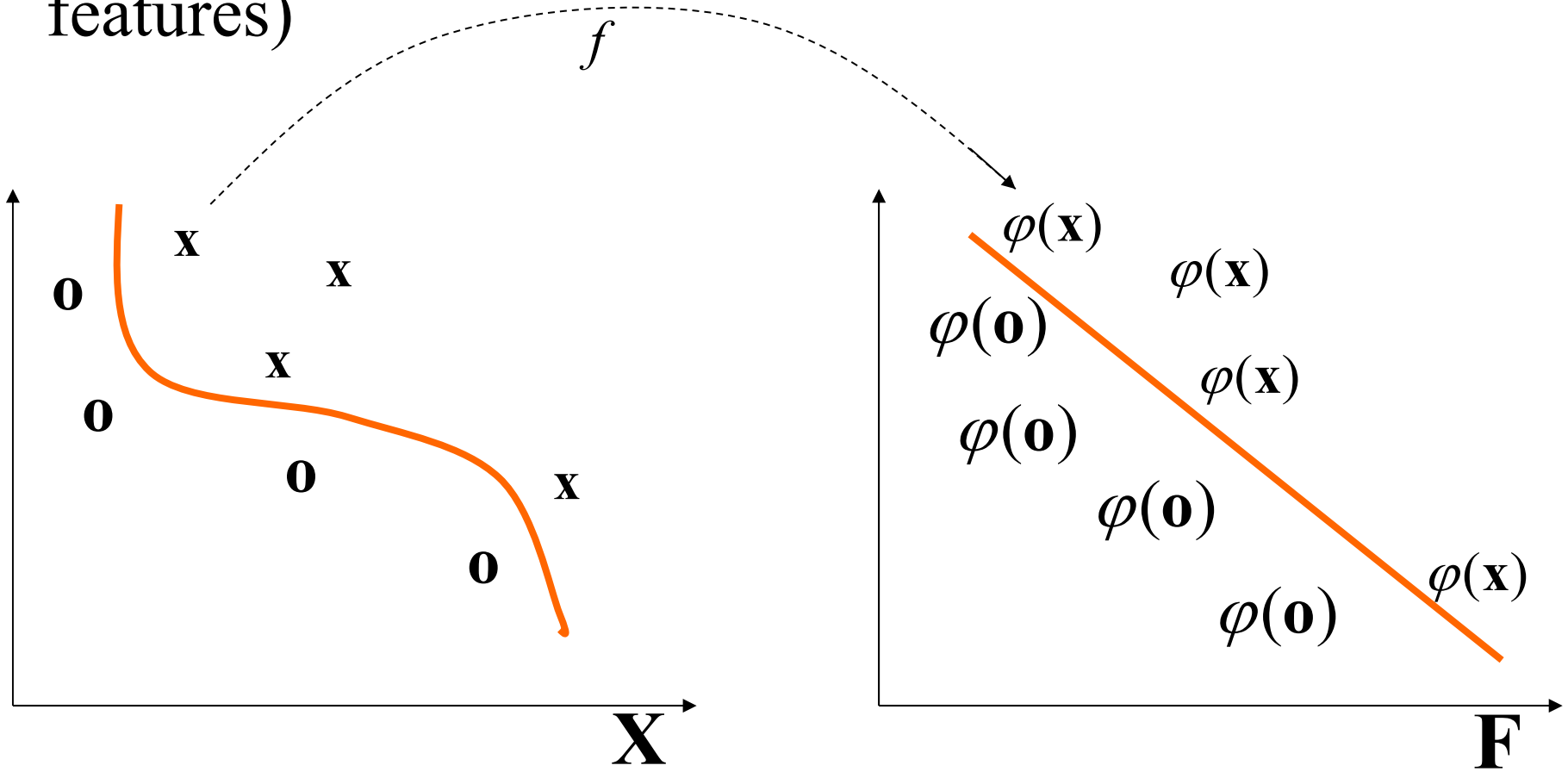
This makes learning much more difficult and much more interesting..

Its not cheating if we use a very big set of non-linear features that is task-independent.

Support Vector Machines make it possible to use a huge number of features without much computation or data.

Learning in the Feature Space

- Map data into a feature space where they are linearly separable $\mathbf{x} \rightarrow \varphi(\mathbf{x})$ (i.e. attributes $\rightarrow$ features)



Learning in the Feature Space

Example

- Consider the target function

$$f(m_1, m_2, r) = G\frac{m_1 m_2}{r^2},$$

giving gravitational force between two bodies.

- Observable quantities are masses m_1, m_2 and distance r. A linear machine could not represent it, but a change of coordinates

$$(m_1, m_2, r) \rightarrow (x, y, z) = (\ln m_1, \ln m_2, \ln r)$$

gives the representation

$$g(x, y, z) = \ln f(m_1, m_2, r) = \ln G + \ln m_1 + \ln m_2 - 2\ln r = c + x + y - 2z$$

Learning in the Feature Space

- The task of choosing the most suitable representation is known as *feature selection.*
- The space X is referred to as the input space, while $F = \{\varphi(\mathbf{x}) : \mathbf{x} \in X\}$ is called the feature space.

- Frequently one seeks to find smallest possible set of features that still conveys essential information (dimensionality reduction)

$$\mathbf{x} = (x_1, \ldots, x_n) \rightarrow \varphi(\mathbf{x}) = (\varphi_1(\mathbf{x}), \ldots, \varphi_d(\mathbf{x})), d < n$$

Problems with Feature Space

- Working in high dimensional feature spaces solves the problem of expressing complex functions

BUT:

- There is a computational problem (working with very large vectors)
- And a generalization theory problem (curse of dimensionality)

Implicit Mapping to Feature Space

We will introduce Kernels:

- Solve the computational problem of working with many dimensions
- Can make it possible to use infinite dimensions
  - Efficiently in time/space
- Other advantages, both practical and conceptual

Kernel-Induced Feature Space

- In order to learn non-linear relations we select non-linear features. Hence, the set of hypotheses we consider will be functions of type

$$f(x) = \sum_{i=1}^{N} w_i \varphi_i(x) + b$$

where $\varphi : X \to F$ is a non-linear map from input space to feature space

- In the dual representation, the data points only appear inside dot products

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i \left\langle \varphi(x_i), \varphi(x) \right\rangle + b$$

Kernels

- Kernel is a function that returns the value of the dot product between the images of the two arguments

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

- When using kernels, the dimensionality of space F not necessarily important. We may not even know the map
- Given a function K, it is possible to verify that it is a kernel

Kernels

- One can use linear learning machines in a feature space by simply rewriting it in dual representation and replacing dot products with kernels:

$$\langle x_1, x_2 \rangle \leftarrow K(x_1, x_2) = \langle \varphi(x_1), \varphi(x_2) \rangle$$

Support vector machines:
kernel methods

A kernel function is simply a function that takes two inputs and produces a similarity measure of this pair.

we always deal with a pair of data points, consistently.

This results in shifting the emphasis from directly manipulating the features of the available data into manipulating the similarity between pairs of data points.

clustering algorithms that use a pairwise comparison of similarity or distances are kernel methods.

our concentration will be on supervised learning kernel methods for regression and classification.

Kernel methods

We start by talking about the duality of linear models where we show how to formulate the familiar linear model as a kernel method.

Next, we generalise the ideas of a linear kernel model into a more capable model, namely the radial basis function network.

Finally, we move into sparse kernel methods, support vector machines (SVM).

SVM is widely used due to its versatile and strong guarantees of a global minimum which avoids the issues related to getting stuck in local minima that other methods, such as neural networks

Linear kernel methods and duality representation

design matrix **Φ**: the feature mapping for input vector **x**:

$$\mathbf{\Phi} = \begin{bmatrix} \phi_0(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_n) & \cdots & \phi_{M-1}(\mathbf{x}_n) \\ \vdots & \vdots & \vdots \\ \phi_0(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{bmatrix}$$

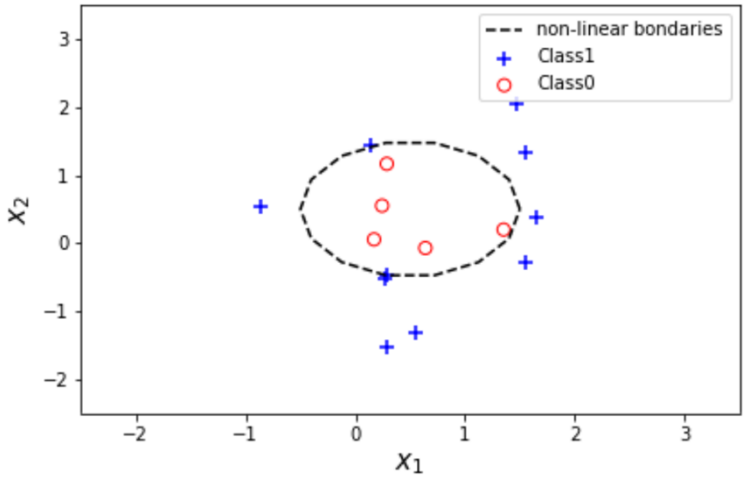
dot product kernel function

$$k(x, x') = \boldsymbol{\phi}^\top(x)\boldsymbol{\phi}(x')$$

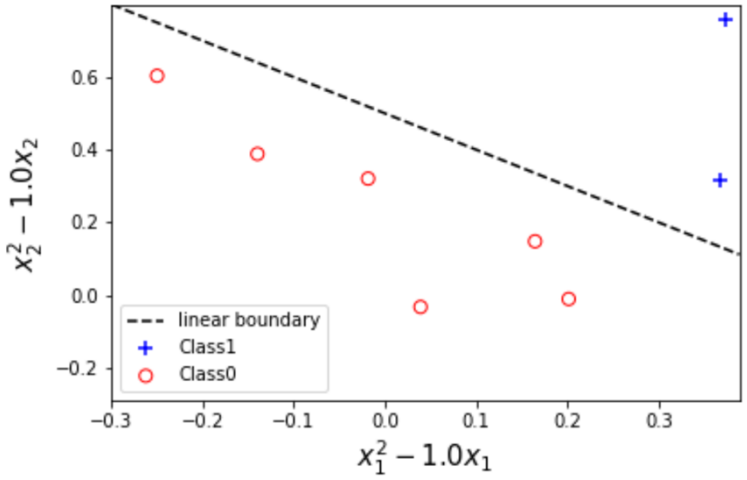
dot product of two vectors provides a way to measure their similarity

$$\boldsymbol{\phi}(x).\boldsymbol{\phi}(x') = \boldsymbol{\phi}^\top(x)\boldsymbol{\phi}(x') = \|\boldsymbol{\phi}(x)\|\|\boldsymbol{\phi}(x')\|cos(\alpha)$$

Same direction: α=0 and cos(0)=1, so ‖ϕ(x)‖‖ϕ(x′)‖
Orthogonality: cos(π/2)=0, then 0.
Opposite direction: −‖ϕ(x)‖‖ϕ(x′)‖ since cos(π)=−1.



(a) Before mapping



(b) After mapping

Linear kernel methods and duality representation

design matrix **Φ**: the feature mapping for input vector **x**:

$$\mathbf{\Phi} = \begin{bmatrix} \phi_0(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_n) & \cdots & \phi_{M-1}(\mathbf{x}_n) \\ \vdots & \vdots & \vdots \\ \phi_0(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{bmatrix}$$

dot product kernel function

$$k(x, x') = \boldsymbol{\phi}^\top(x)\boldsymbol{\phi}(x')$$

Gram matrix: similarity measure for all pairs of data points using the dot product kernel

$$\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top$$

nonparametric models
(models that do not use weights parameters to estimate a solution)

Linear kernel methods and duality representation

Gram matrix: similarity measure for all pairs of data points using the dot product kernel

$$\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^{\top}$$

nonsparse kernel methods

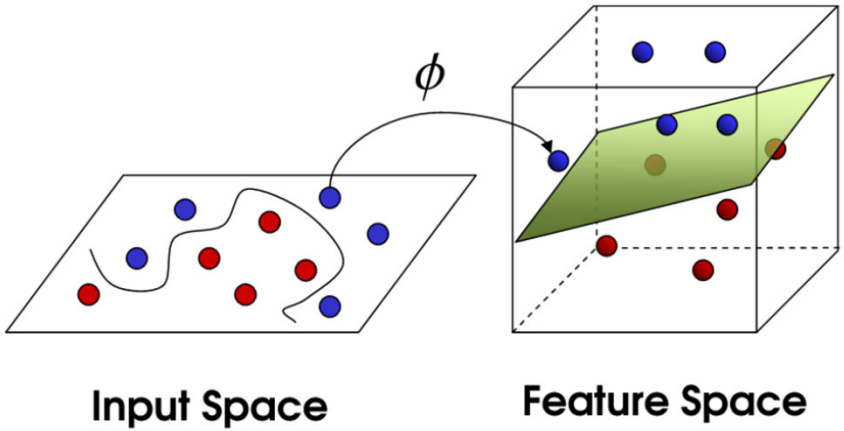
- Radial basis function network
- Gaussian processes

Sparse kernel methods

- Support vector machines

Kernel methods

What if the input data points are non-linearly separable?



Support vector machines

a linearly separable binary classification problem

$$y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

where b is the bias and w is a weight vector.

N input vectors x, 1,...,N,
with corresponding target labels of t, 1,...,N.

Binary: $t \in \{-1, +1\}$.

Support vector machines

A new data point x is classified according to the sign of y(x):

$$\text{if } y(\mathbf{x}) > 0 \text{ then } \mathbf{x} \text{ is positive}$$

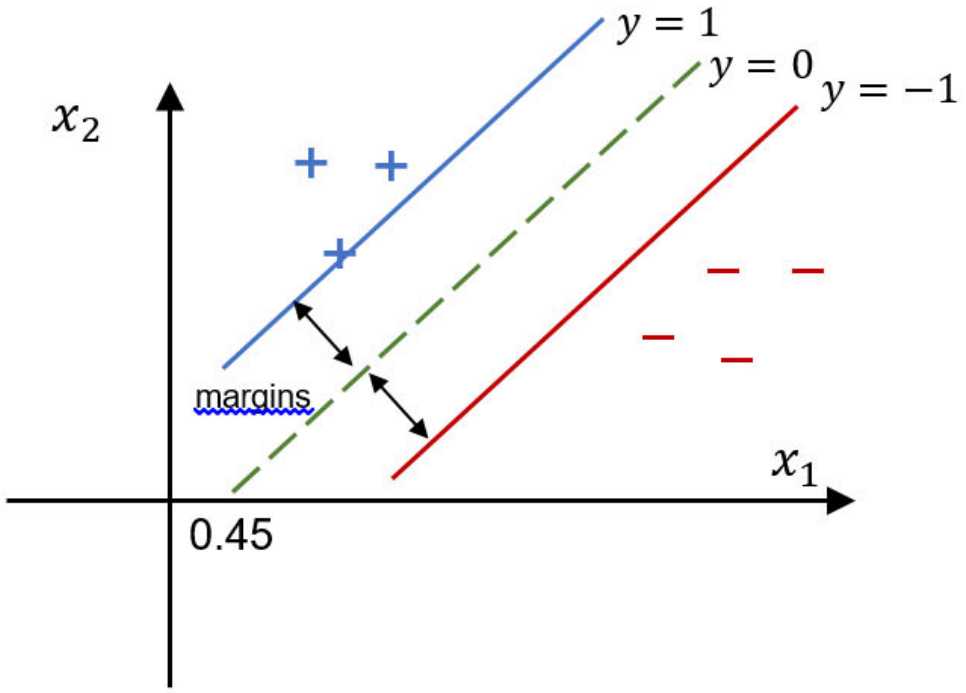
$$\text{if } y(\mathbf{x}) < 0 \text{ then } \mathbf{x} \text{ is negative}$$

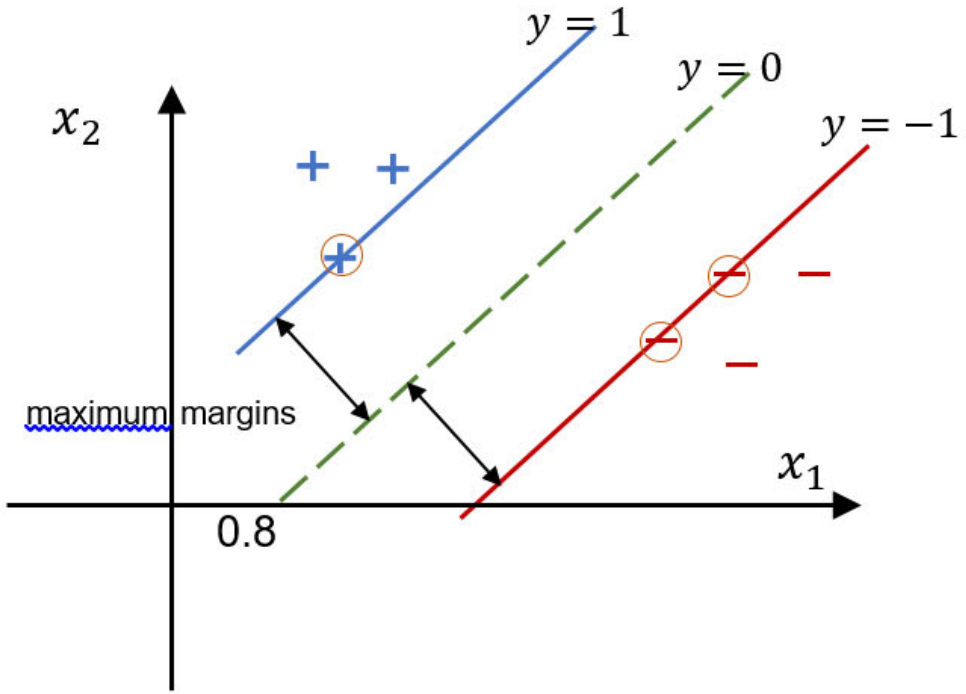
This means that y=0 draws a line between positive and negative data points.

SVM is to simply add a safety margins on the two sides of the line that separates the two classes so that we maximise the differentiability of the two classes given the available data.

In other words, instead of using 0 in both conditions that differentiate between the negative and positive cases, we use two opposite values to specify a wide street that separates between the negative and positive cases.

Support vector machines



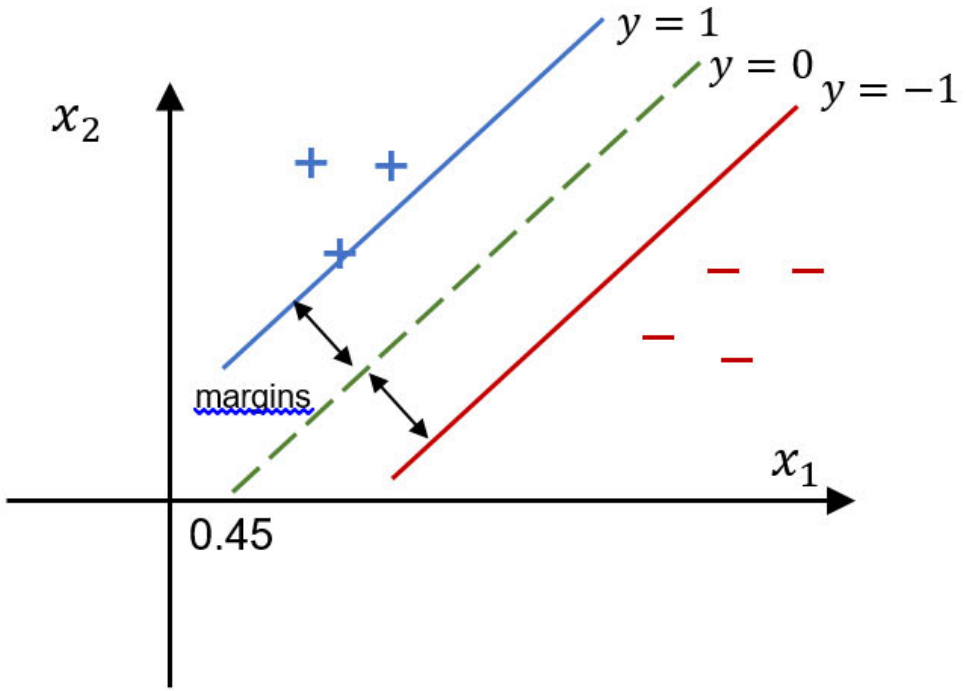


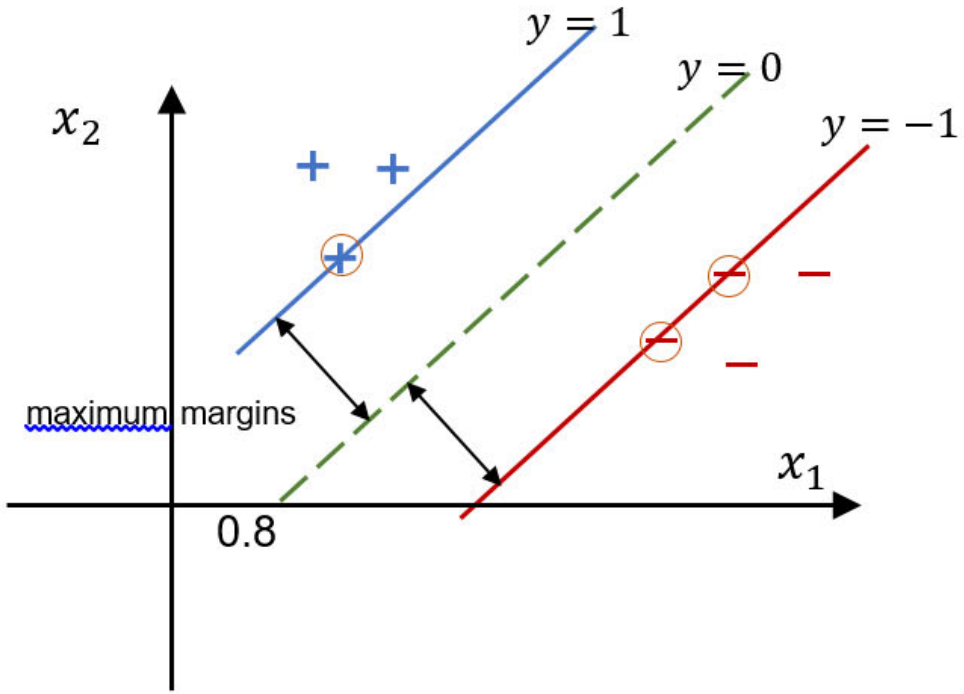
when $\mathbf{x}_n$ is positive then $y\left(\mathbf{x}_n\right) > +1$

when $\mathbf{x}_n$ is negative then $y\left(\mathbf{x}_n\right) < -1$

In other words, instead of using 0 in both conditions that differentiate between the negative and positive cases, we use two opposite values to specify a wide street that separates between the negative and positive cases.

Support vector machines





when $\mathbf{x}_n$ is positive then $t_n = +1$

when $\mathbf{x}_n$ is negative then $t_n = -1$

Support vector machines

$$\text{when } \mathbf{x}_n \text{ is positive then } y\left(\mathbf{x}_n\right) > +1$$

$$\text{when } \mathbf{x}_n \text{ is negative then } y\left(\mathbf{x}_n\right) < -1$$

all of the above four requirements for negative and positive classes can be formulated in one condition

$$t_n(\mathbf{w}^\top \mathbf{x}_n + b) - 1 \geq 0$$

for those data points that lie on the margins of the street,
which define the linear boundaries of a binary class problem

Support vector machines

$$t_n(\mathbf{w}^\top \mathbf{x}_n + b) - 1 \geq 0$$

The inequality defines a set of constraints

Lagrange method gives a simple way to combine the minimisation and the constraints in one formula to streamline the process of finding a minimiser that satisfies the constraints.

Hence the targets are:

$$\text{Minimise } \|\mathbf{w}\|^2$$

$$\text{Subject to constraints } t_n \left(\mathbf{w}^\top \mathbf{x}_n + b\right) - 1 \geq 0 \quad n = 1, \ldots, N$$

Support vector machines

$$\text{Minimise} \left|\left|\mathbf{w}\right|\right|^2$$

$$\text{Subject to constraints} \, t_n \left(\mathbf{w}^\top \mathbf{x}_n + b\right) - 1 \geq 0 \quad n = 1, \ldots, N$$

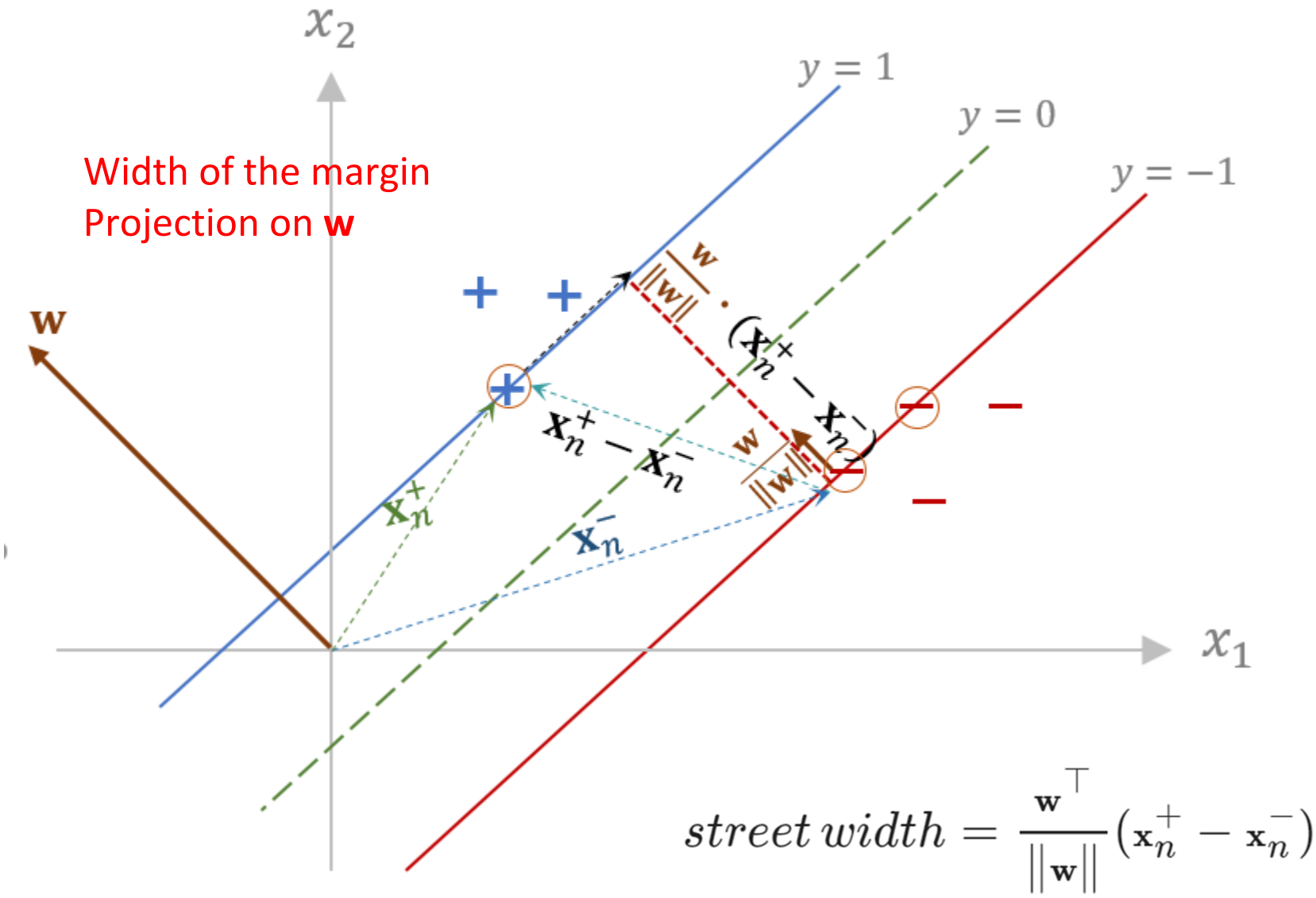
Maximise the margin means to maximise the width of the street that separates the two classes.

We assume that **w** is perpendicular to the street that separates the classes, hence conveniently the *street width* can be expressed as

$$street\, width = \frac{\mathbf{w}^\top}{\left|\left|\mathbf{w}\right|\right|}(\mathbf{x}_n^+ - \mathbf{x}_n^-)$$

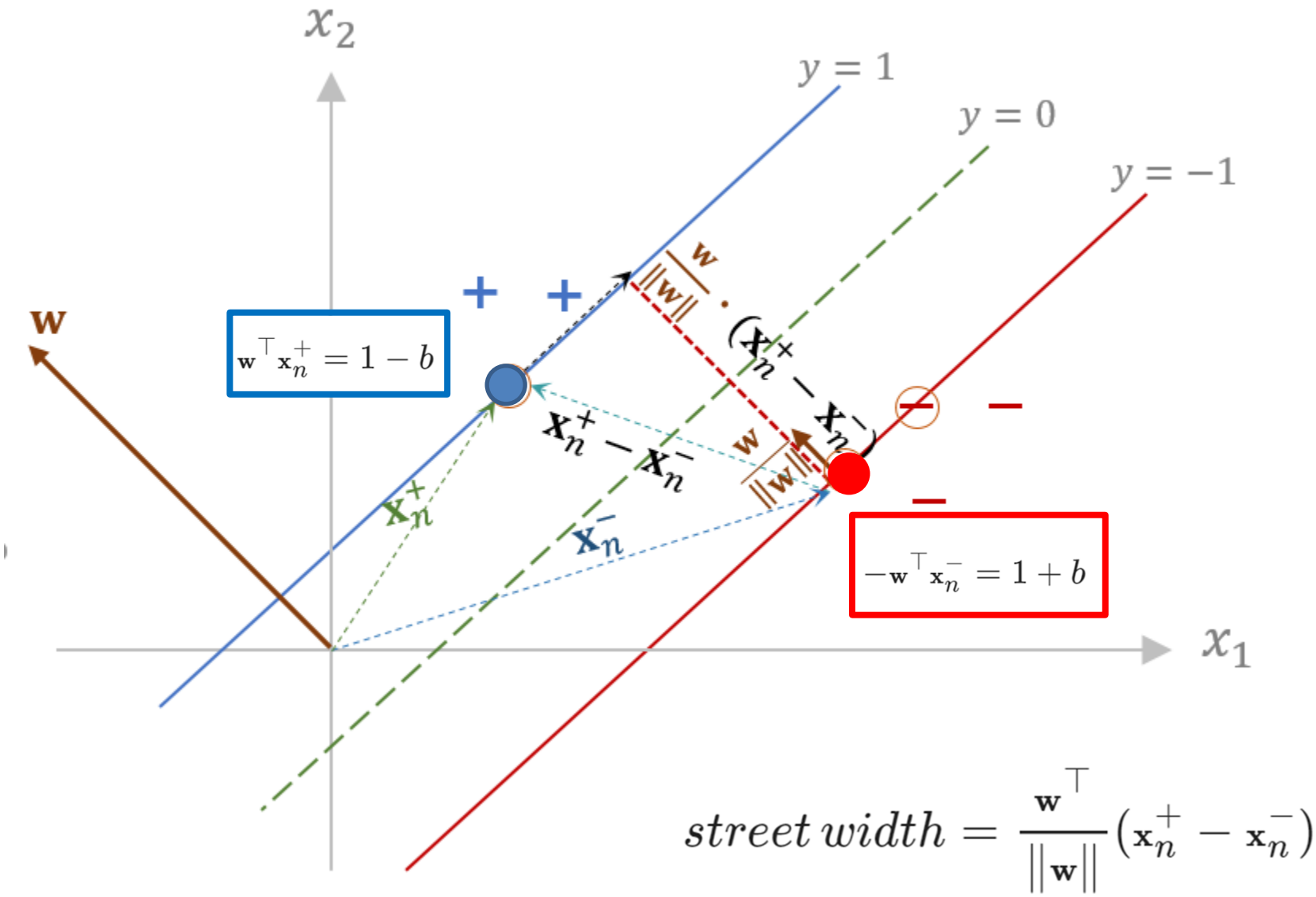
The vector difference between two samples from our dataset.
We call such points the support vectors for the binary classification problem.

Support vector machines



$$street\ width = \frac{\mathbf{w}^\top}{\|\mathbf{w}\|}(\mathbf{x}_n^+ - \mathbf{x}_n^-)$$

Support vector machines



Support vector machines

$\mathbf{w}^\top \mathbf{x}_n^+ = 1 - b$

$$street\ width = \frac{\mathbf{w}^\top}{\|\mathbf{w}\|}(\mathbf{x}_n^+ - \mathbf{x}_n^-)$$

$-\mathbf{w}^\top \mathbf{x}_n^- = 1 + b$

$$\begin{aligned} street\ width &= \frac{1}{\|\mathbf{w}\|}(\mathbf{w}^\top \mathbf{x}_n^+ - \mathbf{w}^\top \mathbf{x}_n^-) \\ &= \frac{1}{\|\mathbf{w}\|}(1 - b + 1 + b) \\ &= \frac{2}{\|\mathbf{w}\|} \end{aligned}$$

Support vector machines

$$
\begin{aligned}
street\,width &= \frac{1}{\|\mathbf{w}\|}(\mathbf{w}^\top \mathbf{x}_n^+ - \mathbf{w}^\top \mathbf{x}_n^-) \\
&= \frac{1}{\|\mathbf{w}\|}(1 - b + 1 + b) \\
&= \frac{2}{\|\mathbf{w}\|}
\end{aligned}
$$

maximising the street width is achieved by maximising 2/‖w‖
which can be achieved by minimising ‖w‖
which in turn can be achieved by *minimising* ‖w‖^2.

Note that we use only two supports to calculate the width while we still have to honour the constraints for all support vectors that are on the gutter.

So now our problems becomes a quadratic programming problem and is formulated as:

$$
\begin{aligned}
\textbf{Minimising} \frac{1}{2}\|\mathbf{w}\|^2 \\
\textbf{Subject to constraints} t_n(\mathbf{w}^\top \mathbf{x}_n + b) - 1 \geq 0
\end{aligned}
$$

Support vector machines

$$\textbf{Minimising} \frac{1}{2} \|\mathbf{w}\|^2$$

$$\textbf{Subject to constraints} t_n(\mathbf{w}^\top \mathbf{x}_n + b) - 1 \geq 0$$

The good news is that now we have a *convex* problem that has a *global optimum*.

This is great since solving it means that we have no local minima issues as in a neural networks loss function.

Support vector machines

$$\textbf{Minimising} \frac{1}{2}\|\mathbf{w}\|^2$$

$$\textbf{Subject to constraints} t_n(\mathbf{w}^\top \mathbf{x}_n + b) - 1 \geq 0$$

Lagrange comes to the rescue to solve this optimisation problem under the given constraints.

The Lagrangian combines the above two requirements (minimisation under constraints) together in one convenient mathematical quantity that we aim to minimise.

In other words, we want to:

$$Maximise\, L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} \alpha_n [t_n(\mathbf{w}^\top \mathbf{x}_n + b) - 1]$$

where α are Lagrange multipliers satisfying α≥0.

Support vector machines

$$\textbf{Minimising} \frac{1}{2}\|\mathbf{w}\|^2$$

$$\textbf{Subject to constraints} t_n(\mathbf{w}^\top \mathbf{x}_n + b) - 1 \geq 0$$

For the support vectors that lie on the gutters we say that the constraints are active, while for the other data points we say that the constraints are inactive.

This will be apparent later where we will see that the Lagrange multipliers are 0 for the inactive and none 0 for the support which constitutes the **sparsity** aspect of support vector machines.

$$Maximise\, L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} \alpha_n [t_n(\mathbf{w}^\top \mathbf{x}_n + b) - 1]$$

where α are Lagrange multipliers satisfying $\alpha \geq 0$.

Support vector machines

$$Maximise\, L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} \alpha_n [t_n(\mathbf{w}^\top \mathbf{x}_n + b) - 1]$$

To maximise this Lagrangian function we need to take the derivatives with respect to the weights **w** and **b** and we set them to 0

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{m=1}^{N} \alpha_m t_m \mathbf{x}_m = 0$$

$$\mathbf{w}^\top = \sum_{m=1}^{N} \alpha_m t_m \mathbf{x}_m^\top$$

$$\frac{\partial L}{\partial b} = \sum_{n=1}^{N} \alpha_m t_m \mathbf{x}_m = 0$$

$$\sum_{m=1}^{N} \alpha_m t_m = 0$$

Support vector machines

$$Maximise\, L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} \alpha_n [t_n(\mathbf{w}^\top \mathbf{x}_n + b) - 1]$$

$$\mathbf{w}^\top = \sum_{m=1}^{N} \alpha_m t_m \mathbf{x}_m^\top$$

Substitute w in the Lagrangian eq

This will give us the **dual representation** of the maximum margin problem

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\left(\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m \left[\mathbf{x}_m^\top \mathbf{x}_n\right]\right)$$

all dot products of the different samples

$$\sum_{n=1}^{N} \alpha_n t_n \left[\mathbf{x}_n^\top \mathbf{x}\right] + b \geq 1 \quad \text{then } \mathbf{x} \in \text{positive class}$$

$$\sum_{n=1}^{N} \alpha_n t_n \left[\mathbf{x}_n^\top \mathbf{x}\right] + b \leq 1 \quad \text{then } \mathbf{x} \in \text{negative class}$$

Support vector machines

Feature mapping

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\left(\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m \left[\boldsymbol{\phi}_m^{\top}\boldsymbol{\phi}_n\right]\right)$$

$$\sum_{n=1}^{N} \alpha_n t_n \left[\boldsymbol{\phi}_n^{\top}\boldsymbol{\phi}\right] + b \geq 0 \quad \text{then } \mathbf{x} \in \text{positive class}$$

$$\sum_{n=1}^{N} \alpha_n t_n \left[\boldsymbol{\phi}_n^{\top}\boldsymbol{\phi}\right] + b \leq 0 \quad \text{then } \mathbf{x} \in \text{negative class}$$

Support vector machines

Kernel function

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\left(\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m k(\phi_m, \phi_n)\right)$$

$$\sum_{n=1}^{N} \alpha_n t_n k\left(\phi_n^{\top}\phi\right) + b \geq 0 \quad \text{then } \mathbf{x} \in \text{positive class}$$

$$\sum_{n=1}^{N} \alpha_n t_n k\left(\phi_n^{\top}\phi\right) + b \leq 0 \quad \text{then } \mathbf{x} \in \text{negative class}$$

The gain is that by using a more complex kernels we can obtain more complex representations for more complex relationships between the classes.

We can choose a kernel that maps the input space into a richer space that turns a non-linear classification problem into a linearly separable classification problem in the new space.

Support vector machines

dual representation of the maximum margin problem

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\left(\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m \left[\mathbf{x}_m^\top \mathbf{x}_n\right]\right)$$

all dot products of the different samples

Notice that all we have are the dot products

α **will be obtained, so it can tell a new datapoint by the sign**

$$\sum_{n=1}^{N} \alpha_n t_n \left[\mathbf{x}_n^\top \mathbf{x}\right] + b \geq 1 \quad \text{then } \mathbf{x} \in \text{positive class}$$

$$\sum_{n=1}^{N} \alpha_n t_n \left[\mathbf{x}_n^\top \mathbf{x}\right] + b \leq 1 \quad \text{then } \mathbf{x} \in \text{negative class}$$

Support vector machines

dual representation of the maximum margin problem

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\left(\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m \left[\mathbf{x}_m^\top \mathbf{x}_n\right]\right)$$

Notice that all we have are the dot products

inner products provide some measure of 'similarity'

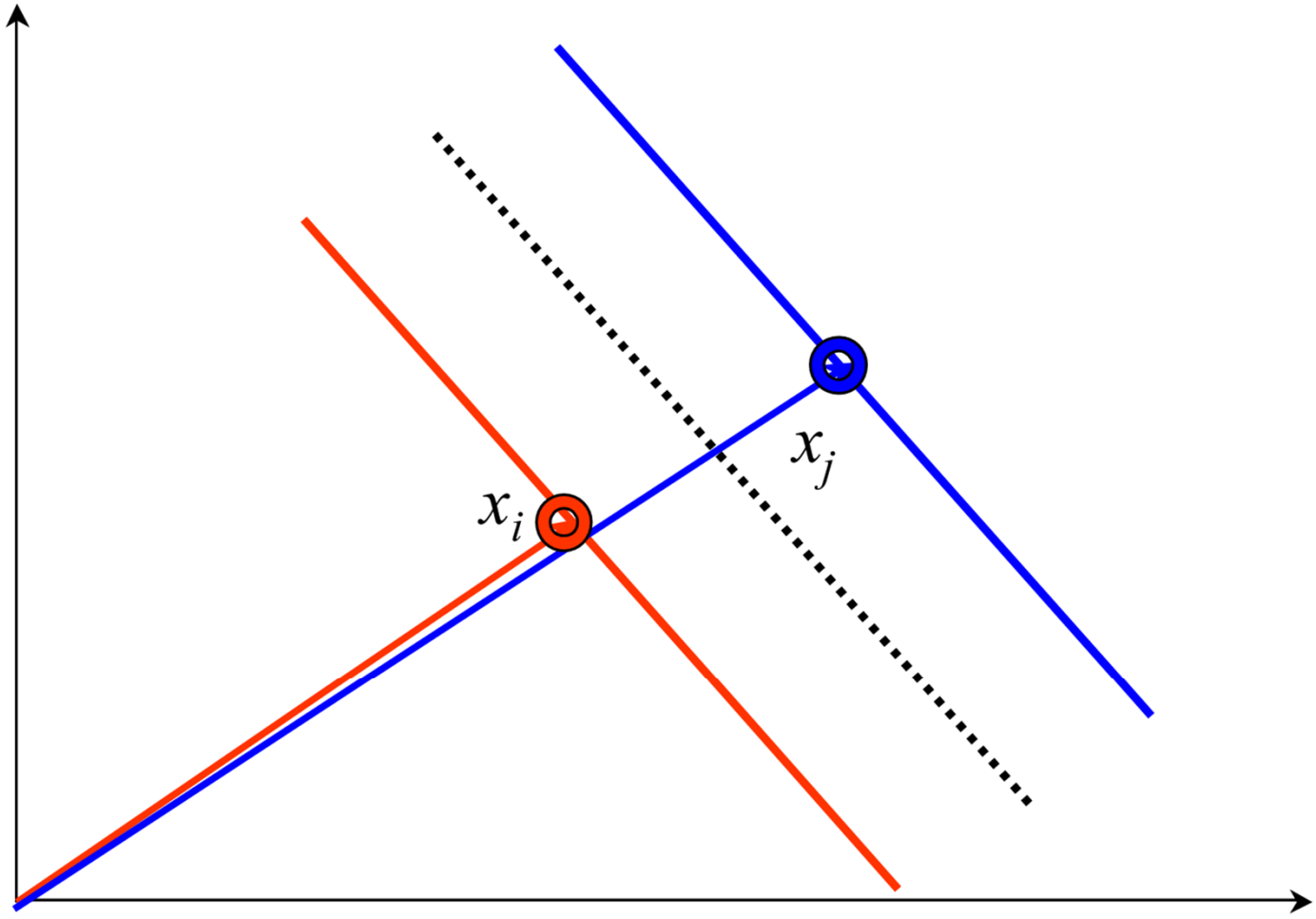
Case 1. If two features are completely dissimilar, their dot product is 0, and they don't contribute to L.

Case 2. If two features are completely alike, their dot product is 1.

- Subcase 1: both **x** predict the same output value **t** (either +1 or –1). Then product will be positive. But this would decrease the value of **L** (since it would subtract from the first term sum). So, the algorithm downgrades similar feature vectors that make the same prediction.
- Subcase 2: both **x** make opposite predictions about the output value **t** (one is +1, the other –1), then the product is negative and we are subtracting it, so this adds to the sum, maximizing **L**. This is precisely the examples we are looking for: the critical ones that tell the two classses apart.

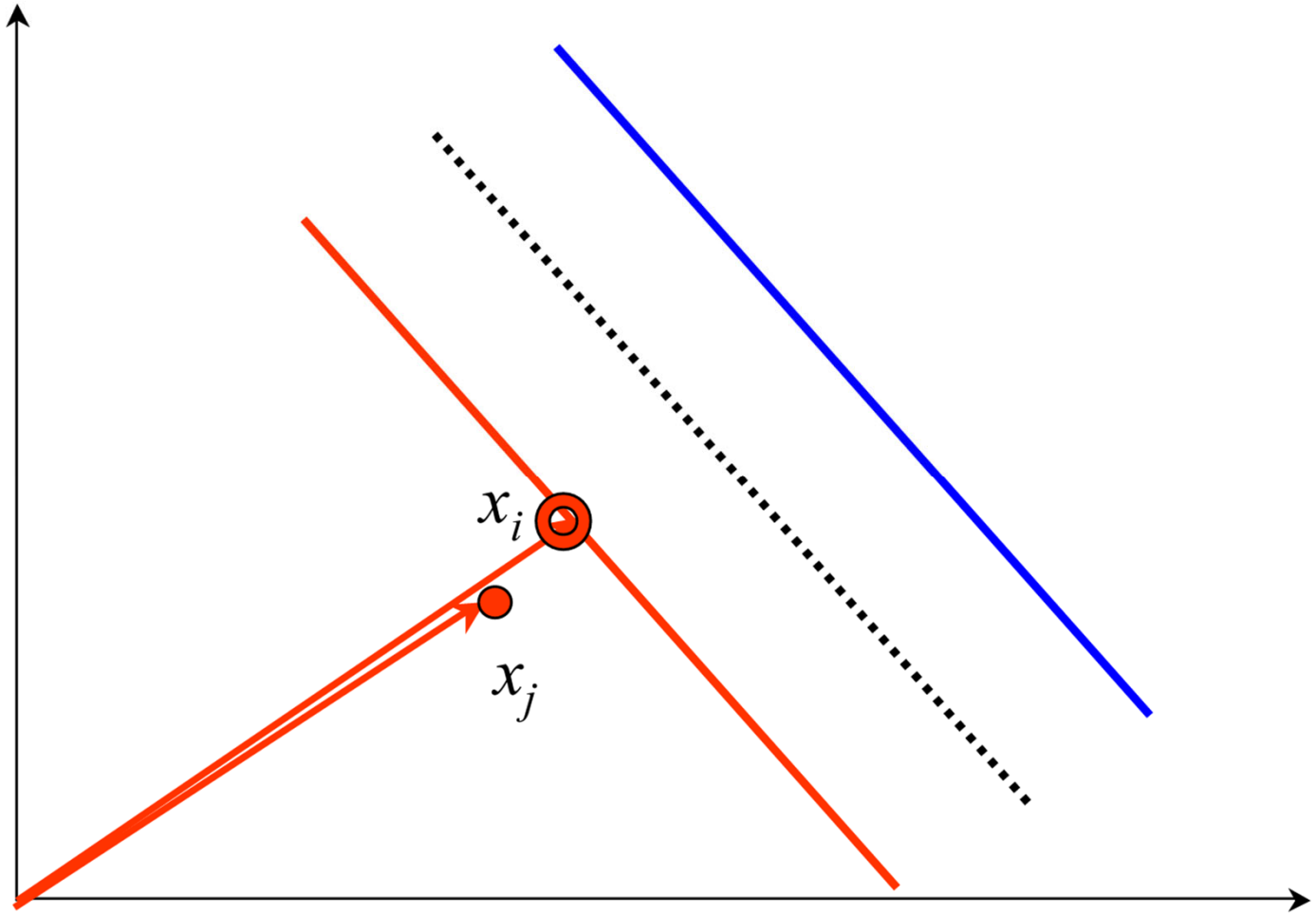
Support vector machines

2 very similar vectors that predict **different** classes tend to maximize the margin width



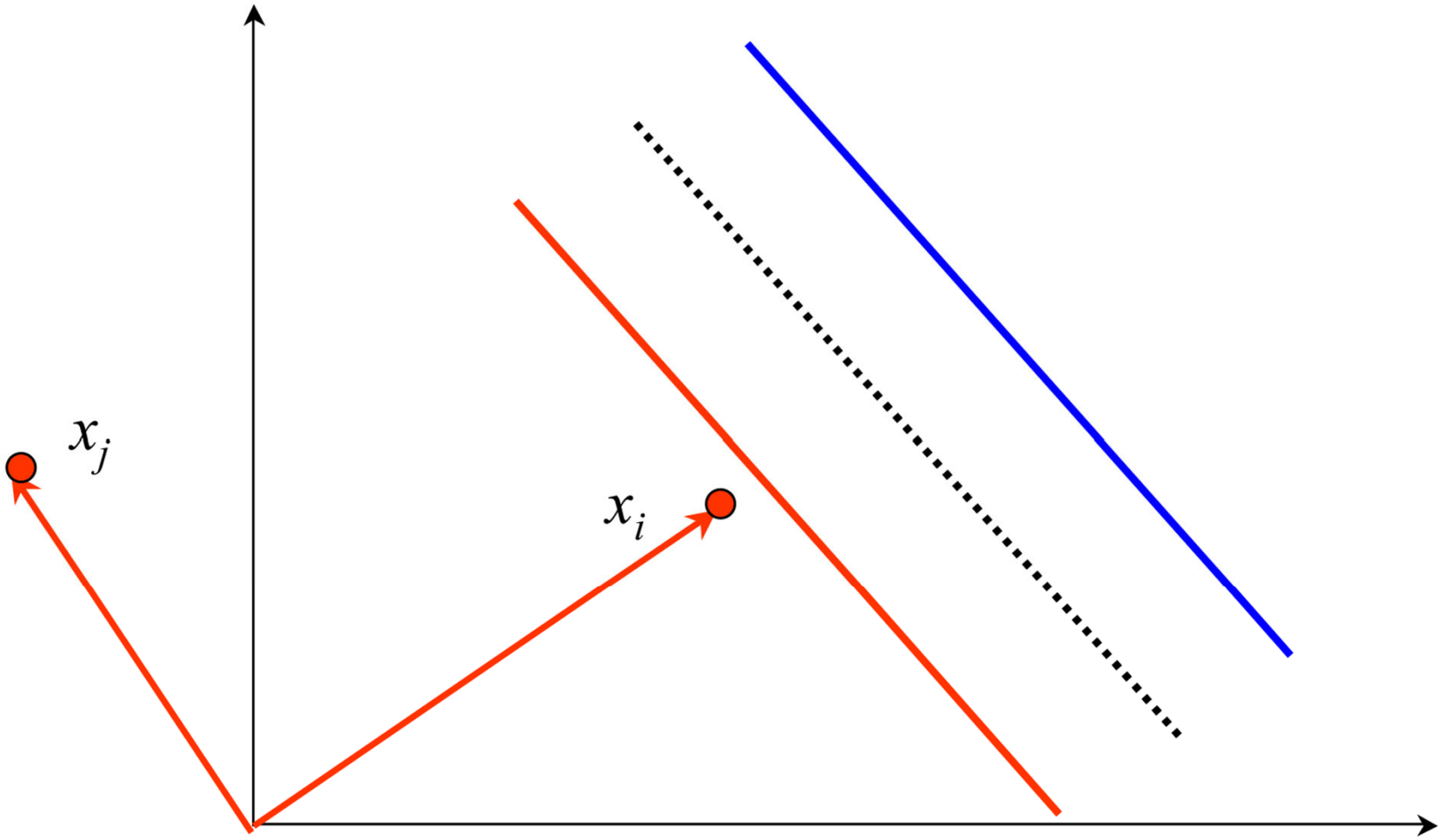
Support vector machines

2 very similar vectors that predict **same** classes are redundant



Support vector machines

2 dissimilar (orthogonal) vectors don’t count at all



The Kernel Matrix (Gram Matrix)

K=

K(1,1)	K(1,2)	K(1,3)	…	K(1,m)
K(2,1)	K(2,2)	K(2,3)	…	K(2,m)
…	…	…	…	…
K(m,1)	K(m,2)	K(m,3)	…	K(m,m)

The Kernel Matrix

- The central structure in kernel machines
- Information ‘bottleneck’: contains all necessary information for the learning algorithm
- Fuses information about the data AND the kernel
- Many interesting properties:

Mercer's Theorem

- The kernel matrix is Symmetric Positive Definite
- Any symmetric positive definite matrix can be regarded as a kernel matrix, that is as an inner product matrix in some space
- More formally, **Mercer's Theorem:** Every (semi) positive definite, symmetric function is a kernel: i.e. there exists a mapping φ such that it is possible to write:

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

Definition of Positive Definiteness:

$$\int K(x,z) f(x) f(z) dx dz \geq 0, \quad \forall f \in \mathbf{L}_2$$

Mercer's Theorem cont.

- Eigenvalues expansion of Mercer's Kernels:

$$K(x_1, x_2) = \sum_i \lambda_i \langle \varphi_i(x_1), \varphi_i(x_2) \rangle$$

- That is: the eigenvalues act as features!

Mercer's Theorem

There is a "kernel function" K such that

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

then we do not need to know or compute φ at all!!

That is, the kernel function defines inner products in the transformed space.

Or, it defines similarity in the transformed space.

Examples of Kernels

Simple examples of kernels:

Polynomial of degree d

$$K(x,z) = (\langle x,z \rangle + 1)^d$$

Gaussian **R**adial **B**asis **F**unction

$$K(x,z) = e^{-\|x-z\|^2/2\sigma^2}$$

Sigmoidal

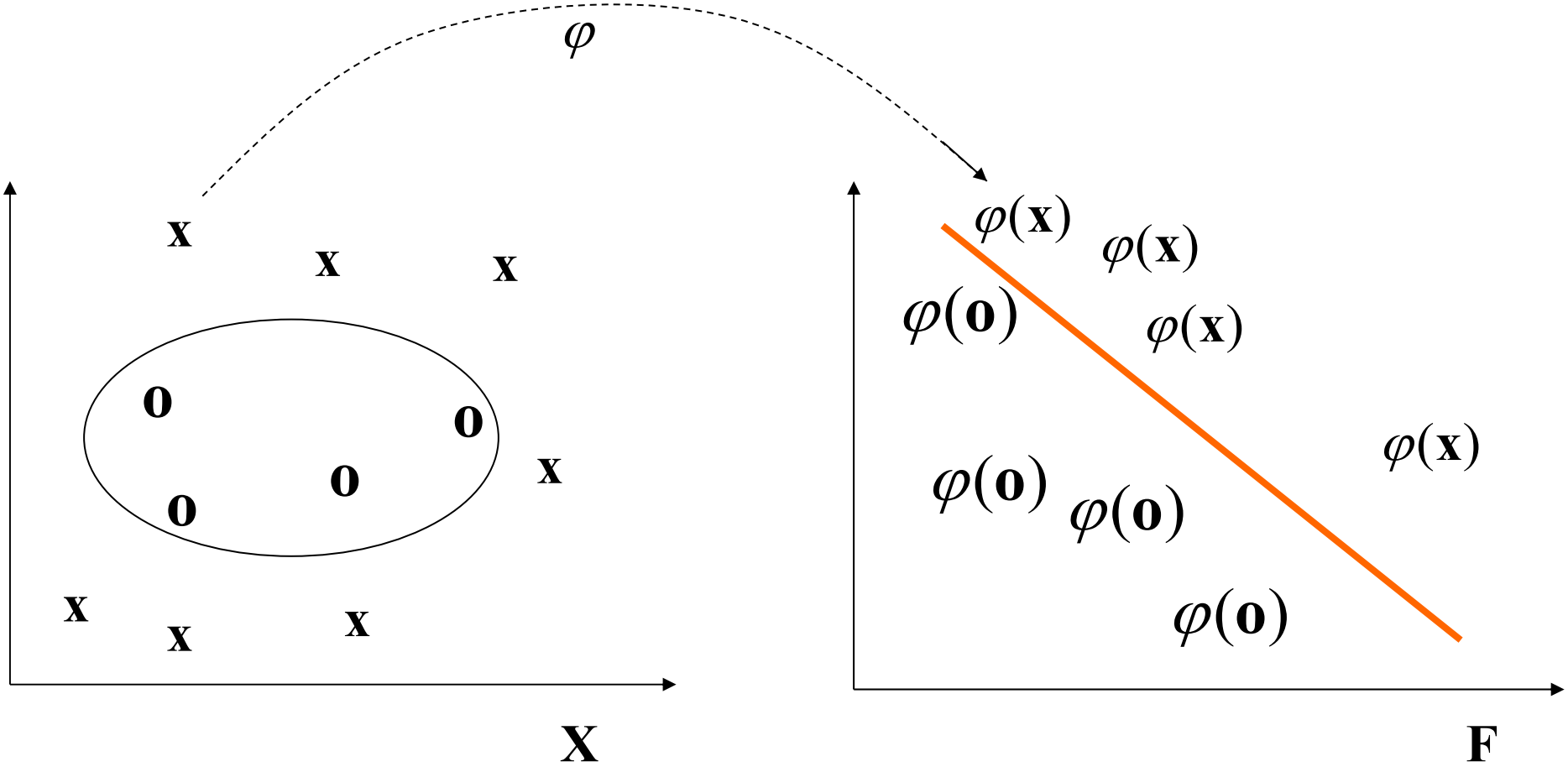
$$K(x,z) = \tanh(\langle x,z \rangle - \theta)$$

Example: Polynomial Kernels

$$x = (x_1, x_2);$$
$$z = (z_1, z_2);$$

$$\langle x, z \rangle^2 = (x_1 z_1 + x_2 z_2)^2$$
$$= x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 z_1 x_2 z_2$$
$$= \left\langle (x_1^2, x_2^2, \sqrt{2} x_1 x_2), (z_1^2, z_2^2, \sqrt{2} z_1 z_2) \right\rangle$$
$$= \langle \varphi(x), \varphi(z) \rangle$$

Example



Making Kernels

The set of kernels is closed under some operations. If K,K' are kernels, then:

- K+K' is a kernel
- cK is a kernel, if c>0
- aK+bK' is a kernel, for a,b>0
- Etc…
- One can make complex kernels from simple ones: modularity!

A bad kernel

- .. Would be a kernel whose kernel matrix is mostly diagonal: all points orthogonal to each other, no clusters, no structure….

1	0	0	…	0
0	1	0	…	0
		1		
…	…	…	…	…
0	0	0	…	1

No Free Kernel

- In mapping in a space with too many irrelevant features, kernel matrix becomes diagonal
- Need some prior knowledge of target so choose a good kernel

SVMs are Linear Learning Machines, that :

- Use a dual representation
- Operate in a kernel induced feature space (that is:

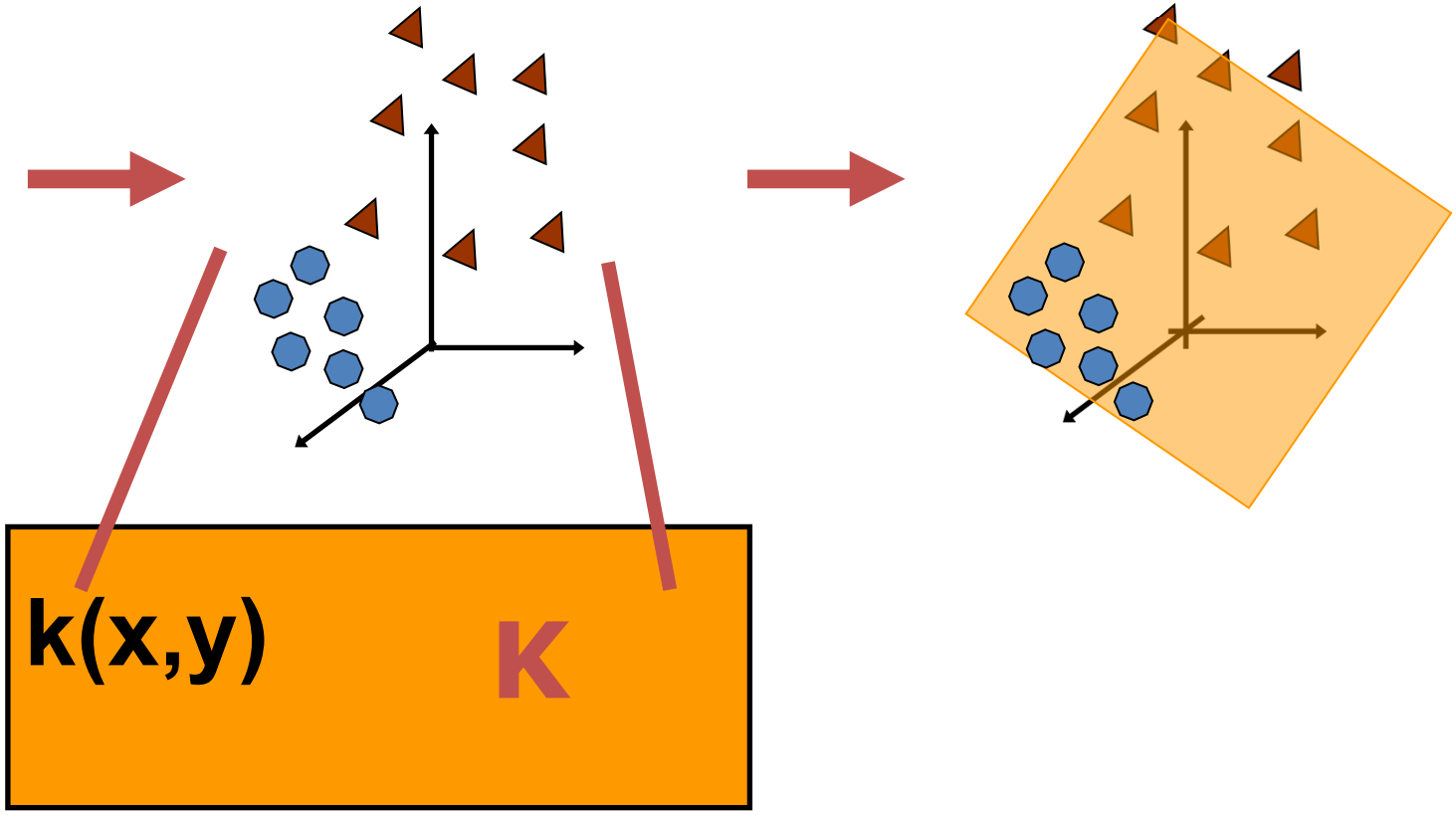
$$f(x) = \sum \alpha_i y_i \langle \varphi(x_i), \varphi(x) \rangle + b$$

is a linear function in the feature space implicitly defined by K)

Take-home message:
SVM

Data **Embedding** **Linear algorithm**

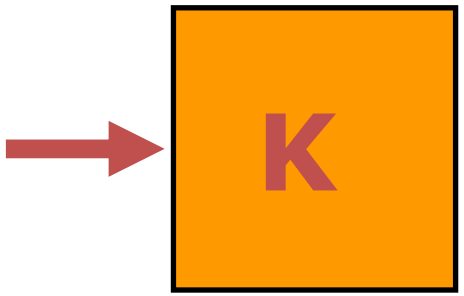
$\{(x_i, y_i)\}$

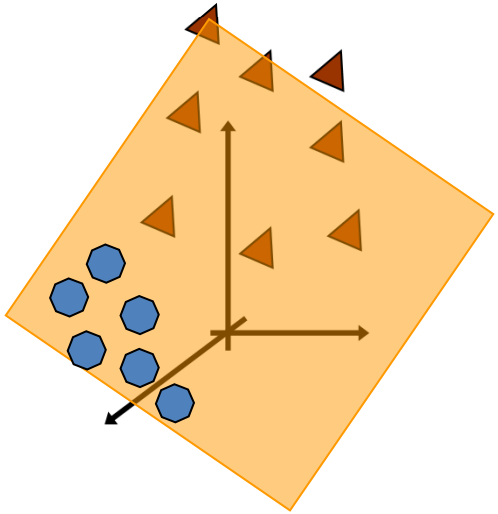


Take-home message:
SVM

Data | **Embedding** | **Linear algorithm**

$\{(x_i, y_i)\}$





Kernel design | **Kernel algorithm**

Support vector machines

valid kernels:

$$
\begin{aligned}
k(x,x') &= ck_1(\mathbf{x},\mathbf{x}') \\
k(x,x') &= f(\mathbf{x})k_1(\mathbf{x},\mathbf{x}')f(\mathbf{x}') \\
k(x,x') &= q(k_1(\mathbf{x},\mathbf{x}')) \\
k(x,x') &= \exp(k_1(\mathbf{x},\mathbf{x}')) \\
k(x,x') &= k_1(\mathbf{x},\mathbf{x}') + k_2(\mathbf{x},\mathbf{x}') \\
k(x,x') &= k_1(\mathbf{x},\mathbf{x}')k_2(\mathbf{x},\mathbf{x}') \\
k(x,x') &= k_3(\boldsymbol{\phi}(\mathbf{x}),\boldsymbol{\phi}(\mathbf{x}')) \\
k(x,x') &= \mathbf{x}^\top \mathbf{A}\mathbf{x}' \\
k(x,x') &= k_a(\mathbf{x}_a,\mathbf{x}'_a) + k_b(\mathbf{x}_b,\mathbf{x}'_b) \\
k(x,x') &= k_a(\mathbf{x}_a,\mathbf{x}'_a)k_b(\mathbf{x}_b,\mathbf{x}'_b)
\end{aligned}
$$

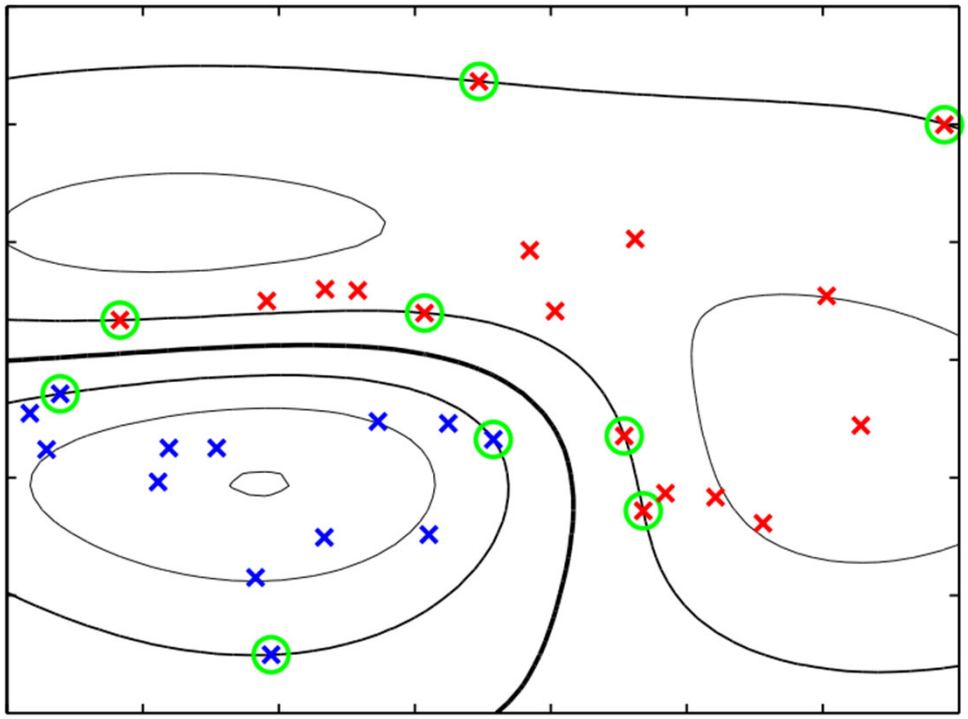
We assume:

- $k_1(\mathbf{x},\mathbf{x}')$ and $k_2(\mathbf{x},\mathbf{x}')$ are valid kernels.
- $c > 0$ is a constant.
- $f(\cdot)$ is any function.
- $q(\cdot)$ is a polynomial with nonnegative coefficients.
- $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$.
- $k_3(\cdot,\cdot)$ is a valid kernel in $\mathbb{R}^M$.
- $\mathbf{A}$ is a symmetric positive semi-definite matrix.
- $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$.
- k_a and k_b are valid kernel functions over their respective spaces.

Example: The Gaussian kernel

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

Support vector machines



from Bishop: Example of synthetic data from two classes in two dimensions showing contours of constant y(x) obtained from a support vector machine having a Gaussian kernel function.

Also shown are the decision boundary, the margin boundaries, and the support vectors.

Note that the data is linearly separable in the Gaussian-kernel space but not in the original space

Classification

I) Instance-based methods:
   1) Nearest neighbor
II) Probabilistic models:
   1) Naïve Bayes
   2) **Logistic Regression**
III) Linear Models:
   1) **Perceptron**
   2) **Support Vector Machine**

IV) Decision Models:
   1) Decision Trees