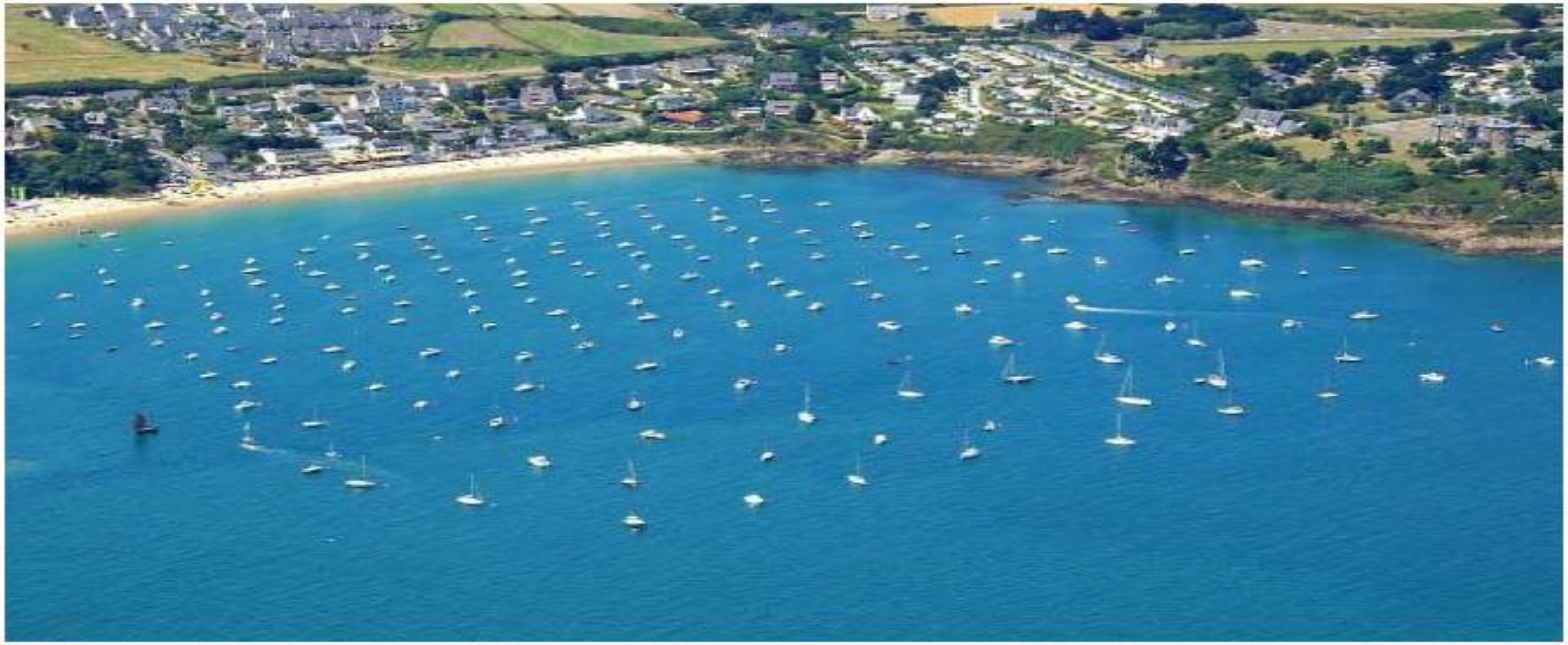
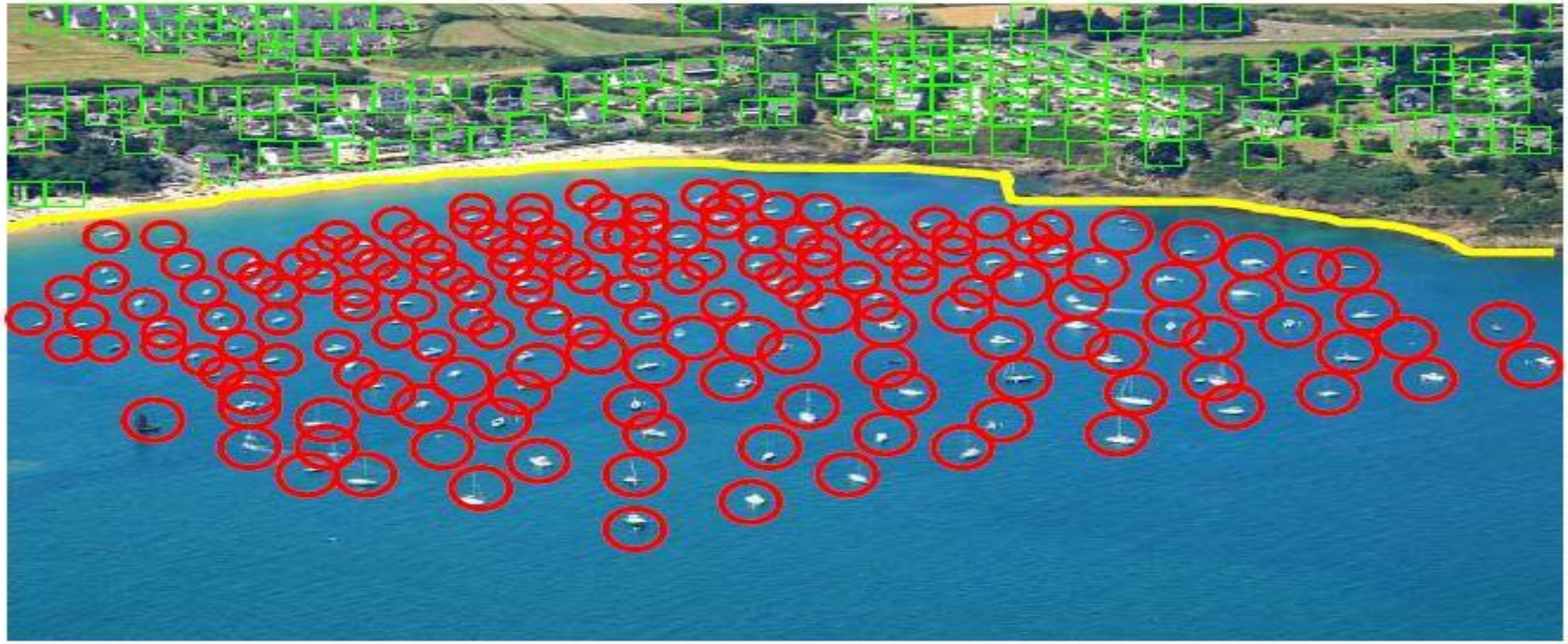
Support Vector Machines

Basic principles of classification



- Want to classify objects as boats and houses.

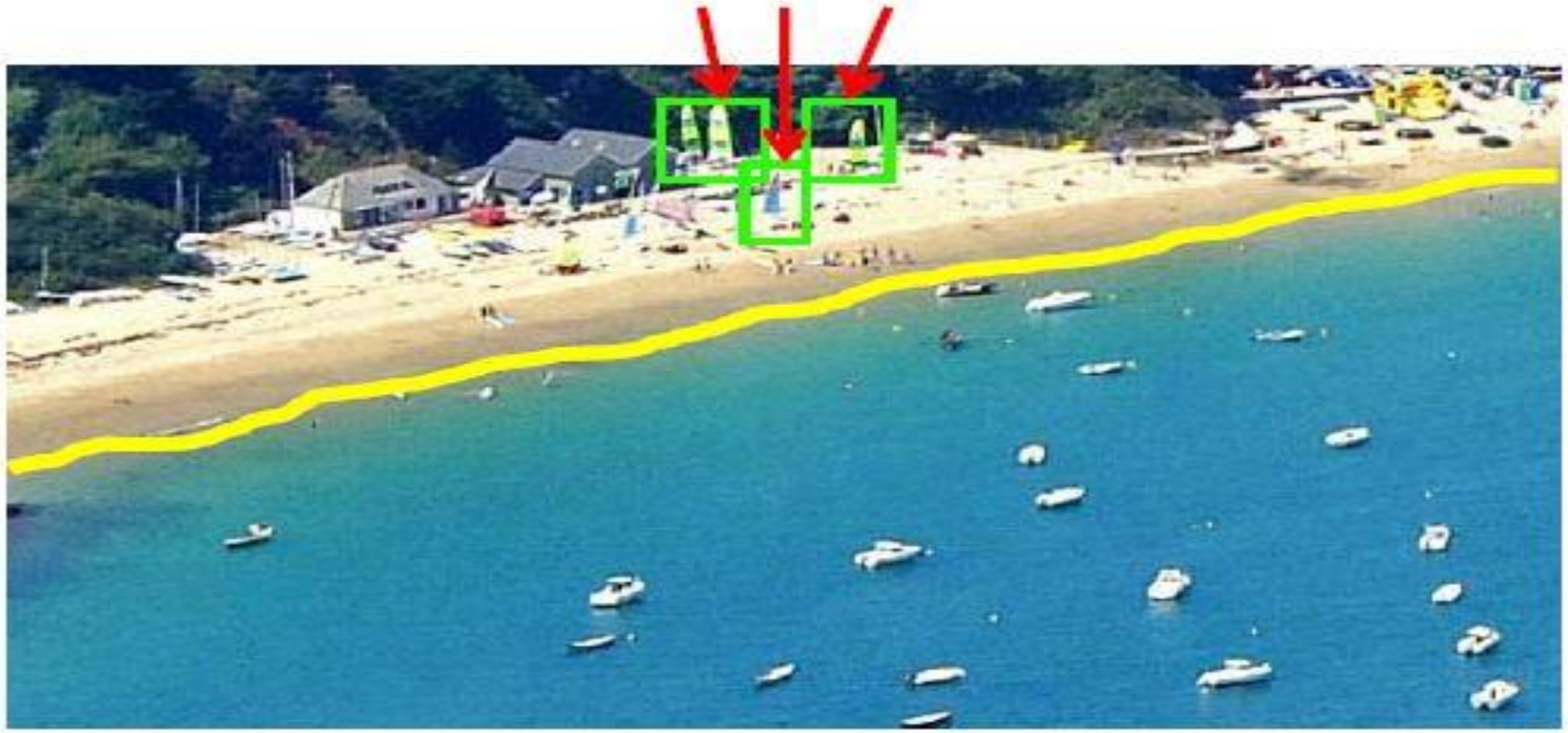
Basic principles of classification



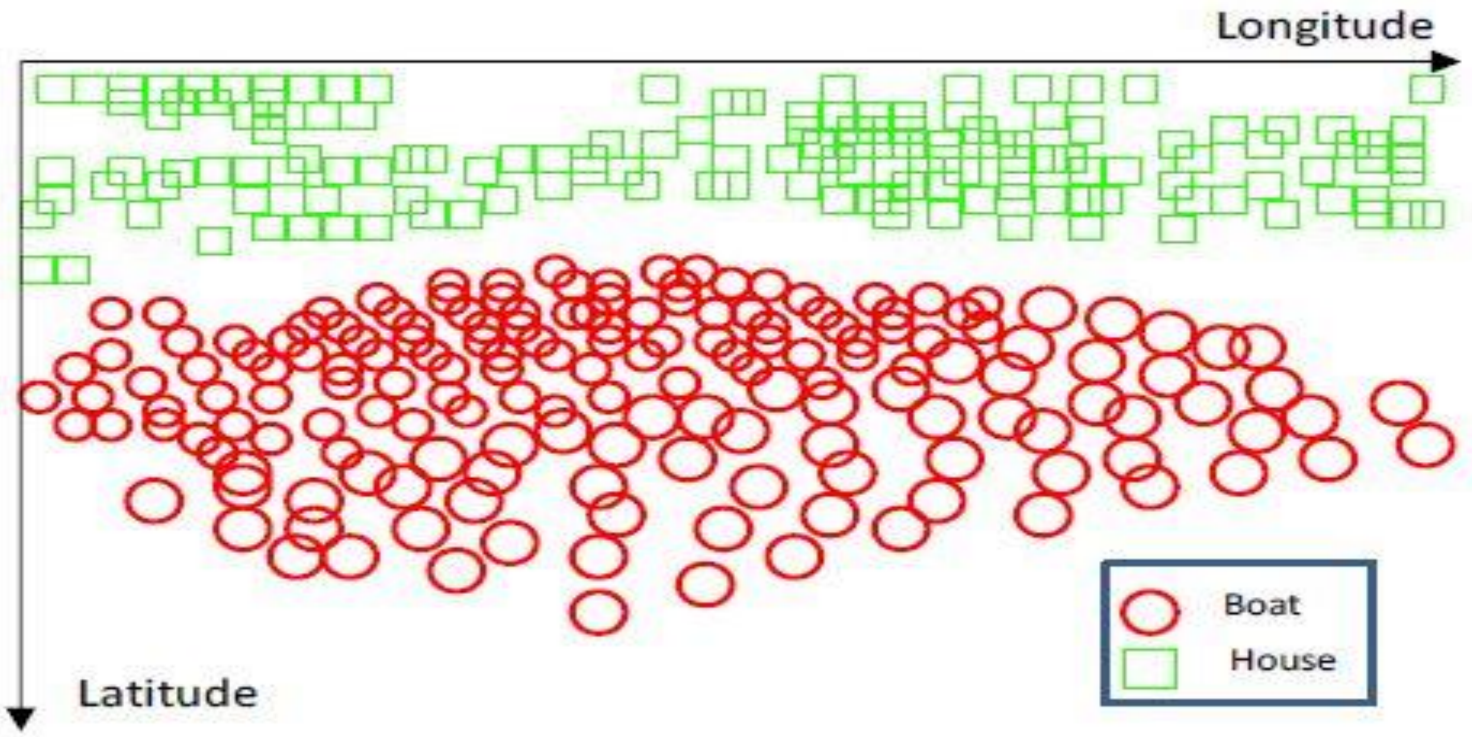
- All objects before the coast line are boats and all objects after the coast line are houses.
- Coast line serves as a *decision surface* that separates two classes.

Basic principles of classification

These boats will be misclassified as houses

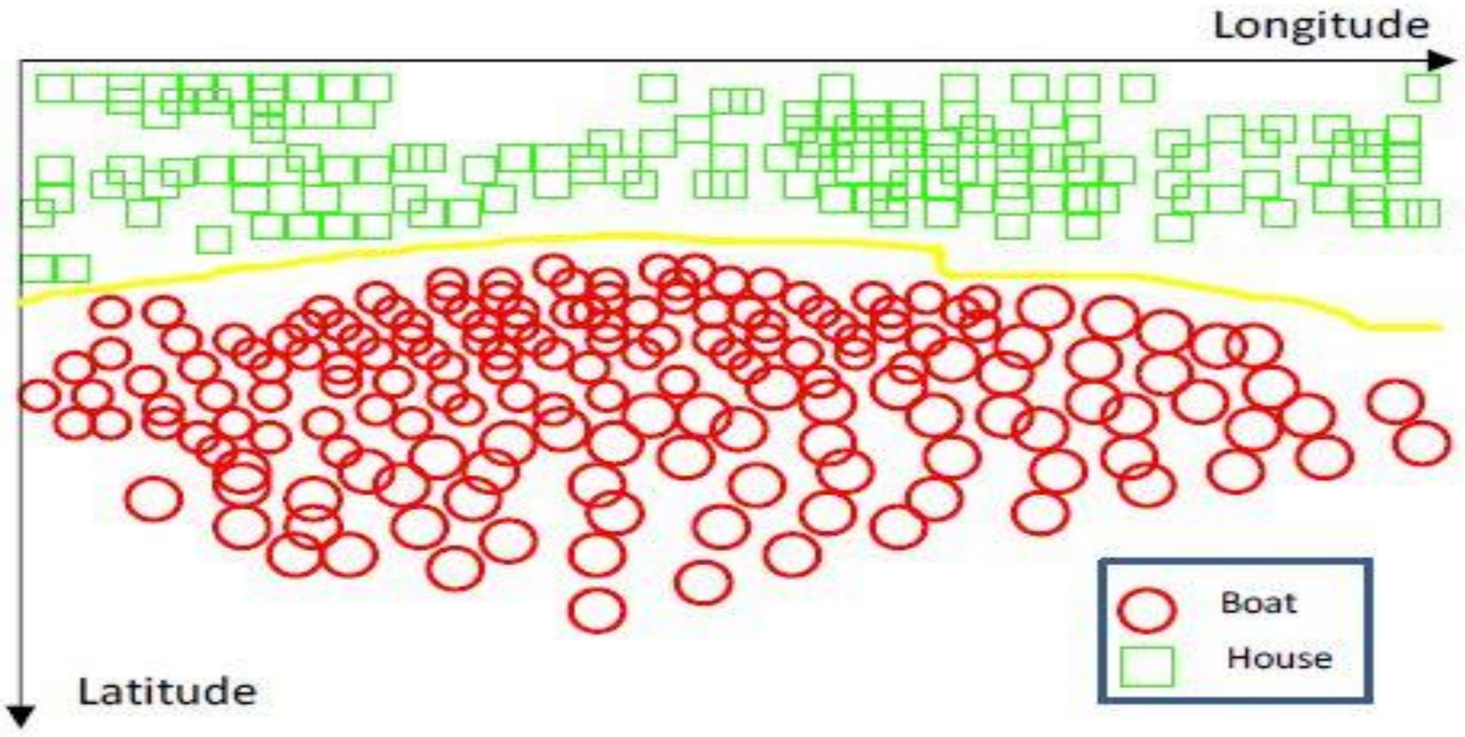


Basic principles of classification



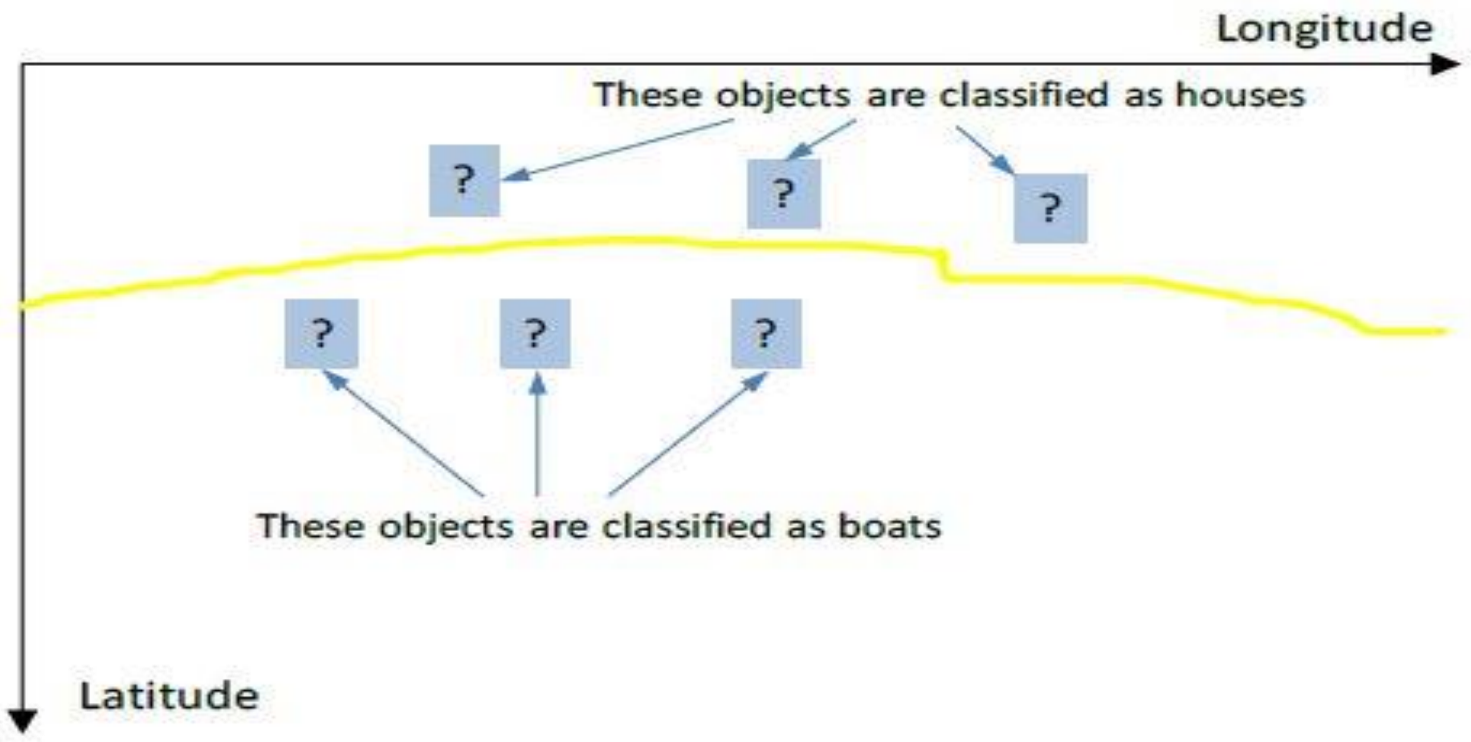
- The methods that build classification models (i.e., "*classification algorithms*") operate very similarly to the previous example.
- First all objects are represented geometrically.

Basic principles of classification



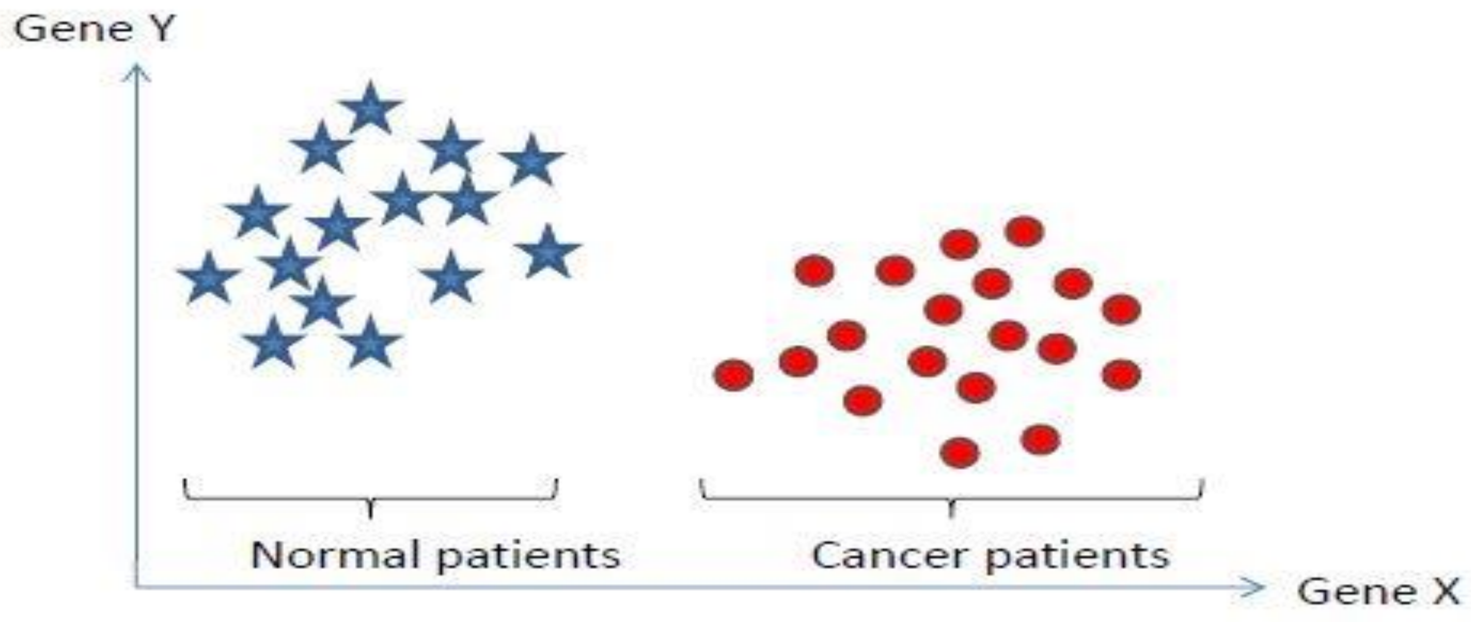
Then the algorithm seeks to find a decision surface that separates classes of objects

Basic principles of classification


Longitude
These objects are classified as houses
?
?
?
?
?
?
These objects are classified as boats
Latitude

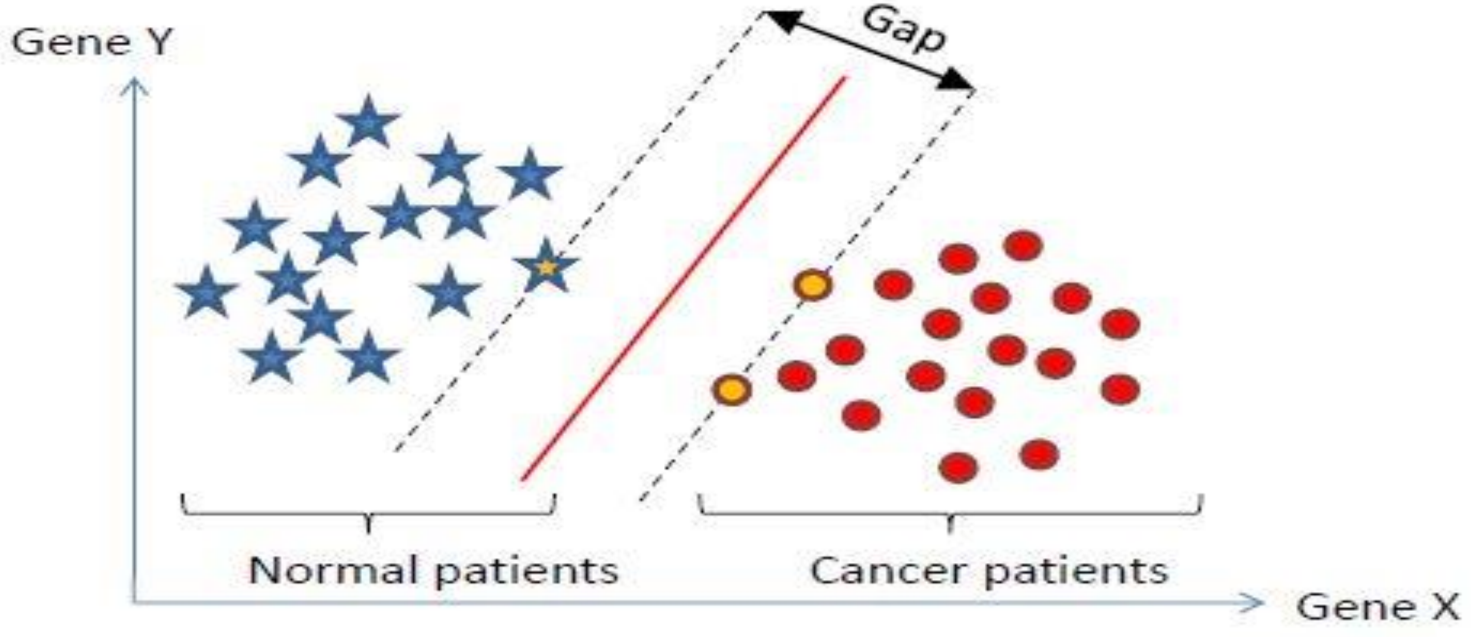
Unseen (new) objects are classified as “boats” if they fall below the decision surface and as “houses” if the fall above it

Main ideas of SVMs



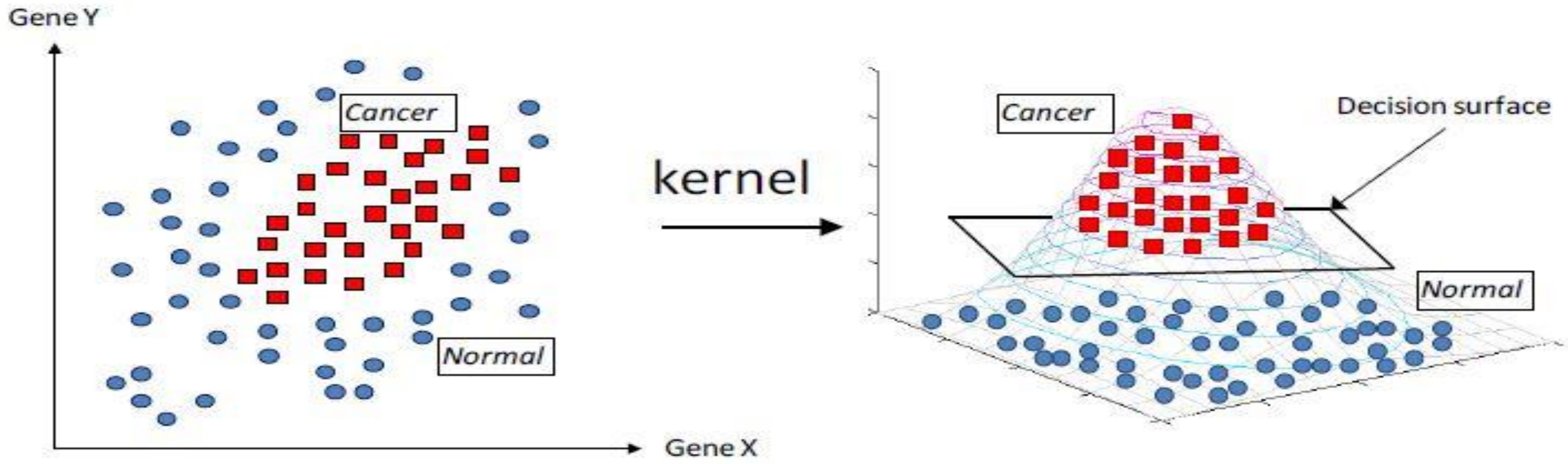
- Consider example dataset described by 2 genes, gene X and gene Y
- Represent patients geometrically (by “vectors”)

Main ideas of SVMs



- Find a linear decision surface ("hyperplane") that can separate patient classes <u>and</u> has the largest distance (i.e., largest "gap" or "margin") between border-line patients (i.e., "support vectors");

Main ideas of SVMs

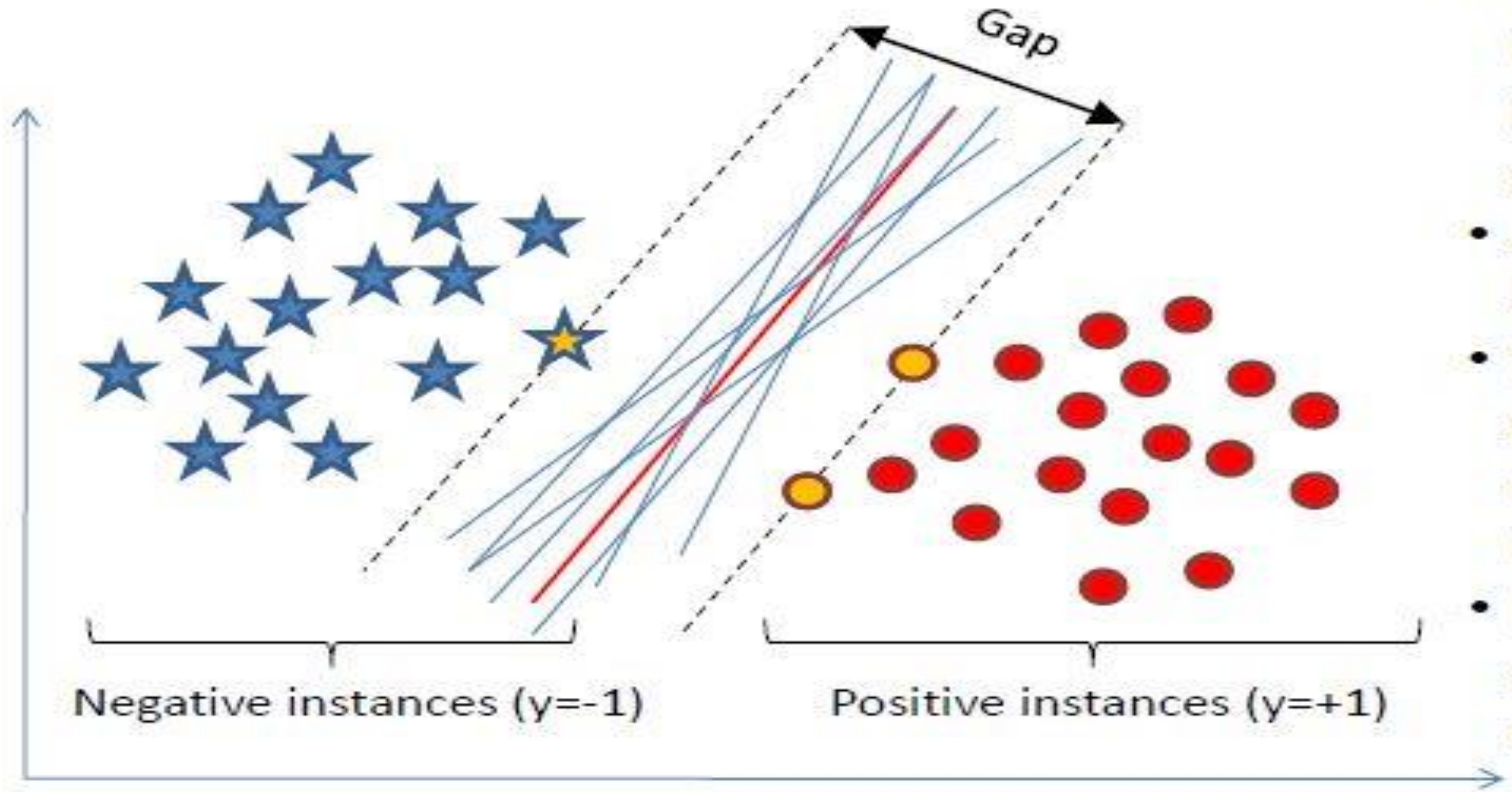


- If such linear decision surface does not exist, the data is mapped into a much higher dimensional space ("feature space") where the separating decision surface is found;
- The feature space is constructed via very clever mathematical projection ("kernel trick").

Support vector machines for binary classification: classical formulation

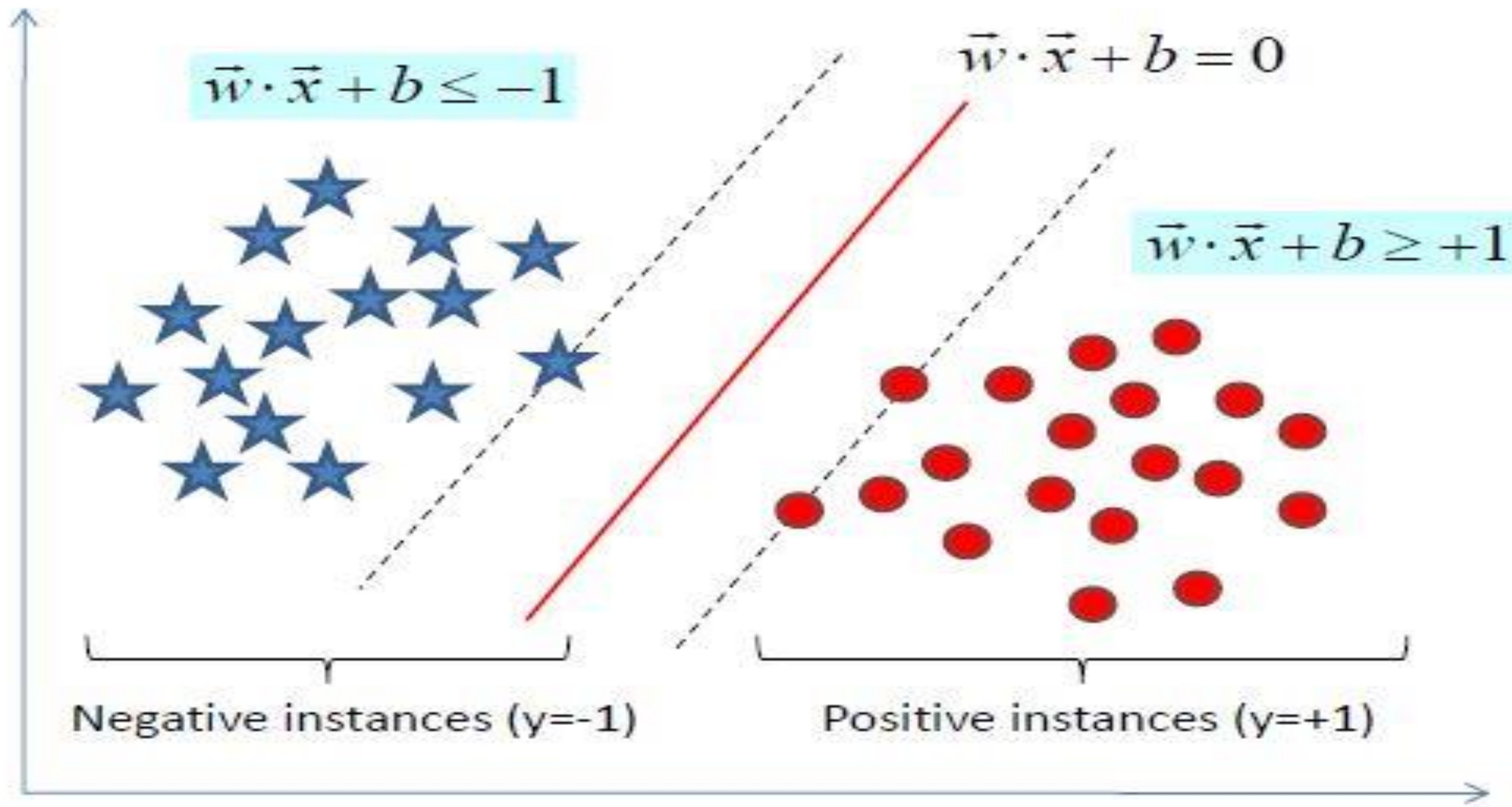
Case 1: Linearly separable data; "Hard-margin" linear SVM

Given training data: $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N \in R^n$
$y_1, y_2, \ldots, y_N \in \{-1, +1\}$



- Want to find a classifier (hyperplane) to separate negative instances from the positive ones.
- An infinite number of such hyperplanes exist.
- SVMs finds the hyperplane that maximizes the gap between data points on the boundaries (so-called "support vectors").
- If the points on the boundaries are not informative (e.g., due to noise), SVMs will not do well.

Statement of linear SVM classifier



$\vec{w}\cdot\vec{x}+b\le -1$
$\vec{w}\cdot\vec{x}+b=0$
$\vec{w}\cdot\vec{x}+b\ge +1$
Negative instances (y=-1)
Positive instances (y=+1)

In addition we need to impose constraints that all instances are correctly classified. In our case:

$$\vec{w}\cdot\vec{x}_i+b\le -1 \text{ if } y_i=-1$$
$$\vec{w}\cdot\vec{x}_i+b\ge +1 \text{ if } y_i=+1$$

Equivalently:

$$\boxed{y_i(\vec{w}\cdot\vec{x}_i+b)\ge 1}$$

<u>In summary:</u>

Want to minimize $\frac{1}{2}\|\vec{w}\|^2$ subject to $y_i(\vec{w}\cdot\vec{x}_i+b)\ge 1$ for $i=1,\dots,N$

Then given a new instance x, the classifier is $f(\vec{x})=sign(\vec{w}\cdot\vec{x}+b)$

SVM optimization problem: Primal formulation

Minimize $\boxed{\frac{1}{2}\sum_{i=1}^{n} w_i^2}$ subject to $\boxed{y_i(\vec{w}\cdot\vec{x}_i + b) - 1 \geq 0}$ for $i = 1,\ldots,N$

Objective function

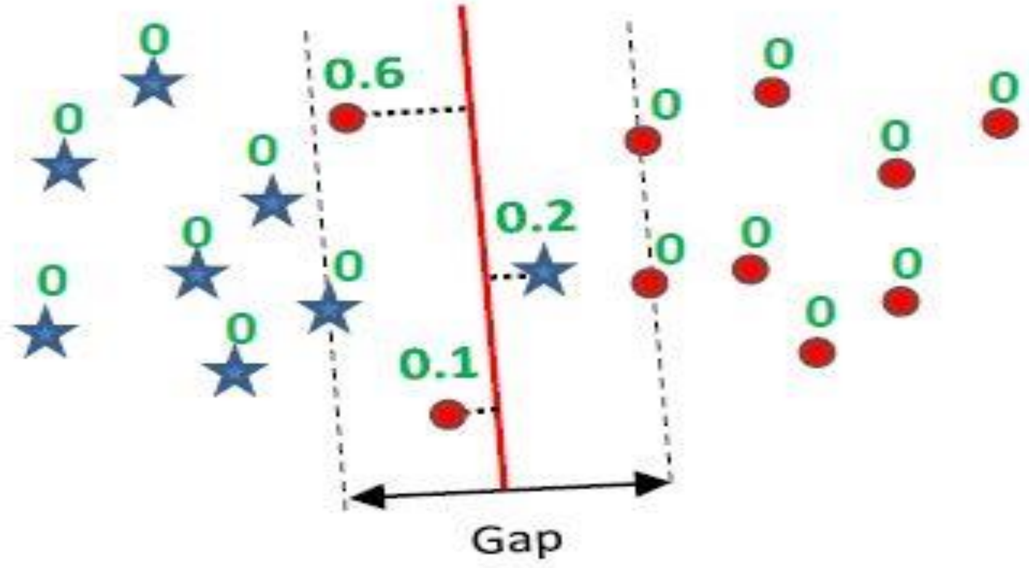
Constraints

- This is called "*primal formulation of linear SVMs*".
- It is a convex quadratic programming (QP) optimization problem with *n* variables (w_i, $i = 1,\ldots,n$), where *n* is the number of features in the dataset.

Case 2: Not linearly separable data; "Soft-margin" linear SVM

What if the data is not linearly separable? E.g., there are outliers or noisy measurements, or the data is slightly non-linear.

Want to handle this case without changing the family of decision functions.



Approach:

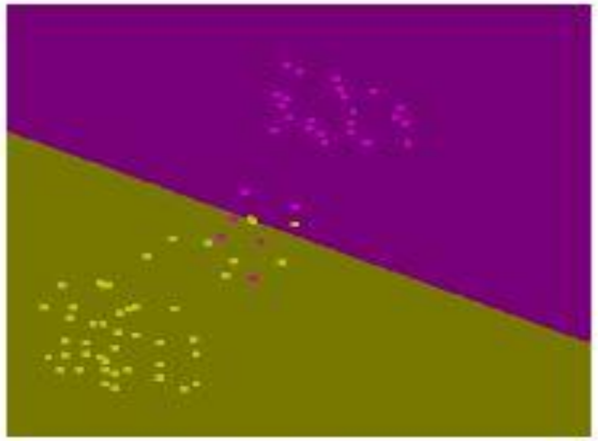
Assign a "slack variable" to each instance $\xi_i \geq 0$, which can be thought of distance from the separating hyperplane if an instance is misclassified and 0 otherwise.

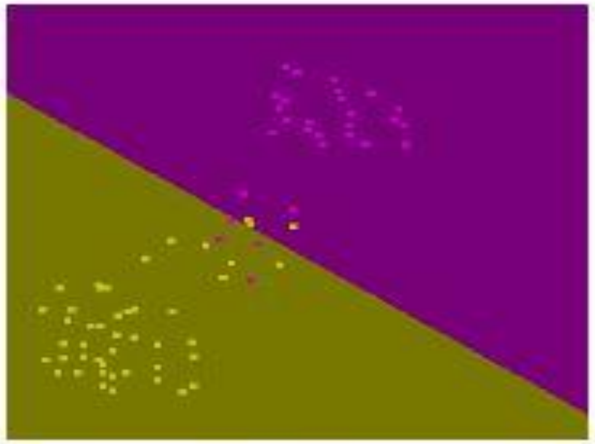
Want to minimize $\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\xi_i$ subject to $y_i(\vec{w}\cdot\vec{x}_i + b) \geq 1 - \xi_i$ for $i = 1,\ldots,N$

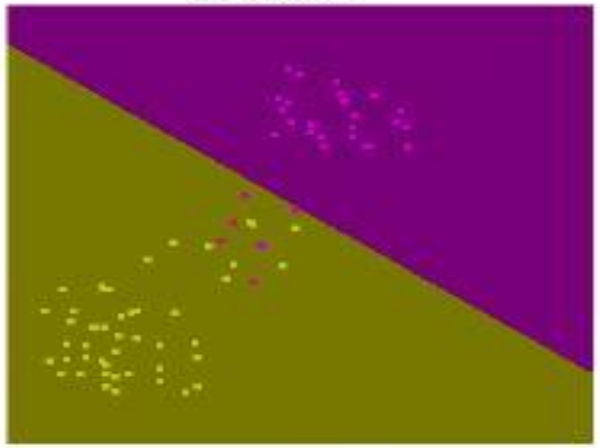
Then given a new instance x, the classifier is $f(x) = sign(\vec{w}\cdot\vec{x} + b)$

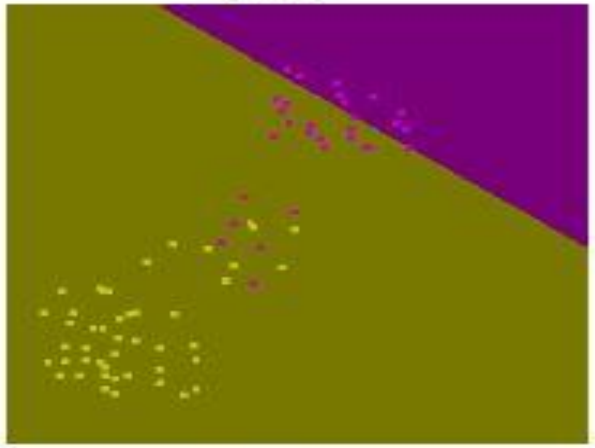
Parameter *C* in soft-margin SVM

Minimize $\frac{1}{2}\|\vec{w}\|^2 + \boxed{C}\sum_{i=1}^{N}\xi_i$ subject to $y_i(\vec{w}\cdot\vec{x}_i + b) \geq 1 - \xi_i$ for $i = 1,\ldots,N$


C=100

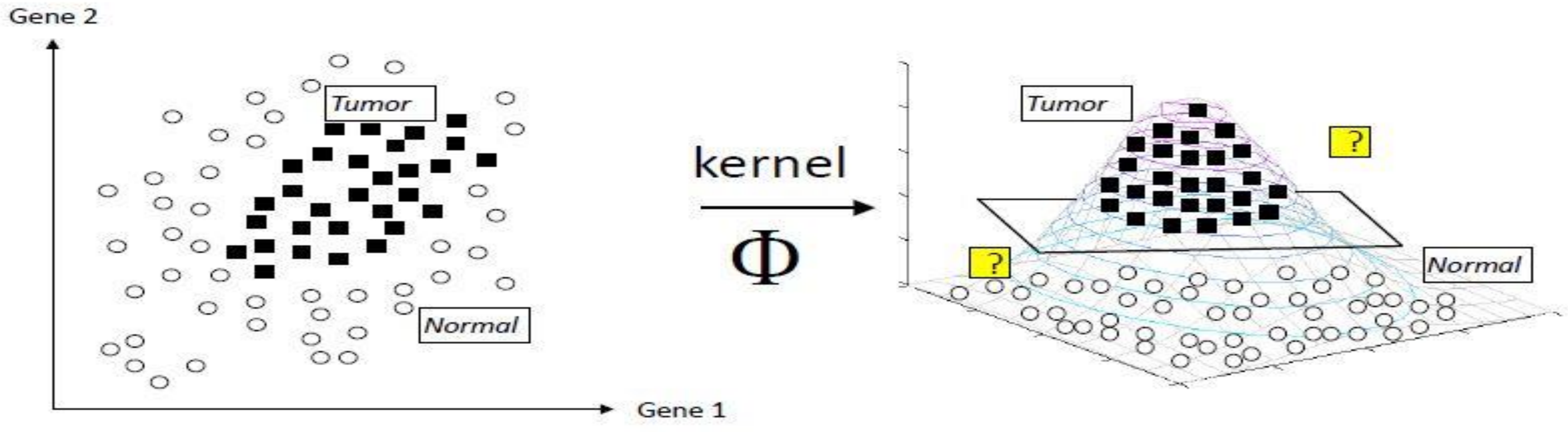

C=1


C=0.15


C=0.1

- When *C* is very large, the soft-margin SVM is equivalent to hard-margin SVM;
- When *C* is very small, we admit misclassifications in the training data at the expense of having w-vector with small norm;
- *C* has to be selected for the distribution at hand as it will be discussed later in this tutorial.

Case 3: Not linearly separable data; Kernel trick



Data is not linearly separable in the input space

Data is linearly separable in the feature space obtained by a kernel

$$\Phi : \mathbf{R}^N \to \mathbf{H}$$

Kernel trick

Original data $\vec{x}$ (in input space)

$$f(x) = sign(\vec{w} \cdot \vec{x} + b)$$

$$\vec{w} = \sum_{i=1}^{N} \alpha_i y_i \vec{x}_i$$

Data in a higher dimensional feature space $\Phi(\vec{x})$

$$f(x) = sign(\vec{w} \cdot \Phi(\vec{x}) + b)$$

$$\vec{w} = \sum_{i=1}^{N} \alpha_i y_i \Phi(\vec{x}_i)$$

$$f(x) = sign(\sum_{i=1}^{N} \alpha_i y_i \Phi(\vec{x}_i) \cdot \Phi(\vec{x}) + b)$$

$$f(x) = sign(\sum_{i=1}^{N} \alpha_i y_i K(\vec{x}_i, \vec{x}) + b)$$

Therefore, we do not need to know Φ explicitly, we just need to define function $K(\cdot, \cdot)$: $\mathbb{R}^N \times \mathbb{R}^N \rightarrow \mathbb{R}$.

Not every function $\mathbb{R}^N \times \mathbb{R}^N \rightarrow \mathbb{R}$ can be a valid kernel; it has to satisfy so-called Mercer conditions. Otherwise, the underlying quadratic program may not be solvable.

Popular kernels

A kernel is a dot product in *some* feature space:

$$K(\vec{x}_i, \vec{x}_j) = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$$

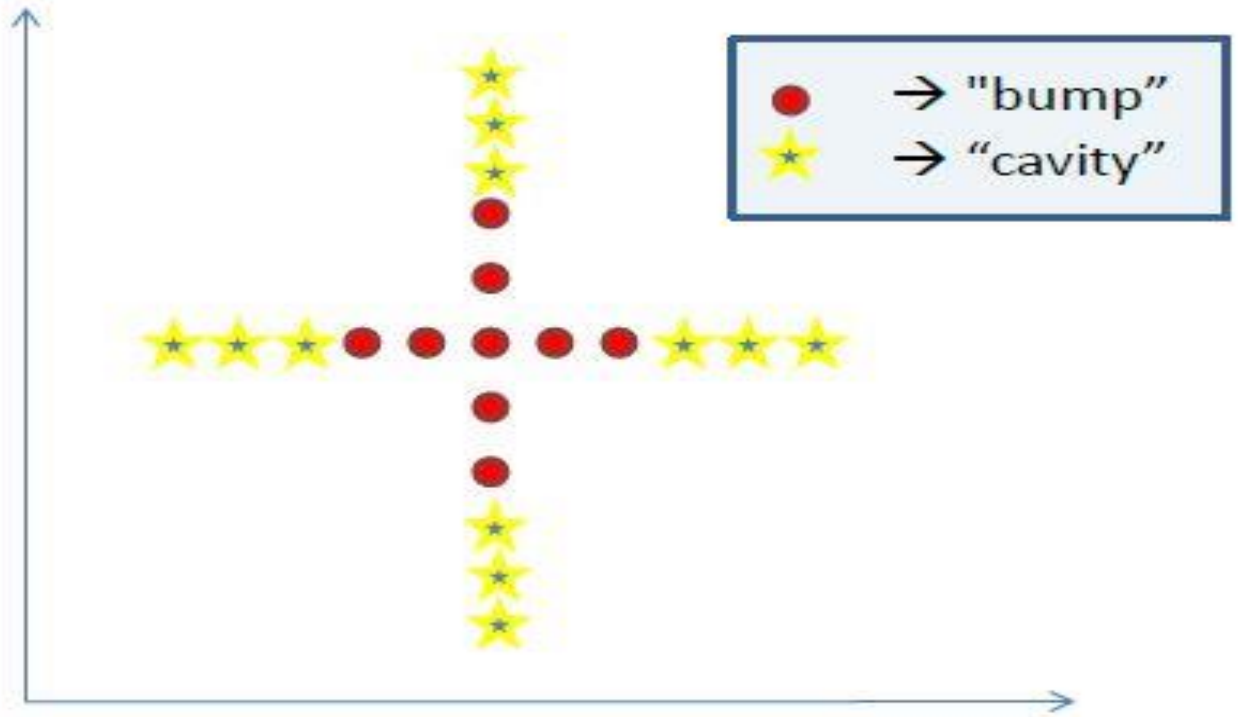
Examples:

$K(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j$	Linear kernel
$K(\vec{x}_i, \vec{x}_j) = \exp(-\gamma \left\| \vec{x}_i - \vec{x}_j \right\|^2)$	Gaussian kernel
$K(\vec{x}_i, \vec{x}_j) = \exp(-\gamma \left\| \vec{x}_i - \vec{x}_j \right\|)$	Exponential kernel
$K(\vec{x}_i, \vec{x}_j) = (p + \vec{x}_i \cdot \vec{x}_j)^q$	Polynomial kernel
$K(\vec{x}_i, \vec{x}_j) = (p + \vec{x}_i \cdot \vec{x}_j)^q \exp(-\gamma \left\| \vec{x}_i - \vec{x}_j \right\|^2)$	Hybrid kernel
$K(\vec{x}_i, \vec{x}_j) = \tanh(k\vec{x}_i \cdot \vec{x}_j - \delta)$	Sigmoidal

Understanding the Gaussian kernel

Consider Gaussian kernel: $K(\vec{x}, \vec{x}_j) = \exp(-\gamma \left\| \vec{x} - \vec{x}_j \right\|^2)$

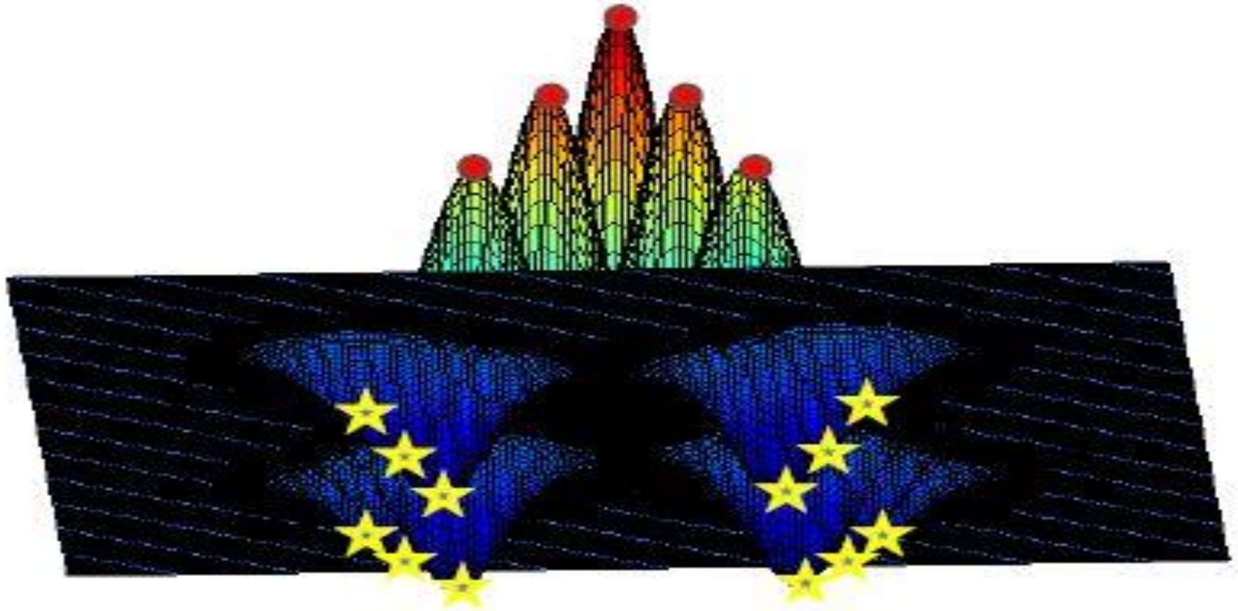
Geometrically, this is a "bump" or "cavity" centered at the training data point $\vec{x}_j$:

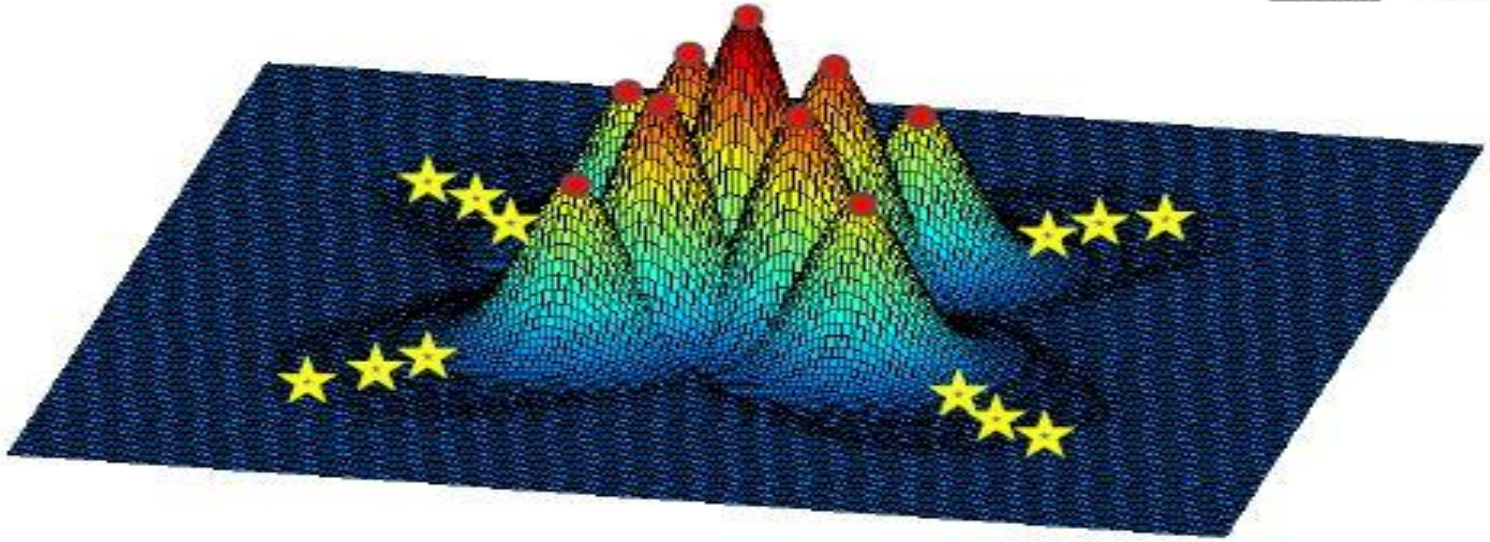


The resulting mapping function is a **combination** of bumps and cavities.

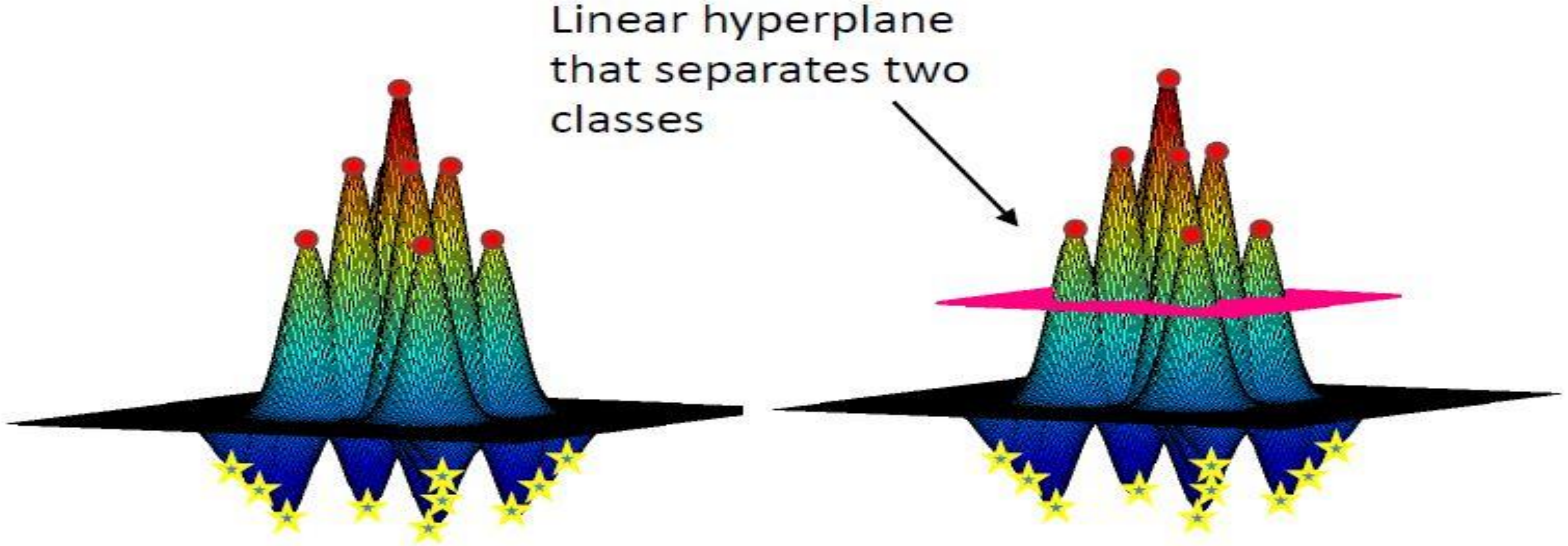
Understanding the Gaussian kernel

Several more views of the data is mapped to the feature space by Gaussian kernel





Understanding the Gaussian kernel



Understanding the polynomial kernel

Consider polynomial kernel: $K(\vec{x}_i, \vec{x}_j) = (1 + \vec{x}_i \cdot \vec{x}_j)^3$

Assume that we are dealing with 2-dimensional data (i.e., in $\mathbb{R}^2$). Where will this kernel map the data?

2-dimensional space

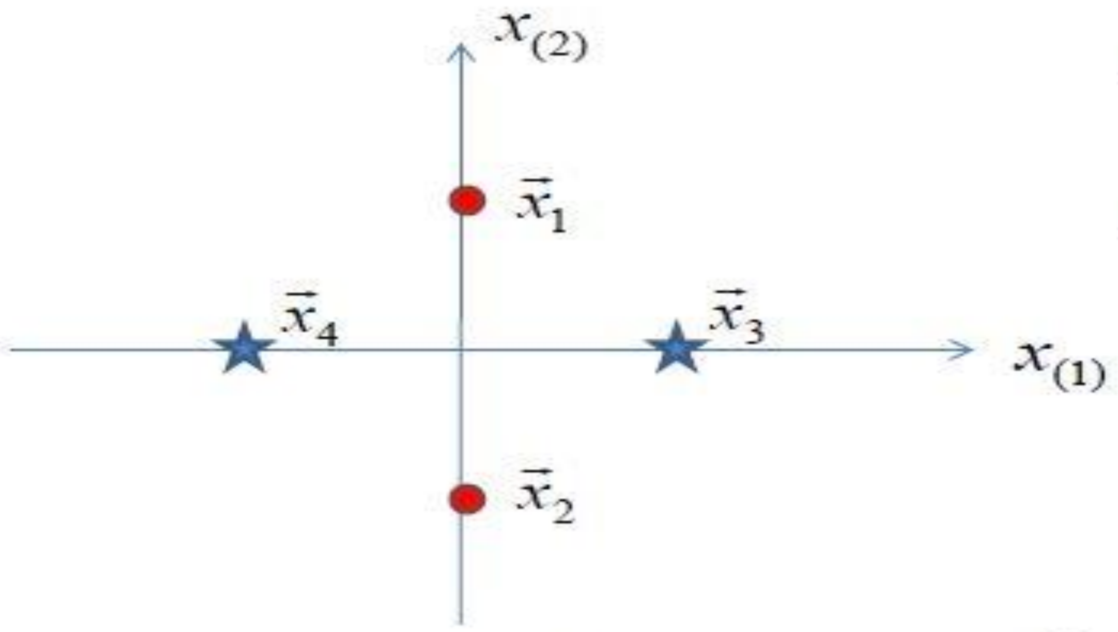
$$x_{(1)} \quad x_{(2)}$$

$$\Big\downarrow \text{ kernel}$$

10-dimensional space

$$1 \quad x_{(1)} \quad x_{(2)} \quad x_{(1)}^2 \quad x_{(2)}^2 \quad x_{(1)}x_{(2)} \quad x_{(1)}^3 \quad x_{(2)}^3 \quad x_{(1)}x_{(2)}^2 \quad x_{(1)}^2 x_{(2)}$$

Example of benefits of using a kernel



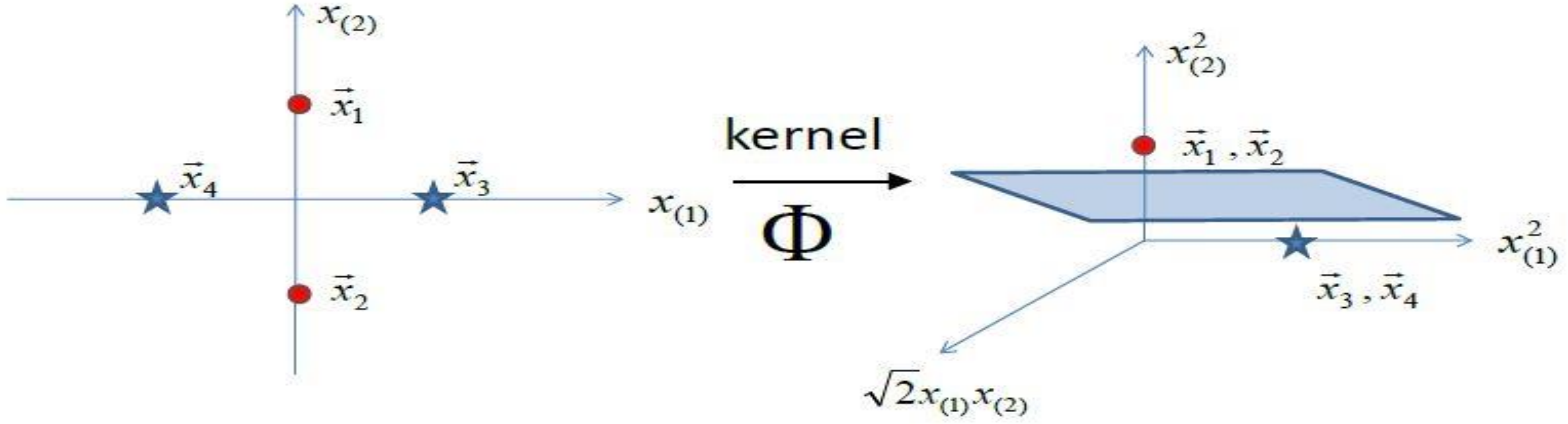
- Data is not linearly separable in the input space ($\mathbb{R}^2$).
- Apply kernel $K(\vec{x},\vec{z}) = (\vec{x}\cdot\vec{z})^2$ to map data to a higher dimensional space (3-dimensional) where it is linearly separable.

$$K(\vec{x},\vec{z}) = (\vec{x}\cdot\vec{z})^2 = \left[\begin{pmatrix} x_{(1)} \\ x_{(2)} \end{pmatrix} \cdot \begin{pmatrix} z_{(1)} \\ z_{(2)} \end{pmatrix}\right]^2 = \left[x_{(1)}z_{(1)} + x_{(2)}z_{(2)}\right]^2 =$$

$$= x_{(1)}^2 z_{(1)}^2 + 2x_{(1)}z_{(1)}x_{(2)}z_{(2)} + x_{(2)}^2 z_{(2)}^2 = \begin{pmatrix} x_{(1)}^2 \\ \sqrt{2}x_{(1)}x_{(2)} \\ x_{(2)}^2 \end{pmatrix} \cdot \begin{pmatrix} z_{(1)}^2 \\ \sqrt{2}z_{(1)}z_{(2)} \\ z_{(2)}^2 \end{pmatrix} = \Phi(\vec{x})\cdot\Phi(\vec{z})$$

Example of benefits of using a kernel

Therefore, the explicit mapping is $\Phi(\vec{x}) = \begin{pmatrix} x_{(1)}^2 \\ \sqrt{2}x_{(1)}x_{(2)} \\ x_{(2)}^2 \end{pmatrix}$



$x_{(2)}$
$\vec{x}_1$
$\vec{x}_4$
$\vec{x}_3$
$x_{(1)}$
$\vec{x}_2$
kernel
Φ
$x_{(2)}^2$
$\vec{x}_1, \vec{x}_2$
$x_{(1)}^2$
$\vec{x}_3, \vec{x}_4$
$\sqrt{2}x_{(1)}x_{(2)}$