



# CSC4007 Advanced Machine Learning

## **Lesson 05:** Support Vector Machine

by Vien Ngo  
EEECS / ECIT / DSSC

# Support Vector Machine: Introduction

- Support vector machine (SVM) algorithms are among the best “off-the-shelf” supervised learning algorithms.
- Key notes about SVM:
  - **Margins** (the idea of separating data with “*large gap*”)
  - **Features with potentially infinite dimensions?**
  - A SVM is a discriminant classifier formally defined by a separating hyperplane.

# Outline

- Support vector machine (SVM) for binary-classification
- Support vector machine for regression
- Support vector machine for multi-class classification

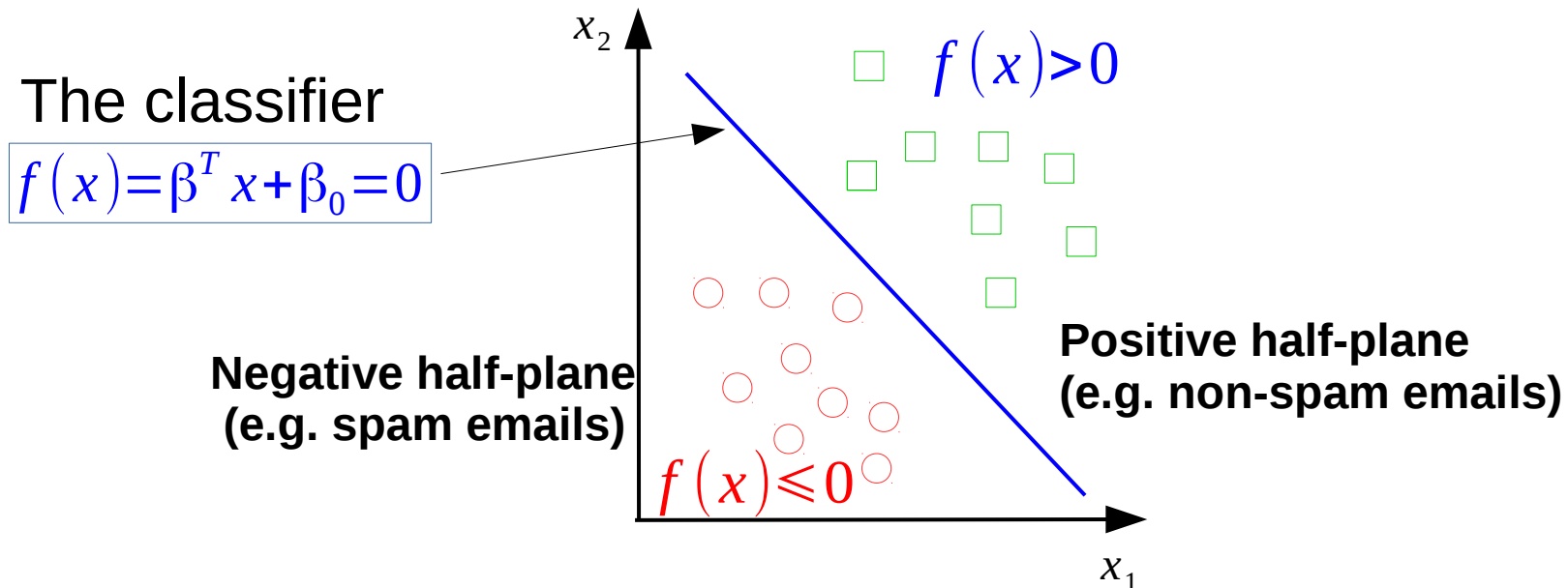
# Outline

- **Support vector machine (SVM) for binary-classification**
- Support vector machine for regression
- Support vector machine for multi-class classification

# Binary Classification: Revisit

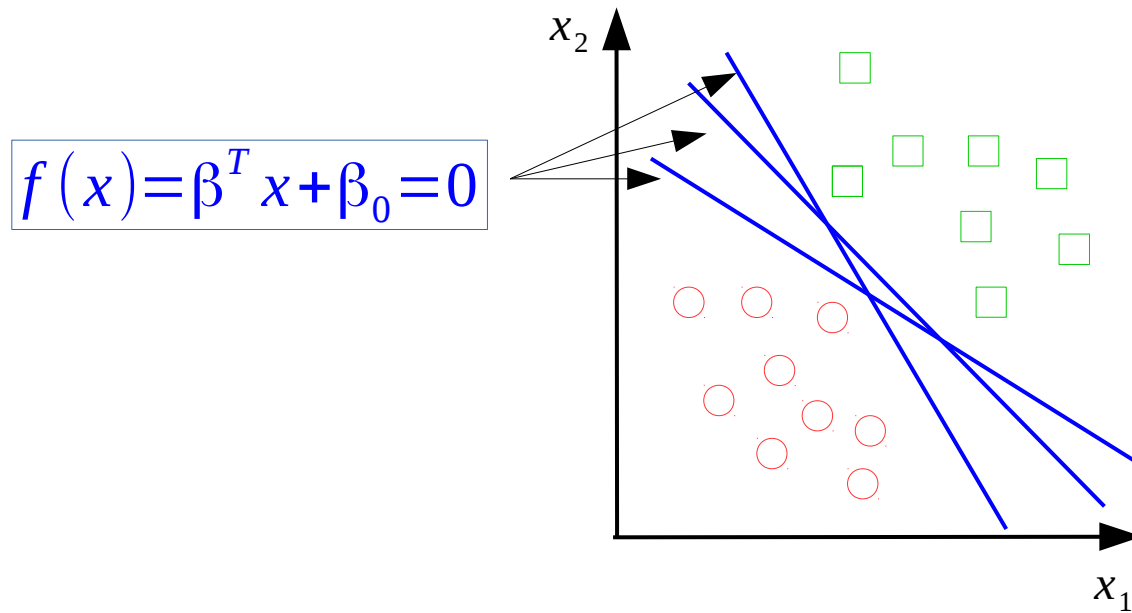
- **Binary classifier** (assuming the output label  $y$  is  $\{-1, +1\}$ ):
  - linear discriminant function: e.g.  $f(x) = \beta_0 + \beta^T x = \beta_0 + \beta_1 x_1 + \beta_2 x_2$
- **The decision boundary is at  $f(x) = \beta^T x + \beta_0 = 0$ . Prediction is:**
  - If  $f(x) > 0$ , then  $y = +1$
  - If  $f(x) \leq 0$ , then  $y = -1$

$y = \text{sign}(f(x)) = \text{sign}(\beta^T x + \beta_0)$



# Which classifier?

- Which linear classifier? e.g. when data is linearly separable

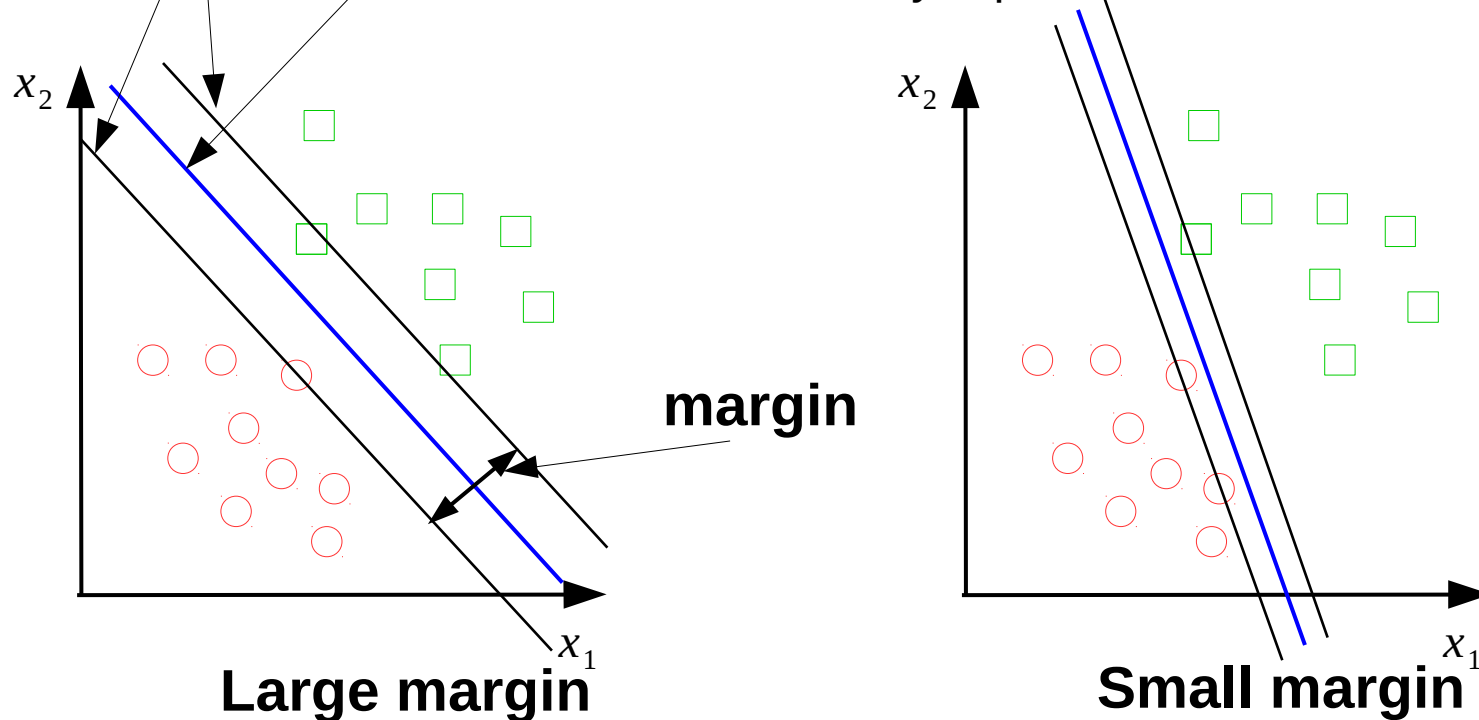


**All these classifiers have an accuracy of 100%**

# Which classifier?

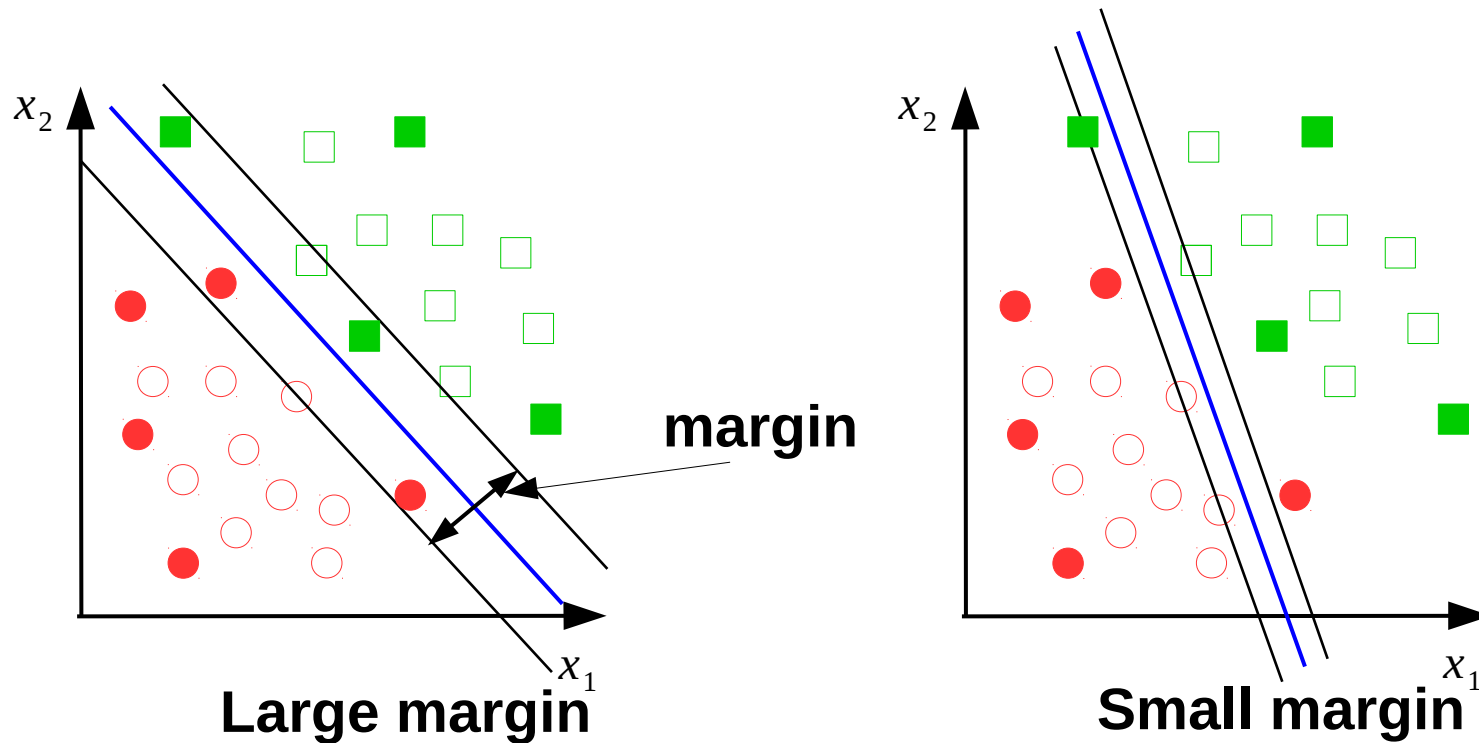
The two parallel hyper-planes crossing to points that are closest to the classifier

The classifier:  
drawn at the middle of two hyper-planes



- **Large margin** is preferred (a “*buffer zone*” around the boundary): because the **chance** of a new point still on the **correct** side is **higher**.
  - e.g. you are driving on a road with bigger lanes, so if your driving becomes a bit *sleepy*, then *your chance to be still on your correct lane must be higher than driving on a road with smaller lanes.* (= ***noisy data generation***)

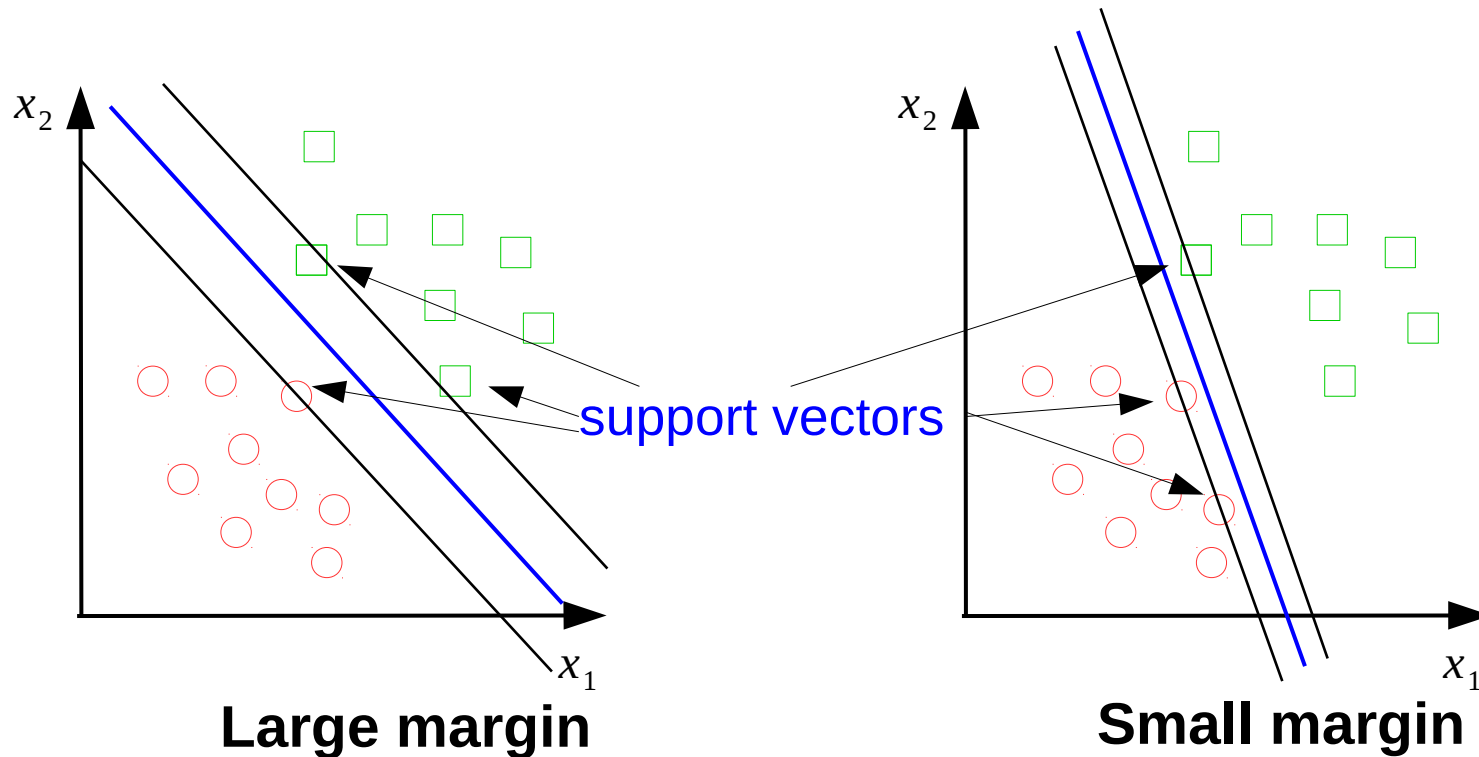
# Which classifier?



- **Large margin** is preferred (a “*buffer zone*” around the boundary): because the **chance** of a new point still on the **correct** side is **higher**.

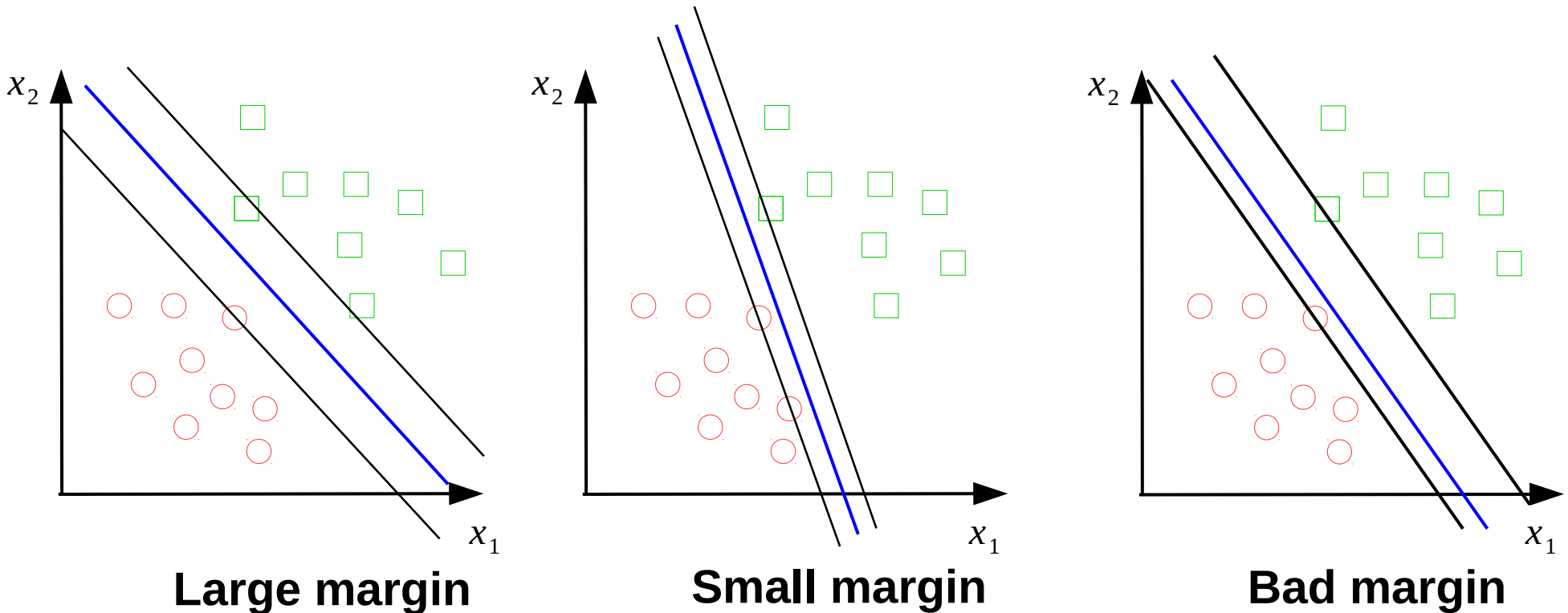


# Which classifier?



- The line that maximizes the margin is a “*good bet*”.
  - The model class of “hyper-planes with a margin of  $m$ ” has the **best generalization ability** if  $m$  is big.
  - Try to separate positive and negative instances far away from each other
- This maximum-margin separator is determined by a subset of the datapoints.
  - Datapoints in this subset are called “support vectors”.
  - It will be computationally useful if only a small fraction of the datapoints are support vectors, because we use the support vectors to decide which side of the separator a test case is on.

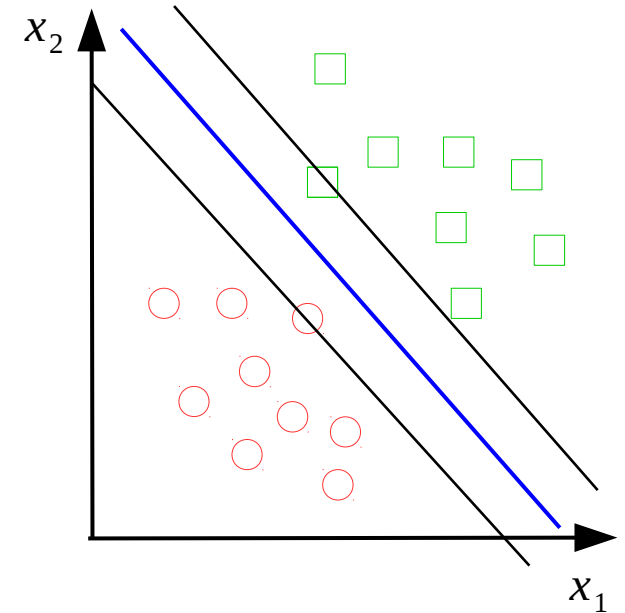
# Why Max Margin?



- Intuitively this feels safest.
- If we've made a small error in the location of the boundary, this gives us least chance of causing a misclassification.
- Empirically it works very very well.
- Leave-one-out cross validation is easy since the model is immune to removal of any non support-vector datapoints

# Binary Classification: Revisit

- **prediction rule is**
  - If  $f(x) = \beta^T x + \beta_0 > 0$  then  $y = +1$
  - If  $f(x) = \beta^T x + \beta_0 \leq 0$  then  $y = -1$
- we can always scale  $\beta$  such that
  - If  $f(x) = \beta^T x + \beta_0 \geq 1$  then  $y = +1$
  - If  $f(x) = \beta^T x + \beta_0 \leq -1$  then  $y = -1$



If there is *epsilon* margin, then divide both sides of the inequality by *epsilon* (this scaling helps us to compute the margin more effectively)

# Max-Margin Problem: Setting

- Given a data set  $D = \{x_i, y_i\}_{i=1}^n$  for binary classification
  - Assume the data is **linearly separable**

- Find a linear classifier of  $\beta$  that has:

- **Decision rule:**

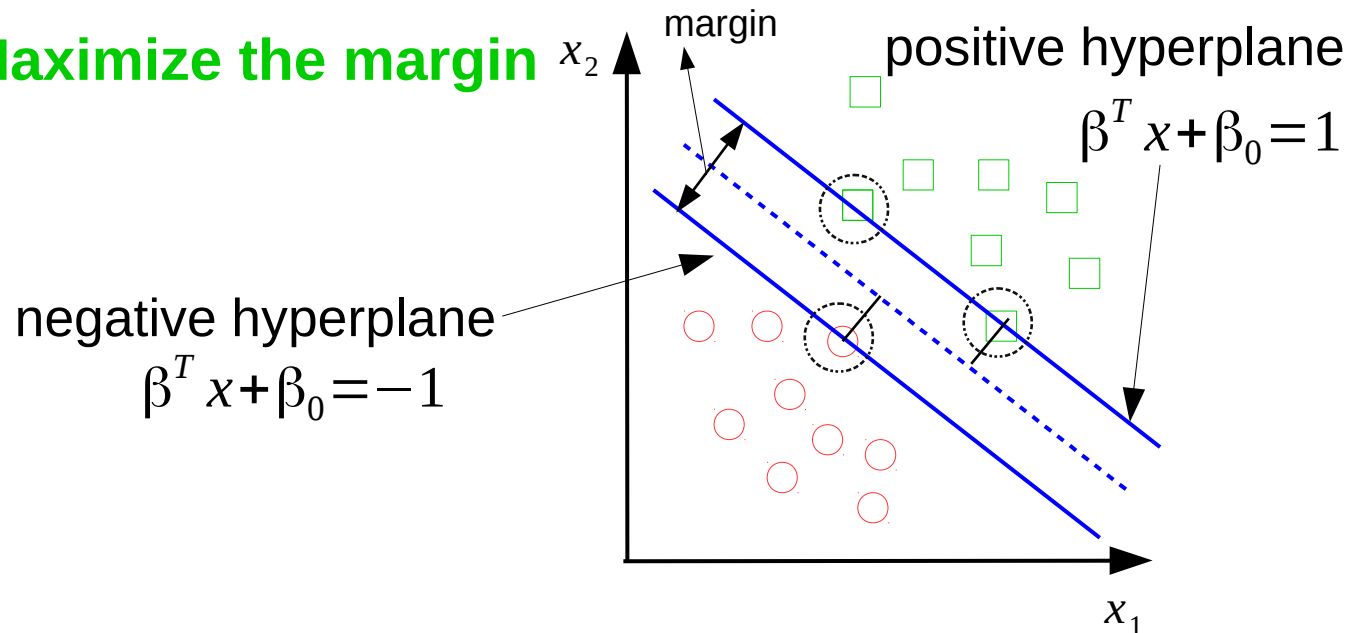
- If  $f(x) = \beta^T x + \beta_0 \geq 1$  then  $y = +1$
- If  $f(x) = \beta^T x + \beta_0 \leq -1$  then  $y = -1$

- $D = \{x_i, y_i\}_{i=1}^n$  **are correctly classified**

- **Objective: Maximize the margin**

Exist a (hyper)-plane to separate the data

$$y_i(\beta^T x_i + \beta_0) \geq 1$$



# Max-margin problem: Computing the margin?

- Given a data set  $D = \{x_i, y_i\}_{i=1}^n$  for binary classification
- The margin** is the **distance** between two **+/- hyperplanes**
- Distance from a point to a line:  **$d=??$**  (example: compute  $d$  in 2D)

**+) normal vector** of the line  $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$  is  $\beta = (\beta_1, \beta_2)$

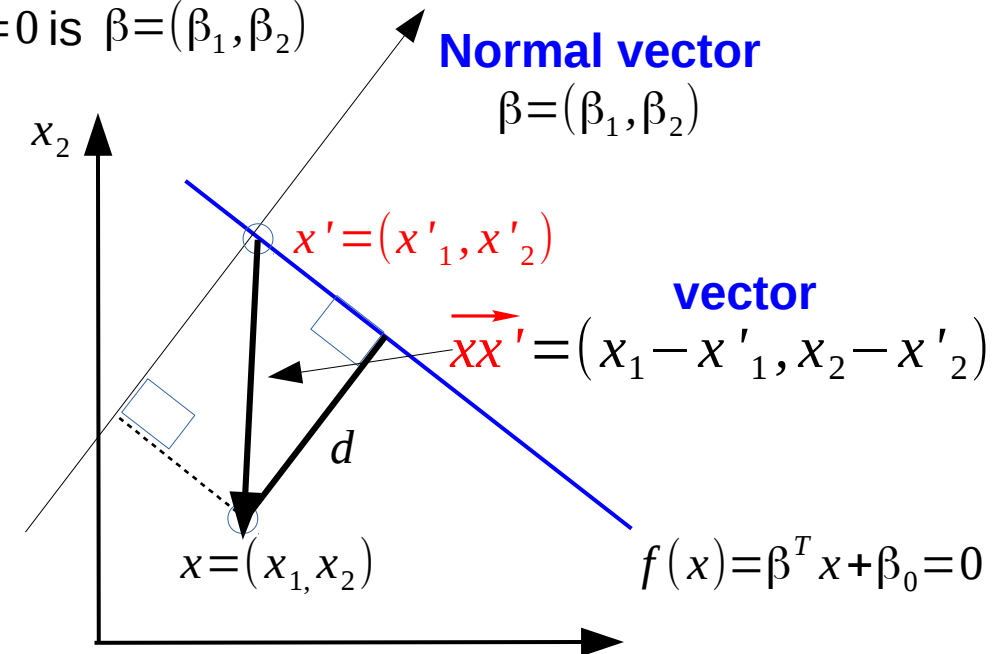
**+)  $x'$**  is any point on the line  $f(x)$

$$\beta^T x' + \beta_0 = \beta_0 + \beta_1 x'_1 + \beta_2 x'_2 = 0$$

**+) so,  $d$  is also equal to the projection of the vector  $\overrightarrow{xx'}$  onto the normal vector.** The length of this projection is computed as

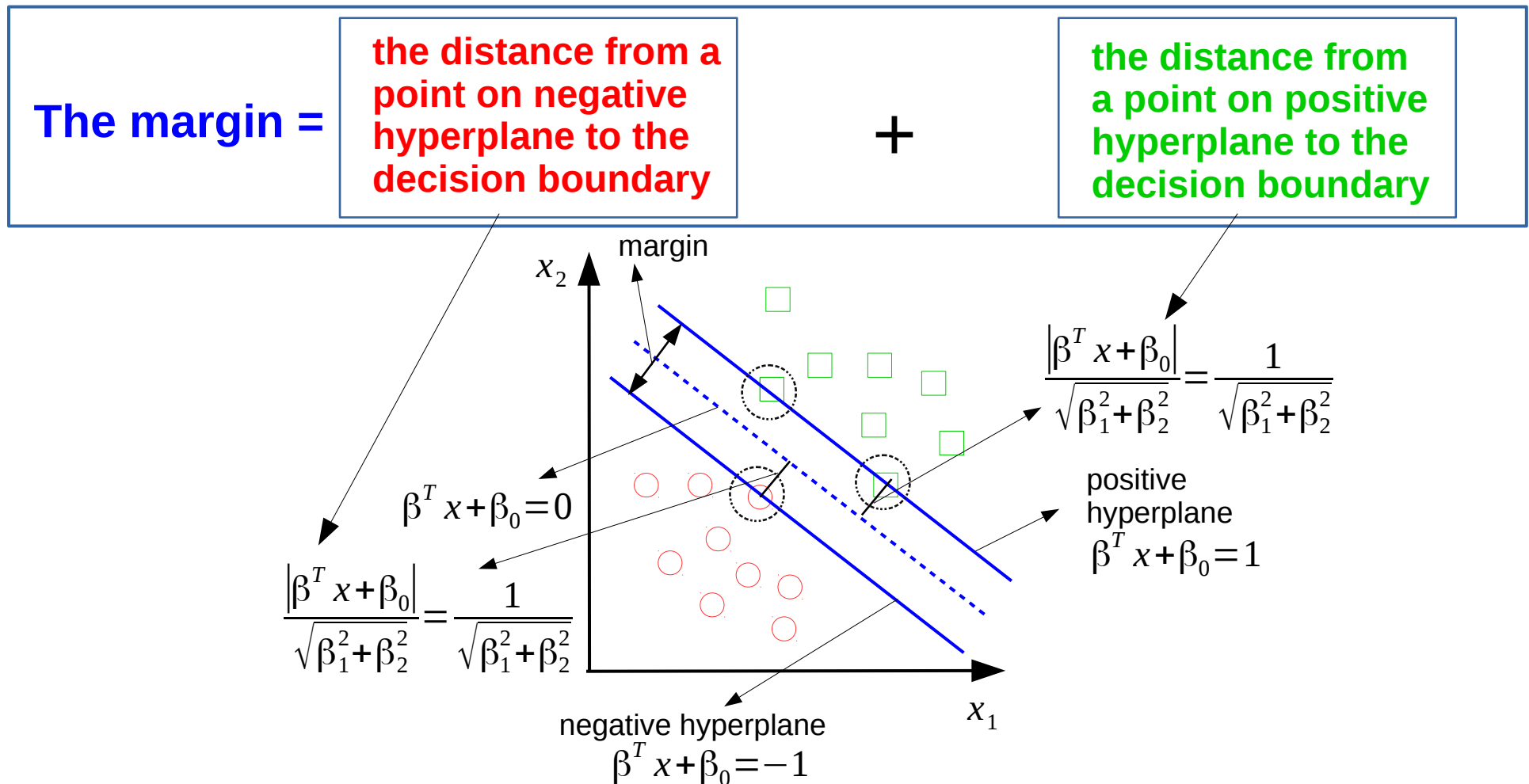
$$d = \frac{|\overrightarrow{xx'}^T \beta|}{\text{length}(\beta)} = \frac{|(x_1 - x'_1)\beta_1 + (x_2 - x'_2)\beta_2|}{\sqrt{\beta_1^2 + \beta_2^2}}$$

$$= \frac{|x_1\beta_1 + x_2\beta_2 - x'_1\beta_1 - x'_2\beta_2|}{\sqrt{\beta_1^2 + \beta_2^2}} = \frac{|\beta_0 + x_1\beta_1 + x_2\beta_2 - \beta_0 - x'_1\beta_1 - x'_2\beta_2|}{\sqrt{\beta_1^2 + \beta_2^2}} = \frac{|\beta_0 + x_1\beta_1 + x_2\beta_2|}{\sqrt{\beta_1^2 + \beta_2^2}}$$



# Max-margin method: Computing the margin?

- Given a data set  $D = \{x_i, y_i\}_{i=1}^n$  for binary classification
- The margin** is the **distance** between two **+/- hyperplanes**



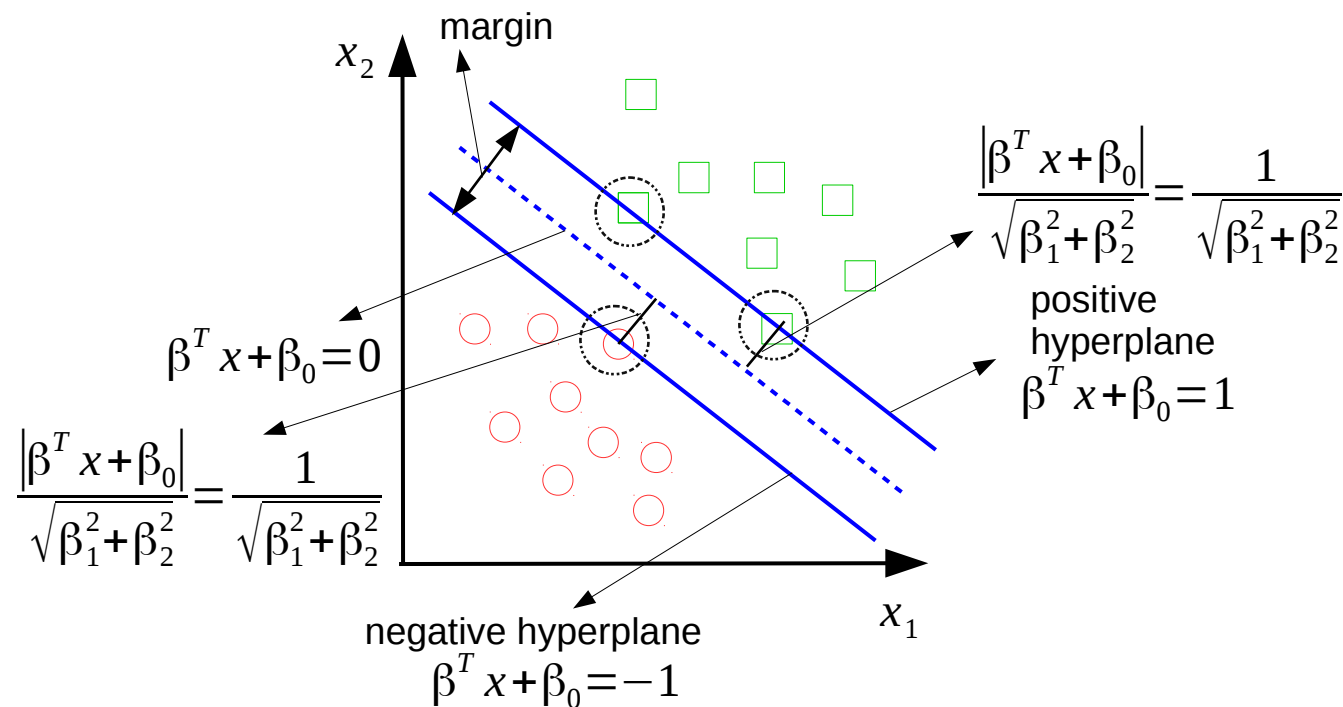
# Max-margin method: Computing the margin?

- The computed **margin** is


In 2-D: 
$$\frac{2}{\sqrt{\beta_1^2 + \beta_2^2}} = \frac{2}{\|\beta\|}$$

In d-D (d-dimensional): 
$$\frac{2}{\sqrt{\beta_1^2 + \beta_2^2 + \dots + \beta_d^2}} = \frac{2}{\|\beta\|}$$


$f(x) = \beta_0 + \beta^T x = \beta_0 + \beta_1 x_1 + \dots + \beta_d x_d$



# Support vector machine: Max-margin method

- Maximize  $\frac{2}{\|\beta\|}$   **The margin**  $\frac{2}{\sqrt{\beta_1^2 + \beta_2^2 + \dots + \beta_d^2}}$

**Subject that:**  $\min_{i=1,2,\dots,n} y_i(\beta^T x_i + \beta_0) = 1$

 **Minimum at: points on the positive and negative hyper-planes**  
Points outside the positive and negative hyper-planes would make:

$$y_i(\beta^T x_i + \beta_0) > 1$$

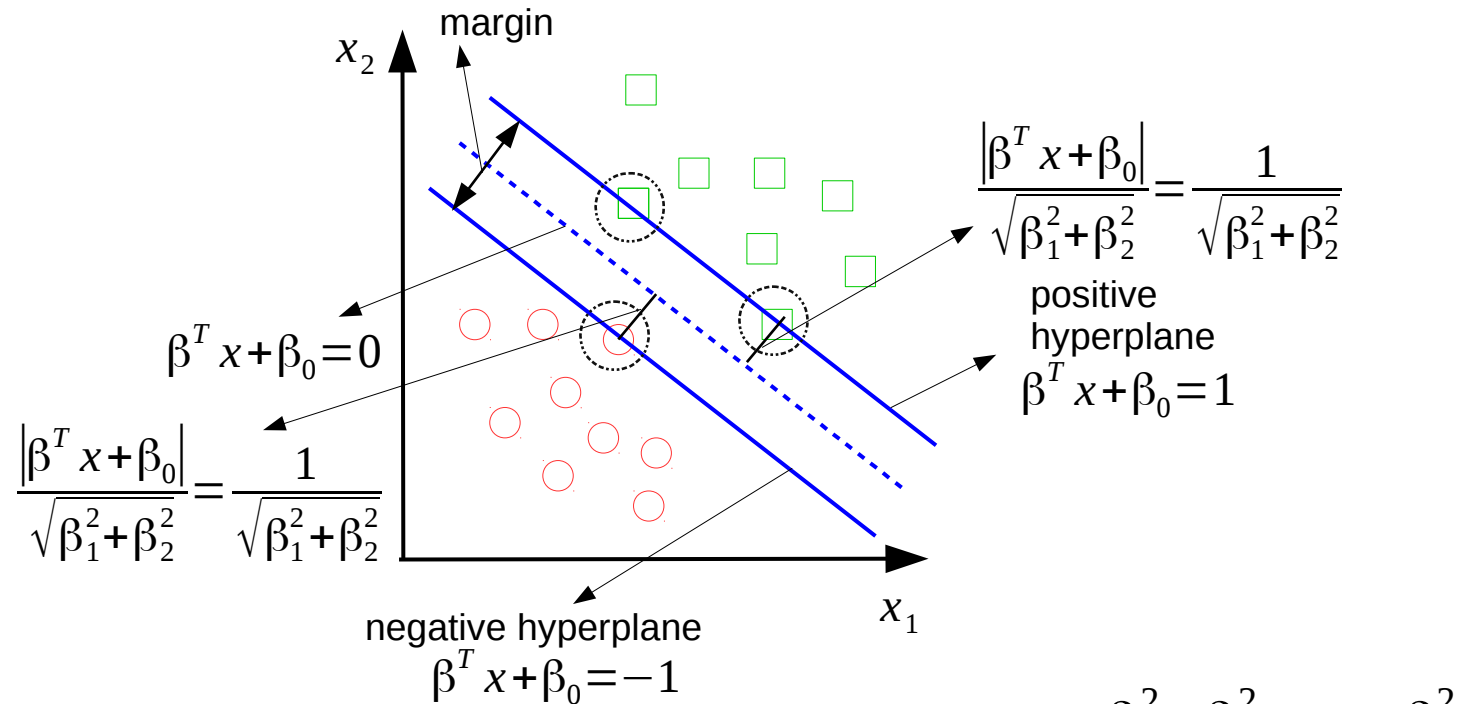
- Equivalently, we go to minimize  $\frac{\|\beta\|^2}{2}$    $\frac{\beta_1^2 + \beta_2^2 + \dots + \beta_d^2}{2}$

**Subject that:** for all  $i$ ,  $y_i(\beta^T x_i + \beta_0) \geq 1$

In order to maximize the margin, we need to minimize  $\|\beta\|$ . With the condition that there are no datapoints between the positive and negative planes.



# Support vector machine: Max-margin method



- Minimize  $\frac{\|\beta\|^2}{2}$   $\longleftarrow \frac{\beta_1^2 + \beta_2^2 + \dots + \beta_d^2}{2}$

**Subject that:** for all  $i$ ,  $y_i(\beta^T x_i + \beta_0) \geq 1$

In order to maximize the margin, we need to minimize  $\|\beta\|^2$ . With the condition that there are **no datapoints between the positive and negative planes**.

# Support vector machine: Quadratic program

- Minimize:  $\frac{\|\beta\|^2}{2}$  **subject that:** for all  $i$ ,  $y_i(\beta^T x_i + \beta_0) \geq 1$
- At **optimum**:
  - $\|\beta\|^2$  is **minimized** = **maximized the margin**
  - **All constraints are satisfied** => **all training data points are correctly classified (linearly separated).**
- This is tricky but it's a convex problem. There is only one optimum and we can find it without fiddling with learning rates or weight decay or early stopping.
  - don't worry about the optimization problem. It has been solved. It's called quadratic programming (e.g. quadratic programming solutions can also be found using gradient descent methods).
- It takes time proportional to  $n^2$  which is really bad for very big datasets
  - so for big datasets we end up doing approximate optimization!

Where  $n$  is the number of training instances

# Support vector machine: Solution

- The classifier function after being optimized (after using quadratic programming)
  - Notice that it relies on an inner product between the test point  $x$  and the support vectors  $x_i$

$$f(x) = \beta_0 + \beta^T x = \sum_{x_i \text{ is support vectors}} \alpha_i x^T x_i \quad \alpha_i \text{ is weight associated with a support vector } i$$

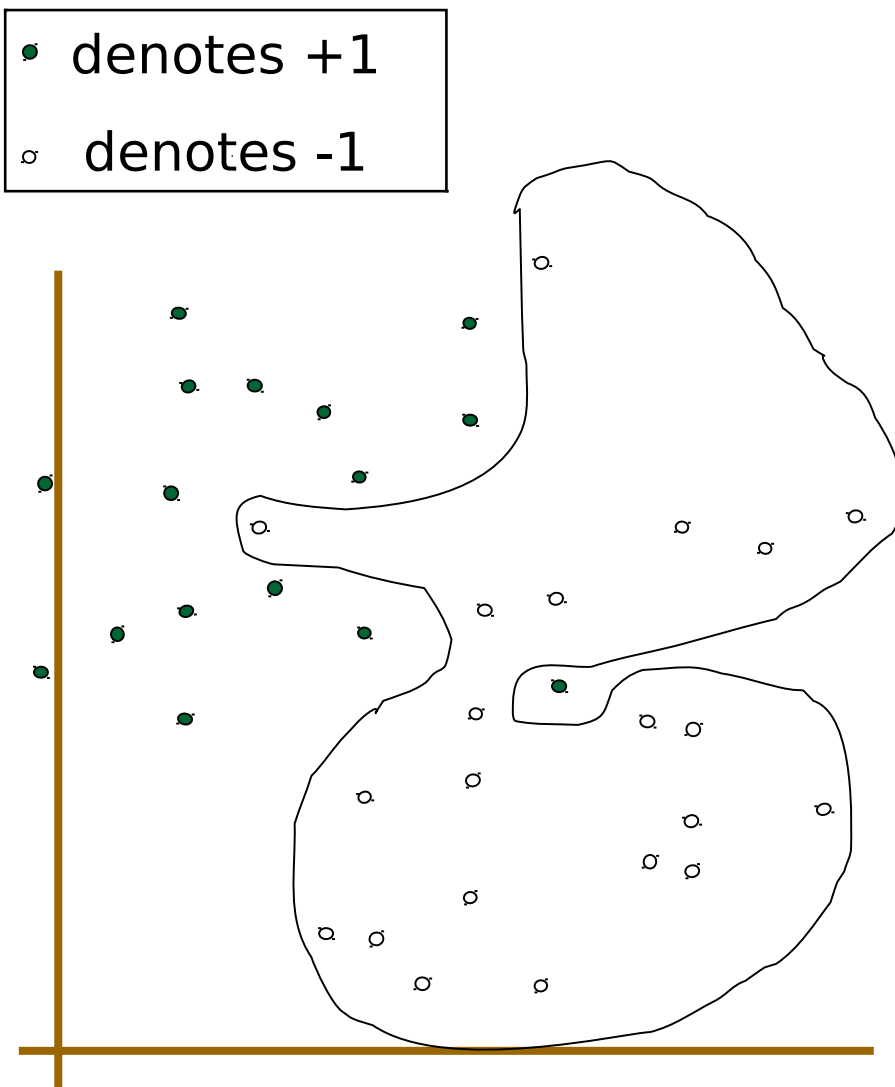
Decisions:

If  $f(x) \geq 1$ ,  $y = +1$

If  $f(x) \leq -1$ ,  $y = -1$

- We need to decide which side of the separating hyperplane a test point lies on and this requires us to compute a **inner product**.
- We can express this inner product as a weighted average of inner products with the stored support vectors
  - This could still be slow if there are a lot of support vectors .

# Dataset with noise or non-linearity



- **Hard Margin:** So far we require all data points be classified correctly
  - No training error
- **What if the training set is noisy and not-linearly separable?**
  - **Solution 1: use very powerful features**  
(means with potentially infinite features)

**OVERFITTING!**

- **Solution 2: accept error tolerance**  
(use slack variables)

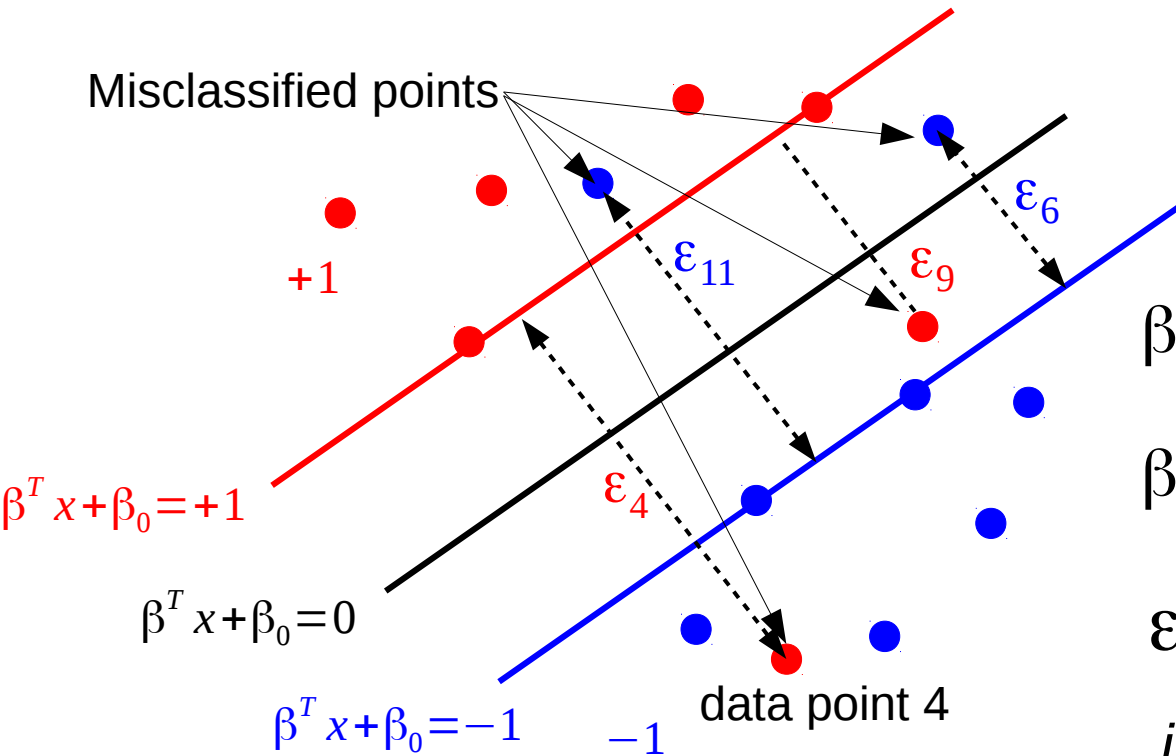
# Dataset with noise or non-linearity

- Use a much bigger set of features.
  - This looks as if it would make the computation hopelessly slow, but in the next part of the lecture we will see how to use the “**kernel**” **trick** to make the computation fast even with huge numbers of features.
- Extend the definition of maximum margin to allow non-separating planes and error tolerance.
  - This can be done by using “slack” variables

# Soft margin classification

**Slack variables  $\epsilon_i$  can be added to allow misclassification of difficult or noisy examples.**

Slack variables are constrained to be non-negative. When they are greater than zero they allow us to cheat by putting the plane closer to the datapoint than the margin. So we need to minimize the amount of cheating.



$$\beta^T x_i + \beta_0 \geq 1 - \varepsilon_i \quad \text{for positive cases}$$

$$\beta^T x_i + \beta_0 \geq -1 + \varepsilon_i \text{ for negative cases}$$

$$\varepsilon_i \geq 0 \quad \text{for all cases}$$

$i$  is the index of data point  $i$

# Soft margin classification

**Slack variables  $\varepsilon_i$**  can be added to allow misclassification of difficult or noisy examples.

What should our quadratic optimization criterion be changed to?

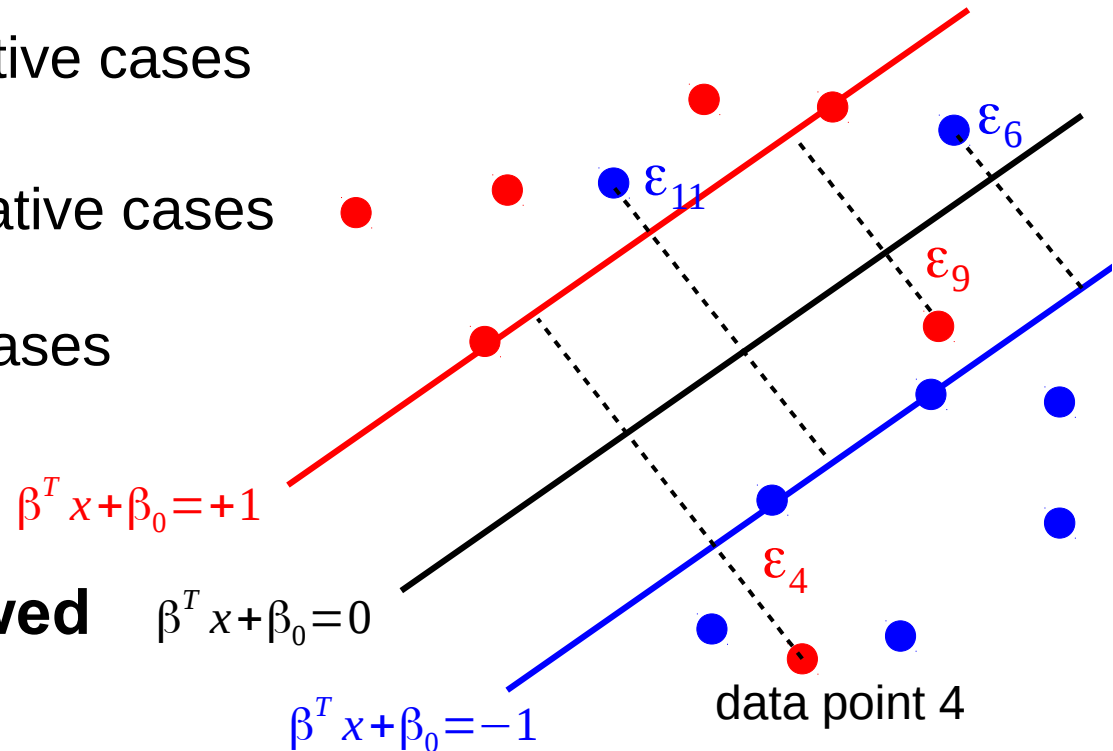
Minimize:  $\frac{\|\beta\|^2}{2} + C(\varepsilon_1 + \varepsilon_2 + \dots + \varepsilon_n)$

Such that:  $\beta^T x_i + \beta_0 \geq 1 - \varepsilon_i$  for positive cases

$\beta^T x_i + \beta_0 \geq -1 + \varepsilon_i$  for negative cases

$\varepsilon_i \geq 0$  for all cases

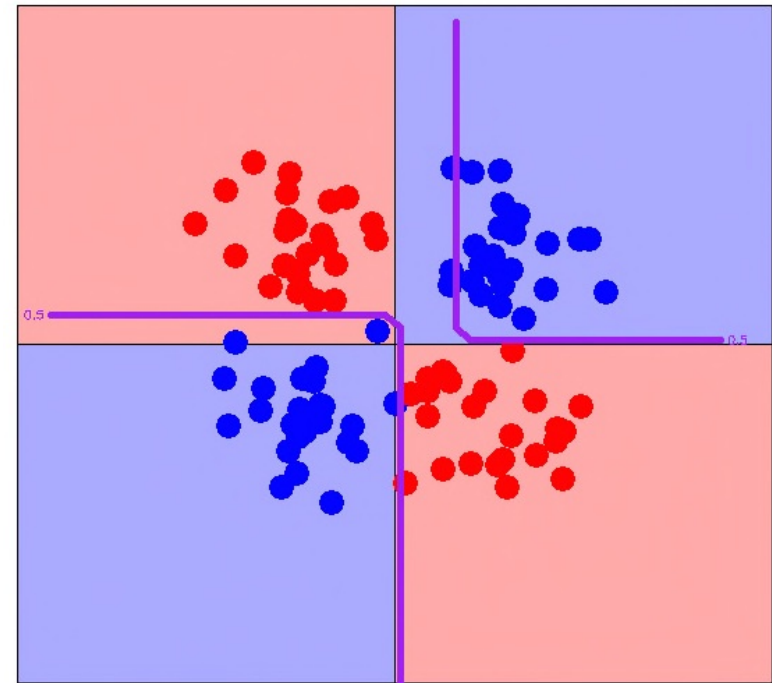
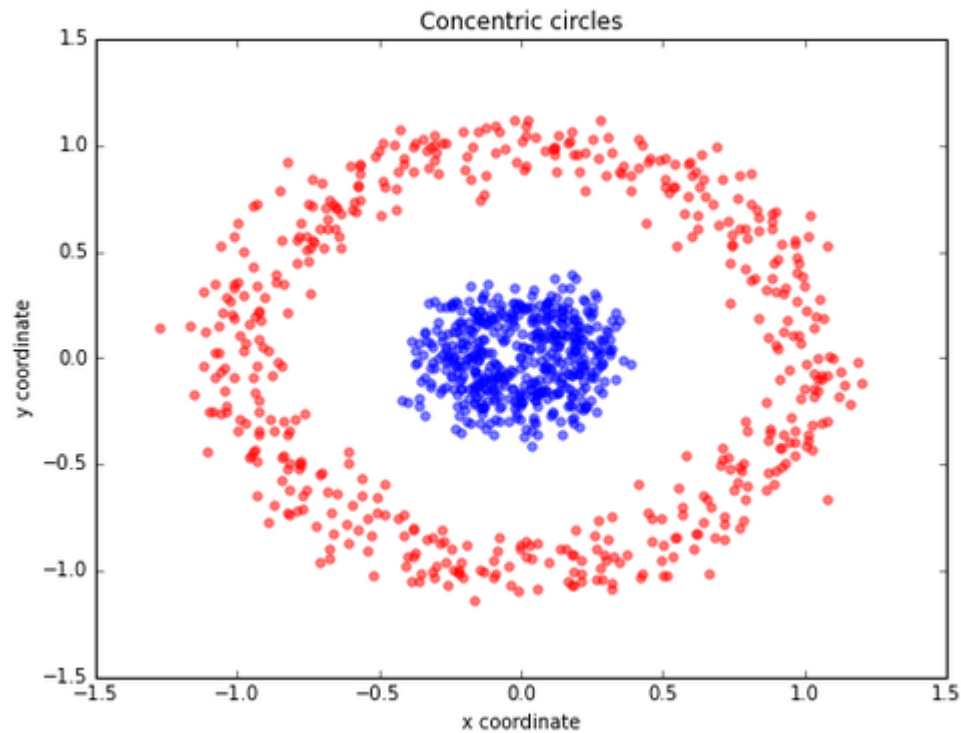
**Hyper-parameter  $C$  can be viewed as a way to control overfitting.**



# Non-linearly separable data

- When the data is **not linearly separable**?

—





# A potential problem and a magic solution

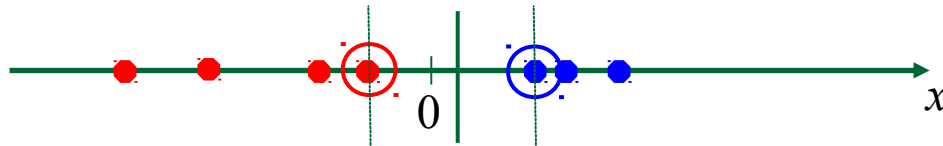
- If we map the input vectors into a **very** high-dimensional feature space, surely the task of finding the maximum-margin separator becomes computationally intractable?
  - The mathematics is all linear, which is good, but the vectors have a huge number of components.
  - So taking the scalar product of two vectors is very expensive.
- The way to keep things tractable is to use

**“the kernel trick”**

- The kernel trick makes your brain hurt when you first learn about it, but its actually very simple.

# Non-linear SVMs

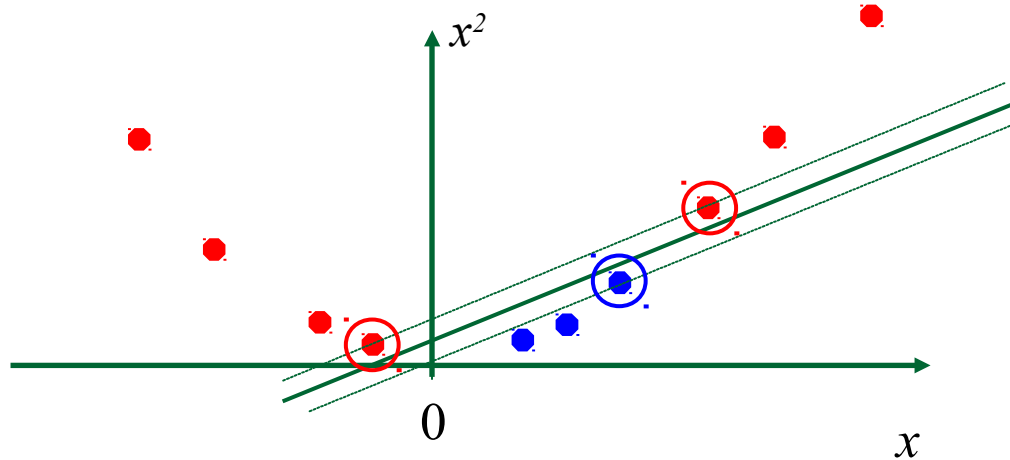
- Datasets that are linearly separable with some noise work out great:



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



- How about... mapping data to a higher-dimensional space:

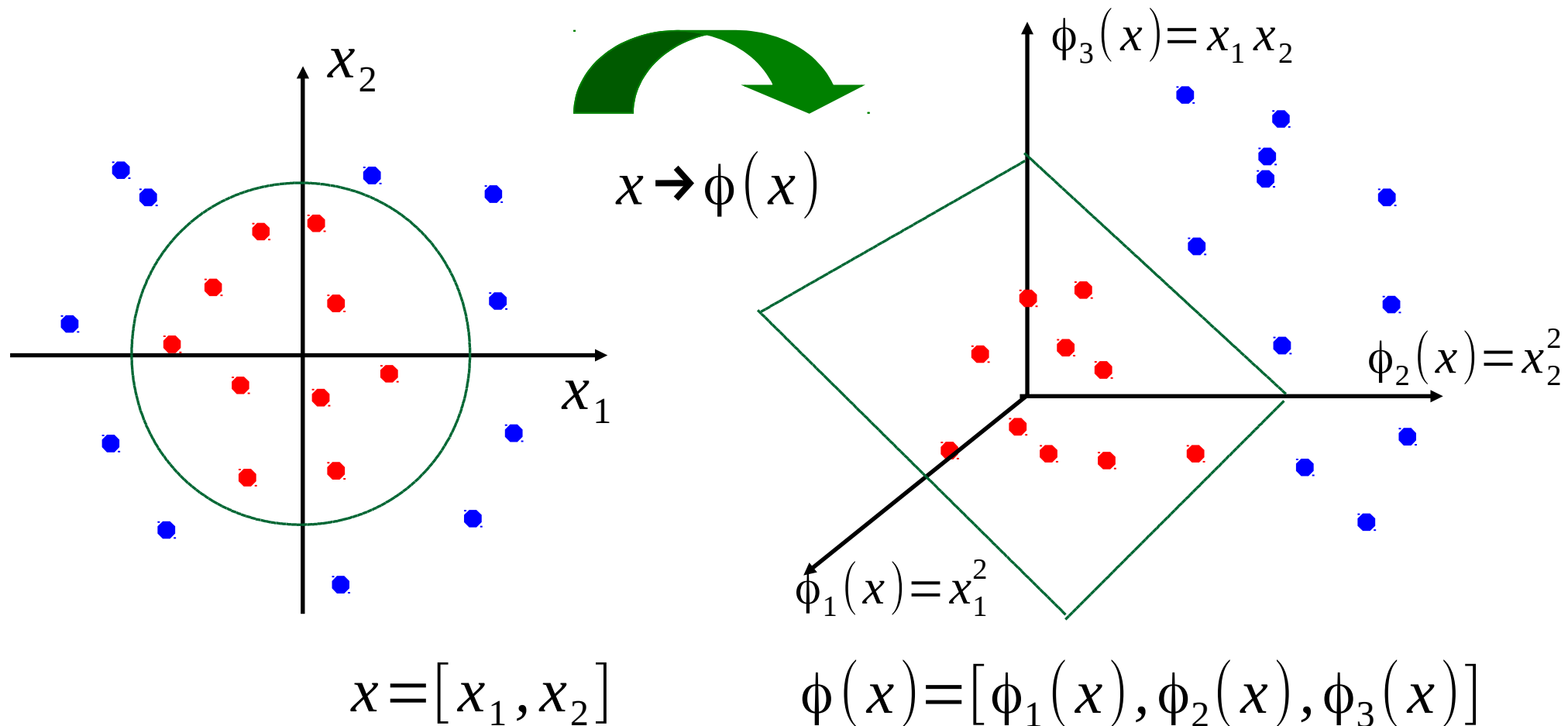


# Non-linear SVM - Overview

- SVM locates a separating hyperplane in the feature space and classify points in that space
- It does not need to represent the space explicitly, simply by defining a kernel function
- The kernel function plays the role of the inner product in the feature space.

# Non-linear SVM: Feature mapping

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



# Non-linear SVM: Solution

- The classifier function after being optimized
  - Notice that it relies on an *inner product* between the test point  $x$  and the support vectors  $x_i$

$$f(x) = \beta_0 + \phi(x)^T \beta = \sum_{x_i \text{ is support vectors}} \alpha_i \phi(x)^T \phi(x_i)$$

$\alpha_i$  is weight associated with a support vector  $i$

Decisions:

If  $f(x) \geq 1$ ,  $y = +1$

If  $f(x) \leq -1$ ,  $y = -1$

When using slack variables:

$\beta^T \phi(x_i) + \beta_0 \geq 1 - \varepsilon_i$  for positive cases

$\beta^T \phi(x_i) + \beta_0 \geq -1 + \varepsilon_i$  for negative cases

$\varepsilon_i \geq 0$  for all cases

# The “Kernel Trick”

- The linear classifier relies on inner product between vectors

$$k(x_i, x) = x_i^T x$$

- If every data point is mapped into high-dimensional space via some transformation  $x \rightarrow \phi(x)$ , the inner product becomes:

$$k(x_i, x) = \phi(x_i)^T \phi(x)$$

- A kernel function is some function that corresponds to an inner product in some expanded feature space.

- Example: – Polynomial:  $k(x, x') = (x^T x' + c)^d$   
 Let's verify for  $d = 2$ ,  $\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$ :  

$$k(x, x') = ((x_1, x_2) \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} + 1)^2 \quad \leftarrow \text{Quadratic features} \quad d=2, c=1$$

$$= (x_1x'_1 + x_2x'_2 + 1)^2$$

$$= 1 + 2x_1x'_1 + 2x_2x'_2 + x_1^2x'^2_1 + 2x_1x_2x'_1x'_2 + x_2^2x'^2_2$$

$$= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)(1, \sqrt{2}x'_1, \sqrt{2}x'_2, x'^2_1, \sqrt{2}x'_1x'_2, x'^2_2)^T$$

$$= \phi(x)^T \phi(x')$$

# What the kernel trick achieves

- All of the computations that we need to do to find the maximum-margin separator can be expressed in terms of inner products between pairs of datapoints (in the high-dimensional feature space).
- These inner products are the only part of the computation that depends on the dimensionality of the high-dimensional space.
  - So if we had a fast way to do the scalar products we would not have to pay a price for solving the learning problem in the high-D space.
- The kernel trick is just a magic way of doing inner products a whole lot faster than is usually possible.
  - It relies on choosing a way of mapping to the high-dimensional feature space that allows fast inner products.

# Some Commonly Used Kernel Functions

- Linear:  $k(x_i, x) = x_i^T x$
- Polynomial of power  $d$ :  $k(x_i, x) = (x_i^T x + c)^d$
- Gaussian (radial-basis function network):

$$k(x_i, x) = e^{-\frac{\|x_i - x\|^2}{2\sigma^2}}$$

$\sigma, c, d$  : is a hyperparameters of these kernels

Optimal Choice? e.g. use cross-validation to select it





# SVM: Performance

- Support Vector Machines work very well in practice.
  - The user must choose the kernel function and its parameters, but the rest is automatic.
  - The test performance is very good.
- They can be expensive in time and space for big datasets
  - The computation of the maximum-margin hyper-plane depends on the **square** of the number of training cases.
  - We need to store all the support vectors.
- SVM's are very good if you have no idea about what structure to impose on the task.

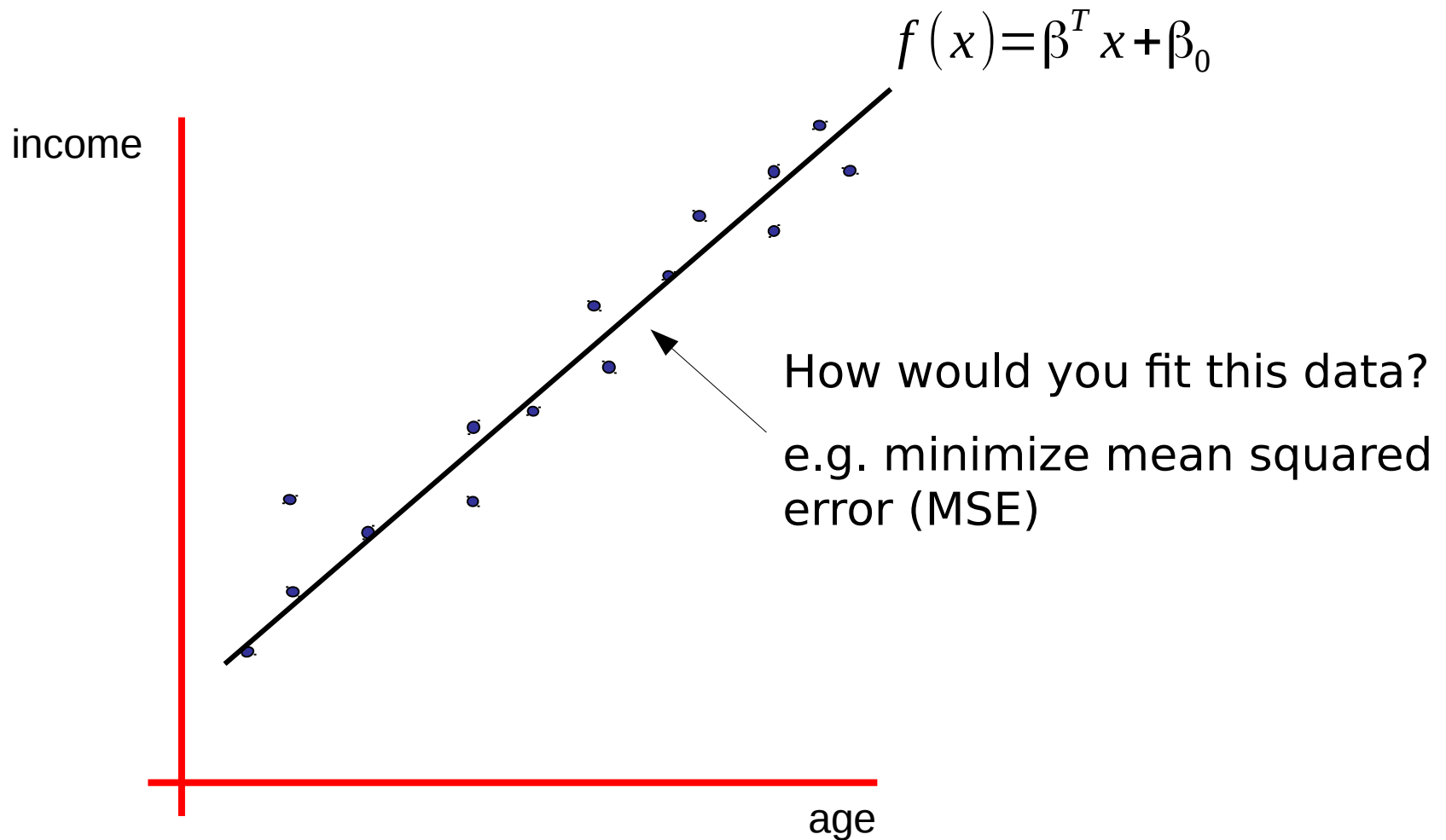
# Outline

- Support vector machine (SVM) for binary-classification
- **Support vector machine for regression**
- Support vector machine for multi-class classification

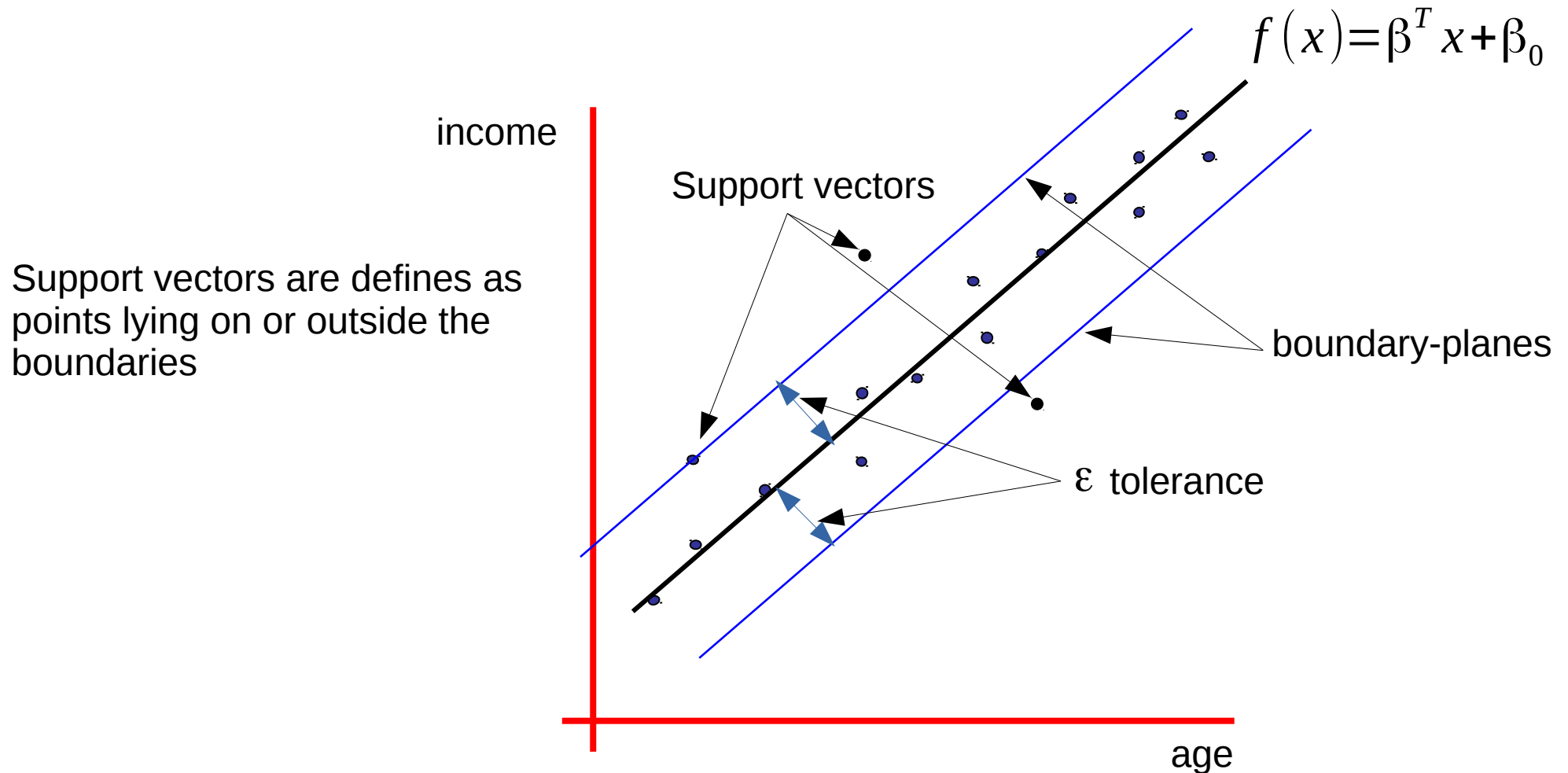
# Support Vector Regression (SVR)

- SVR is support vector machine for regression
- In SVR there are two planes (boundary planes) other than the regressor which creates a margin .
- **Support vectors in SVR:** These are the data points which are closest to the boundary planes (must be outside). The distance of the points is minimum or least.
- **Regressor:** In SVM this is basically the separation line/classifier between the data classes. In SVR we are going to define it as the line that will help us predict the continuous value or target value.

# Linear Regression: Revisit

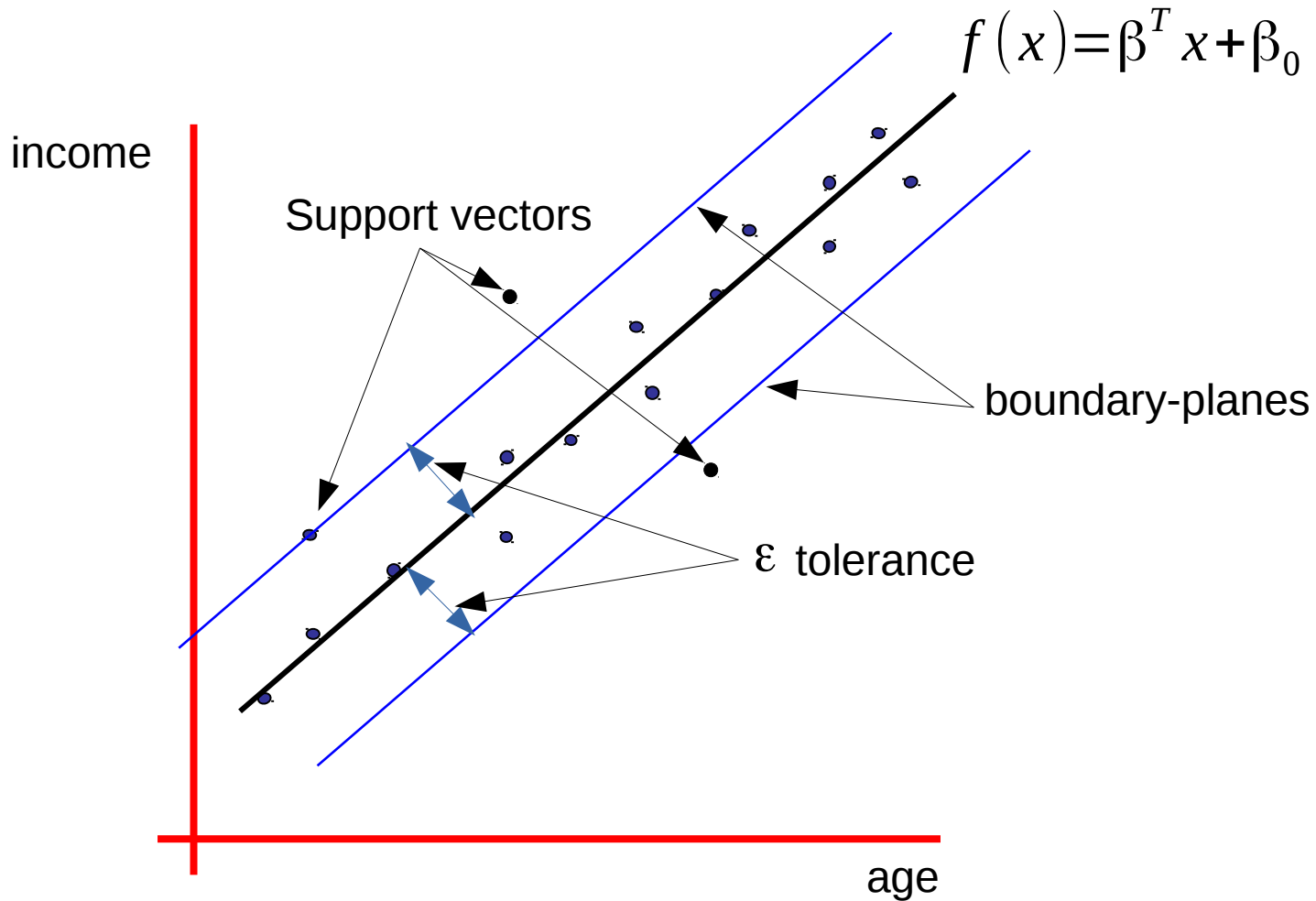


# Support Vector Regression



- The value of  $\varepsilon$  can affect the number of support vectors used to construct the regression function.
- The bigger  $\varepsilon$ , the fewer support vectors are selected.
- On the other hand, bigger  $\varepsilon$ -values results in more 'flat' estimates. Hence, both  $C$  and  $\varepsilon$ -values affect model complexity (but in a different way).

# Support Vector Regression



- Similar to SVM, SVR problems can also be solved as a quadratic program (.e.g using gradient-descent to find optimum of this program)

# Outline

- Support vector machine (SVM) for binary-classification
- Support vector machine for regression
- **Support vector machine for multi-class classification**

# Multi-Class SVM

- Approaches: Not straightforwardly (assume there are  $K$  classes)
  - One against One (  $K(K-1) / 2$  ) binary Classifiers required

Outputs of the classifiers are aggregated to make the final decision.

- One against All ( $K$  binary Classifiers required):

It trains  $K$  binary classifiers, each of which separates one class from the other  $(K-1)$  classes. Given a data point  $x$ , the binary classifier with the largest output determines the class of  $x$ . **(Each classifier is a binary SVM, which is used as a discriminant function)**



# SVM Applications

- **SVM has been used successfully in many real-world problems**
  - ✓ text (and hypertext) categorization
  - ✓ image classification
  - ✓ bioinformatics (protein classification, cancer classification)
  - ✓ hand-written character recognition
  - ✓ etc.

# Application 1: Cancer classification

- High dimensional  
-  $d > 1000$ ;  $n < 100$
- Imbalanced  
- less positive samples
- Kernel

$$k[x, x'] = e^{\frac{-\|x - x'\|^2}{2\sigma^2}}$$

| Genes    |     |     |       |     |
|----------|-----|-----|-------|-----|
| Patients | g-1 | g-2 | ..... | g-d |
| P-1      |     |     |       |     |
| p-2      |     |     |       |     |
| .....    |     |     |       |     |
| p-n      |     |     |       |     |

$x_1$

- Many irrelevant features —————> Gene is a long sequence of DNA nucleotides
- Noisy

SVM is sensitive to noisy (mis-labeled) data ☹

# Application 2: Text categorization

- Task: The classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content.
  - email filtering, web searching, sorting documents by topic, etc..
- A document can be assigned to more than one category, so this can be viewed as a series of binary classification problems, one for each category



Multi-class SVM (**One against All**)

# Representation of Text

Information Retrieval's vector space model (bag-of-words representation)

- A document is represented by a vector indexed by a pre-fixed set or dictionary of terms
- Values of a feature entry can be binary or weights: for each word in the dictionary, we define one feature as

$$\phi_i(x) = TF(word_i) \log\left(\frac{n}{DF(i)}\right)$$

} features  $x \rightarrow \phi(x)$

- Normalization, stop words, word stems
- Document  $x$  is mapped to features:  $x \rightarrow \phi(x)$

where  $x = [word_1, word_2, word_3, \dots]$

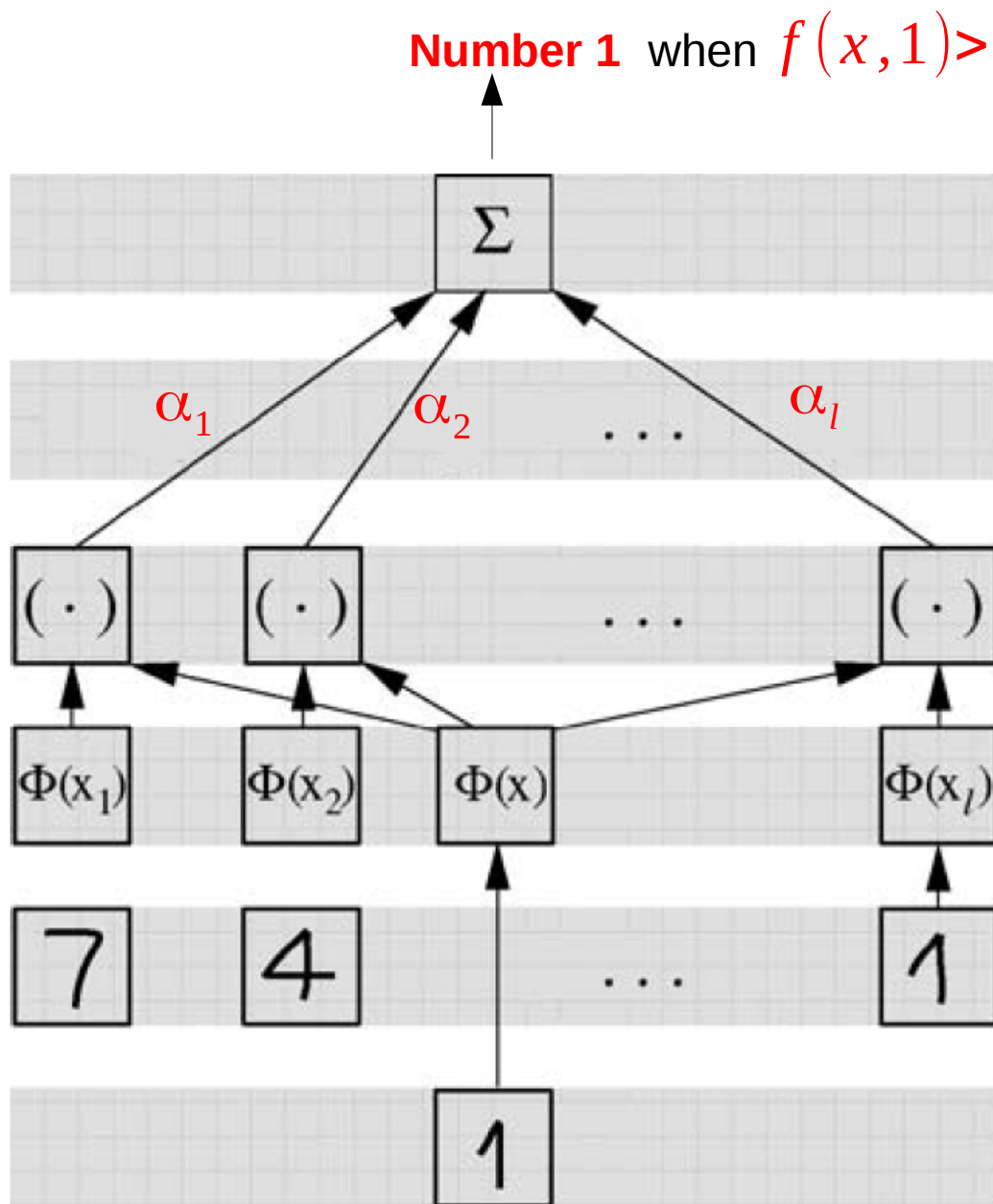
$TF(word_i)$  means how many times does the  $i$ th word occur in document  $x$   
 $DF(i)$  counts the documents containing the  $word_i$  at least once.

$n$  is the number of documents

# Text categorization using SVM

- **The distance between two documents is  $k(x_i, x) = \phi(x_i)^T \phi(x)$**
- **$k(x_i, x) = \phi(x_i)^T \phi(x)$  is a valid kernel, SVM can be used with for discrimination.**
- **Why SVM?**
  - High dimensional input space
  - Few irrelevant features (dense concept)
  - Sparse document vectors (sparse instances)
  - Text categorization problems are linearly separable

# SVM: Hand-written digit recognition



**Number 1** when  $f(x, 1) > f(x, j), j = [2, 3, \dots, 9]$  (discriminant function)

Outputs: Multi-class SVM (one vs. all)

$$f(x, j) = \alpha_1 k(x, x_1) + \dots + \alpha_l k(x, x_l)$$

With weights

Computing inner product:

$$\phi(x)^T \phi(x_i) = k(x, x_i)$$

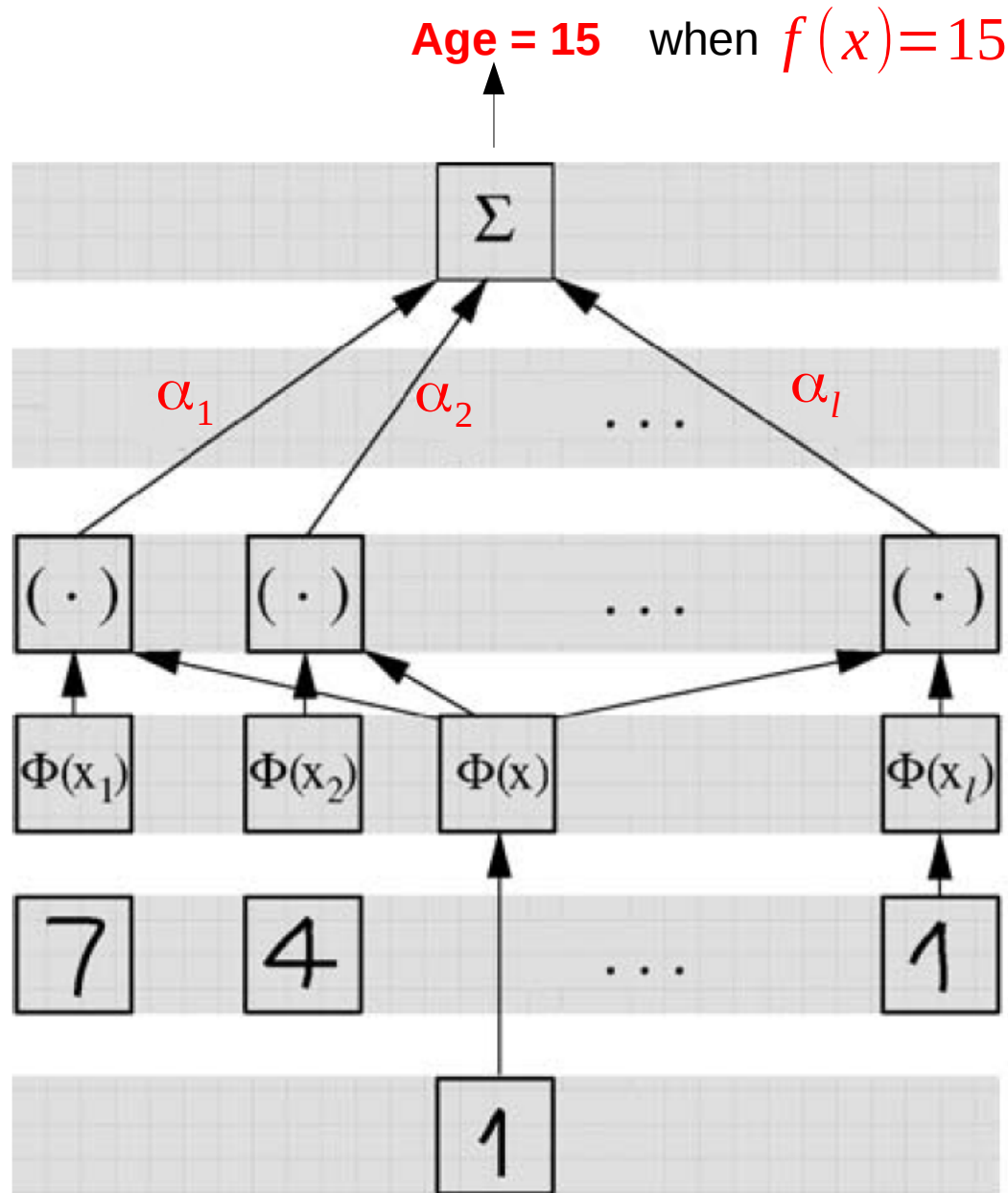
Feature maps:

$$\phi(x), \phi(x_1), \phi(x_2), \dots$$

Support vectors:  $x_1, x_2, \dots$

Test data (e.g. an image):  $x$

# SVR: Predict Ages based on Hand-written Digits



Outputs:

$$f(x) = \alpha_1 k(x, x_1) + \dots + \alpha_l k(x, x_l)$$

With weights

Computing inner product:

$$\phi(x)^T \phi(x_i) = k(x, x_i)$$

Feature maps:

$$\phi(x), \phi(x_1), \phi(x_2), \dots$$

Support vectors:  $x_1, x_2, \dots$

Test data (e.g. an image):  $x$

# Summary

- SVM for linearly separable data
- SVM for non-linearly separable data
- SVM is a very practical and widely used method
- Kernel trick offers learning with non-linear data, but avoids explicit feature designs.
- Advanced topics in SVM: SVR, Multi-class SVM