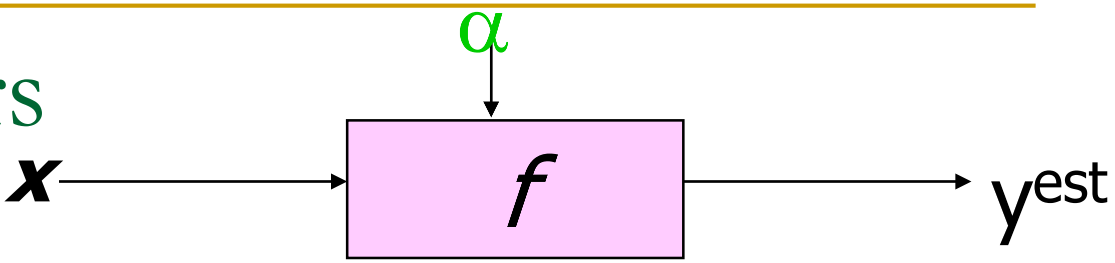
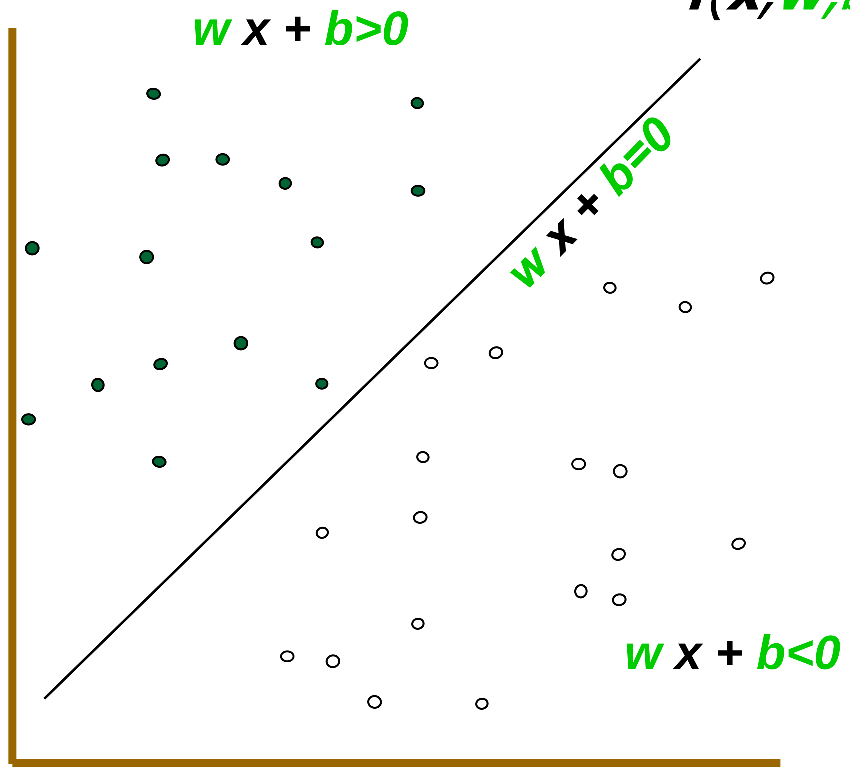
Support Vector Machines

Linear Classifiers



$f(\mathbf{x},\mathbf{w},b) = \mathrm{sign}(\mathbf{w}\,\mathbf{x} + b)$

- ● denotes +1
- ○ denotes -1



How would you classify this data?

How do we characterize “power”?

- Different machines have different amounts of “power”.
- Tradeoff between:
  - More power: Can model more complex classifiers but might overfit.
  - Less power: Not going to overfit, but restricted in what it can model.
- How do we characterize the amount of power?

Some definitions

- Given some machine **f**
- And under the assumption that all training points *(x_k,y_k)* were drawn i.i.d from some distribution.
- And under the assumption that future test points will be drawn from the same distribution
- Define

$$R(\alpha) = \mathrm{TESTERR}(\alpha) = E\left[\frac{1}{2}\left|y - f(x,\alpha)\right|\right] = \begin{matrix}\text{Probability of} \\ \text{Misclassification}\end{matrix}$$

Official terminology

Terminology we'll use

Some definitions

- Given some machine **f**
- And under the assumption that all training points *(x_k,y_k)* were drawn i.i.d from some distribution.
- And under the assumption that future test points will be drawn from the same distribution
- Define

$$R(\alpha) = \text{TESTERR}(\alpha) = E\left[\frac{1}{2}\left|y - f(x,\alpha)\right|\right] = \begin{array}{c}\text{Probability of}\\ \text{Misclassification}\end{array}$$

Official terminology

Terminology we'll use

$$R^{emp}(\alpha) = \text{TRAINERR}(\alpha) = \frac{1}{R}\sum_{k=1}^{R}\frac{1}{2}\left|y_k - f(x_k,\alpha)\right| = \begin{array}{c}\text{Fraction Training}\\ \text{Set misclassified}\end{array}$$

R = #training set data points

Vapnik-Chervonenkis dimension

$$\text{TESTERR}(\alpha) = E\left[\frac{1}{2}\left|y - f(x,\alpha)\right|\right] \qquad \text{TRAINERR}(\alpha) = \frac{1}{R}\sum_{k=1}^{R}\frac{1}{2}\left|y_k - f(x_k,\alpha)\right|$$

- Given some machine **f**, let *h* be its VC dimension.
- *h* is a measure of **f**'s power (*h* does not depend on the choice of training set)
- Vapnik showed that with probability 1-η

$$\text{TESTERR}(\alpha) \le \text{TRAINERR}(\alpha) + \sqrt{\frac{h(\log(2R/h)+1)-\log(\eta/4)}{R}}$$

This gives us a way to estimate the error on future data based only on the training error and the VC-dimension of ***f***

What VC-dimension is used for

$$\text{TESTERR}(\alpha) = E\left[\frac{1}{2}\left|y - f(x,\alpha)\right|\right] \qquad \text{TRAINERR}(\alpha) = \frac{1}{R}\sum_{k=1}^{R}\frac{1}{2}\left|y_k - f(x_k,\alpha)\right|$$

- Given some machine **f**, let *h* be its VC
- *h* is a measure of **f**'s power.
- Vapnik showed th

TES

$$\frac{\quad 1) - \log(\eta/4)}{R}$$

y to estimate the error on future data based only on the training error and the VC-dimension of f

But given machine **f**, how do we define and compute h?

Shattering

- Machine f can *shatter* a set of points $x_1, x_2 .. x_r$ if and only if…

  For every possible training set of the form (x_1,y_1) , (x_2,y_2) ,… (x_r ,y_r)

  …There exists some value of α that gets zero training error.

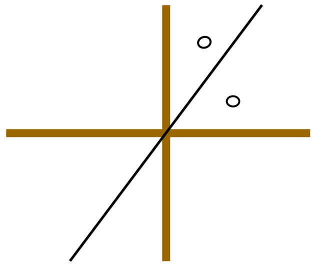
There are 2^r such training sets to consider, each with a different combination of +1's and –1's for the y's

Shattering

- Machine f can *shatter* a set of points $x_1, x_2 .. X_r$ if and only if…

  For every possible training set of the form (x_1,y_1) , (x_2,y_2) ,… (x_r ,y_r)

  …There exists some value of α that gets zero training error.
- Question: Can the following f shatter the following points?



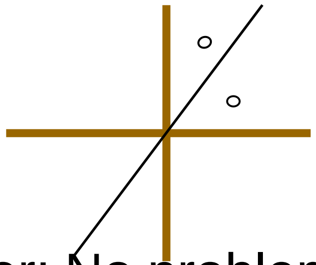
f(x,w) = sign(x.w)

Shattering

- Machine f can *shatter* a set of points $x_1, x_2 .. X_r$ if and only if…

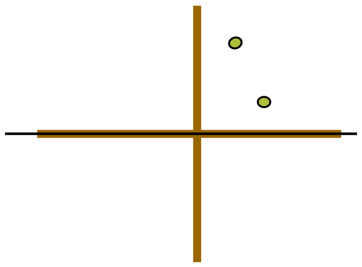
  For every possible training set of the form $(x_1,y_1) , (x_2,y_2) ,\ldots (x_r ,y_r)$

  …There exists some value of α that gets zero training error.
- Question: Can the following f shatter the following points?

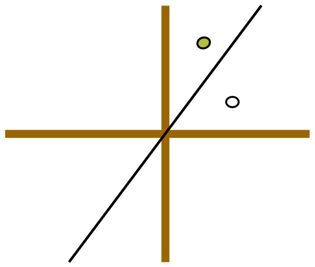


$f(x,w) = \text{sign}(x.w)$

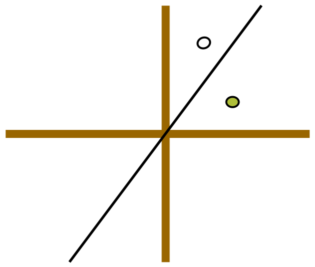
- Answer: No problem. There are four training sets to consider



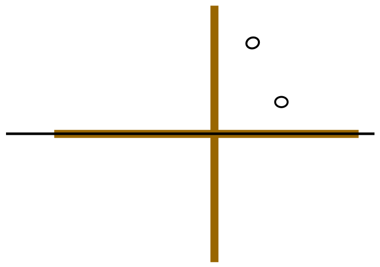
w=(0,1)



w=(-2,3)



w=(2,-3)



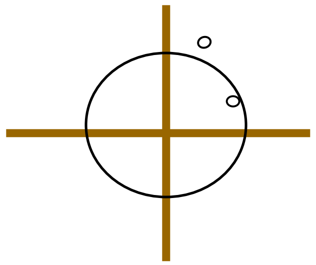
w=(0,-1)

Shattering

- Machine f can *shatter* a set of points $x_1, x_2 .. X_r$ if and only if…

  For every possible training set of the form (x_1,y_1) , (x_2,y_2) ,… (x_r ,y_r)

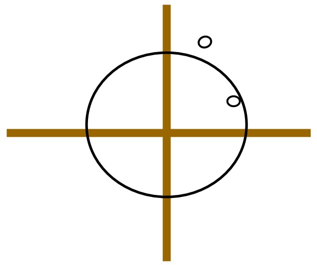
  …There exists some value of α that gets zero training error.
- Question: Can the following f shatter the following points?



$f(x,b) = \text{sign}(x.x-b)$

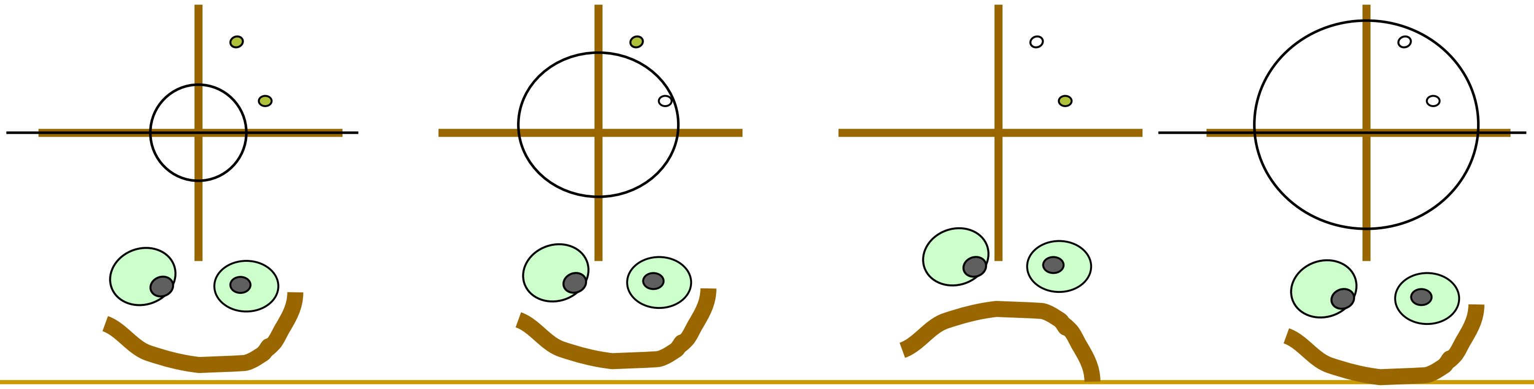
Shattering

- Machine f can *shatter* a set of points x_1, x_2 .. X_r if and only if…
  For every possible training set of the form (x_1,y_1) , (x_2,y_2) ,… (x_r ,y_r)
  …There exists some value of α that gets zero training error.
- Question: Can the following f shatter the following points?



f(x,b) = sign(x.x-b)

- Answer: No way my friend.



Definition of VC dimension

Given machine ***f***, the VC-dimension *h* is

The maximum number of points that can be arranged so that ***f*** shatter them.

Example: What's VC dimension of f(x,b) = sign(x.x-b)

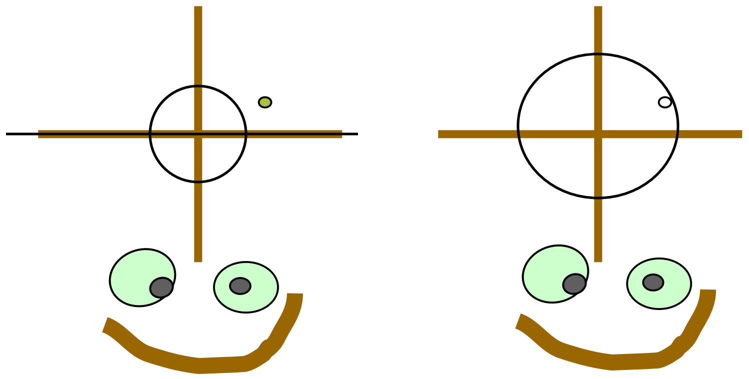
VC dim of trivial circle

Given machine ***f***, the VC-dimension *h* is

The maximum number of points that can be arranged so that ***f*** shatter them.

Example: What's VC dimension of f(x,b) = sign(x.x-b)

Answer = 1: we can't even shatter two points! (but it's clear we can shatter 1)



Reformulated circle

Given machine ***f***, the VC-dimension *h* is

The maximum number of points that can be arranged so that ***f*** shatter them.

Example: For 2-d inputs, what's VC dimension of f(x,q,b) = sign(qx.x-b)

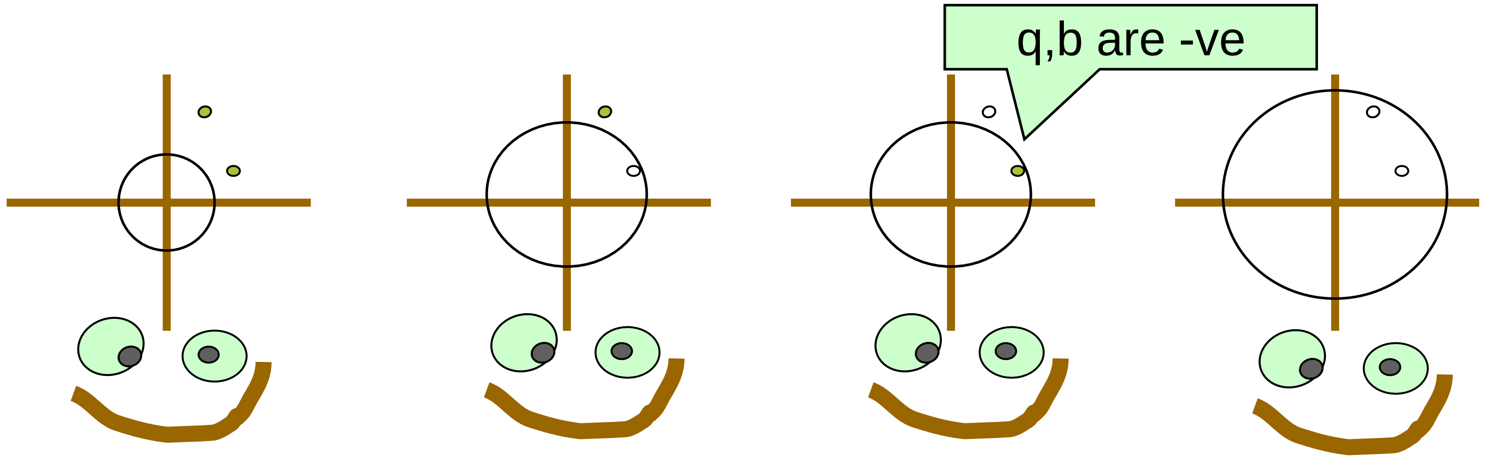
Reformulated circle

Given machine ***f***, the VC-dimension *h* is

The maximum number of points that can be arranged so that ***f*** shatter them.

Example: What's VC dimension of f(x,q,b) = sign(qx.x-b)

- Answer = 2

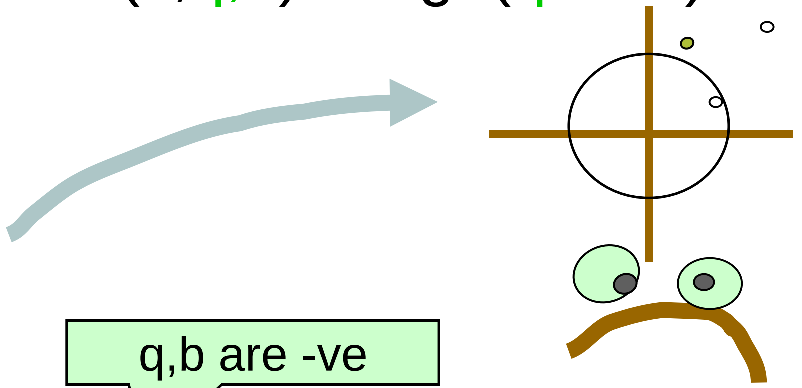


Reformulated circle

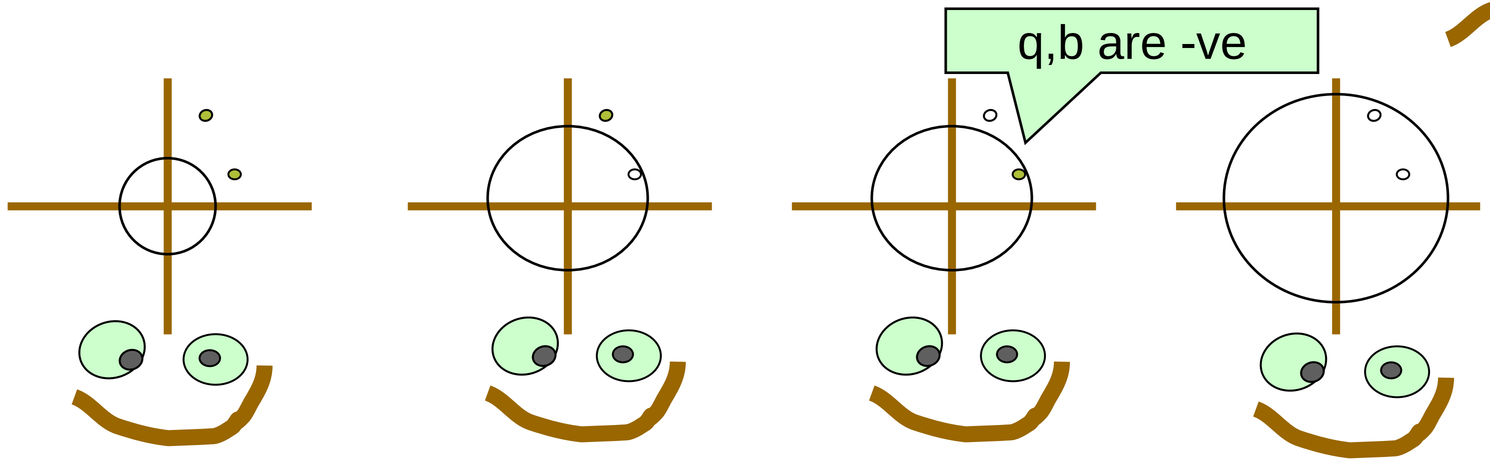
Given machine ***f***, the VC-dimension *h* is

> The maximum number of points that can be arranged so that ***f*** shatter them.

Example: What's VC dimension of f(x,q,b) = sign(qx.x-b)



- Answer = 2 (clearly can't do 3)



Vapnik-Chervonenkis dimension

$$\text{TESTERR}(\alpha) = E\left[\frac{1}{2}\left|y - f(x,\alpha)\right|\right] \qquad \text{TRAINERR}(\alpha) = \frac{1}{R}\sum_{k=1}^{R}\frac{1}{2}\left|y_k - f(x_k,\alpha)\right|$$

- Given some machine **f**, let *h* be its VC dimension.
- *h* is a measure of **f**'s power (*h* does not depend on the choice of training set)
- Vapnik showed that with probability 1-η

$$\text{TESTERR}(\alpha) \le \text{TRAINERR}(\alpha) + \sqrt{\frac{h(\log(2R/h)+1) - \log(\eta/4)}{R}}$$

This gives us a way to estimate the error on future data based only on the training error and the VC-dimension of ***f***

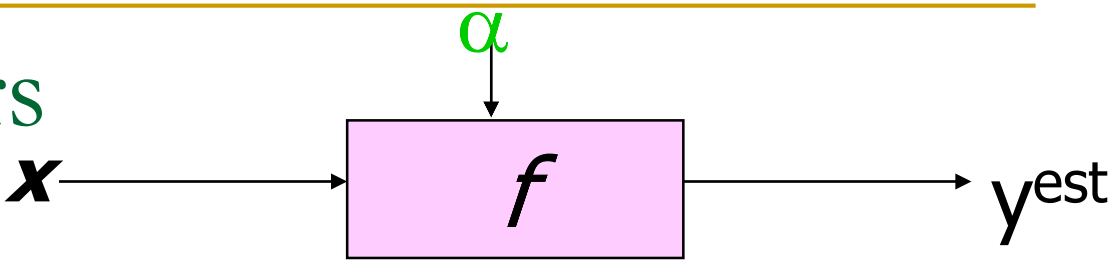
Definition of VC dimension

Given machine ***f***, the VC-dimension *h* is

The maximum number of points that can be arranged so that ***f*** shatter them.

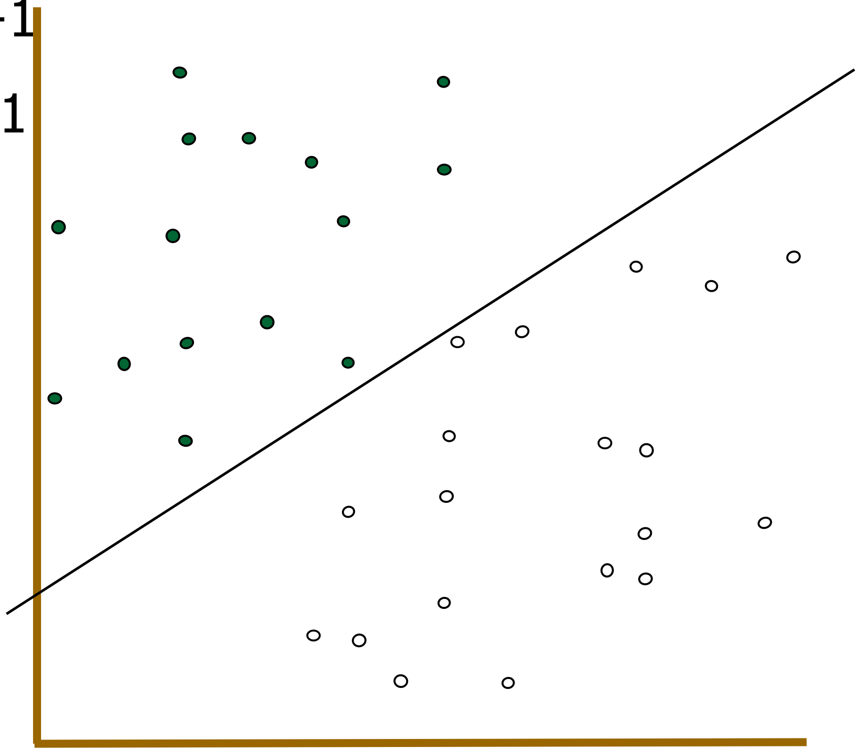
Example: What's VC dimension of f(x,b) = sign(x.x-b)

Linear Classifiers



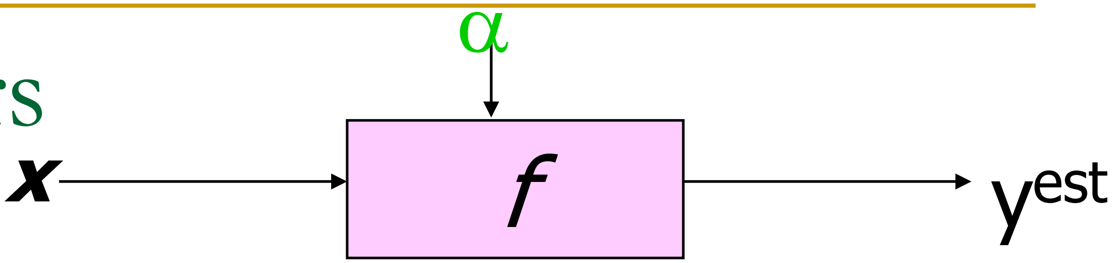
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$

- ● denotes +1
- ○ denotes -1



How would you classify this data?

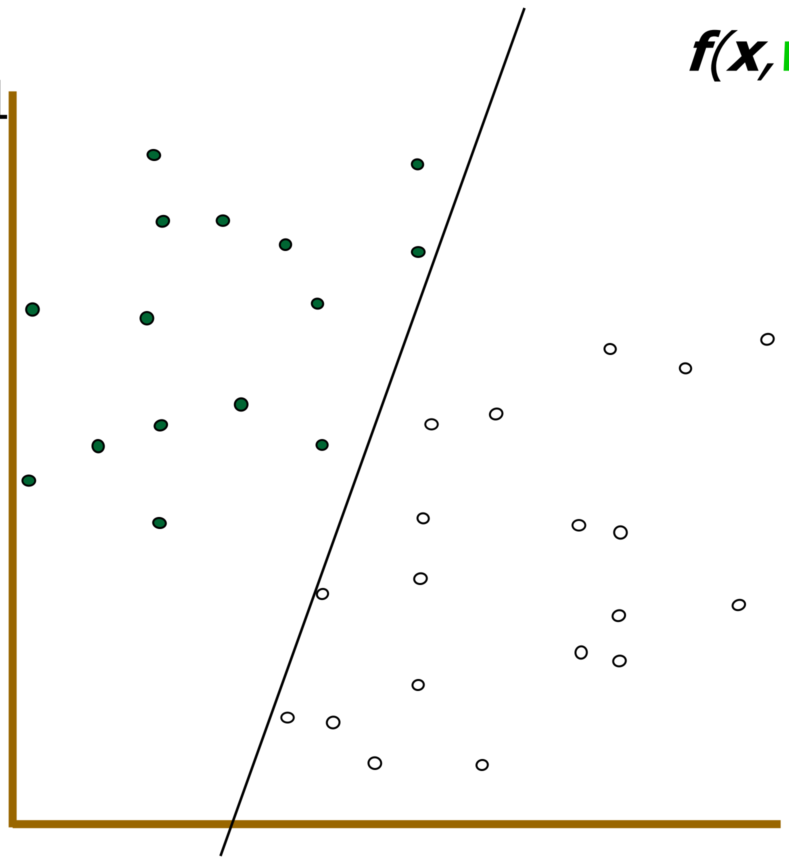
Linear Classifiers



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$

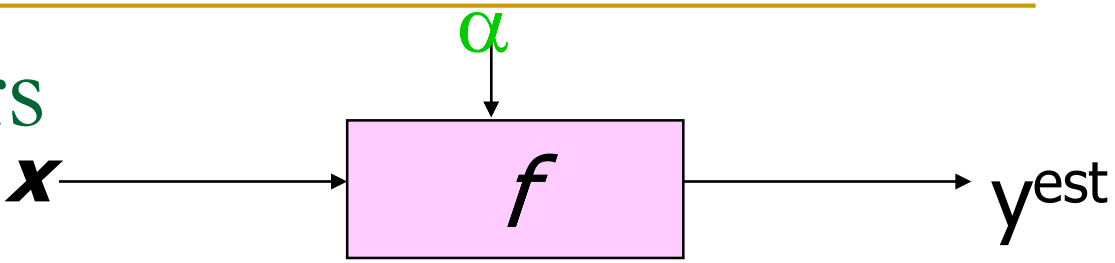
● denotes +1

○ denotes -1

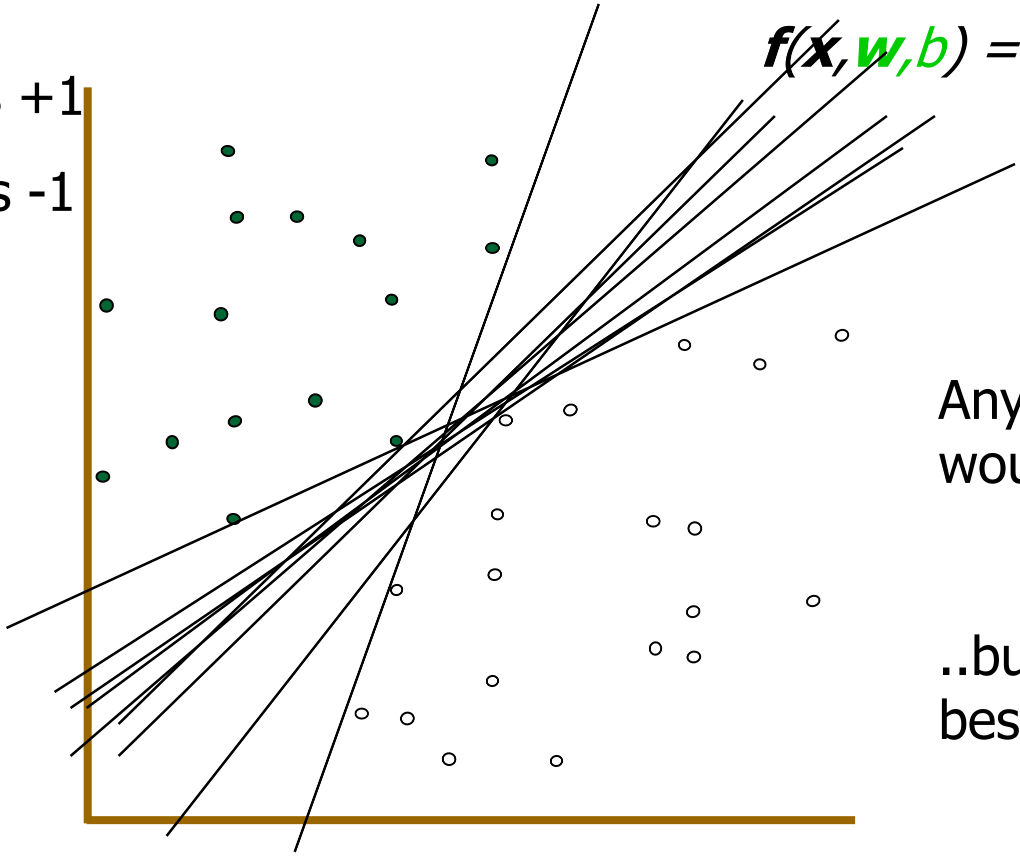


How would you classify this data?

Linear Classifiers



$f(\boldsymbol{x},\boldsymbol{w},b) = sign(\boldsymbol{w}\,\boldsymbol{x} + b)$

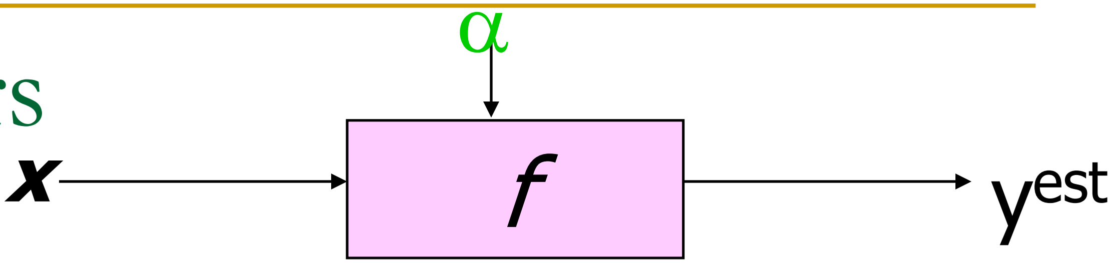


- ● denotes +1
- ○ denotes -1

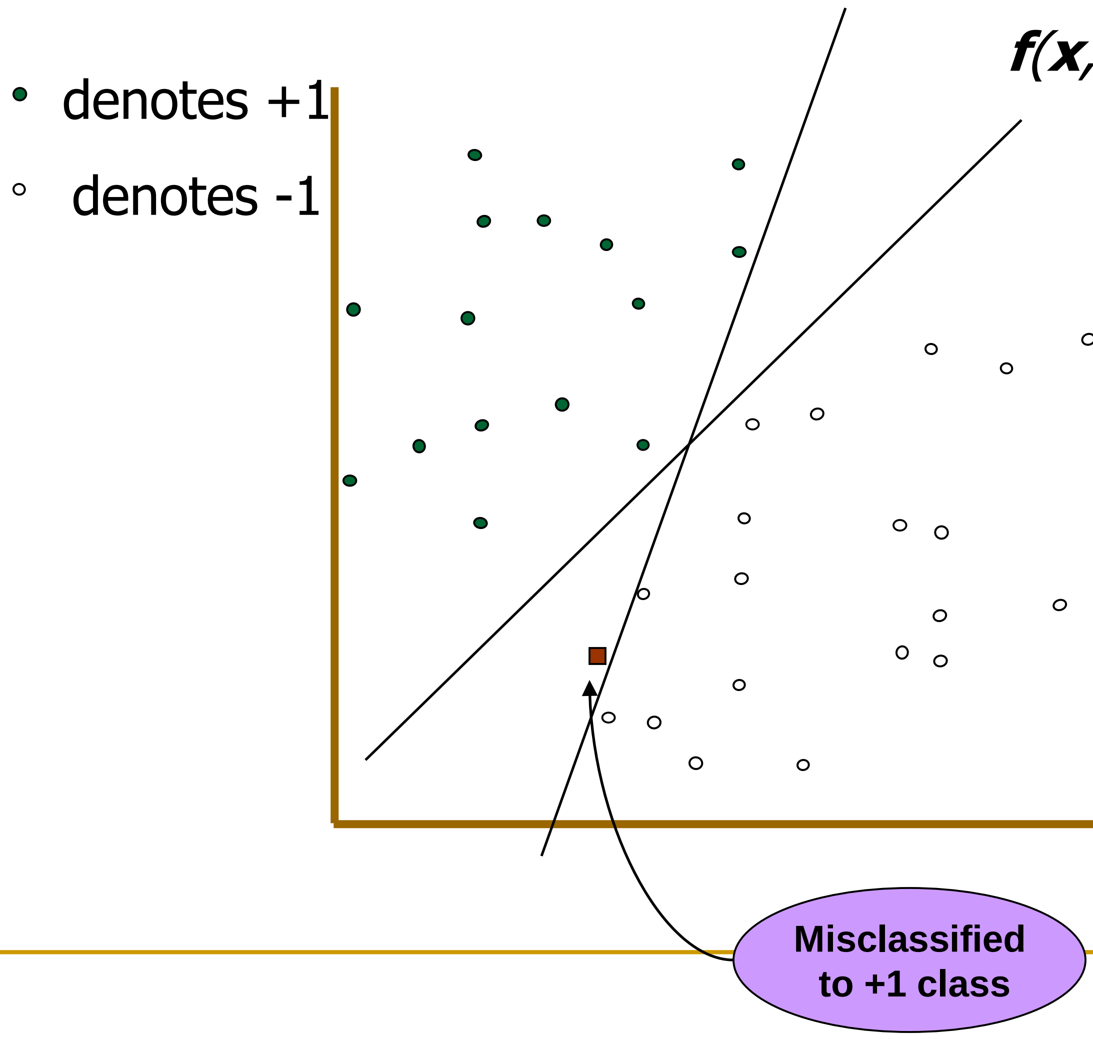
Any of these would be fine..

..but which is best?

Linear Classifiers

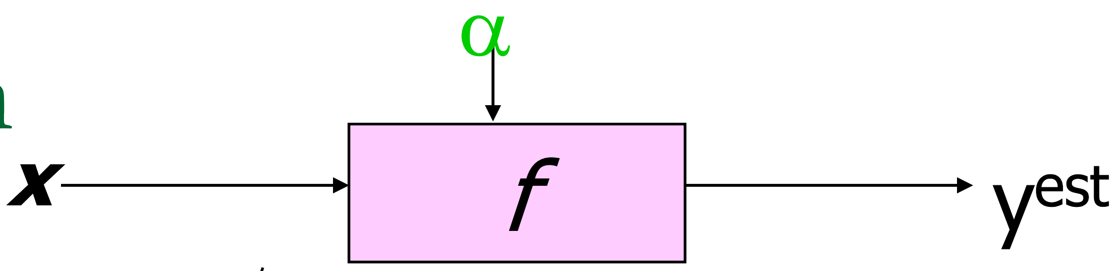


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$

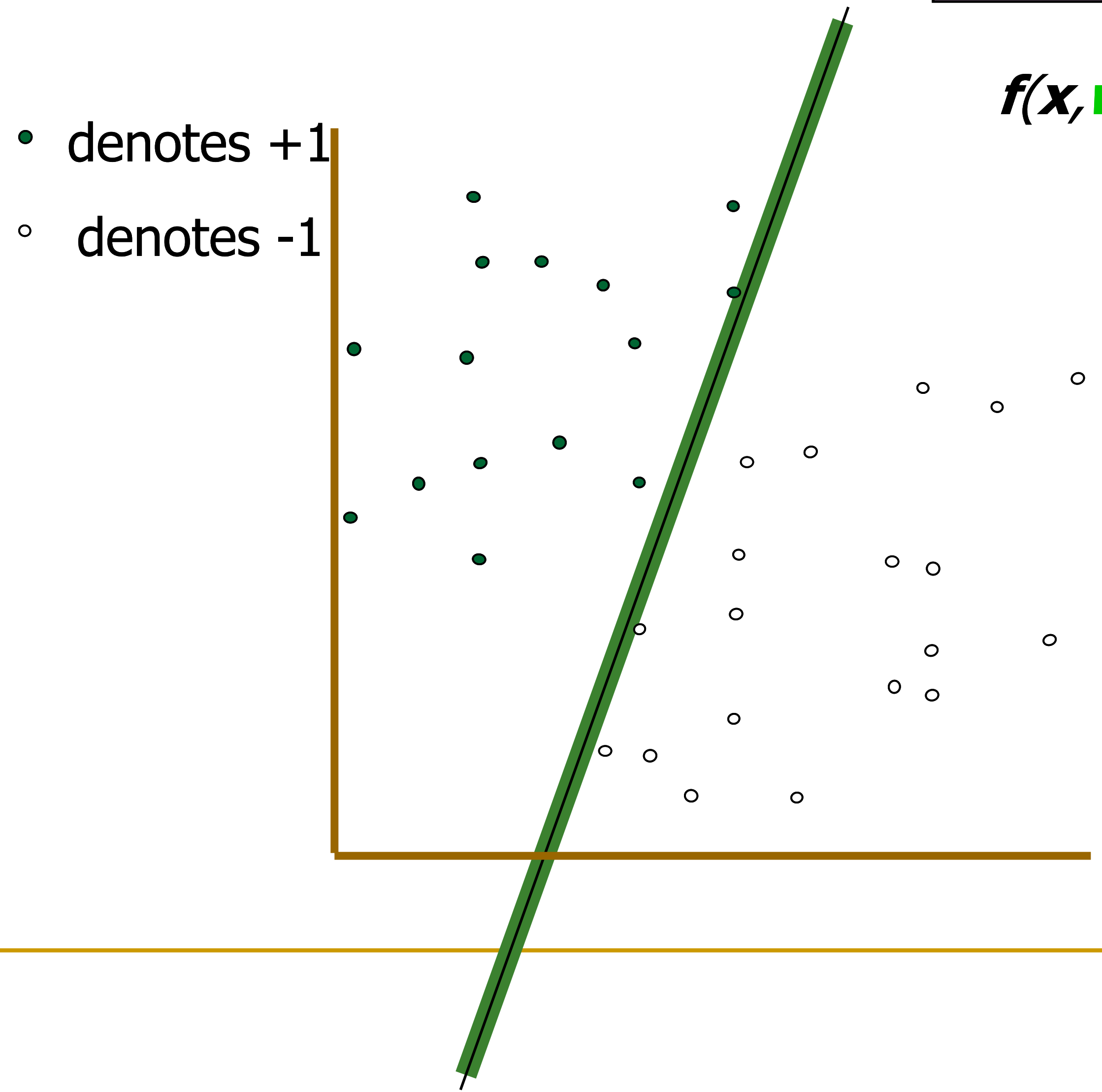


How would you classify this data?

Classifier Margin

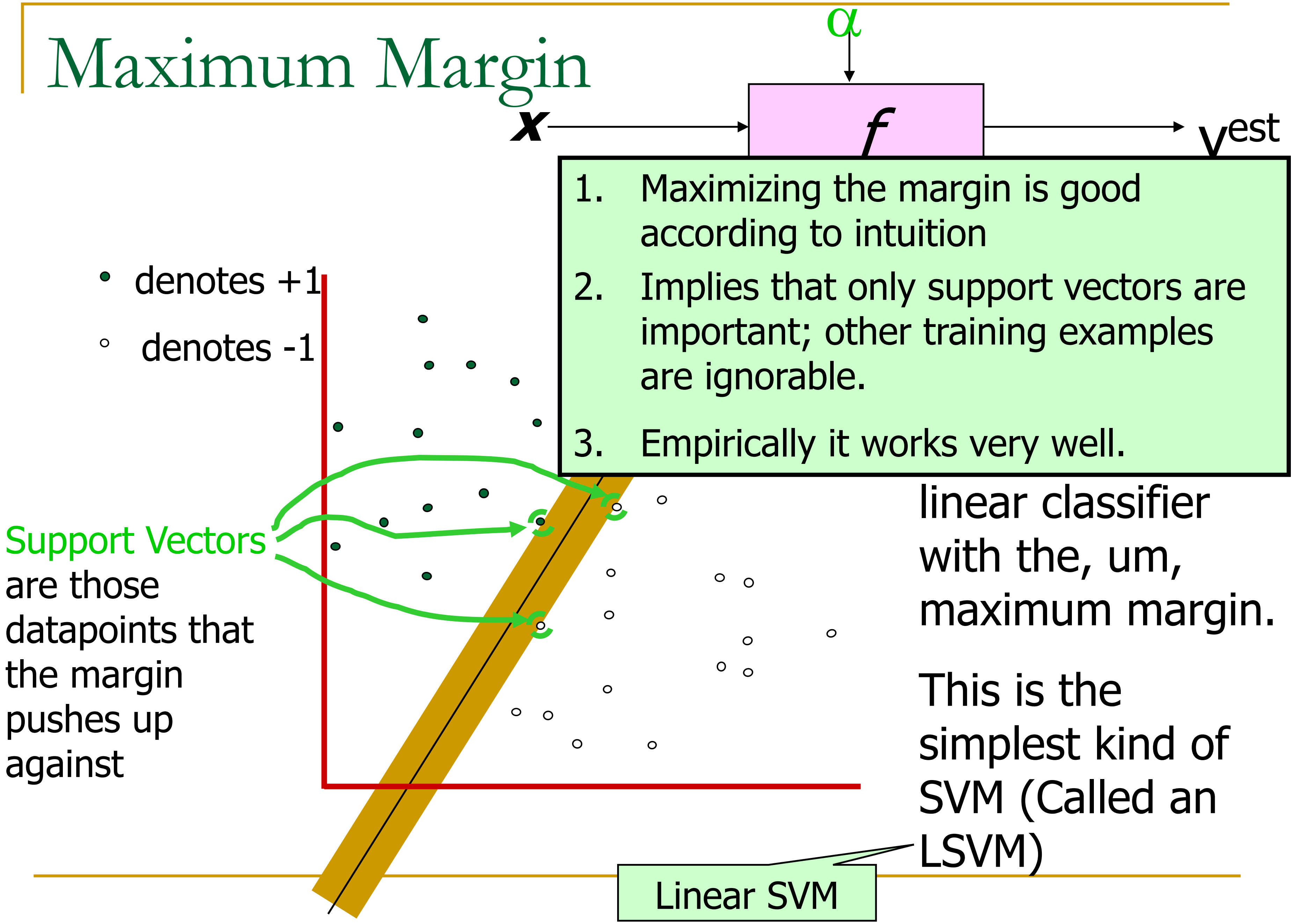


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



1. Maximizing the margin is good according to intuition
2. Implies that only support vectors are important; other training examples are ignorable.
3. Empirically it works very well.

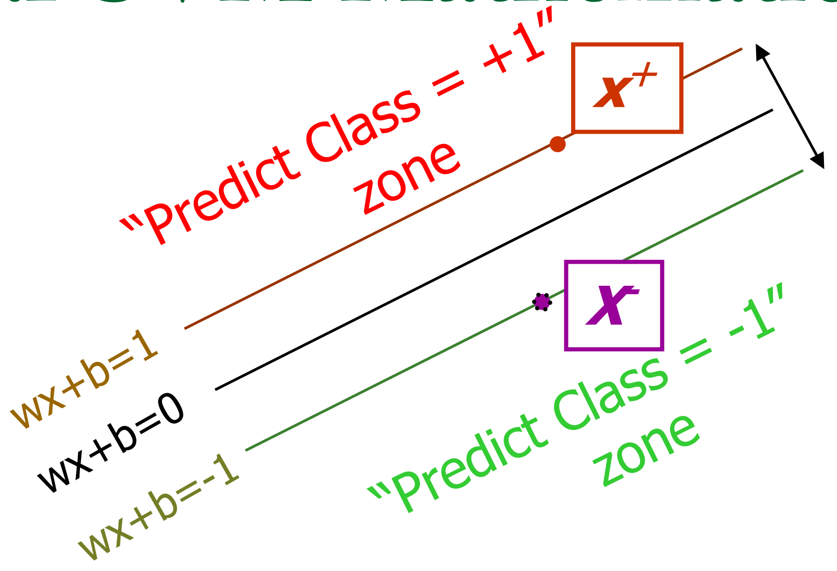
Support Vectors are those datapoints that the margin pushes up against

linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Linear SVM

Linear SVM Mathematically



M=Margin Width

What we know:

- ***w*** . ***x***$^+$ + *b* = +*1*
- ***w*** . ***x***$^-$ + *b* = -*1*
- ***w*** . (***x***$^+$-***x***$^-$) = *2*

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

Linear SVM Mathematically

- Goal: **1) Correctly classify all training data**

$$\left.\begin{aligned} wx_i + b &\geq 1 \quad \text{if } y_i = +1 \\ wx_i + b &\leq 1 \quad \text{if } y_i = -1 \end{aligned}\right\}$$

$$y_i(wx_i + b) \geq 1 \quad \text{for all i}$$

2) Maximize the Margin $M = \frac{2}{|w|}$

same as minimize $\frac{1}{2} w^t w$

- **We can formulate a Quadratic Optimization Problem and solve for w and b**

- Minimize $\Phi(w) = \frac{1}{2} w^t w$

  subject to $y_i(wx_i + b) \geq 1 \qquad \forall i$

Solving the Optimization Problem

Find **w** and b such that
$\mathbf{\Phi}(\mathbf{w}) = ½\ \mathbf{w}^{\mathrm{T}}\mathbf{w}$ is minimized;
and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + b) \geq 1$

- **Need to optimize a *quadratic* function subject to *linear* constraints.**
- **Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.**
- **The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:**

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - ½\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x_i}^{\mathrm{T}}\mathbf{x_j}$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- The solution has the form:

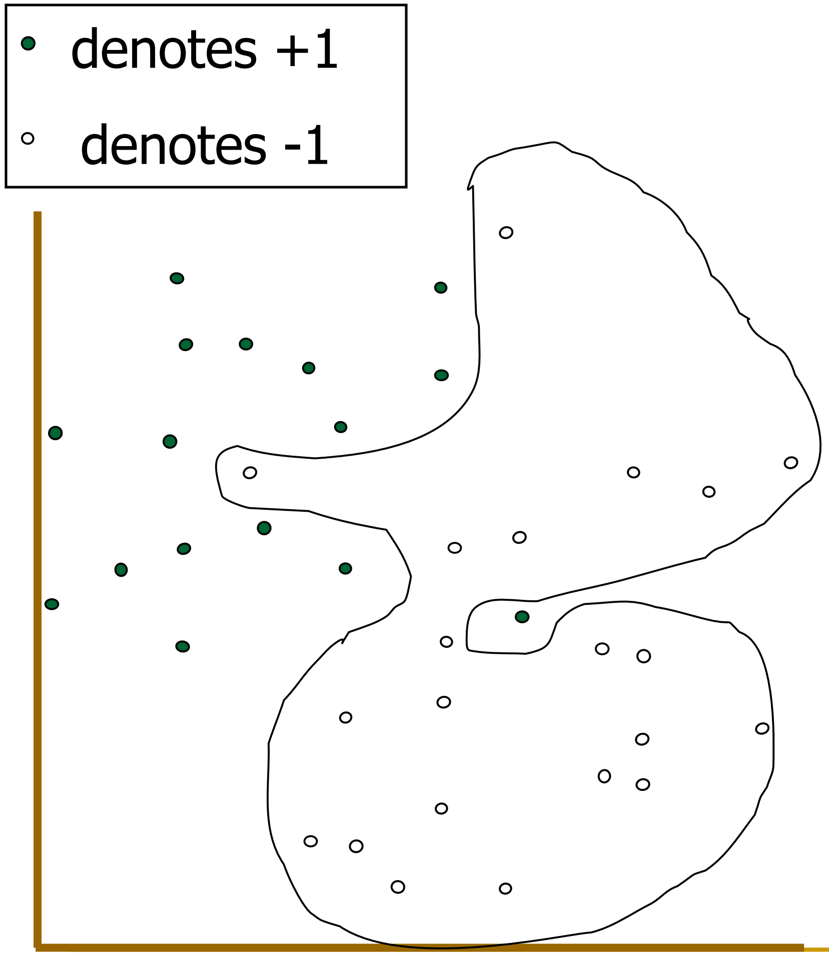
$$\boxed{\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w}^{\mathbf{T}}\mathbf{x_k} \text{ for any } \mathbf{x_k} \text{ such that } \alpha_k \neq 0}$$

- Each non-zero α_i indicates that corresponding $\mathbf{x_i}$ is a support vector.
- Then the classifying function will have the form:

$$\boxed{f(\mathbf{x}) = \Sigma \alpha_i y_i \mathbf{x_i}^{\mathbf{T}}\mathbf{x} + b}$$

- Notice that it relies on an *inner product* between the test point **x** and the support vectors $\mathbf{x_i}$.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x_i}^{\mathbf{T}}\mathbf{x_j}$ between all pairs of training points.

Dataset with noise



- **Hard Margin: So far we require all data points be classified correctly**

  - No training error

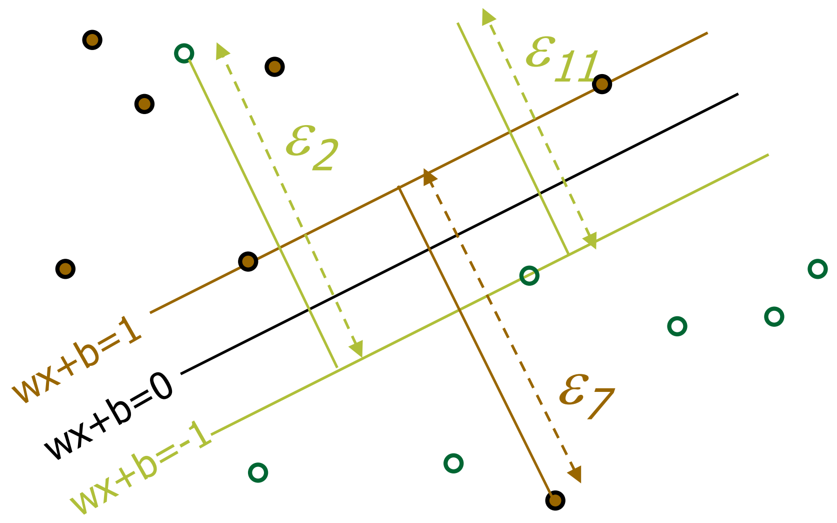
- **What if the training set is noisy?**

  - Solution 1: use very powerful kernels

OVERFITTING!

Soft Margin Classification

***Slack variables ξi* can be added to allow misclassification of difficult or noisy examples.**



What should our quadratic optimization criterion be?

Minimize

$$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

Hard Margin v.s. Soft Margin

- **The old formulation:**

Find **w** and *b* such that

$\boldsymbol{\Phi}(\mathbf{w}) = ½\ \mathbf{w}^T\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

$y_i\ (\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

- **The new formulation incorporating slack variables:**

Find **w** and *b* such that

$\boldsymbol{\Phi}(\mathbf{w}) = ½\ \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

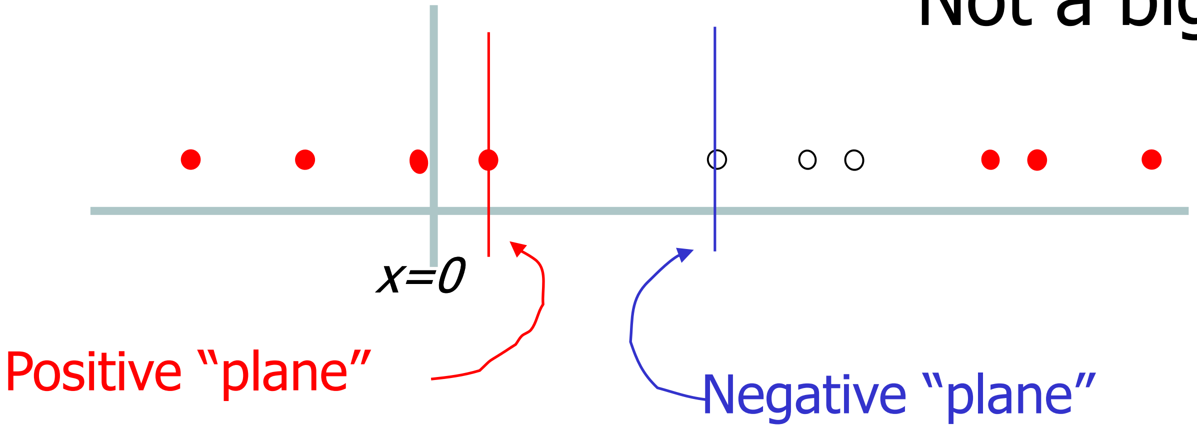
$y_i\ (\mathbf{w}^T\mathbf{x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

- **Parameter *C* can be viewed as a way to control overfitting.**

Hard 1-dimensional Dataset

What would SVMs do with this data?

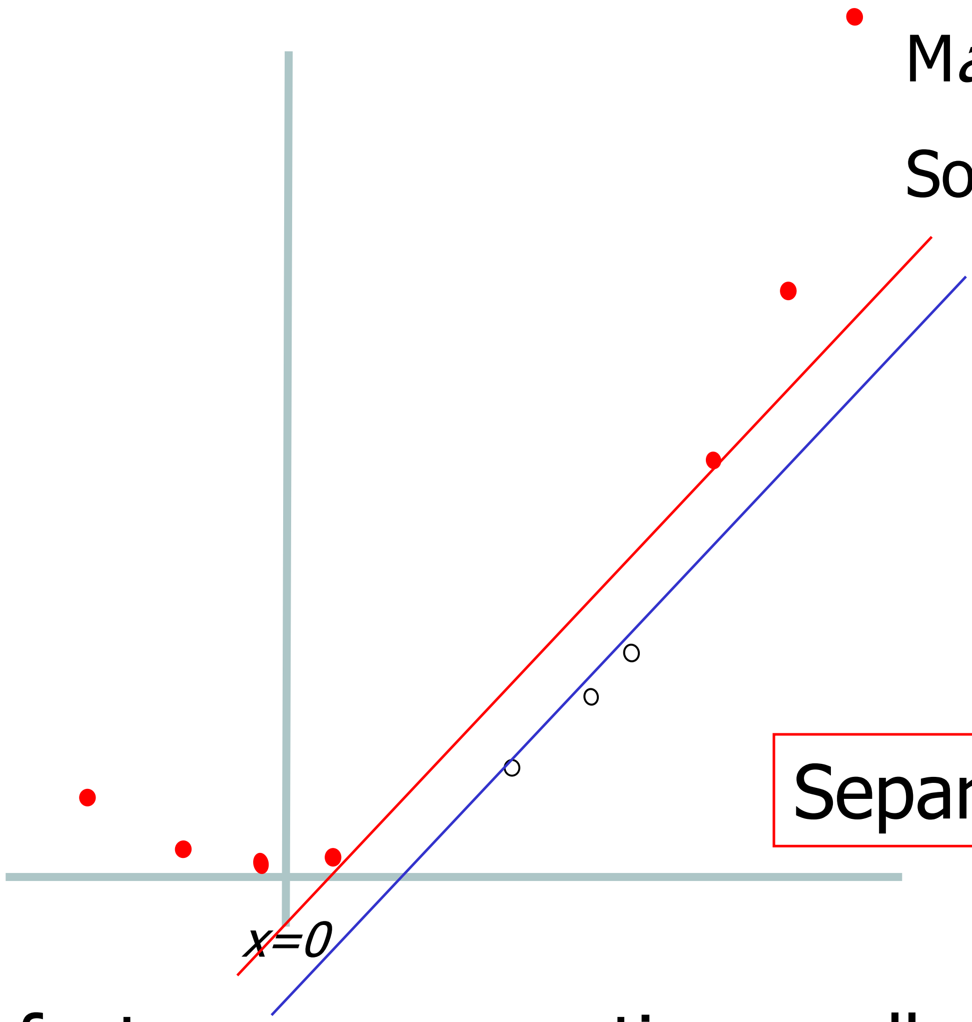
Not a big surprise



Doesn't look like slack variables will save us this time…

taken from Andrew W. Moore

Hard 1-dimensional Dataset



M*ake up a new feature!*

Sort of…

… computed from original feature(s)

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

Separable! MAGIC!

New features are sometimes called *basis functions.*

Now drop this "augmented" data into our linear SVM.

taken from Andrew W. Moore

Kernels and Linear Classifiers

Let $\vec{x} = [\vec{x}_1, \vec{x}_2] \in \mathbb{R}^2$ be a vectorial represenation of object $x \in \mathcal{X}$

Let $\phi : \mathcal{X} \to \mathcal{K} \subset \mathbb{R}^3$ feature map be given by

$$\phi(\vec{x}) \doteq [\vec{x}_1, \vec{x}_2^2, \vec{x}_1\vec{x}_2]^T \in \mathcal{K} \subset \mathbb{R}^3$$

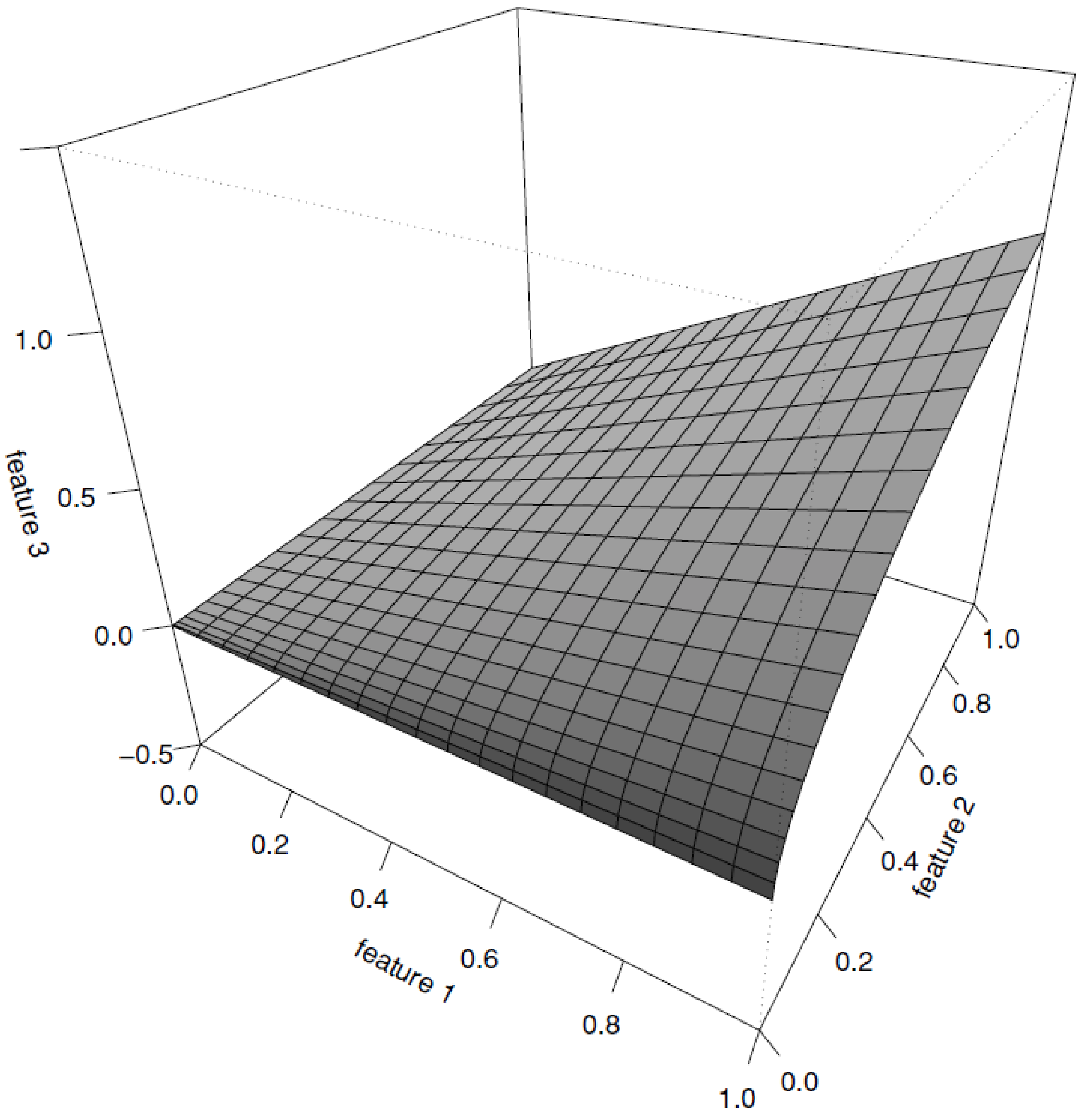
Def. Feature space: $\mathcal{K}$

We will use linear classifiers in this feature space.

In the original space $\mathbb{R}^2$ for a given $\mathbf{w} \in \mathbb{R}^3$ the decision surface is:

$$\boxed{\tilde{X}_0(\mathbf{w}) = \{\vec{x} \in \mathbb{R}^2 \mid w_1\vec{x}_1 + w_2\vec{x}_2^2 + w_3\vec{x}_1\vec{x}_2 = 0\}}$$

- This is nonlinear in $\vec{x} \in \mathbb{R}^2$
- This is linear in the feature space $\phi(\vec{x}) \in \mathcal{K} \subset \mathbb{R}^3$



$$\phi(\vec{x}) \doteq [\vec{x}_1, \vec{x}_2^2, \vec{x}_1\vec{x}_2]^T \in \mathcal{K} \subset \mathbb{R}^3 \text{ feature map}$$

Picture is taken from R. Herbrich

Kernels and Linear Classifiers

$$\phi(\vec{x}) \doteq [\phi_1(\vec{x}), \phi_2(\vec{x}), \phi_3(\vec{x})] \doteq [\vec{x}_1, \vec{x}_2^2, \vec{x}_1\vec{x}_2]^T$$

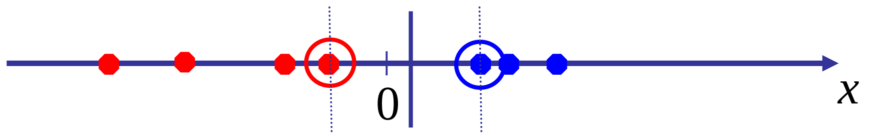
Feature functions

- We seek for a small set of basis vectors $\{\phi_i\}$ which allows perfect discrimination between the classes in $\mathcal{X}$ (**Feature selection**)

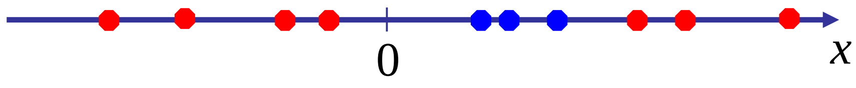
- If we have too many features $\Rightarrow$ overfitting can happen.

Non-linear SVMs

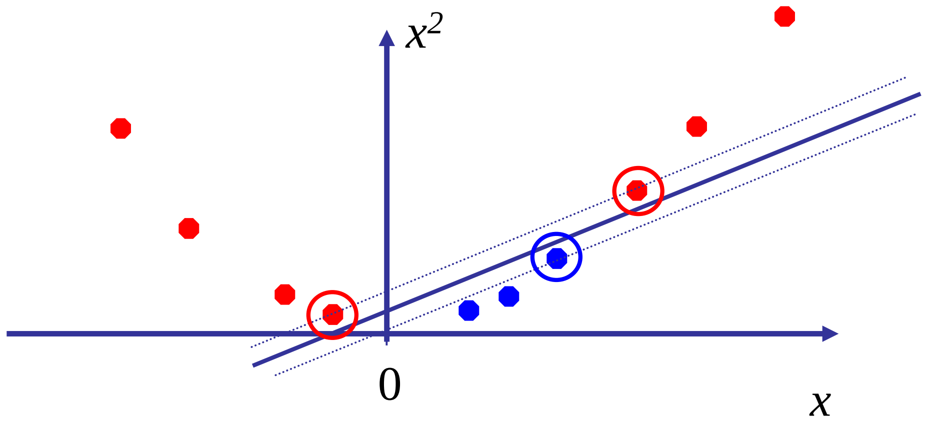
- Datasets that are linearly separable with some noise work out great:



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

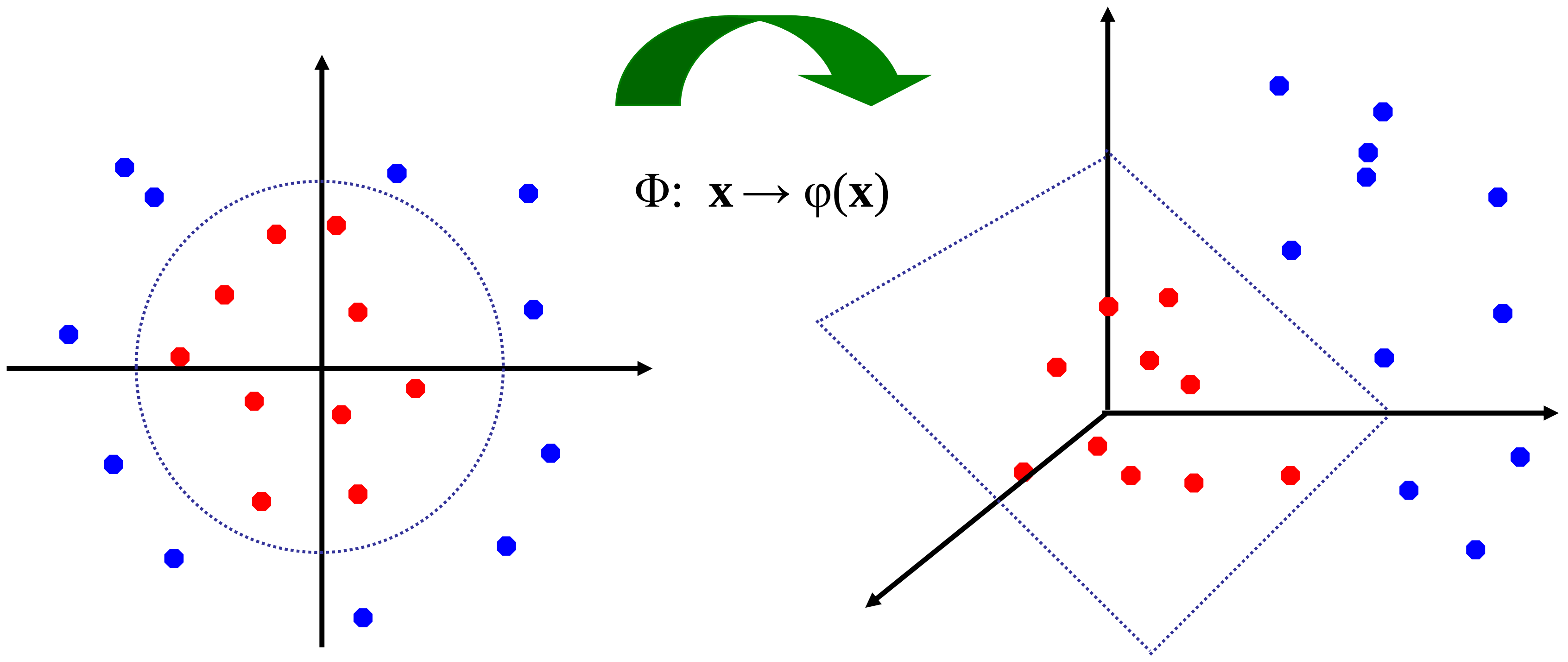


- How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

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



The “Kernel Trick”

- **To produce linear separability in Higher Dimension, the linear classifier relies on dot product between vectors $K(x_i,x_j)=x_i^T x_j$**
- **If every data point is mapped into high-dimensional space via some transformation Φ: x → φ(x), the dot product becomes:**

$$K(x_i,x_j)= \varphi(x_i)^T\varphi(x_j)$$

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

2-dimensional vectors x=[x_1 x_2]; let $K(x_i,x_j)=(1 + x_i^T x_j)^2$,

Need to show that $K(x_i,x_j)= \varphi(x_i)^T\varphi(x_j)$:

$$K(x_i,x_j)=(1 + x_i^T x_j)^2,$$
$$= 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$$
$$= [1\ \ x_{i1}^2\ \ \sqrt{2}\, x_{i1}x_{i2}\ \ x_{i2}^2\ \ \sqrt{2}x_{i1}\ \ \sqrt{2}x_{i2}]^T [1\ \ x_{j1}^2\ \ \sqrt{2}\, x_{j1}x_{j2}\ \ x_{j2}^2\ \ \sqrt{2}x_{j1}\ \ \sqrt{2}x_{j2}]$$
$$= \varphi(x_i)^T\varphi(x_j), \quad \text{where } \varphi(x) = [1\ \ x_1^2\ \ \sqrt{2}\, x_1x_2\ \ x_2^2\ \ \sqrt{2}x_1\ \ \sqrt{2}x_2]$$

What Functions are Kernels?

- **For some functions $K(x_i,x_j)$ checking that**

  $K(x_i,x_j)= \varphi(x_i)^T\varphi(x_j)$ can be cumbersome.
- **Mercer's theorem:**

  Every semi-positive definite symmetric function is a kernel
- **Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:**

K=

$K(\mathbf{x}_1,\mathbf{x}_1)$	$K(\mathbf{x}_1,\mathbf{x}_2)$	$K(\mathbf{x}_1,\mathbf{x}_3)$	…	$K(\mathbf{x}_1,\mathbf{x}_N)$
$K(\mathbf{x}_2,\mathbf{x}_1)$	$K(\mathbf{x}_2,\mathbf{x}_2)$	$K(\mathbf{x}_2,\mathbf{x}_3)$		$K(\mathbf{x}_2,\mathbf{x}_N)$
…	…	…	…	…
$K(\mathbf{x}_N,\mathbf{x}_1)$	$K(\mathbf{x}_N,\mathbf{x}_2)$	$K(\mathbf{x}_N,\mathbf{x}_3)$	…	$K(\mathbf{x}_N,\mathbf{x}_N)$

Examples of Kernel Functions

- Linear: $K(\mathbf{x_i},\mathbf{x_j}) = \mathbf{x_i}^T\mathbf{x_j}$

- Polynomial of power *p*: $K(\mathbf{x_i},\mathbf{x_j}) = (1 + \mathbf{x_i}^T\mathbf{x_j})^p$

- Gaussian (radial-basis function network):

$$K(\mathbf{x_i},\mathbf{x_j}) = \exp\left(-\frac{\left\|\mathbf{x_i} - \mathbf{x_j}\right\|^2}{2\sigma^2}\right)$$

- Sigmoid: $K(\mathbf{x_i},\mathbf{x_j}) = \tanh(\beta_0 \mathbf{x_i}^T\mathbf{x_j} + \beta_1)$

Non-linear SVMs Mathematically

- **Dual problem formulation:**

Find $\alpha_1 \ldots \alpha_N$ such that
$Q(\alpha) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(\mathrm{x}_i, \mathrm{x}_j)$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

- **The solution is:**

$$\boxed{f(\mathrm{x}) = \Sigma\alpha_i y_i K(\mathrm{x}_i, \mathrm{x}_j) + b}$$

- **Optimization techniques for finding α_i's remain the same!**

Nonlinear 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 dot product in the feature space.

Properties of SVM

- **Flexibility in choosing a similarity function**
- **Sparseness of solution when dealing with large data sets**

  - only support vectors are used to specify the separating hyperplane
- **Ability to handle large feature spaces**

  - complexity does not depend on the dimensionality of the feature space
- **Overfitting can be controlled by soft margin approach**
- **Nice math property: a simple convex optimization problem which is guaranteed to converge to a single global solution**
- **Feature Selection**

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

Weakness of SVM

- **It is sensitive to noise**

  - A relatively small number of mislabeled examples can dramatically decrease the performance

- **It only considers two classes**

  - how to do multi-class classification with SVM?

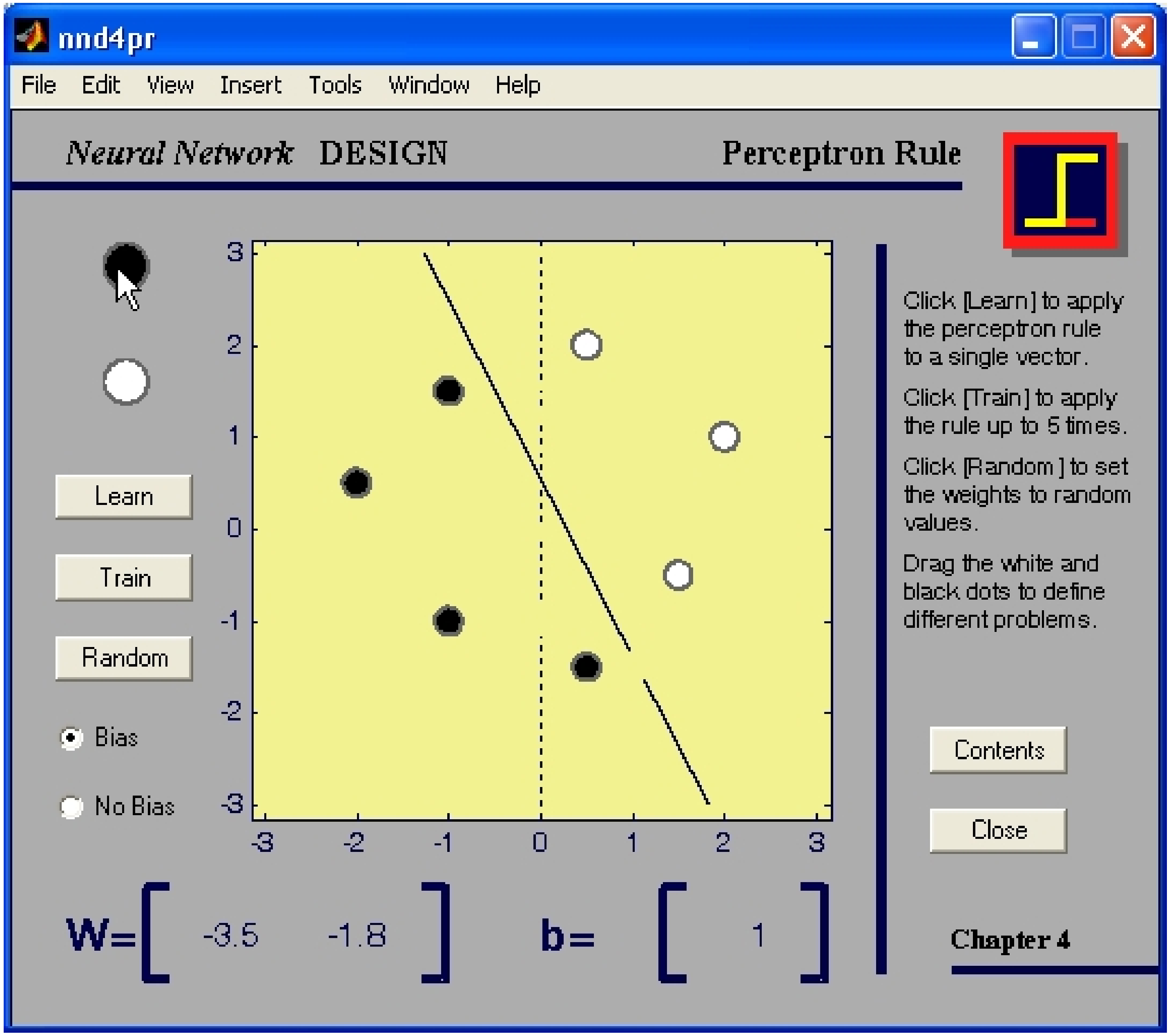
  - Answer:

  1) with output arity m, learn m SVM's

  - **SVM 1 learns "Output==1" vs "Output != 1"**
  - **SVM 2 learns "Output==2" vs "Output != 2"**
  - **:**
  - **SVM m learns "Output==m" vs "Output != m"**

  2)To predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.

Back to the Perceptron Example



Click [Learn] to apply the perceptron rule to a single vector.

Click [Train] to apply the rule up to 5 times.

Click [Random] to set the weights to random values.

Drag the white and black dots to define different problems.

$W = [\ -3.5 \quad -1.8\ ]$ $\quad b = [\ 1\ ]$

The Perceptron

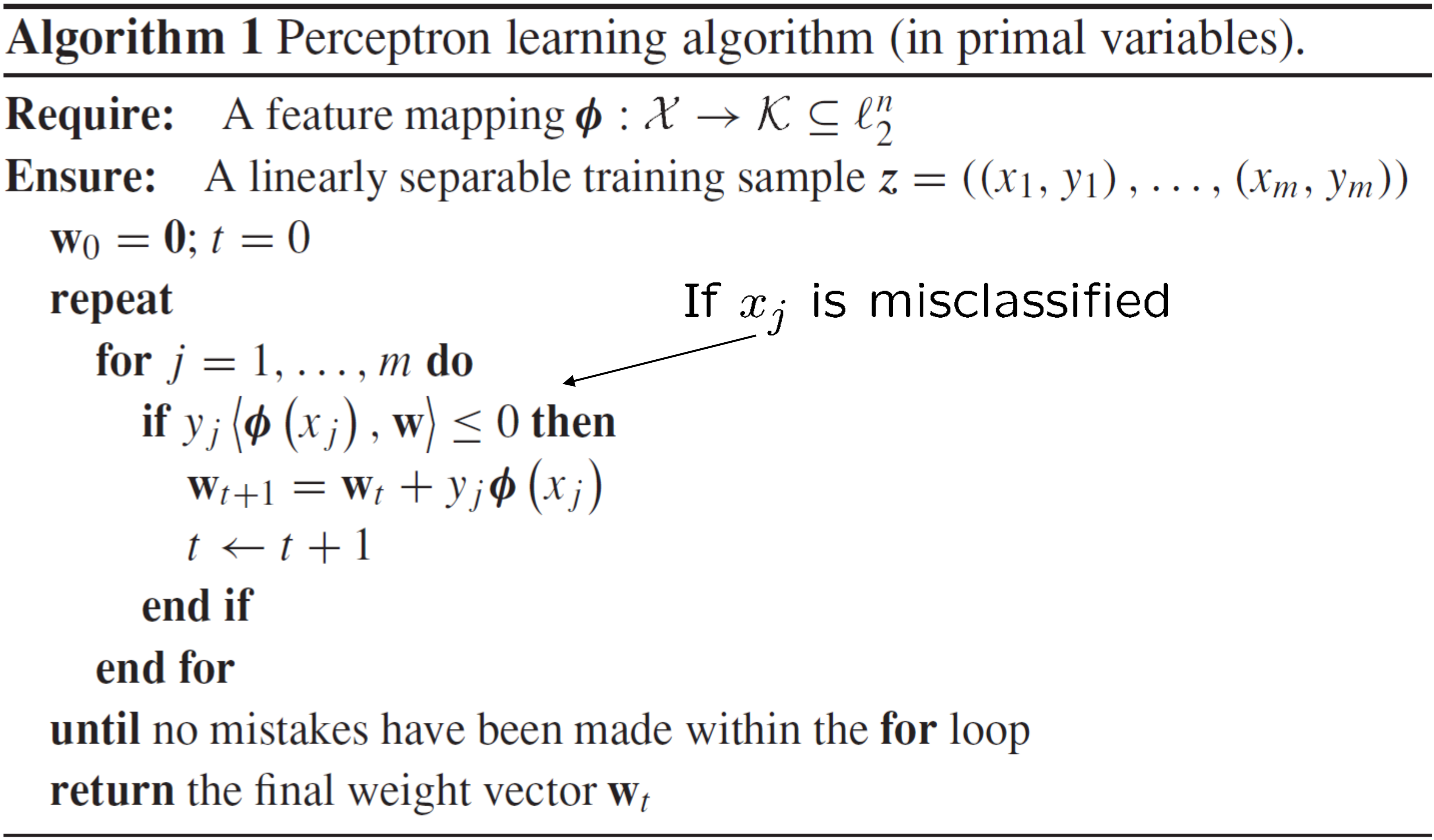
- **The primal algorithm in the feature space**

$D = \{(x_i, y_i), i = 1, \ldots, m\}$ training data set.

$\mathbf{x}_i = \phi(x_i) \in \mathcal{K} \subset \mathbb{R}^n$ feature map.

```
1., $\mathbf{w} = 0 \in \mathbb{R}^n$
2., $\forall\ (x_i, y_i),\ i = 1, \ldots, m$, evaluate $sign(y_i\langle \mathbf{x}_i, \mathbf{w}\rangle)$
3., If $x_i$ is misclassified $(sign(y_i\langle \mathbf{x}_i, \mathbf{w}\rangle) < 0)$
        then $\mathbf{w} := \mathbf{w} + y_i\mathbf{x}_i$
4., If no mistakes occur $\Rightarrow$ STOP
```

The primal algorithm in the feature space



Algorithm 1 Perceptron learning algorithm (in primal variables).

```
Require: A feature mapping $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{K} \subseteq \ell_2^n$
Ensure: A linearly separable training sample $\boldsymbol{z} = ((x_1, y_1), \ldots, (x_m, y_m))$
  $\mathbf{w}_0 = \mathbf{0}$; $t = 0$
  repeat
    for $j = 1, \ldots, m$ do
      if $y_j \langle \boldsymbol{\phi}(x_j), \mathbf{w} \rangle \leq 0$ then
        $\mathbf{w}_{t+1} = \mathbf{w}_t + y_j \boldsymbol{\phi}(x_j)$
        $t \leftarrow t + 1$
      end if
    end for
  until no mistakes have been made within the for loop
  return the final weight vector $\mathbf{w}_t$
```

If x_j is misclassified

Picture is taken from R. Herbrich

The Perceptron

We start at $\mathbf{w}_0 = 0 \in \mathcal{K} \subset \mathbb{R}^n$

$m=$ num of training examples,
$n = dim(\mathcal{K})$,

$t=$ num of mistakes so far

$\Rightarrow \mathbf{w}_t = \sum_{i=1}^{m} \alpha_i \phi(x_i) = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \in \mathbb{R}^n$ at time step t

Thus instead of tuning n variables
$\mathbf{w} = (w_1, \ldots, w_n)$ (**Primal variables**)
in the large n-dimensional feautre space $\mathcal{K}$, it is
enough to learn $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$ values (**Dual variables**).

The Perceptron

The Dual Algorithm in the feature space

$D = \{(x_i, y_i), i = 1, \ldots, m\}$ training data set.

$\mathbf{x}_i = \phi(x_i) \in \mathcal{K} \subset \mathbb{R}^n$ feaure map, $i = 1, \ldots, m$

$t=$ num of mistakes so far

$\Rightarrow \mathbf{w}_t = \sum_{i=1}^{m} \alpha_i \phi(x_i) = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \in \mathbb{R}^n$ at time step t

We update $\boldsymbol{\alpha_t} \in \mathbb{R}^m$ whenever a mistake occurs

1., $\boldsymbol{\alpha}_0 = 0 \in \mathbb{R}^m$

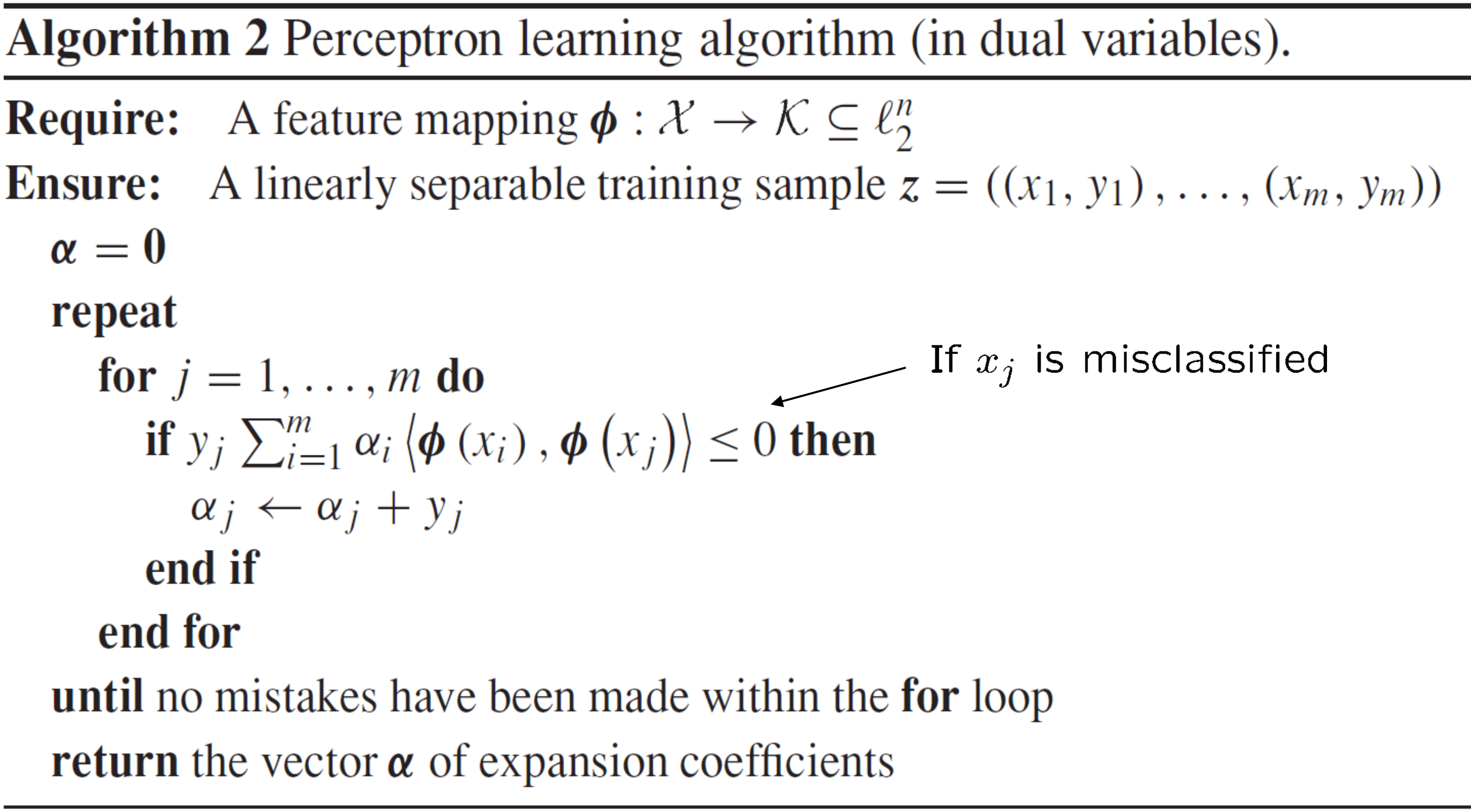
2., $\forall j = 1, \ldots, m$ evaluate

$$y_j \langle \mathbf{x}_j, \mathbf{w}_t \rangle = y_j \langle \mathbf{x}_j, \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \rangle = y_j \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}_j, \mathbf{x}_i \rangle$$

3., If x_j is misclassified $(y_j \langle \mathbf{x}_j, \mathbf{w}_t \rangle < 0)$ then update $\boldsymbol{\alpha}_t \in \mathcal{K}$

4., If no mistakes occur $\Rightarrow$ STOP

The Dual Algorithm in the feature space



Algorithm 2 Perceptron learning algorithm (in dual variables).

```
Require: A feature mapping $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{K} \subseteq \ell_2^n$
Ensure: A linearly separable training sample $\mathbf{z} = ((x_1, y_1), \ldots, (x_m, y_m))$
  $\boldsymbol{\alpha} = \mathbf{0}$
  repeat
    for $j = 1, \ldots, m$ do
      if $y_j \sum_{i=1}^{m} \alpha_i \langle \boldsymbol{\phi}(x_i), \boldsymbol{\phi}(x_j) \rangle \leq 0$ then
        $\alpha_j \leftarrow \alpha_j + y_j$
      end if
    end for
  until no mistakes have been made within the for loop
  return the vector $\boldsymbol{\alpha}$ of expansion coefficients
```

If x_j is misclassified

Picture is taken from R. Herbrich

The Dual Algorithm in the feature space

For the classification of a new object (x, y)
we have to evaluate

$$y \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

We don't have to know the actual values of $\mathbf{x} = \phi(x)$!

It is enough to know the inner products

$$\langle \mathbf{x}, \mathbf{x}_i \rangle \quad \forall i = 1, \ldots, m$$

between the object and the training points

Kernels

Definition: (kernel)

We are given $\phi : \mathcal{X} \to \mathcal{K} \subset l_2^n$ feautre mapping.

The **kernel** $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is the corresponding inner product function:

$$\boxed{k(x_i, x_j) \doteq \langle \underbrace{\phi(x_i)}_{\mathbf{x}_i}, \underbrace{\phi(x_j)}_{\mathbf{x}_j} \rangle_{\mathcal{K}} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle_{\mathcal{K}}}$$

Kernels

Definition: (Gram matrix, kernel matrix)

Gram matrix $G \in \mathbb{R}^{m\times m}$ of kernel k at $\{x_1, \ldots, x_m\}$:

$$\left.\begin{array}{l}\text{Given a kernel } k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \\ \text{and a training set } \{x_1, \ldots, x_m\}\end{array}\right\} \Rightarrow G_{ij} \doteq k(x_i, x_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

Definition: (Feature space, kernel space)

$$\mathcal{K} \doteq span\{\phi(x) \mid x \in \mathcal{X}\} \subset \mathbb{R}^n$$

Kernel technique

Definition:

Matrix $G \in \mathbb{R}^{m \times m}$ is positive semidefinite (PSD)
$\Leftrightarrow G$ is symmetric, and $0 \leq \boldsymbol{\beta}^T G \boldsymbol{\beta}\ \forall \boldsymbol{\beta} \in \mathbb{R}^{m \times m}$

$$\left.\begin{array}{l}\text{Given a kernel } k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \\ \text{and a training set } \{x_1, \ldots, x_m\}\end{array}\right\} \Rightarrow G_{ij} \doteq k(x_i, x_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle_{\mathcal{K}}$$

Lemma:

The Gram matrix is symmetric, PSD matrix.

Proof:

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_m] \in \mathbb{R}^{n \times m} \Rightarrow G = \mathbf{X}^T \mathbf{X} \in \mathbb{R}^{m \times m}$$

$$0 \leq \langle \mathbf{X}\boldsymbol{\beta}, \mathbf{X}\boldsymbol{\beta} \rangle_{\mathcal{K}} = \boldsymbol{\beta}^T G \boldsymbol{\beta}$$

Kernel technique

We already know that several algorithms use the **kernel values** only (...and NOT the **feature values**)!

Key idea:

Choose a nice kernel function k

rather than an ugly feautre mapping

$\phi : \mathcal{X} \to \mathbb{R}^n$

Kernel technique

We have seen so far how to build a kernel $k(\cdot,\cdot)$ from a given feature map $\phi : \mathcal{X} \rightarrow \mathbb{R}^n$

Now we want to do the opposite:

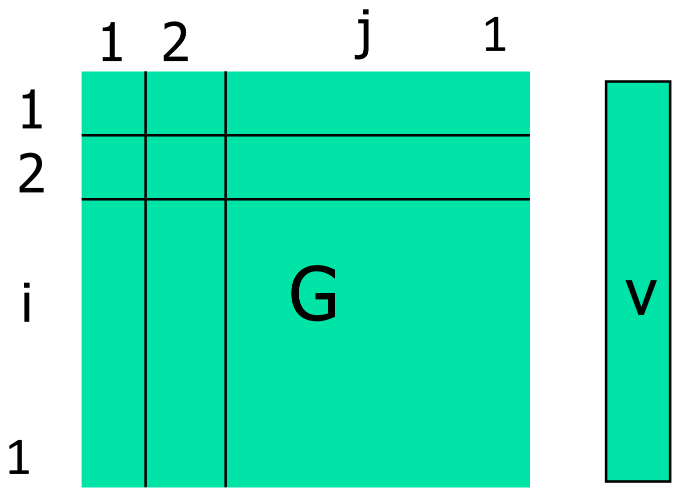
A function $k(\cdot,\cdot)$ is kernel $\Leftrightarrow$ there exists a feature space $\mathcal{K}$ and feature map $\phi : \mathcal{X} \rightarrow \mathcal{K}$, such that $k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle_{\mathcal{K}}$

Let us try to find ϕ and $\mathcal{K}$!

From Vector domain to Functions

- Observe that each vector v = (v[1], v[2], ..., v[n])
  is a mapping from the integers {1,2,..., n} to <

- We can generalize this easily to **INFINITE** domain
  w = (w[1], w[2], ..., w[n], ...)
  where w is mapping from {1,2,...} to <



$$(T_G v)(i) \doteq (Gv)(i) = \sum_{\underbrace{j=1}_{\int_{\mathcal{X}}}}^{\infty} \underbrace{G_{ij}}_{k(i,j)} \underbrace{v_j}_{f(j)}$$

From Vector domain to Functions

From integers we can further extend to

- $<$ or
- $<^m$
- Strings
- Graphs
- Sets
- Whatever
- …

Kernels

We don't need the $\mathcal{K} \subset l_2^n$ assumption. It is enough if $\mathcal{K}$ is a complete inner product (Hilbert) space.

Definition: inner product, Hilbert spaces

$\langle \cdot, \cdot \rangle : \mathcal{K} \times \mathcal{K} \to \mathbb{R}$ is an inner product in vector space $\mathcal{K}$, iff for all vectors $x, y, z \in \mathcal{K}$ and all scalars $a \in \mathbb{R}$:

* Symmetry: $\langle x, y \rangle = \langle y, x \rangle$.

* Linearity in the first argument:
$\langle ax, y \rangle = a\langle x, y \rangle$, $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$.

* Positive-definite: $\langle x, x \rangle \geq 0$ with equality only for $x = 0$.

This is more general than the inner product in $\mathbb{R}^n = l_2^n$

Examples:

- space of square integrable functions $L_2(\mathcal{X})$,
- space of square summable infinite series l_2

Integral operators, eigenfunctions

Definition: Eigenvalue, Eigenfunction

- λ is the eigenvalue,
- $\Psi \in L_2(\mathcal{X})$ is the eigenfunction

  of integral opreator $(T_k f)(\cdot) = \int_{\mathcal{X}} k(\cdot, x) f(x) dx$

$$\Leftrightarrow \begin{cases} \int_{\mathcal{X}} k(x, \bar{x}) \psi(\bar{x}) d\bar{x} = \lambda \psi(x) \quad \forall x \in \mathcal{X} \\ \\ \|\psi\|_{L_2}^2 \doteq \int_{\mathcal{X}} \psi^2(x) dx = 1 \end{cases}$$

The previous $Gv = \lambda v$ is a special case of this, when $\mathcal{X} = \{x_1, \ldots, x_r\}$ is a finite set.

Positive (semi) definite operators

Definition: Positive Definite Operator

$k(\cdot,\cdot)$ is symmetric kernel,

$$\Rightarrow (T_k f)(\cdot) \doteq \int_{\mathcal{X}} k(\cdot, x) f(x) dx$$

$T_k : L_2(\mathcal{X}) \to L_2(\mathcal{X})$ operator is positive semi definit

$$\Leftrightarrow \int_{\mathcal{X}} \int_{\mathcal{X}} k(\tilde{x}, x) f(x) f(\tilde{x}) dx d\tilde{x} \geq 0 \quad \forall f \in L_2(\mathcal{X})$$

The previous $v^T G v \geq 0$ is a special case of this, when $\mathcal{X} = \{x_1, \ldots, x_r\}$ is a finite set.

Mercer's theorem

$$(*)\quad \begin{cases} k(\cdot,\cdot) \in L_2(\mathcal{X} \times \mathcal{X}), \\ k \text{ is symmetric: } k(x,\tilde{x}) = k(\tilde{x},x) \\ (T_k f)(\cdot) = \int_{\mathcal{X}} k(\cdot,x) f(x) dx \text{ operator is pos. semi definit} \\ \psi_i,\ i = 1,2,\ldots \text{ are the eigenfunctions of } T_k \\ \text{with eigenvalues } \lambda_i \end{cases}$$

$$\Rightarrow \begin{cases} (\lambda_1, \lambda_2, \ldots) \in l_1, \quad \lambda_i \geq 0\ \forall i \\ \psi_i \in L_\infty(\mathcal{X}), \quad \forall i = 1,2,\ldots \\ k(x,\tilde{x}) = \sum_{i=1}^{\infty} \lambda_i \psi_i(x) \psi_i(\tilde{x}) \quad \forall x, \tilde{x} \end{cases}$$

2 variables

1 variable

Mercer's theorem

We like the Mercer's theorem becuase of the **expansion**:

$$k(x,\tilde{x}) = \sum_{i=1}^{\infty} \lambda_i \psi_i(x)\psi_i(\tilde{x}) \quad \forall x, \tilde{x}$$

It shows the **existence of the feature map** $\phi : \mathcal{X} \to \mathcal{K} \subset l_2$

Let $\mathcal{K} \doteq l_2$,
and let $\phi(x) \doteq (\sqrt{\lambda_1}\psi_1(x), \sqrt{\lambda_2}\psi_2(x), \ldots)^T$

$$\begin{aligned}
&\Rightarrow \langle \phi(x), \phi(\tilde{x})\rangle_{l_2} \\
&= (\sqrt{\lambda_1}\psi_1(x), \sqrt{\lambda_2}\psi_2(x), \ldots)^T(\sqrt{\lambda_1}\psi_1(x), \sqrt{\lambda_2}\psi_2(x), \ldots) \\
&= \sum_{i=1}^{\infty} \lambda_i \psi_i(x)\psi_i(\tilde{x}) = k(x,\tilde{x})
\end{aligned}$$

...☺

$\psi(x) = (\psi_1(x), \psi_2(x), \ldots)$ is known as **Mercer map**

A nicer characterization

The (*) condition in the Mercer's theorem is a bit ugly, but we have a nicer form that characterizes when a function $k(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel

(i.e. scalar product in some inner product space)

Theorem: nicer kernel characterization

$k(\cdot,\cdot)$ is a (Mercer) kernel

$\Leftrightarrow$ $(T_k f)(\cdot)$ is a pos. semi definite operator

$\Leftrightarrow$ $G = (k(x_i, x_j))_{ij}^r \in \mathbb{R}^{r \times r}$ Gram matrix is pos. semi definite $\forall r$, $\forall (x_1, \ldots, x_r) \in \mathcal{X}^r$

Vapnik-Chervonenkis dimension

$$\text{TESTERR}(\alpha) = E\left[\frac{1}{2}\left|y - f(x,\alpha)\right|\right] \qquad \text{TRAINERR}(\alpha) = \frac{1}{R}\sum_{k=1}^{R}\frac{1}{2}\left|y_k - f(x_k,\alpha)\right|$$

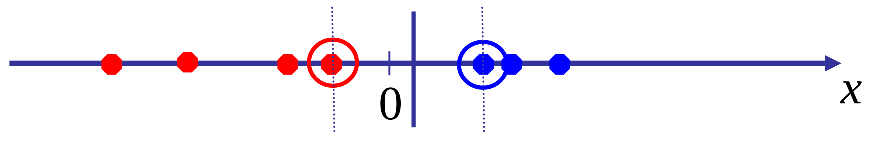
- Given some machine **f**, let *h* be its VC dimension.
- *h* is a measure of **f**'s power (*h* does not depend on the choice of training set)
- Vapnik showed that with probability 1-η

$$\text{TESTERR}(\alpha) \leq \text{TRAINERR}(\alpha) + \sqrt{\frac{h(\log(2R/h)+1) - \log(\eta/4)}{R}}$$

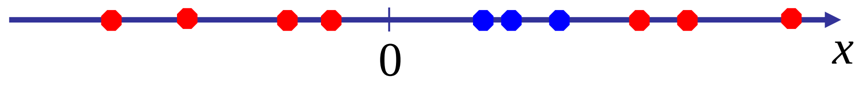
This gives us a way to estimate the error on future data based only on the training error and the VC-dimension of ***f***

Non-linear SVMs

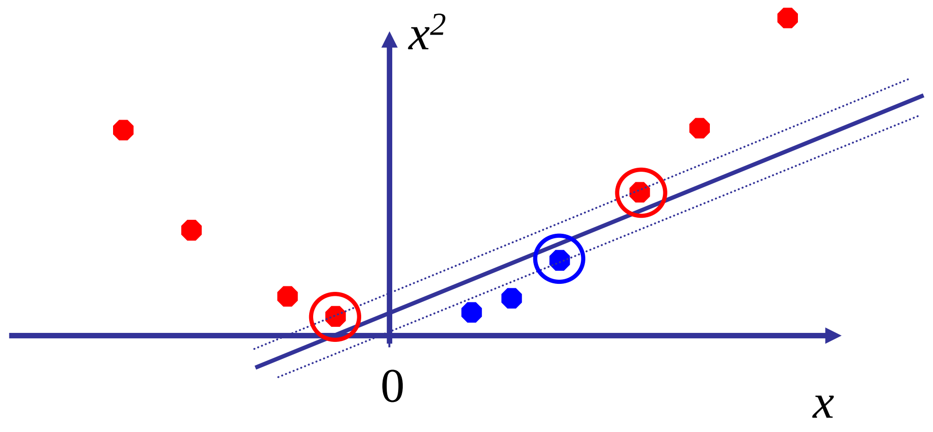
- Datasets that are linearly separable with some noise work out great:



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



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



The “Kernel Trick”

- **To produce linear separability in Higher Dimension, the linear classifier relies on dot product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^T\mathbf{x}_j$**
- **If every data point is mapped into high-dimensional space via some transformation Φ: x → φ(x), the dot product becomes:**

$$K(\mathbf{x}_i,\mathbf{x}_j)= \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$$

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

2-dimensional vectors x=$[x_1\ \ x_2]$; let $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^T\mathbf{x}_j)^2$,

Need to show that $K(\mathbf{x}_i,\mathbf{x}_j)= \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$:

$$K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^T\mathbf{x}_j)^2,$$

$$= 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$$

$$= [1\ \ x_{i1}^2\ \ \sqrt{2}\, x_{i1}x_{i2}\ \ x_{i2}^2\ \ \sqrt{2}x_{i1}\ \ \sqrt{2}x_{i2}]^T [1\ \ x_{j1}^2\ \ \sqrt{2}\, x_{j1}x_{j2}\ \ x_{j2}^2\ \ \sqrt{2}x_{j1}\ \ \sqrt{2}x_{j2}]$$

$$= \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j), \quad \text{where } \varphi(\mathbf{x}) = [1\ \ x_1^2\ \ \sqrt{2}\, x_1x_2\ \ x_2^2\ \ \sqrt{2}x_1\ \ \sqrt{2}x_2]$$

Examples of Kernel Functions

- Linear: $K(\mathbf{x_i},\mathbf{x_j}) = \mathbf{x_i}^T\mathbf{x_j}$

- Polynomial of power p: $K(\mathbf{x_i},\mathbf{x_j}) = (1+ \mathbf{x_i}^T\mathbf{x_j})^p$

- Gaussian (radial-basis function network):

$$K(\mathbf{x_i},\mathbf{x_j}) = \exp\left(-\frac{\left\|\mathbf{x_i}-\mathbf{x_j}\right\|^2}{2\sigma^2}\right)$$

- Sigmoid: $K(\mathbf{x_i},\mathbf{x_j}) = \tanh(\beta_0\mathbf{x_i}^T\mathbf{x_j} + \beta_1)$

Mercer's theorem

$$(*)\quad \begin{cases} k(\cdot,\cdot) \in L_2(\mathcal{X} \times \mathcal{X}), \\ k \text{ is symmetric: } k(x,\tilde{x}) = k(\tilde{x},x) \\ (T_k f)(\cdot) = \int_{\mathcal{X}} k(\cdot,x) f(x) dx \text{ operator is pos. semi definit} \\ \psi_i,\ i = 1,2,\ldots \text{ are the eigenfunctions of } T_k \\ \text{with eigenvalues } \lambda_i \end{cases}$$

$$\Rightarrow \begin{cases} (\lambda_1, \lambda_2, \ldots) \in l_1, \quad \lambda_i \geq 0\ \forall i \\ \psi_i \in L_\infty(\mathcal{X}), \quad \forall i = 1,2,\ldots \\ k(x,\tilde{x}) = \sum_{i=1}^{\infty} \lambda_i \psi_i(x) \psi_i(\tilde{x}) \quad \forall x, \tilde{x} \end{cases}$$

2 variables

1 variable

Reproducing Kernel Hilbert Spaces

For a given kernel $k(\cdot,\cdot)$ we already know how to define feature space $\mathcal{K}$, and $\phi : \mathcal{X} \to \mathcal{K}$ feature map (Mercer map):

$$\mathcal{K} = l_2, \text{ and } \phi(x) \doteq (\sqrt{\lambda_1}\psi_1(x), \sqrt{\lambda_2}\psi_2(x), \ldots)^T$$

Now, we show another way using RKHS

$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ given kernel $\Rightarrow \mathcal{F}_0 \doteq \{k(x,\cdot) | x \in \mathcal{X}\}$ function space

We will add inner product to $\mathcal{F}_0$ function space
$\Rightarrow$ Pre-Hilbert space

Completing (closing) a pre-Hilbert space → Hilbert space

Reproducing Kernel Hilbert Spaces

$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ given kernel $\Rightarrow \mathcal{F}_0 \doteq \{k(x,\cdot) | x \in \mathcal{X}\}$ function space

$$(x_1, \ldots, x_r) \text{ given } \Rightarrow f(\cdot) \doteq \sum_{i=1}^{r} \alpha_i k(x_i, \cdot) \quad \in \mathcal{F}_0$$

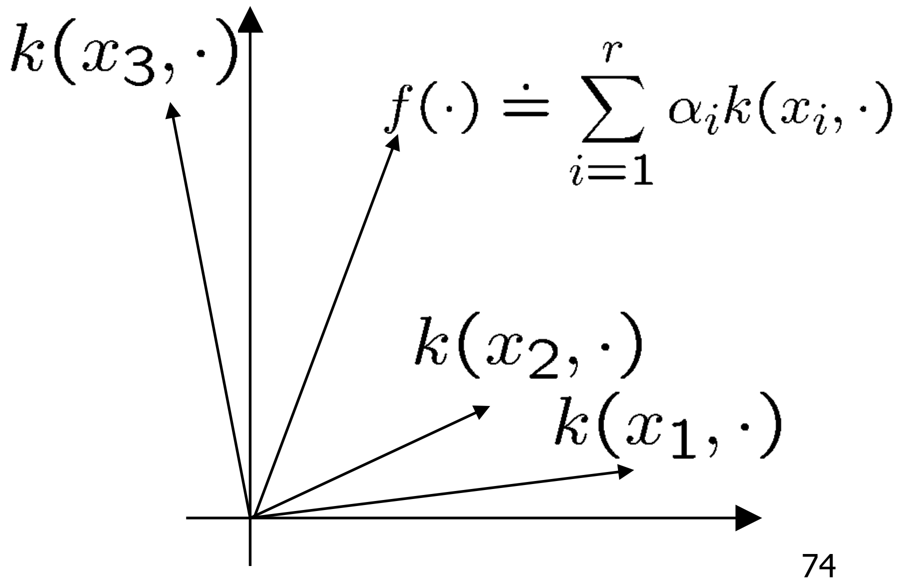
$$(\tilde{x}_1, \ldots, \tilde{x}_s) \text{ given } \Rightarrow g(\cdot) \doteq \sum_{j=1}^{s} \beta_j k(\tilde{x}_j, \cdot) \quad \in \mathcal{F}_0$$

The inner product:

$$\langle f, g \rangle_{\mathcal{F}_0} \doteq \sum_{i=1}^{r} \sum_{j=1}^{s} \alpha_i \beta_j k(x_i, \tilde{x}_j)$$

$$= \sum_{i=1}^{r} \alpha_i g(x_i)$$

$$= \sum_{j=1}^{s} \beta_j f(\tilde{x}_j) \quad (*)$$



Reproducing Kernel Hilbert Spaces

Note:

While for calculating $\langle f, g \rangle_{\mathcal{F}_0}$ we use their representations: $\alpha \in \mathbb{R}^r, \beta \in \mathbb{R}^s, \{x_i\}_{i=1}^r, \{\tilde{x}_j\}_{j=1}^s$
the $\langle f, g \rangle_{\mathcal{F}_0}$ is independent of the representation of f, g

Proof:

If we change $\alpha \in \mathbb{R}^r$ or $x_i \Rightarrow \langle f, g \rangle_{\mathcal{F}_0}$ doesn't change (because of (*)) The same for $\beta \in \mathbb{R}^s$

$$\boxed{\langle f, g \rangle_{\mathcal{F}_0} = \sum_{i_1}^{r} \alpha_i f(x_i) = \sum_{j=1}^{s} \beta_j f(\tilde{x}_j) \quad (*)}$$

Reproducing Kernel Hilbert Spaces

Lemma:

$\langle f, g \rangle$ is an inner product of $\mathcal{F}_0$

$\Rightarrow \mathcal{F}_0$ is pre-Hilbert space

$\mathcal{F} \doteq close(\mathcal{F}_0)$ is a Hilbert space

- **Pre-Hilbert** space:
  Like the Euclidean space with *rational* scalars only
- **Hilbert space**:
  Like the Euclidean space with *real* scalars

Proof:

1., $\langle f, g \rangle_{\mathcal{F}_0} = \langle g, f \rangle_{\mathcal{F}_0}$

2., $\langle cf + dg, h \rangle_{\mathcal{F}_0} = c\langle f, h \rangle_{\mathcal{F}_0} + d\langle g, h \rangle_{\mathcal{F}_0}$, $\forall c, d \in \mathbb{R}$, $\forall f, g, h \in \mathcal{F}_0$

3., $\langle f, f \rangle_{\mathcal{F}_0} \geq 0$

4., $\langle f, f \rangle_{\mathcal{F}_0} = 0 \Leftrightarrow f = 0$

Reproducing Kernel Hilbert Spaces

Lemma: (Reproducing property)

$$\langle f, k(x,\cdot)\rangle_{\mathcal{F}} = f(x)$$

Proof: definition of $\langle f, g\rangle_{\mathcal{F}}$

Lemma: The constructed features match to *k*

Huhh...

$$\langle \underbrace{k(x_i,\cdot)}_{\phi(x_i)}, \underbrace{k(x_j,\cdot)}_{\phi(x_j)} \rangle_{\mathcal{F}} = k(x_i, x_j)$$

Proof: reproducing property

Reproducing Kernel Hilbert Spaces

Proof of property 4.,:

$$0 \leq (f(x))^2 = \langle f, k(x,\cdot)\rangle_{\mathcal{F}}^2, \ \forall x$$

rep. property

$$\langle f, k(x,\cdot)\rangle_{\mathcal{F}}^2 \leq \langle f, f\rangle_{\mathcal{F}} \langle k(x,\cdot), k(x,\cdot)\rangle_{\mathcal{F}} \quad \forall x$$

we need only that <0,0>=0!

Hence, if $\langle f, f\rangle_{\mathcal{F}} = 0 \Rightarrow (f(x))^2 = 0, \ \forall x \in \mathcal{X}$

$$\Rightarrow f(x) = 0, \ \forall x \in \mathcal{X}$$

$$\Rightarrow f = 0$$

The Representer Theorem

In the perceptron problem we could use the dual algorithm, because we had this representation:

$$f(x) \doteq \langle \phi(x), \mathbf{w} \rangle_{\mathcal{K}} = \sum_{i=1}^{m} \alpha_i k(x, x_i)$$

and thus we had to update $\alpha_1, \ldots, \alpha_m$ only, and not $\mathbf{w} \in \mathcal{K}$!

The **Representer theorem** provides us a big class of problems, where the solution can be represented by

$$\boxed{f(\cdot) = \sum_{i=1}^{m} \alpha_i k(x_i, \cdot), \quad \alpha \in \mathbb{R}^m}$$

The Representer Theorem

Theorem:

$$\left.\begin{array}{r} k(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \text{ Mercer kernel on } \mathcal{X} \\ z = (x_1, y_1), \ldots, (x_m, y_m) \in (\mathcal{X} \times \mathcal{Y})^m \text{ training sample} \\ g_{emp} : (\mathcal{X} \times \mathcal{Y} \times \mathbb{R})^m \to \mathbb{R} \cup \{\infty\} \\ g_{reg} : \mathbb{R} \to [0, \infty) \text{ strictly increasing function} \\ \mathcal{F} : \text{ RKHS induced by } k(\cdot,\cdot) \end{array}\right\} \Rightarrow$$

$$\Rightarrow f^* = \arg\min_{f \in \mathcal{F}} R_{reg}[f, z]$$

$$\doteq \arg\min_{f \in \mathcal{F}} \underbrace{g_{emp}[(x_i, y_i, f(x_i))_{i \in \{1 \ldots m\}}]}_{\text{1st term, empirical loss}} + \underbrace{g_{reg}(\|f\|)}_{\text{2nd term, regularization}}$$

admits the following representation:

$$f^*(\cdot) = \sum_{i=1}^{m} \alpha_i k(x_i, \cdot), \quad \alpha = (\alpha_1, \ldots, \alpha_m) \in \mathbb{R}^m$$

The Representer Theorem

Message:

Optimizing in general function classes is difficult, but in RKHS it is only finite! (m) dimensional problem

Proof of Representer Theorem:

$\phi(x) \doteq k(x, \cdot) = \phi(x)(\cdot)$

$x_1, \ldots, x_m$ training samples are given

$$f \in \mathcal{F} \Rightarrow f(\cdot) = \sum_{i=1}^{m} \alpha_i \phi(x_i)(\cdot) + v(\cdot)$$

where $\mathcal{F} \ni v \perp span\{\phi(x_1), \ldots, \phi(x_m)\}$,

thus $\langle v, \phi(x_i) \rangle_{\mathcal{F}} = 0 \quad \forall i = 1, \ldots, m$

Proof of the Representer Theorem

Proof of Representer Theorem

$$f^* = \arg\min_{f\in\mathcal{F}} R_{reg}[f, z] \doteq \arg\min_{f\in\mathcal{F}} \underbrace{g_{emp}[(x_i, y_i, f(x_i))_{i\in\{1\ldots m\}}]}_{1^{st}\text{ term, empirical loss}} + \underbrace{g_{reg}(\|f\|)}_{2^{nd}\text{ term, regularization}}$$

$$\Rightarrow f(x_j) = \langle f, \underbrace{k(x_j, \cdot)}_{\phi(x_j)}\rangle_{\mathcal{F}} = \langle \sum_{i=1}^{m} \alpha_i \phi(x_i) + v, \phi(x_j)\rangle_{\mathcal{F}}$$

$$= \sum_{i=1}^{m} \alpha_i \langle \phi(x_i), \phi(x_j)\rangle_{\mathcal{F}} = \sum_{i=1}^{m} \alpha_i k(x_i, x_j)$$

$\Rightarrow f(x_j)$ depends only on $\alpha_1, \ldots, \alpha_m$, but independent from v!

$\Rightarrow 1^{st}$ term depends only on $\alpha_1, \ldots, \alpha_m$, but not on v

Proof of the Representer Theorem

$$f^* = \arg\min_{f\in\mathcal{F}} R_{reg}[f, z] \doteq \arg\min_{f\in\mathcal{F}} \underbrace{g_{emp}[(x_i, y_i, f(x_i))_{i\in\{1\ldots m\}}]}_{\text{1st term, empirical loss}} + \underbrace{g_{reg}(\|f\|)}_{\text{2nd term, regularization}}$$

Let us examine the 2nd term.

$$g_{reg}(\|f\|) = g_{reg}(\|\sum_{i=1}^{m}\alpha_i\phi(x_i) + v\|)$$

$$= g_{reg}\left(\sqrt{\|\sum_{i=1}^{m}\alpha_i\phi(x_i)\|_{\mathcal{F}}^2 + \|v\|_{\mathcal{F}}^2}\right.$$

since $\mathcal{F} \ni v \perp span\{\phi(x_1), \ldots, \phi(x_m)\}$

$$\geq g_{reg}(\|\sum_{i=1}^{m}\alpha_i\phi(x_i)\|_{\mathcal{F}})$$

with equality only if $v = 0$!

$\Rightarrow$ any minimizer f^* must have $v = 0$

$$\Rightarrow f^*(\cdot) = \sum_{i=1}^{m}\alpha_i k(x_i, \cdot)$$