Support Vector Machines

Reading:

Ben-Hur and Weston, “A User’s Guide to Support Vector Machines”

(linked from class web page)

Notation

- Assume a binary classification problem.
  - Instances are represented by vector $\mathbf{x} \in \Re^n$.
  - Training examples: $\mathbf{x} = (x_1, x_2, \ldots, x_n)$

    $S = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), ..., (\mathbf{x}_\mathrm{m}, t_m) \mid (\mathbf{x}_\mathrm{k}, t_k) \in \Re^n \times \{+1, -1\}\}$

  - Hypothesis: A function $h: \Re^n \rightarrow \{+1, -1\}$.

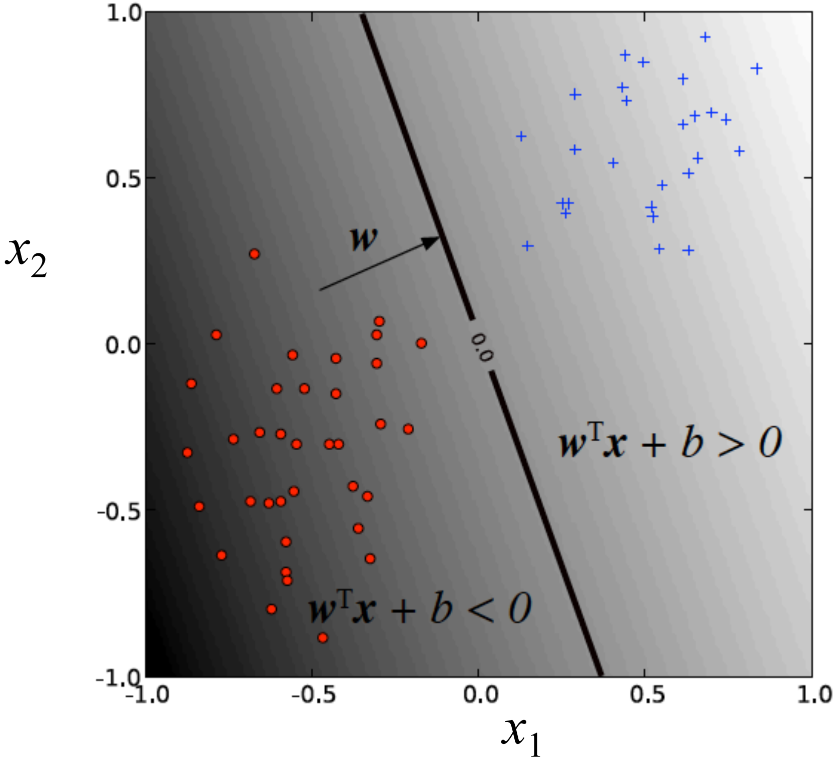
    $h(\mathbf{x}) = h(x_1, x_2, \ldots, x_n) \in \{+1, -1\}$

- Here, assume positive and negative instances are to be separated by the hyperplane

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

Equation of line:

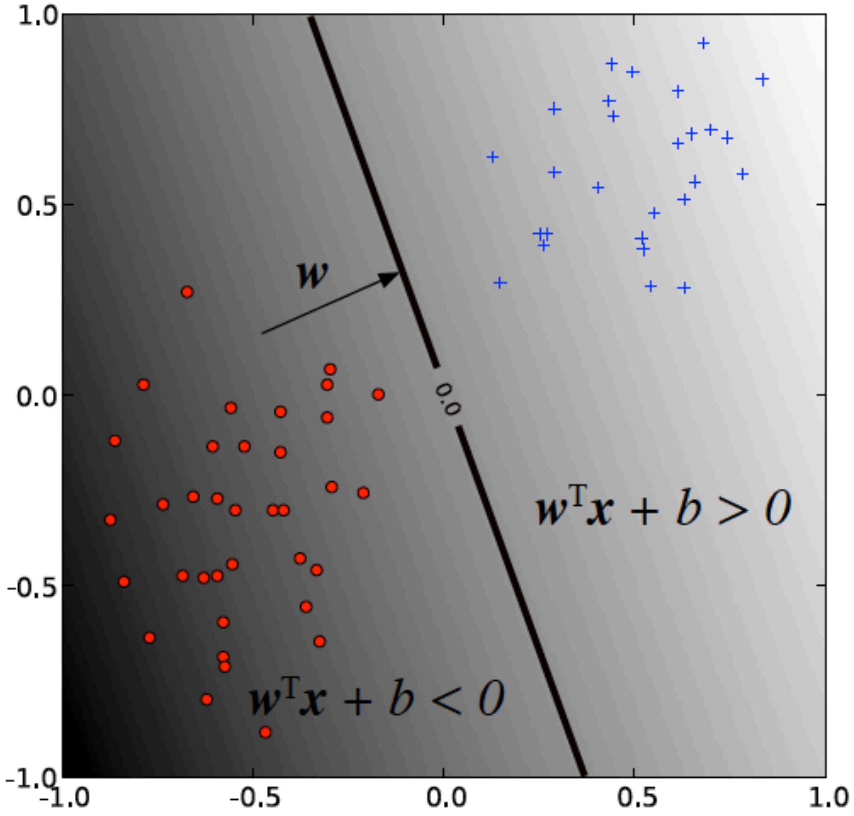
where b is the bias.



$$\mathbf{w} \cdot \mathbf{x} + b = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$$

$$= w_1 x_1 + w_2 x_2 + b = 0$$

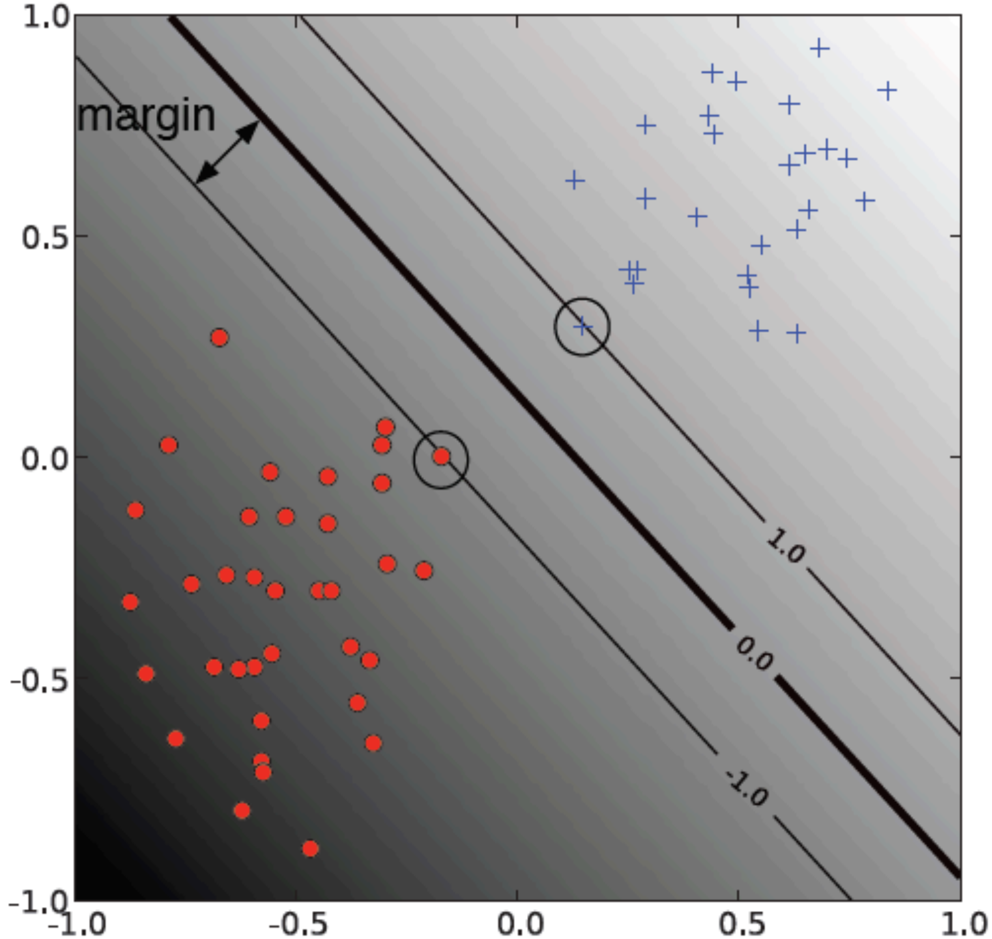
$$\mathbf{w} \cdot \mathbf{x} + b = 0$$



- **Intuition:** the best hyperplane (for future generalization) will "maximally" separate the examples

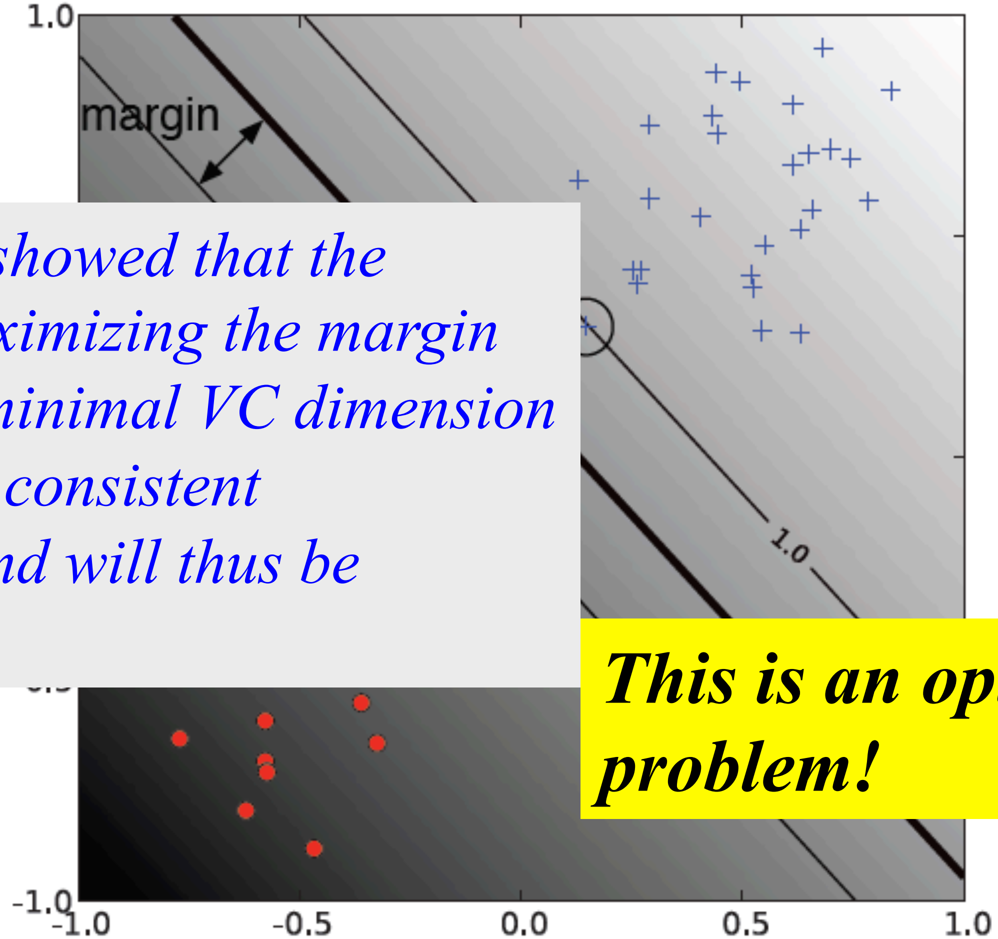
Definition of Margin (with respect to a hyperplane):

Distance from separating hyperplane to nearest positive (or negative) instance.



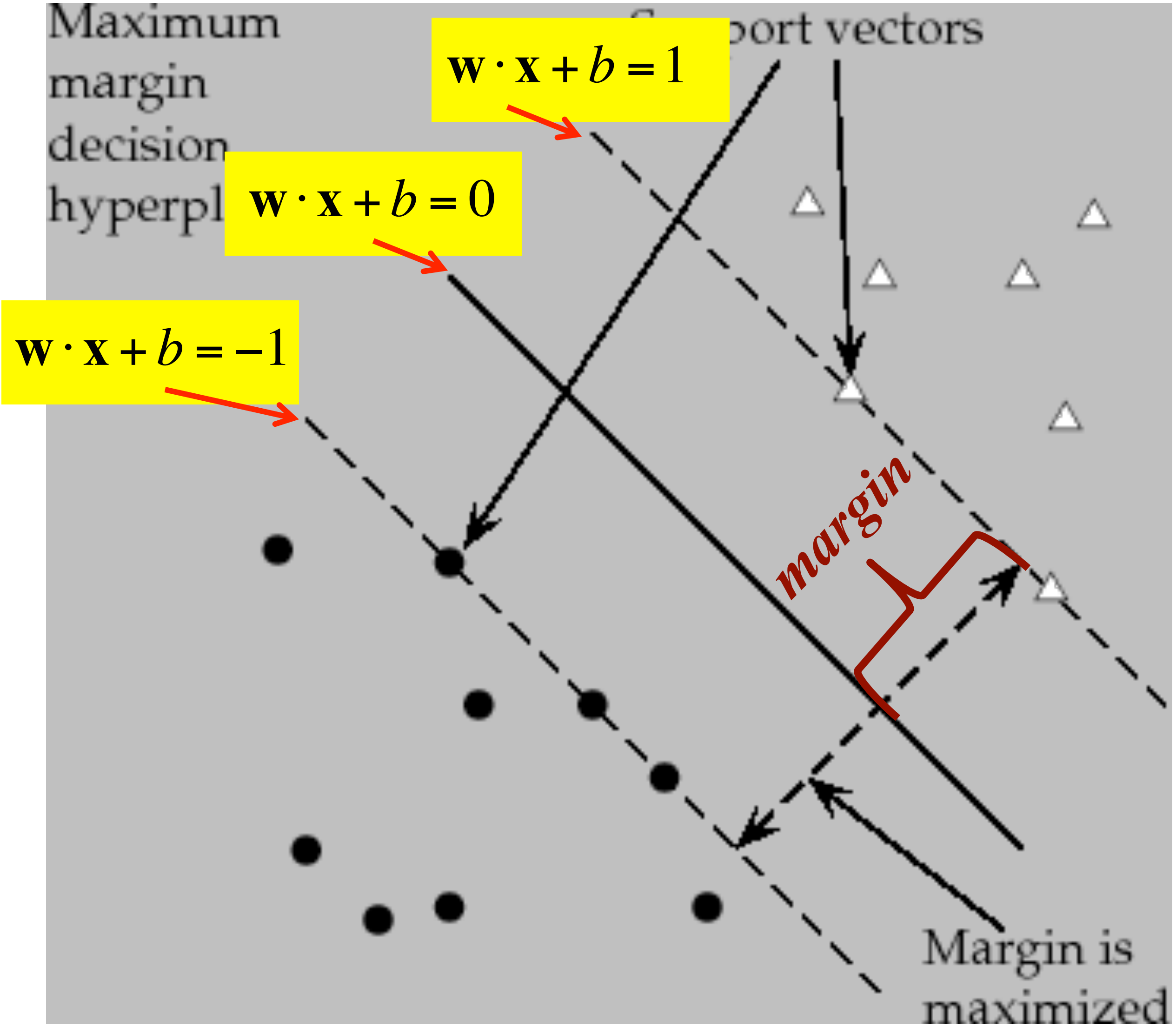
Definition of Margin (with respect to a hyperplane):

Distance from separating hyperplane to nearest positive (or negative) instance.



Vapnik (1979) showed that the hyperplane maximizing the margin of S will have minimal VC dimension in the set of all consistent hyperplanes, and will thus be optimal.

This is an optimization problem!



http://nlp.stanford.edu/IR-book/html/htmledition/img1260.png

Geometry of the margin

$$(\mathbf{w}\cdot\mathbf{x}_1)+b=+1$$
$$(\mathbf{w}\cdot\mathbf{x}_2)+b=-1$$

$$\mathbf{w}\cdot(\mathbf{x}_1-\mathbf{x}_2)=2$$

$$\Rightarrow \left\|(\mathbf{x}_1-\mathbf{x}_2)\right\| = \frac{2}{\|\mathbf{w}\|}$$

where $\|\mathbf{w}\| = \sqrt{\mathbf{w}\cdot\,\mathbf{w}} = \sqrt{w_1{}^2 + w_2{}^2}$

Without changing the problem, we can rescale our data to set *a* = 1

- The length of the margin is

$$\frac{1}{\|\mathbf{w}\|}$$

- So to maximize the margin, we need to minimize $\|\mathbf{w}\|$.

Minimizing ||**w**||

Find **w** and b by doing the following minimization:

$$\min_{\mathbf{w},b}\left(\frac{1}{2}\|\mathbf{w}\|^2\right)$$

subject to:

$$t_k\left(\mathbf{w}\cdot\mathbf{x}_k + b\right) \geq 1, \quad k = 1,\ldots,m$$

$$(t_k \in \{-1,+1\})$$

This is a quadratic, constrained optimization problem. Use "method of Lagrange multipliers" to solve it.

Dual Representation

- It turns out that **w** can be expressed as a linear combination of the training examples:

$$\mathbf{w} = \sum_{\mathbf{x}_k \in S} \alpha_k \, \mathbf{x}_k$$

where $\alpha_k \neq 0$ only if $\mathbf{x}_k$ is a support vector

- The results of the SVM training algorithm (involving solving a quadratic programming problem) are the α_k and the bias *b*.

SVM Classification

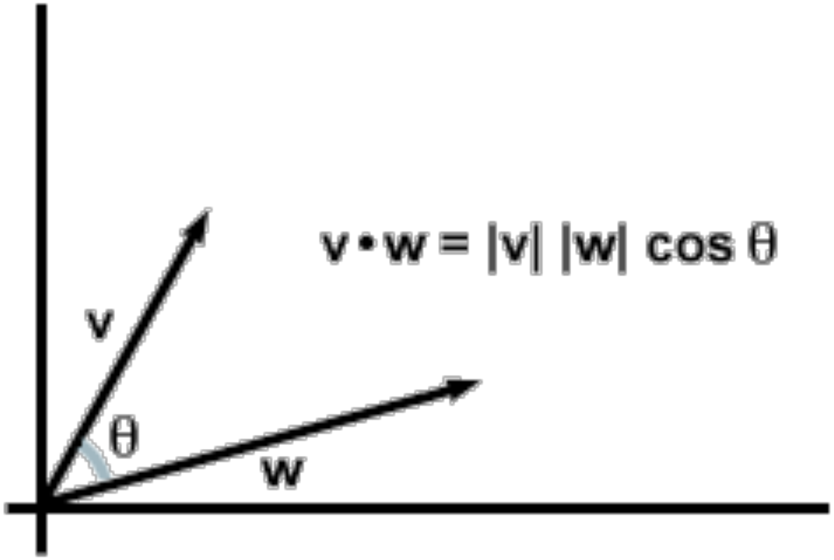
- Classify new example x as follows:

$$h(\mathbf{x}) = \operatorname{sgn}\left(\sum_{k=1}^{m} \alpha_k \left(\mathbf{x} \cdot \mathbf{x}_k \right) + b \right)$$

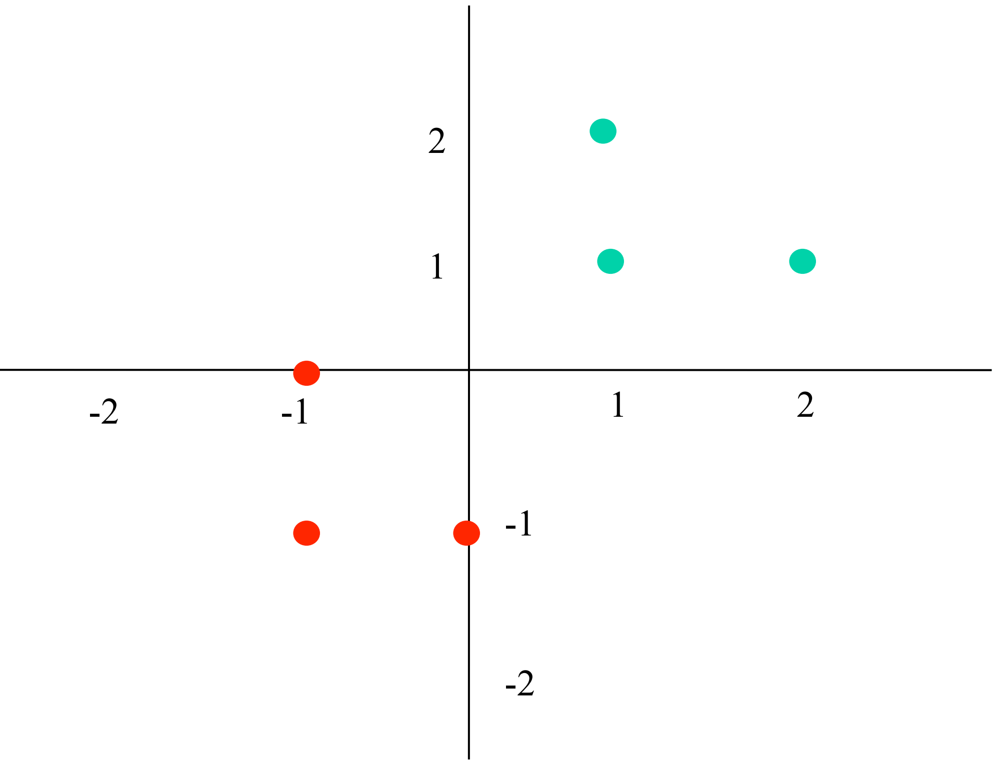
where $|S| = m$. (S is the training set)

- Note that dot product is

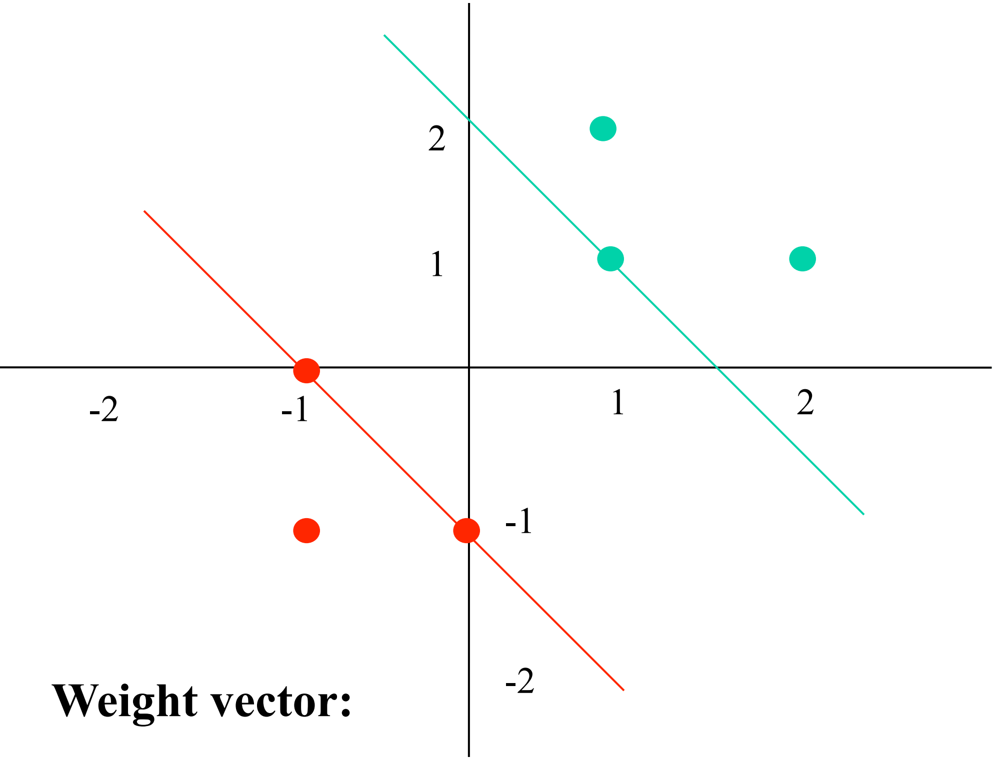
a kind of "similarity measure"



Example



Example



Input to SVM optimzer:

x_1	x_2	$class$
1	1	1
1	2	1
2	1	1
-1	0	-1
0	-1	-1
-1	-1	-1

Output from SVM optimizer:

Support vector	α
$(-1, 0)$	$-.208$
$(1, 1)$	$.416$
$(0, -1)$	$-.208$

$b = -.376$

Weight vector:

$$\mathbf{w} = \sum_{k \in \{\text{training examples}\}} \alpha_k \mathbf{x}_k$$

$$= -.208\,(-1,0) + .416\,(1,1) - .208\,(0,-1)$$

$$= (.624, .624)$$

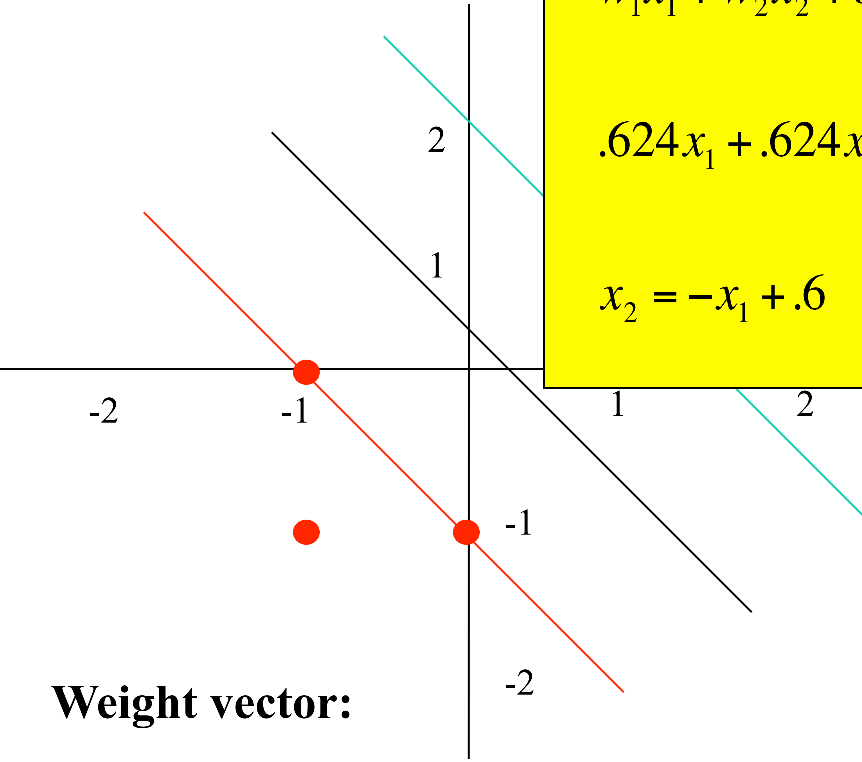
Example

Separation line:

$$w_1 x_1 + w_2 x_2 + b = 0$$

$$.624 x_1 + .624 x_2 - .376 = 0$$

$$x_2 = -x_1 + .6$$



Input to SVM optimzer:

x_1	x_2	$class$
1	1	1
1	2	1
2	1	1
-1	0	-1
0	-1	-1
-1	-1	-1

Output from SVM optimizer:

Support vector	α
$(-1, 0)$	$-.208$
$(1, 1)$	$.416$
$(0, -1)$	$-.208$

$b = -.376$

Weight vector:

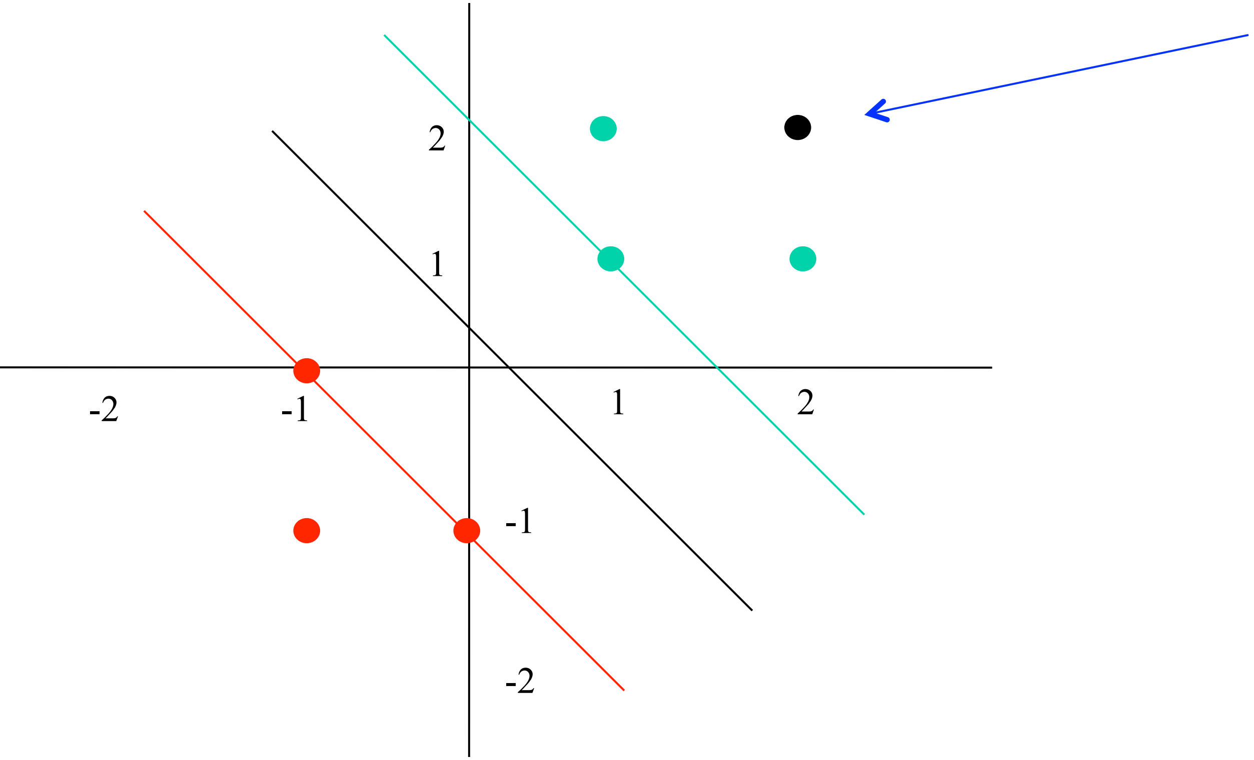
$$\mathbf{w} = \sum_{k \in \{\text{training examples}\}} \alpha_k \mathbf{x}_k$$

$$= -.208\,(-1,0) + .416\,(1,1) - .208\,(0,-1)$$

$$= (.624, .624)$$

Example

Classifying a new point:

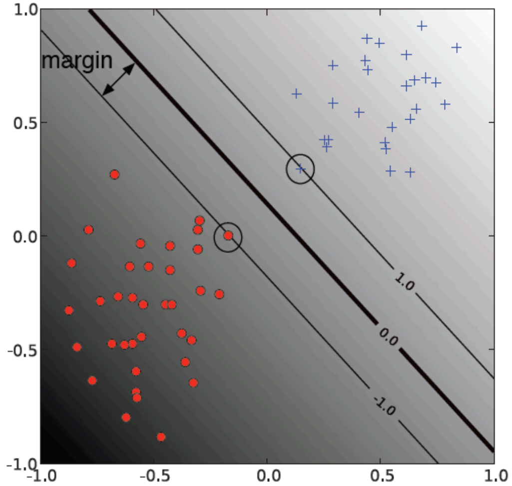


$$h((2,2)) = \operatorname{sgn}\left(\left(\sum_{k=1}^{m}\alpha_k\left(\mathbf{x}_k\cdot\mathbf{x}\right)\right)+b\right)$$

$$= \operatorname{sgn}\left(-.208\left[(-1,0)\cdot(2,2)\right]+.416\left[(1,1)\cdot(2,2)\right]-.208\left[(0,-1)\cdot(2,2)\right]-.376\right)$$

$$= \operatorname{sgn}\left(.416+1.664+.416-.376\right)=+1$$

SVM summary



- Equation of line: $w_1 x_1 + w_2 x_2 + b = 0$

- Define margin using:

$\mathbf{x}_k \cdot \mathbf{w} + b \geq +1$ for positive instances $(t_k = +1)$

$\mathbf{x}_k \cdot \mathbf{w} + b \leq -1$ for negative instances $(t_k = -1)$

- Margin distance: $\dfrac{1}{\|\mathbf{w}\|}$

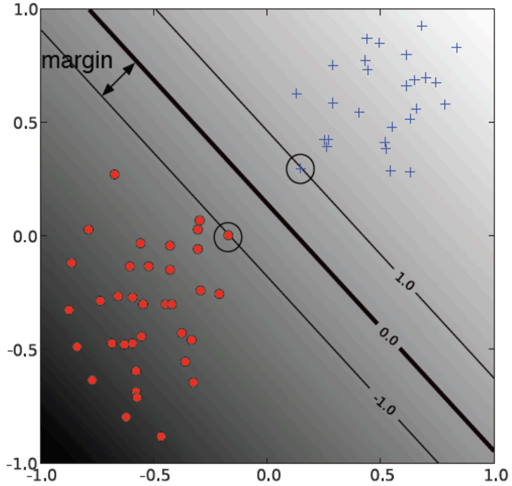
- To maximize the margin, we minimize $\|\mathbf{w}\|$ subject to the constraint that positive examples fall on one side of the margin, and negative examples on the other side:

$$t_k\left(\mathbf{w} \cdot \mathbf{x}_k + b\right) \geq 1, \quad k = 1, \ldots, m$$

where $t_k \in \{-1, +1\}$

- We can relax this constraint using "slack variables"

SVM summary

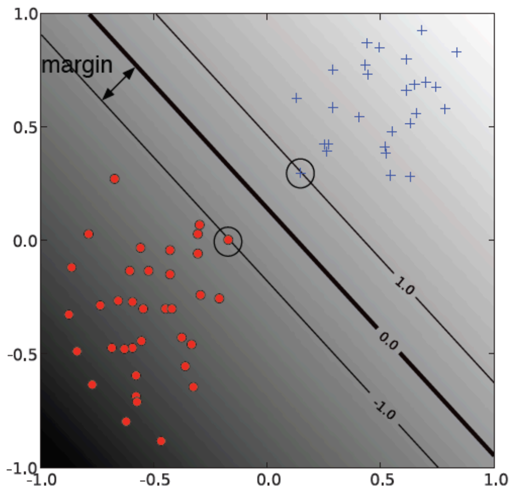


- To do the optimization, we use the **dual formulation:**

$$\mathbf{w} = \sum_{k \in \{\text{training examples}\}} \alpha_k \mathbf{x}_k$$

The results of the optimization "black box" are $\{\alpha_k\}$ and b .

SVM review



• Once the optimization is done, we can classify a new example **x** as follows:

$$h(\mathbf{x}) = \text{class}(\mathbf{x}) = \text{sgn}\left(\mathbf{w} \cdot \mathbf{x} + b\right)$$

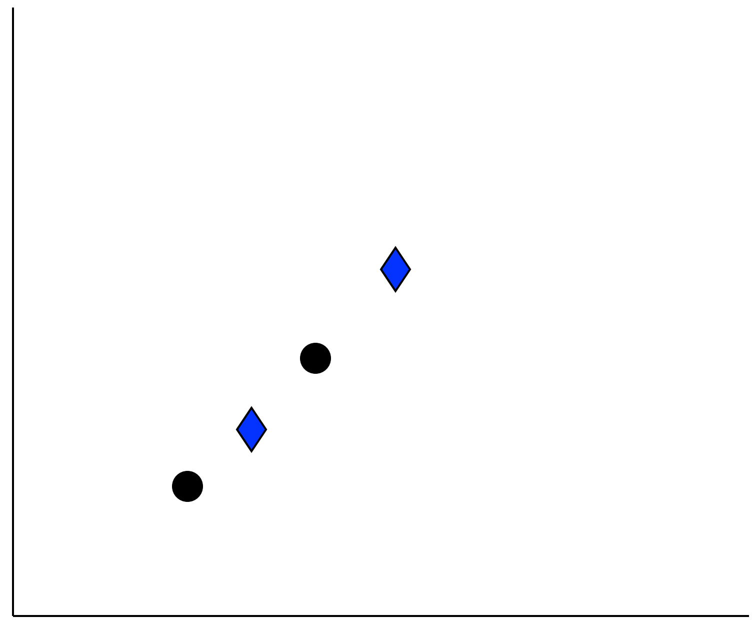
$$= \text{sgn}\left(\left(\sum_{k=1}^{m} \alpha_k \mathbf{x}_k\right) \cdot \mathbf{x} + b\right)$$

$$= \text{sgn}\left(\left(\sum_{k=1}^{m} \alpha_k \left(\mathbf{x}_k \cdot \mathbf{x}\right)\right) + b\right)$$

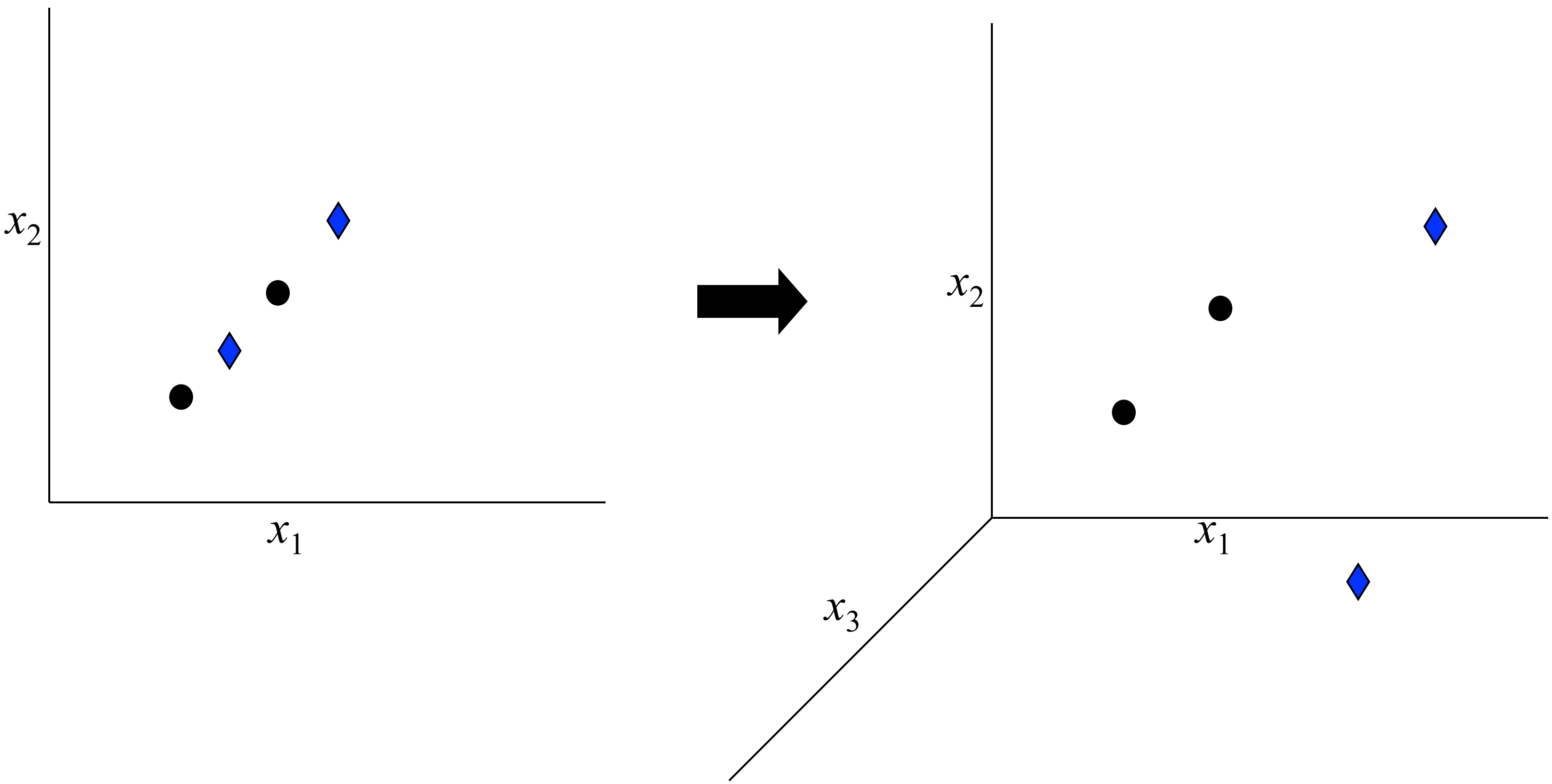
That is, classification is done entirely through a linear combination of dot products with training examples. This is a “kernel” method.

Non-linearly separable training examples

- What if the training examples are not linearly separable?



- Use old trick: Find a function that maps points to a higher dimensional space ("feature space") in which they are linearly separable, and do the classification in that higher-dimensional space.



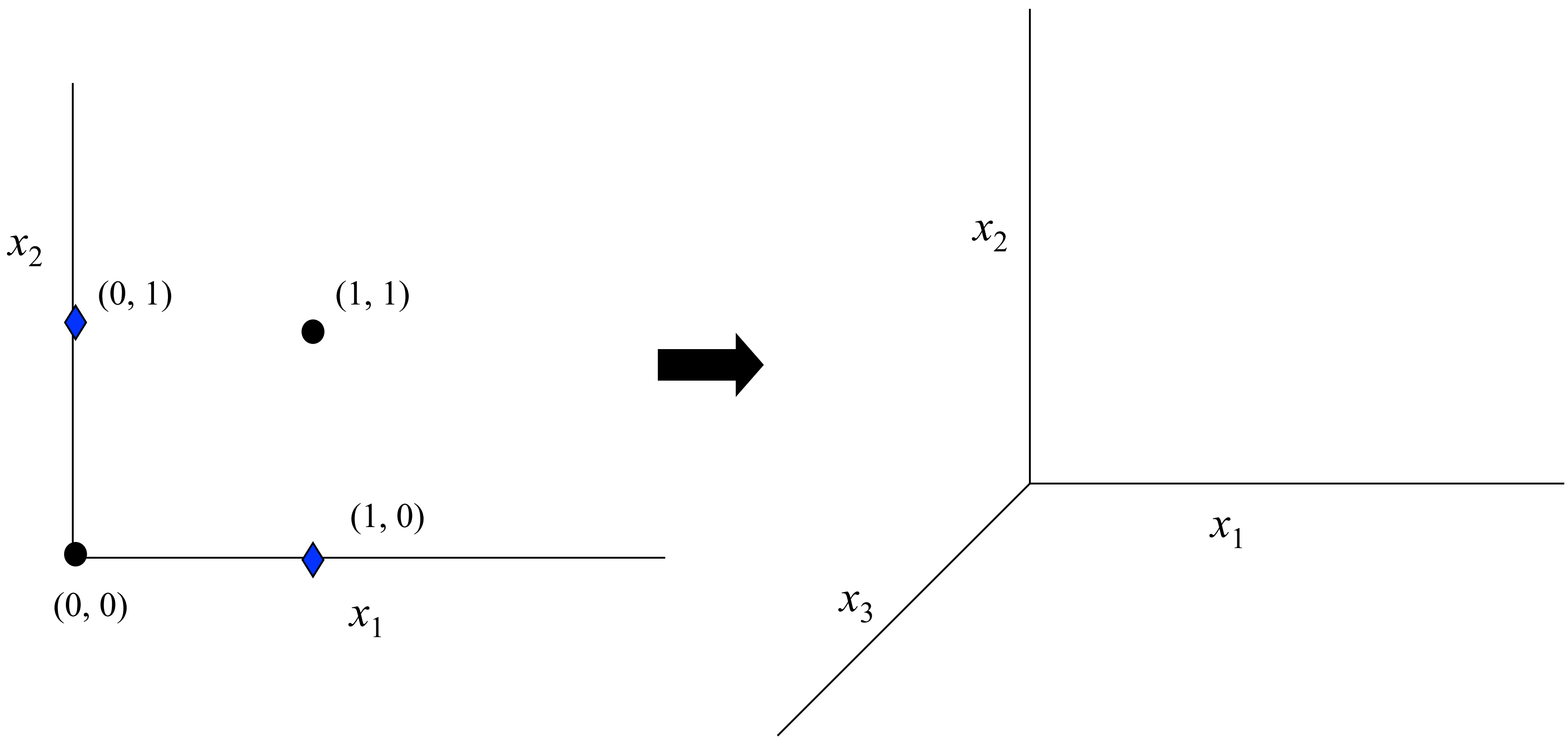
Need to find a function Φ that will perform such a mapping:

$$\Phi: \Re^n \rightarrow F$$

Then can find hyperplane in higher dimensional feature space F, and do classification using that hyperplane in higher dimensional space.

Challenge (work with your neighbor)

Find a 3-dimensional feature space in which XOR is linearly separable.



- **Problem**:

  – Recall that classification of instance **x** is expressed in terms of dot products of **x** and support vectors.

$$\text{Class}(\mathbf{x}) = \text{sgn}\left(\sum_{k \in \{\text{training examples}\}} \alpha_k (\mathbf{x} \cdot \mathbf{x}_k) + b \right)$$

  – The quadratic programming problem of finding the support vectors and coefficients also depends only on dot products between training examples, rather than on the training examples outside of dot products.

– So if each $\mathbf{x}_k$ is replaced by $\Phi(\mathbf{x}_k)$ in these procedures, we will have to calculate $\Phi(\mathbf{x}_k)$ for each k as well as calculate a lot of dot products, $\Phi(\mathbf{x})\cdot \Phi(\mathbf{x}_k)$

– But in general, if the feature space F is high dimensional, $\Phi(\mathbf{x}_\mathrm{i}) \cdot \Phi(\mathbf{x}_\mathrm{j})$ will be expensive to compute.

- **Second trick:**

  – Suppose that there were some magic function,

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

  such that K is cheap to compute even though $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ is expensive to compute.

  – Then we wouldn't need to compute the dot product directly; we'd just need to compute K during both the training and testing phases.

  – The good news is: such K functions exist! They are called "kernel functions", and come from the theory of integral operators.

Example: Polynomial kernel:

Suppose $\mathbf{x} = (x_1, x_2)$ and $\mathbf{z} = (z_1, z_2)$.

$$k(\mathbf{x},\mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$$

$$\text{Let } \Phi(\mathbf{x}) = \left(x_1^{\,2}, \sqrt{2} \cdot x_1 x_2, x_2^2\right).$$

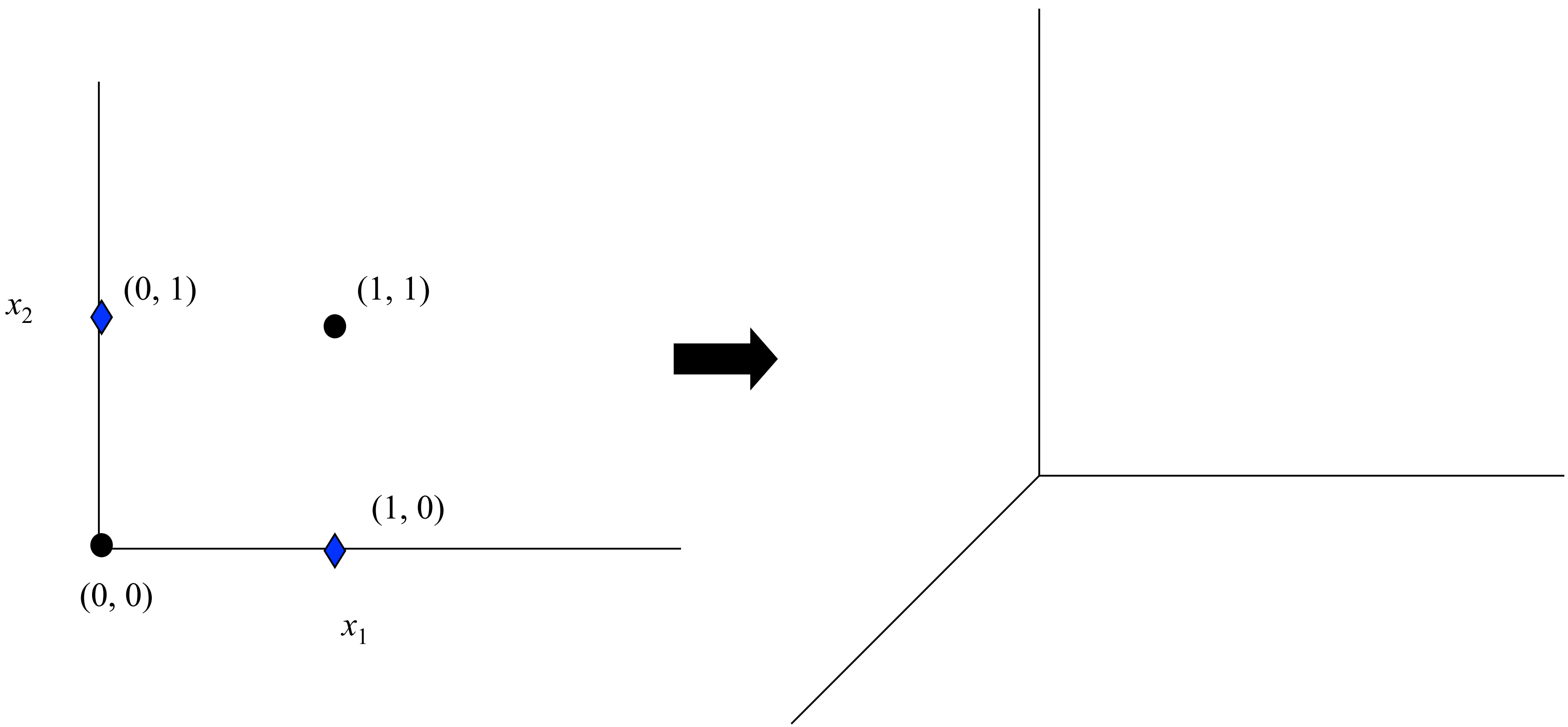
Then :

$$\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) = \left(\begin{pmatrix} x_1^2 \\ \sqrt{2} \cdot x_1 x_2 \\ x_2^2 \end{pmatrix} \cdot \begin{pmatrix} z_1^2 \\ \sqrt{2} \cdot z_1 z_2 \\ z_2^2 \end{pmatrix} \right)$$

$$= \left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \right)^2 = (\mathbf{x} \cdot \mathbf{z})^2 = k(\mathbf{x},\mathbf{z})$$

Exercise

Does the degree-2 polynomial kernel make XOR linearly separable?



Most commonly used kernels

- Linear

$$K(\mathbf{x}, \mathbf{x}_i) = \mathbf{x} \cdot \mathbf{x}_i$$

- Polynomial

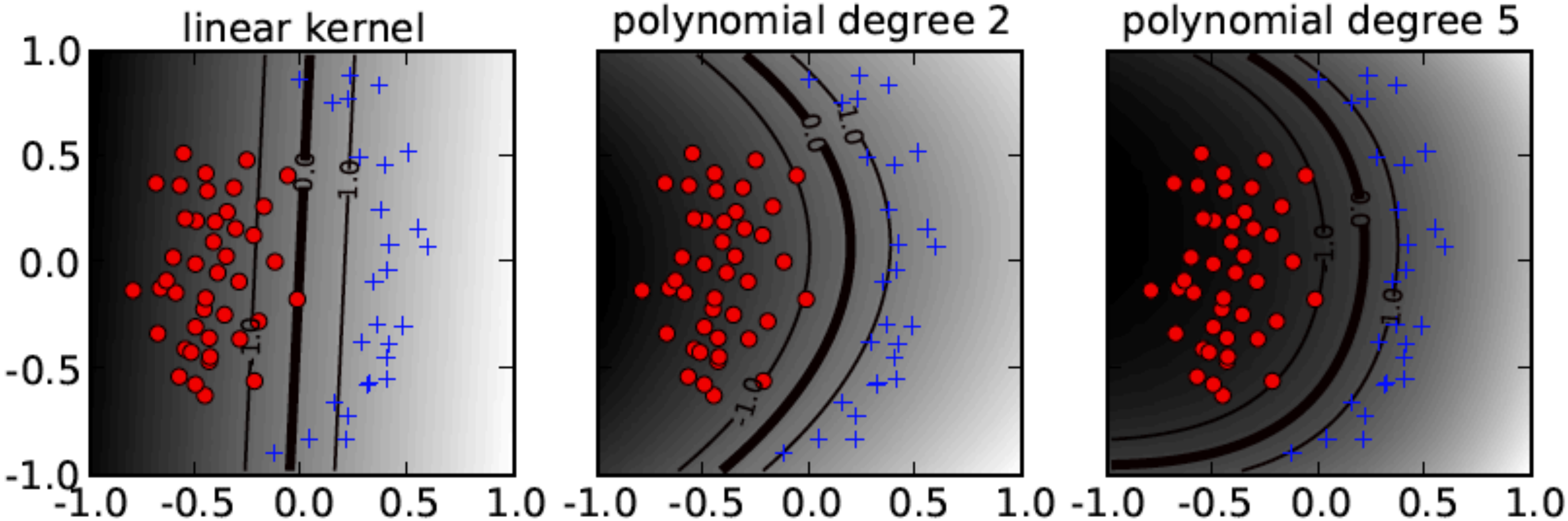
$$K(\mathbf{x}, \mathbf{x}_i) = [(\mathbf{x} \cdot \mathbf{x}_i) + 1]^d$$

- Gaussian (or "radial basis function")

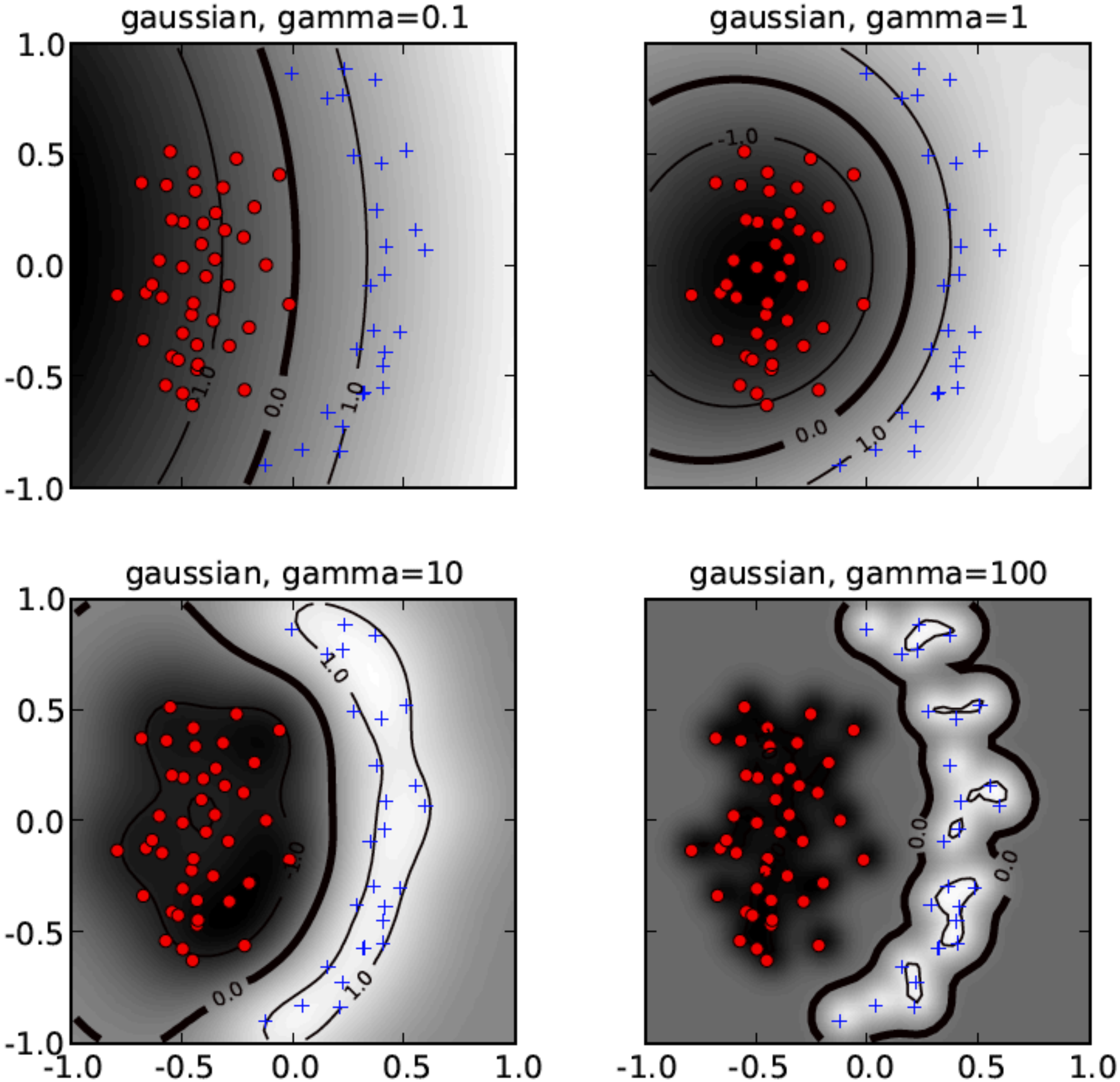
$$K(\mathbf{x}, \mathbf{x}_i) = e^{-\gamma|\mathbf{x}-\mathbf{x}_i|^2}$$

- Sigmoid

$$K(\mathbf{x}, \mathbf{x}_i) = \tanh(a\mathbf{x} \cdot \mathbf{x}_i + b)$$



linear kernel
polynomial degree 2
polynomial degree 5
1.0
0.5
0.0
-0.5
-1.0
-1.0 -0.5 0.0 0.5 1.0
-1.0 -0.5 0.0 0.5 1.0
-1.0 -0.5 0.0 0.5 1.0


gaussian, gamma=0.1
gaussian, gamma=1
gaussian, gamma=10
gaussian, gamma=100
1.0
0.5
0.0
-0.5
-1.0
-1.0
-0.5
0.0
0.5
1.0
-1.0
0.0
1.0

More on Kernels

- So far we've seen kernels that map instances in $\Re^n$ to instances in $\Re^z$ where $z > n$.

- One way to create a kernel: Figure out appropriate feature space $\Phi(\mathbf{x})$, and find kernel function k which defines inner product on that space.

- More practically, we usually don't know appropriate feature space $\Phi(\mathbf{x})$.

- What people do in practice is either:
  1. Use one of the "classic" kernels (e.g., polynomial),

  or

  2. Define their own function that is appropriate for their task, and show that it qualifies as a kernel.

How to define your own kernel

- Given training data ($\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$)

- Algorithm for SVM learning uses *kernel matrix* (also called *Gram matrix*):

$$\mathbf{K}_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j), \text{ for } i, j = 1, \ldots, n$$

- We can choose some function *K,* and compute the kernel matrix **K** using the training data.

- We just have to guarantee that our kernel defines an inner product on some feature space.

- Not as hard as it sounds.

What counts as a kernel?

- Mercer's Theorem: If the kernel matrix **K** is "symmetric positive semidefinite", it defines a kernel, that is, it defines an inner product in some feature space.
- We don't even have to know what that feature space is! It can have a huge number of dimensions.

Structural Kernels

- In domains such as natural language processing and bioinformatics, the similarities we want to capture are often structural (e.g., parse trees, formal grammars).

- An important area of kernel methods is defining *structural kernels* that capture this similarity (e.g., sequence alignment kernels, tree kernels, graph kernels, etc.)

From www.cs.pitt.edu/~tomas/cs3750/kernels.ppt:

- Design criteria - we want kernels to be
  - **valid** – Satisfy Mercer condition of positive semidefiniteness
  - **good** – embody the “true similarity” between objects
  - **appropriate** – generalize well
  - **efficient** – the computation of $K(\mathbf{x}, \mathbf{x}')$ is feasible

Example:
Watson used tree kernels and SVMs to classify question types for Jeopardy! questions

From Moschitti et al., 2011

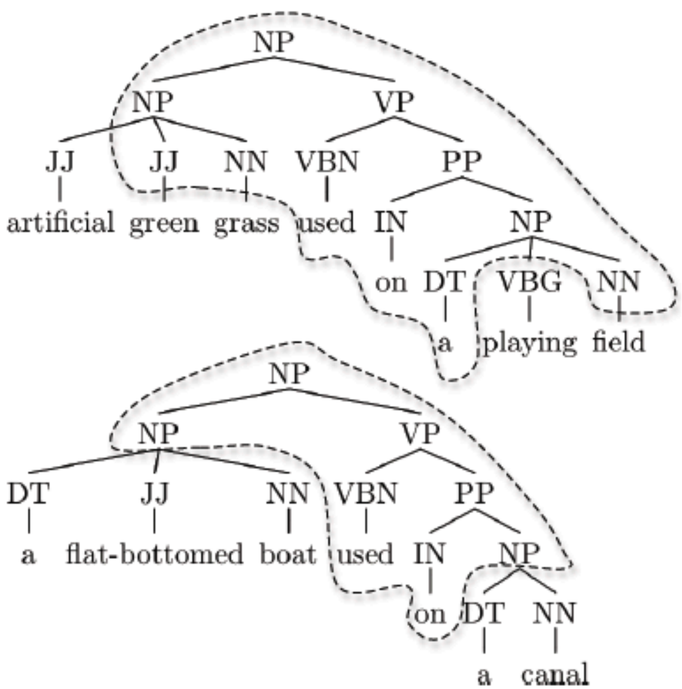


Figure 5: Similarity according to PTK and STK

Kernel Space	Prec.	Rec.	F1
WSK+CSK	70.00	57.19	62.95
PTK-CT+CSK	69.43	60.13	64.45
PTK-CT+WSK+CSK	68.59	62.09	**65.18**
CSK+RBC	47.80	74.51	58.23
PTK-CT+CSK+RBC	59.33	74.84	65.79
BOW+CSK+RBC	60.65	73.53	66.47
PTK-CT+WSK+CSK+RBC	67.66	66.99	**67.32**
PTK-CT+PASS+CSK+RBC	62.46	71.24	66.56
WSK+CSK+RBC	69.26	66.99	**68.11**
ALL	61.42	67.65	64.38

Table 2: Performance of Kernel Combinations using leave-one-out cross-validation.