Support Vector Machines
Part 2: Nonlinear Case

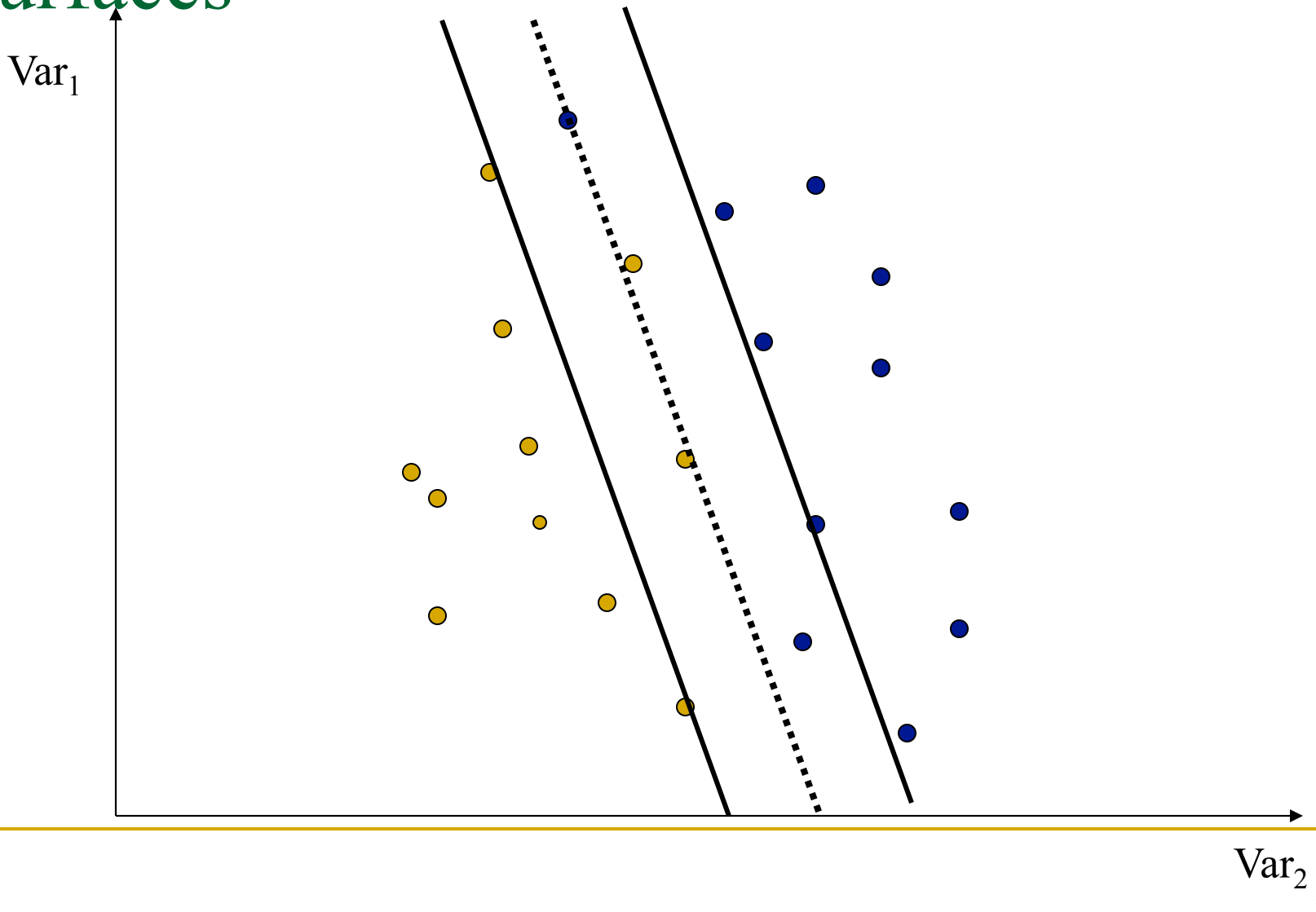
A portion (1/3) of
the slides are taken from
Prof. Andrew Moore's
SVM tutorial at
http://www.cs.cmu.edu/~awm/tutorials
And
Mingyue Tan of UBC

Rao Vemuri
rvemuri@gmail.com

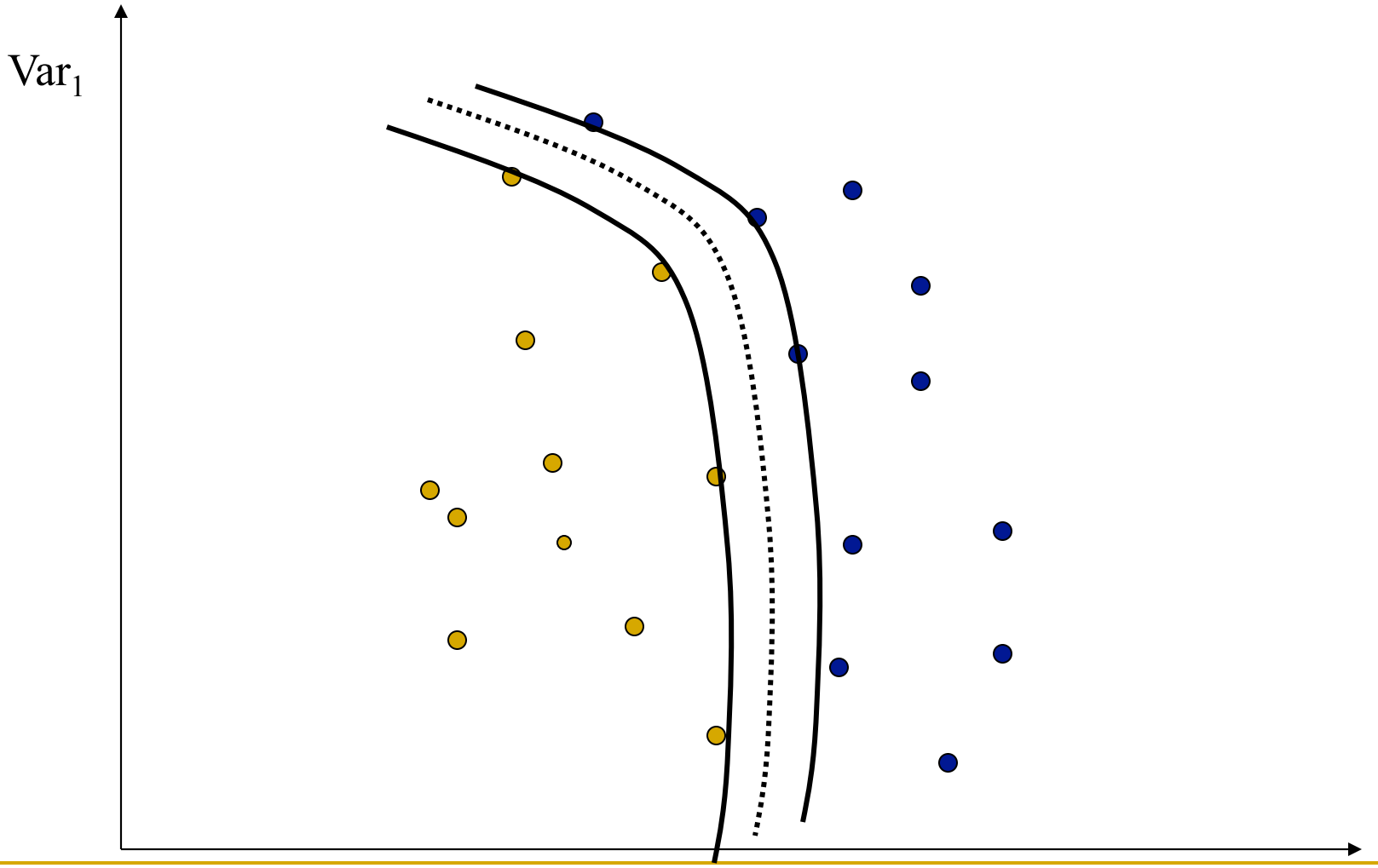
Overview: Part 2

- Mod 7: Nonlinear SVMs
- Mod 8. Applications
  - Gene Data Classification/Text Categorization

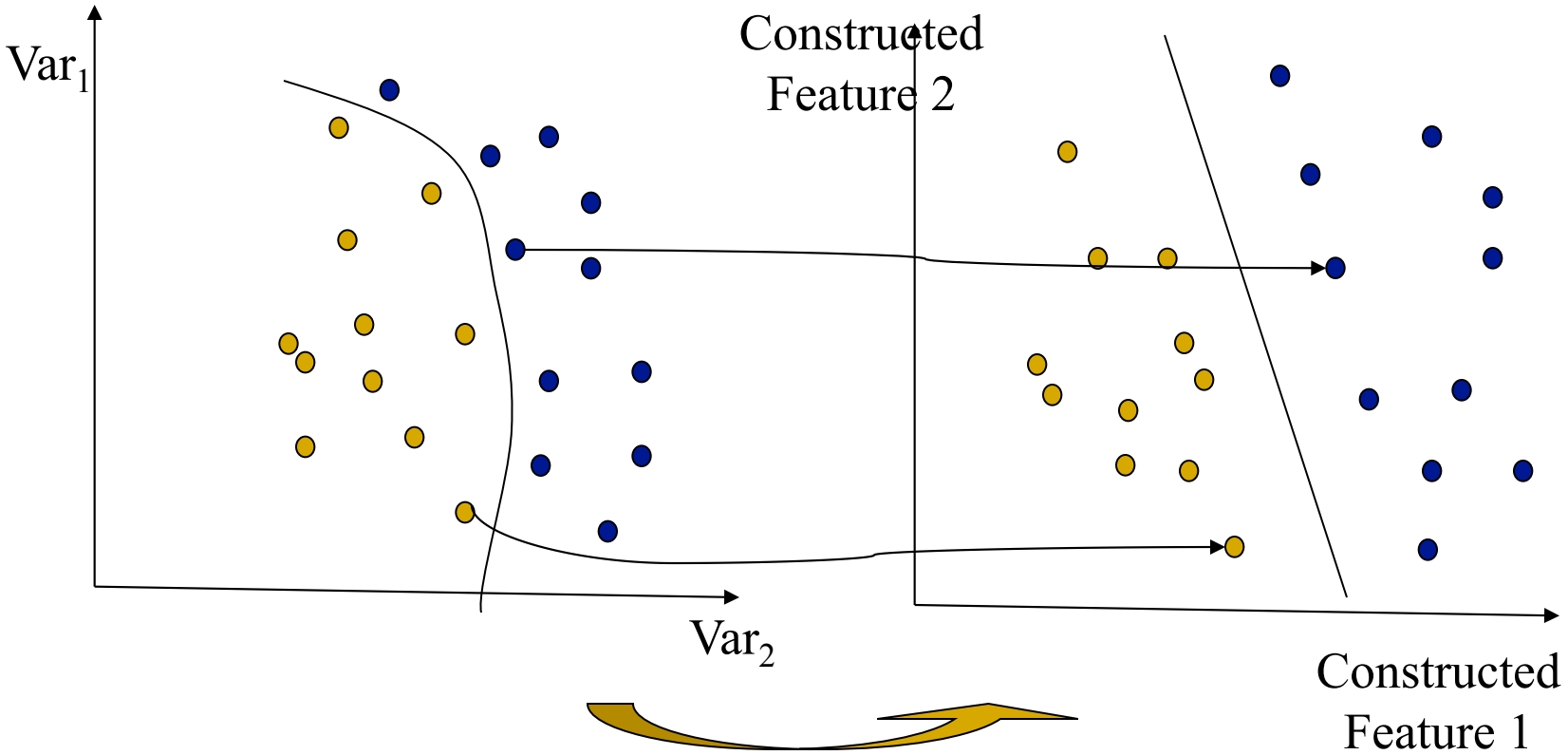
Disadvantages of Linear Decision Surfaces



Advantages of Non-Linear Surfaces



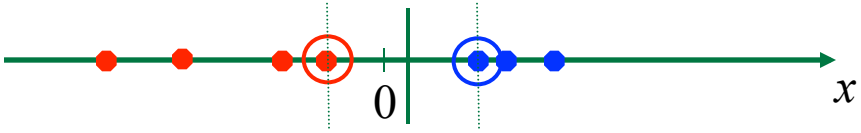
Linear Classifiers in High-Dimensional Spaces



Find function Φ(x) to map to a different space

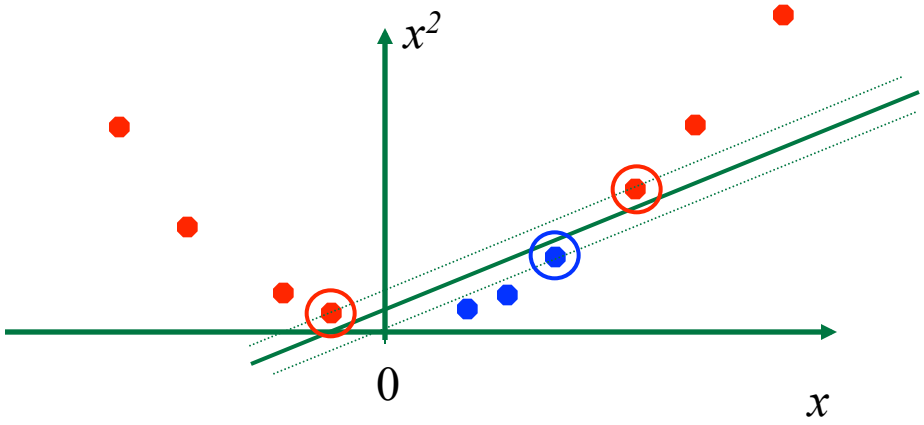
Non-linear SVMs

- 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:



Feature Expansion

- The last figure can be thought of also as a nonlinear basis function in two dimensions
- That is, we used the basis $z = (x, x^2)$
- That is, we expanded the number of features from x to (x, x^2)
- Fitted a support-vector classifier to the expanded space
- This resulted in a non-linear decision boundary in the original space

Example: Quadratic Polynomial

- Suppose we use: $(x_1, x_2, x_1^2, x_2^2, x_1 x_2)$
  instead of just: (x_1, x_2)
- Then the decision boundary would be of the form:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2 = 0$$

- This also leads to a nonlinear decision boundaries in the original space

Example: Cubic Polynomial

- We could have used a cubic for this type of basis expansion.
- Then that cubic would be

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2 +$$
$$\beta_6 x_1^3 + \beta_7 x_2^3 + \beta_8 x_1 x_2^2 + \beta_9 x_1^2 x_2 = 0$$

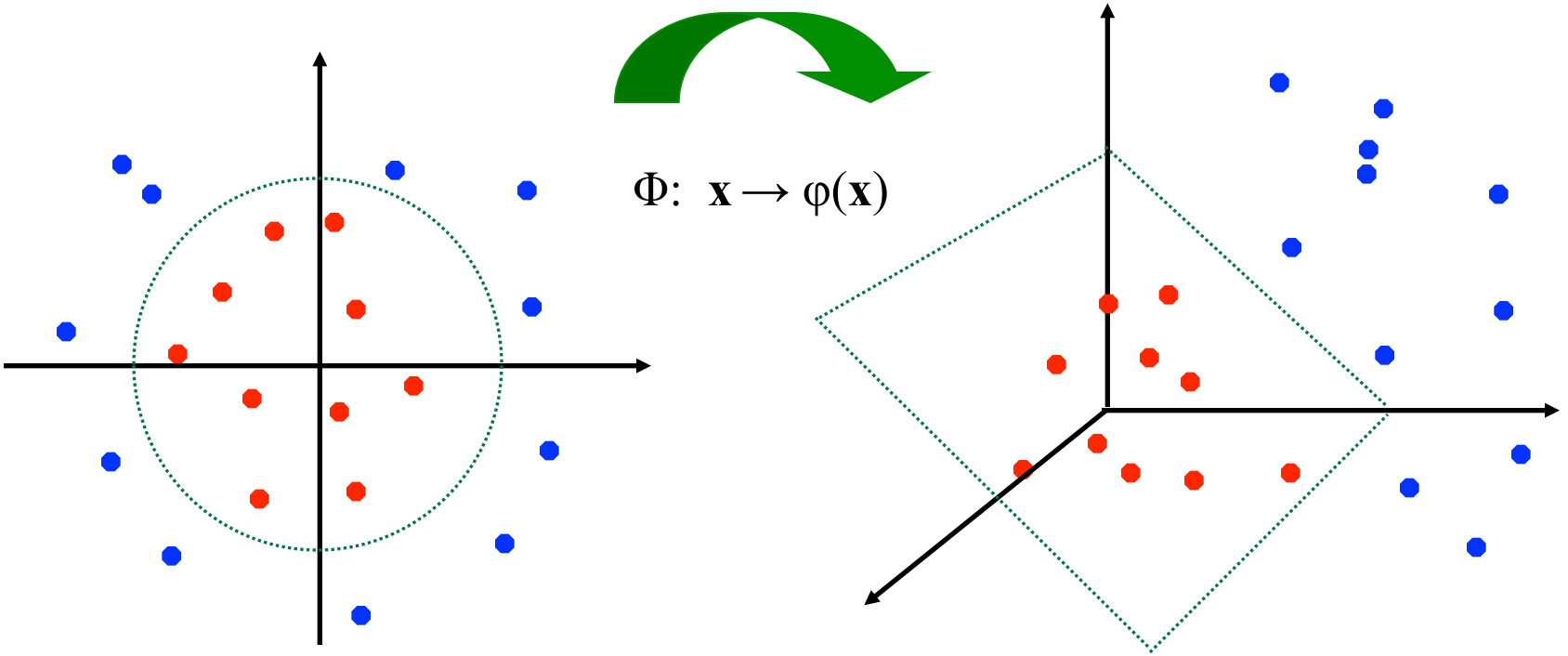
- This would expand from 2-dimensional space to 9-dimensional space

Polynomials Vs Kernels

- Polynomials (especially high-dimensional ones) get wild rather fast.
- There is a more elegant and controlled way to introduce nonlinearities in support-vector classifiers — through the use of kernels.
- Before we discuss these, let us look at a general mapping Φ from (x_1,x_2) to a high 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:



What is the Mapping Function?

- The idea is to achieve linear separation by mapping the data into a higher dimensional space
- Let us call Φ the function that achieves this mapping.
- What is this Φ?

Let us recall the formula we used earlier - Linear SVM lecture:
The dual formulation

Dual Formula – 1 (Linear SVM)

$$\sum_{k=1}^{R}\alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R}\alpha_k\alpha_l Q_{kl}, \quad Q_{kl} = y_k y_l (x_k . x_l)$$

$$subject\ to\ the\ constra\,\mathrm{int}\,s$$

$$0 \le \alpha_k \le C \quad for\ \forall k \quad \sum_{k=1}^{R}\alpha_k y_k = 0$$

- Notice the dot product

Dual in New (Feature) Space

$$\max \sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl} \quad where \; Q_{kl} = y_k y_l (\Phi(x_k).\Phi(x_l))$$

$$s.t. \quad 0 \le \alpha_k \le C, \forall k \quad and \quad \sum_{k=1}^{R} \alpha_k y_k = 0$$

Now define

$$w = \sum_{k \; s.t. \; \alpha_k > 0} \alpha_k y_k \Phi(x_k); \quad b = y_K (1 - \xi_K) - x_K . w_K$$

$$where \quad K = \text{argmax}_k (\alpha_k) K = \arg\max_k (\alpha_k)$$

$$Classify \; with \quad f(x, w, b) = sign(w.\Phi(x) + b)$$

Computational Burden

- Because we're working in a higher-dimension space (and potentially even an infinite-dimensional space), calculating $\varphi(x_i)^T \varphi(x_j)$ may be intractable.
- We must do $R^2/2$ dot products to evaluate Q
- Each dot product requires $m^2/2$ additions and multiplications
- The whole thing costs $R^2m^2/4$.
- Too high!!! Or, does it? Really?

What is Φ

- Any way, We do not know what phi? We still do not know what it is

The Kernel Trick

- Let us replace the *inner product* $(x_i \cdot x_j)$ by

$\Phi(x_i).\ \Phi(x_j)$ to define the operations in the new higher dimensional space

- If there is a "kernel function" K such that

  $K(x_i, x_j) = \Phi(x_i).\ \Phi(x_j) = \Phi(x_i)^T\Phi(x_j)$

  then we do not need to know Φ explicitly.

- This strategy is preferred because the alternative of working with Φ is expensive, computationally.

Example Kernel

- $K(x_i, x_j) = (1 + x_i \cdot x_j)^2 = \varphi(x_i) \cdot \varphi(x_j)$
  for the given φ. If this is true, we can simplify our calculations.
- Re-writing the dual in terms of the kernel,

$$\max_{\alpha \geq 0} \left[\sum \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \right]$$

Decision Function

- To classify a novel instance x once you've learned the optimal αi parameters, all you have to do is calculate

$$f(x) = sign(w.x + b) = sign(\sum \alpha_i y_i K(x_i, x) + b)$$

(by setting $w = \sum \alpha_i y_i \phi(x_i)$ and using the kernel trick).

A Note

- Note that α_i is only non-zero for instances $\varphi(x_i)$ on or near the boundary - those are called the support vectors since they alone specify the decision boundary. We can toss out the other data points once training is complete. Thus, we only sum over the x_i which constitute the support vectors.

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 scalar products between pairs of datapoints (in the high-dimensional feature space).
- These scalar 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 scalar 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 scalar products.

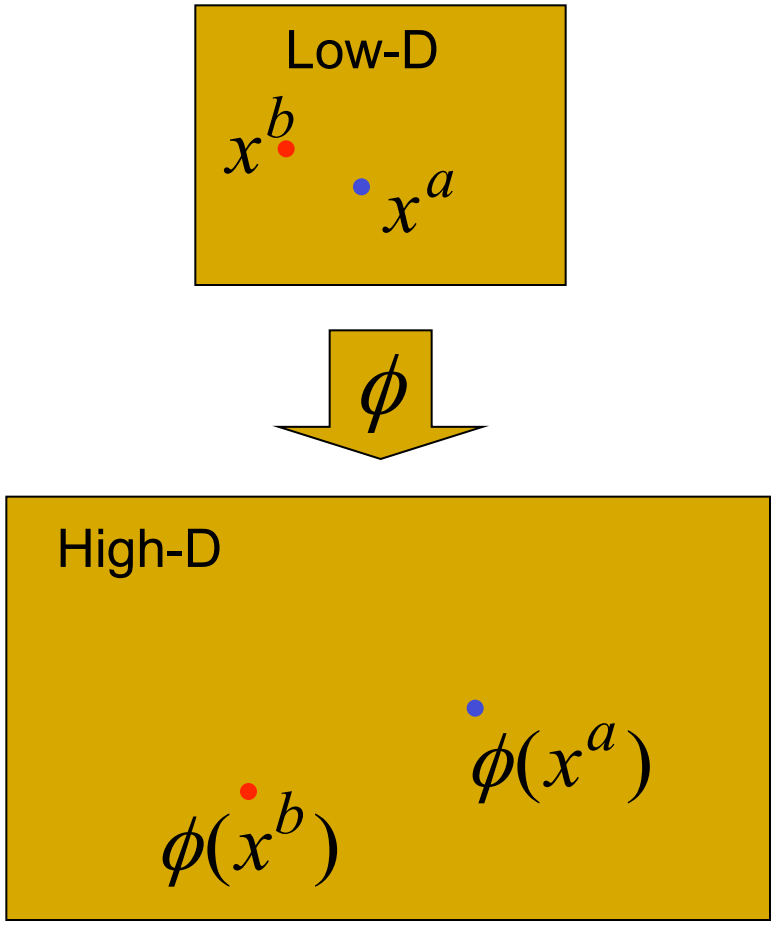
The kernel trick

- For many mappings from a low-D space to a high-D space, there is a simple operation on two vectors in the low-D space that can be used to compute the scalar product of their two images in the high-D space.

$$K(x^a, x^b) = \phi(x^a).\phi(x^b)$$

Letting the kernel do the work

doing the scalar product in the obvious way



Consider a Φ. Φ as shown below

$$\Phi(a).\Phi(b) = \begin{bmatrix} 1 \\ \sqrt{2}a_1 \\ \vdots \\ \sqrt{2}a_m \\ a_1^2 \\ a_2^2 \\ \vdots \\ a_m^2 \\ \sqrt{2}a_1a_2 \\ \sqrt{2}a_1a_3 \\ \\ \sqrt{2}a_1a_m \\ \sqrt{2}a_2a_3 \\ \\ \sqrt{2}a_2a_m \\ \\ \sqrt{2}a_{m-1}a_m \\ \\ \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \sqrt{2}b_1 \\ \vdots \\ \sqrt{2}b_m \\ b_1^2 \\ b_2^2 \\ \vdots \\ b_m^2 \\ \sqrt{2}b_1b_2 \\ \sqrt{2}b_1b_3 \\ \\ \sqrt{2}b_1b_m \\ \sqrt{2}b_2b_3 \\ \\ \sqrt{2}b_2b_m \\ \\ \sqrt{2}b_{m-1}b_m \\ \\ \end{bmatrix}$$

Collecting terms in the dot product

- First term = 1 +
- Next m terms = $\sum_{i=1}^{m} 2a_i b_i$
- Next m terms = $\sum_{i=1}^{m} a_i^2 b_i^2$
- Rest = $\sum_{i=1}^{m} \sum_{j=i+1}^{m} 2a_i a_j b_i b_j$

- Therefore

$$\Phi(a).\Phi(b) = 1 + 2\sum_{i=1}^{m} a_i b_i + \sum_{i=1}^{m} a_i^2 b_i^2 + \sum_{i=1}^{m} \sum_{j=i+1}^{m} 2a_i a_j b_i b_j$$

Out of Curiosity

$$(1+a.b)^2 = (a.b)^2 + 2(a.b) + 1$$

$$= \left(\sum_{i=1}^{m} a_i b_i\right)^2 + 2\left(\sum_{i=1}^{m} a_i b_i\right) + 1$$

$$= \sum_{i=1}^{m}\sum_{j=1}^{m} a_i b_i a_j b_j + 2\left(\sum_{i=1}^{m} a_i b_i\right) + 1$$

$$= \sum_{i=1}^{m} (a_i b_i)^2 + 2\sum_{i=1}^{m}\sum_{j=i+1}^{m} a_i b_i a_j b_j + 2\left(\sum_{i=1}^{m} a_i b_i\right) + 1$$

Both are Same

- Comparing term by term, we see
- $\Phi.\Phi = (1 + a.b)^2$
- But computing the right side is lot more efficient, O(m) (m additions and multiplications)
- Let us call $(1 + a.b)^2 = K(a,b) = Kernel$

Φ in "Kernel Trick" Example

2-dimensional vectors $\mathbf{x} = [x_1 \; x_2]$;

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

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

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

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

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

$$[1 \;\; x_{j1}^2 \;\; \sqrt{2}\, x_{j1}x_{j2} \;\; x_{j2}^2 \;\; \sqrt{2}x_{j1} \;\; \sqrt{2}x_{j2}]$$

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

where $\varphi(\mathbf{x}) = [1 \;\; x_1^2 \;\; \sqrt{2}\, x_1x_2 \;\; x_2^2 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2]$

Other Kernels

- Beyond polynomials there are other high dimensional basis functions that can be made practical by finding the right kernel function

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):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

- Sigmoid: $K(\mathbf{x}_i,\mathbf{x}_j)= \tanh(\beta_0\mathbf{x}_i^T\mathbf{x}_j + \beta_1)$

SVM Software

- Available at www.kernel-machines.org
- LibSVM (C++)
- SVMLight (C)
- Machine learning software that includes SVM
- Torch (C++)
- Spider (Matlab)
- Weka (Java)
- Comparison at http://www.cs.ubc.ca/~murphyk/Software/svm.htm