R·I·T

Support Vector Machines

About the Name...

A Support Vector

A training sample used to define classification boundaries in SVMs

- located near class boundaries

Support Vector Machines

Binary classifiers whose decision boundaries are defined by support vectors

SVMs: Design Principles

Discriminative

Similar to perceptrons, SVMs define linear decision boundaries for two classes directly

- vs. Generative approaches, where decision boundaries defined by estimated posterior probabilities (e.g. LDC, QDC, k-NN)
- Perceptron: decision boundary sensitive to initial weights, choice of η (learning rate), order in which training samples are processed

Maximizing the Margin

Unlike perceptrons, SVMs produce a *unique* boundary between linearly separable classes: the one that maximizes the margin (distance to the decision boundary) for each class

- Often leads to better generalization

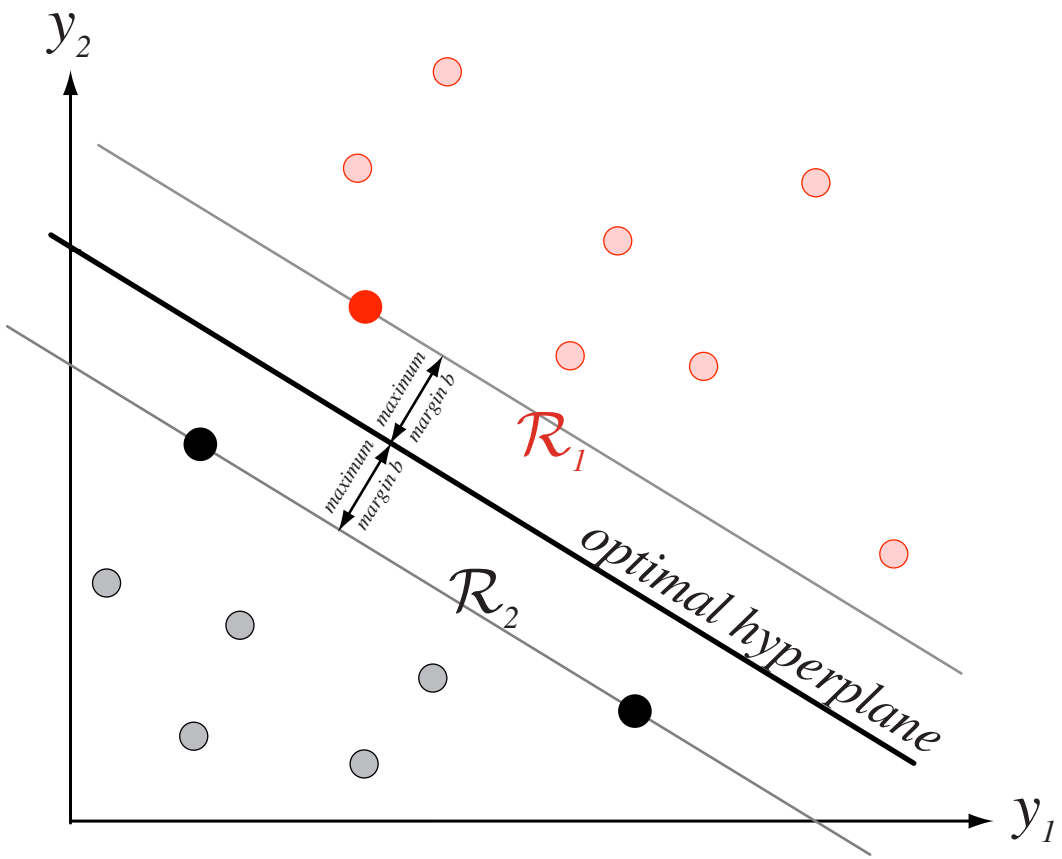


FIGURE 5.19. Training a support vector machine consists of finding the optimal hyperplane, that is, the one with the maximum distance from the nearest training patterns. The support vectors are those (nearest) patterns, a distance *b* from the hyperplane. The three support vectors are shown as solid dots. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons,

More on 'The Margin'

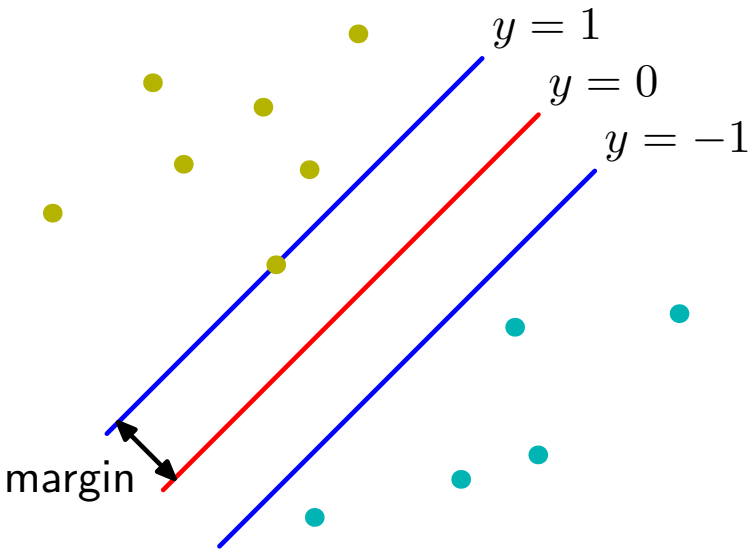
The Margin, *b*

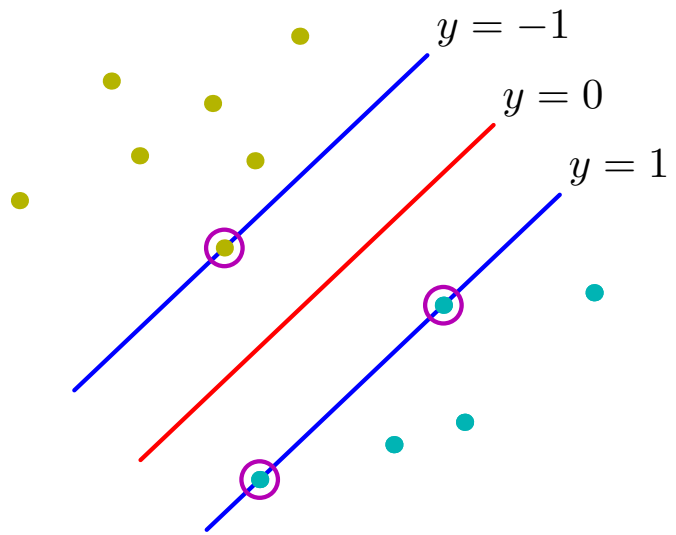
Is the minimum distance of *any* sample to the decision boundary.

Training SVMs

= maximizing the margin, moving the decision boundary as far away from all training samples as possible.

Maximizing the Margin





- At left: a sub-optimal margin
- At right: optimal margin
- y values: for linear function defined by the SVM. For linearly separable data, all training instances correctly classified as -1 or 1 (locations in the margins have y values in (-1,1))

*Bishop, "Pattern Recognition and Machine Learning," p. 327

Binary Classification by SVM

SVMs Define Linear Decision Boundaries

Recall: so do perceptrons, LDCs for two classes, etc.

Classify By the Sign (+/-) of:

N_s: # support vectors x_i (with $\lambda_i > 0$)

$$g(x) = w^T x + w_0$$

$$= \sum_{i=1}^{N_s} \lambda_i \; y_i \; x_i^T x + w_0$$

*Here, y_i refers to class (-1 or 1) for instance x_i

where N_s is the number of support vectors, y_i the class of support vector xi (+1 or -1), and λ_i is a weight (Lagrange multiplier) associated with x_i.

Training/Learning for SVMs

Optimization Problem:

SV signed dot products are constant.

$$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x_i^T x_j} \right)$$

$$subject\ to \sum_{i=1}^{N} \lambda_i y_i = 0$$

$$\forall \mathbf{x_i}, \lambda_i \geq 0$$

Note:

Given a training set, two parameters of g(x) need to be learned/defined: λ_i and w_0

Once λ_i have been obtained, the optimal hyperplane and w_0 may be determined.

R·I·T

But Where are the SVs?

The Support Vectors

...are those with non-zero weights in the learned system, which will lie near decision boundaries between classes.

Data points other than the support vectors can now be discarded, as they do not affect classification.

$$g(x) = w^T x + w_0$$
$$= \sum_{i=1}^{N_s} \lambda_i \, y_i \, x_i^T x + w_0$$

...and the bias w_0?

(classification rule)

$$y_x \left(\sum_{i=1}^{N_S} \lambda_i y_i x_i^T x + \boxed{w_0} \right) = 1$$

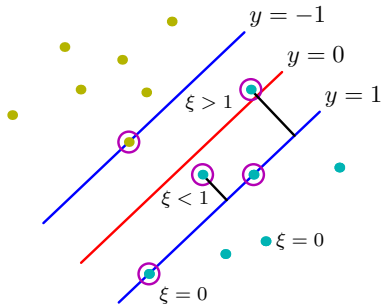
Identity Above Holds for All SVs

But solving for w_0 using a single SV less numerically stable than averaging over all support vectors, as shown below.

$$w_0 = \frac{1}{N_S} \sum_{n \in S} \left(y_n - \sum_{m \in S} \lambda_m y_m x_n^T x_m \right)$$

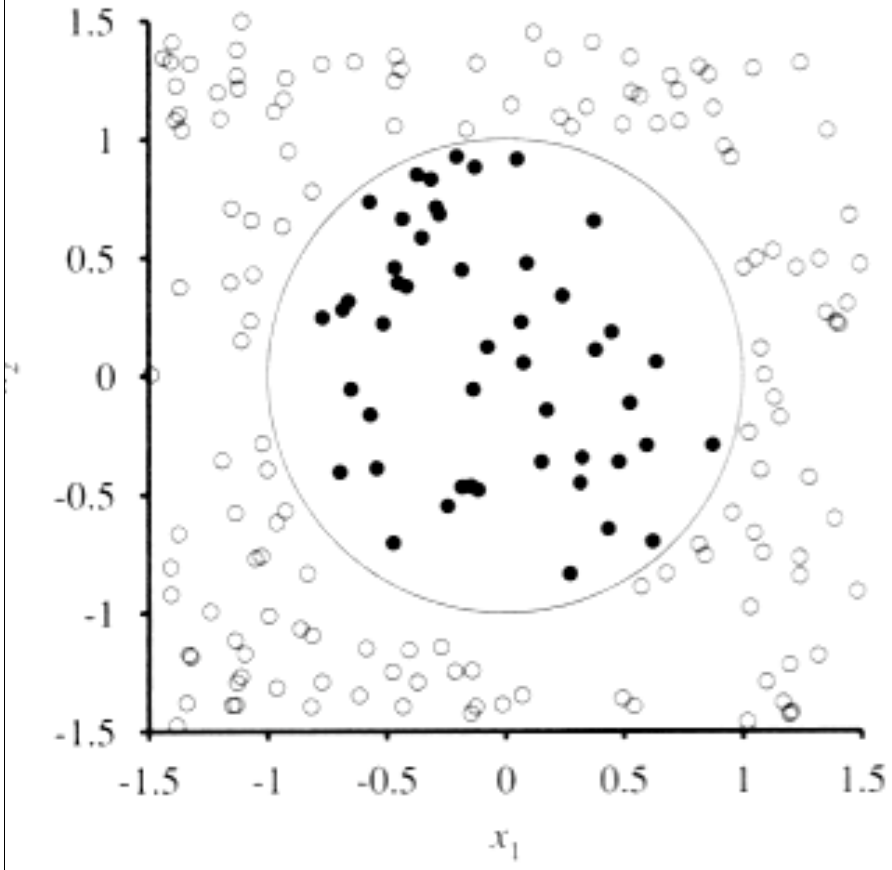
Non-Linearly Separable Classes

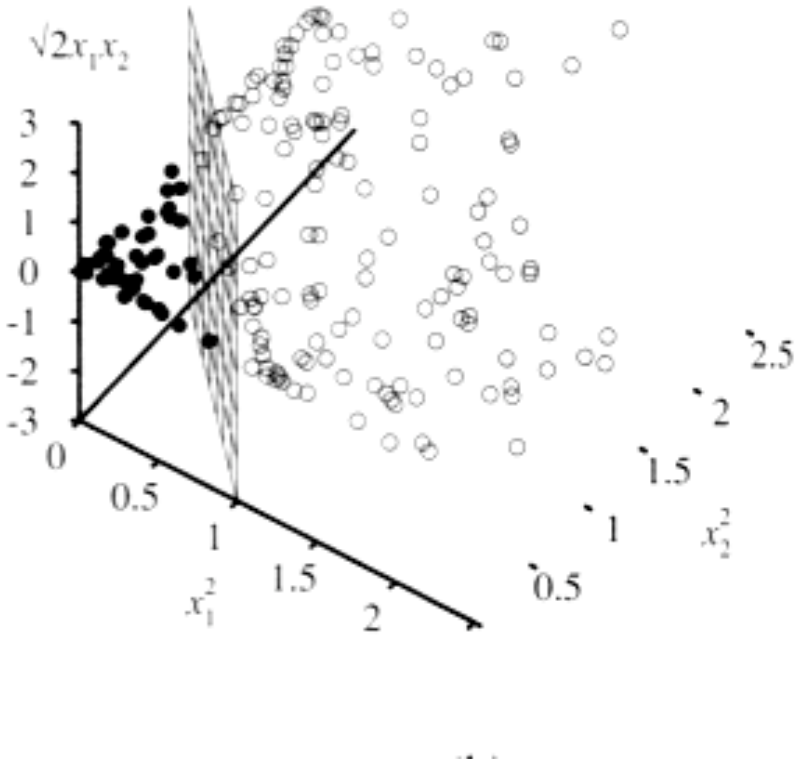
May be handled by using a *soft margin*, in which points may lie, and classification errors may occur (e.g. margin properties defined by tunable parameters for v-SVM).



Often handled by transforming a non-linearly separable feature space into a higher-dimensional one in which classes are linearly separable (the "kernel trick"), and then use a 'plain' SVM for classification.

Example: Restructuring Feature Space
(from Russell & Norvig, 2nd Edition)





Here, mapping is defined by:

$f_1 = x_1^2 \quad f_2 = x_2^2 \quad f_3 = \text{sqrt}(2)x_1x_2$

The "Kernel Trick"

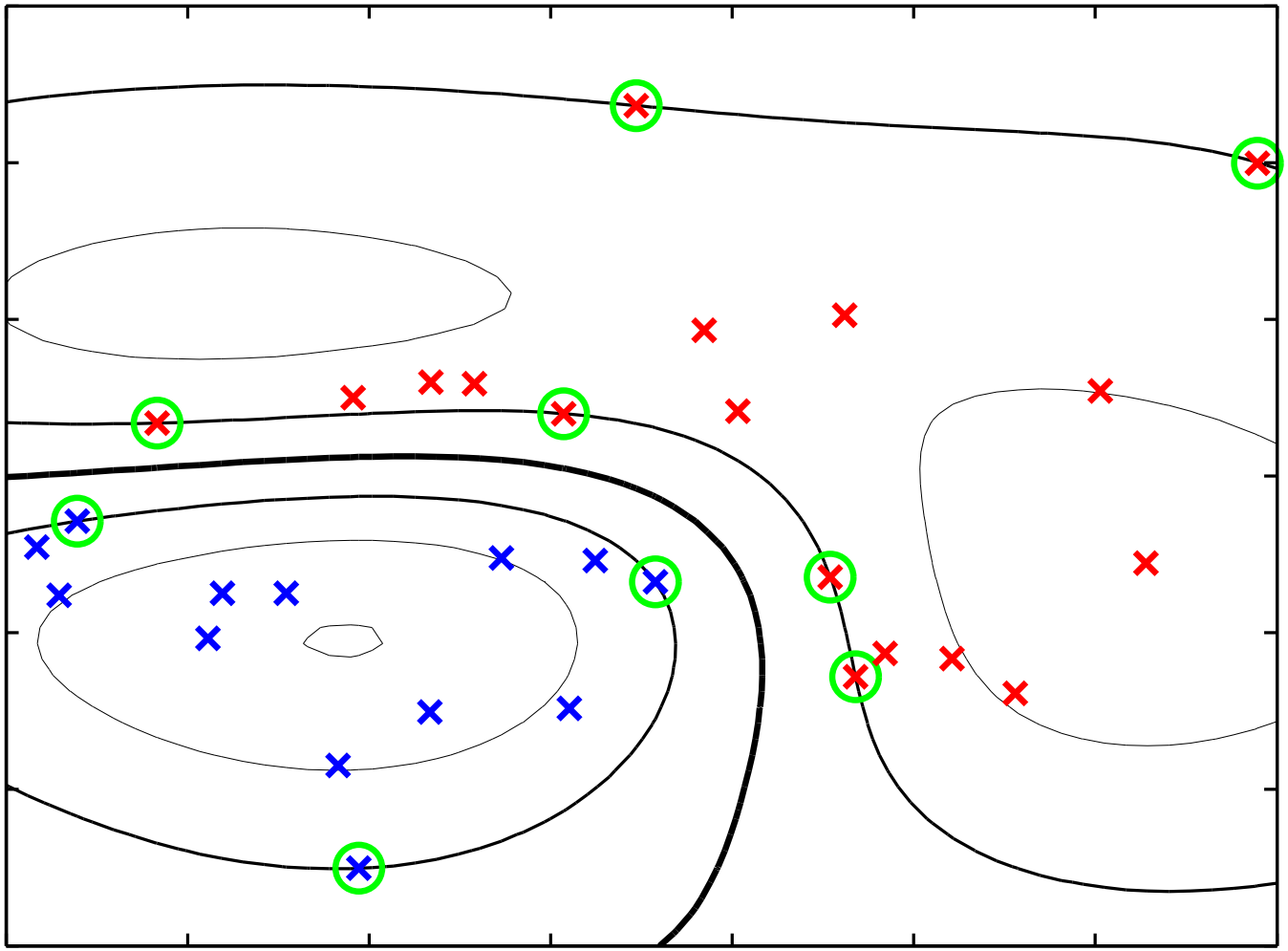
$$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x_i^T x_j} \right)$$

- The expression for optimization above does not depend on the dimensions of the feature vectors, only their inner ('dot') product.
- We can substitute a kernelized version of the inner product (*k*) for the inner product of feature vectors, where *k* uses a non-linear feature mapping phi: $k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$
- After training, we classify according to:

$$g(x) = \sum_{i=1}^{N_s} \lambda_i \; y_i \; k(x_i, x) + w_0$$

Example: Gaussian Kernel

$$k(x, x') = \exp(-||\mathbf{x} - \mathbf{x}'||^2/2\sigma^2)$$



R·I·T

Some Common Kernel Functions

Polynomial (d is degree of polynomial)

$$k(x, x') = (x^T x')^d$$

Gaussian

$$k(x, x') = \exp(-||\mathbf{x} - \mathbf{x}'||^2 / 2\sigma^2)$$

Additional Examples: See Bishop Ch. 6

Handling Multiple Classes

One vs. All

Create one binary classifier per class (~ discriminants $g_i(x)$)

- Most widely used: C (# class) SVMs needed; can select max class

One vs. One

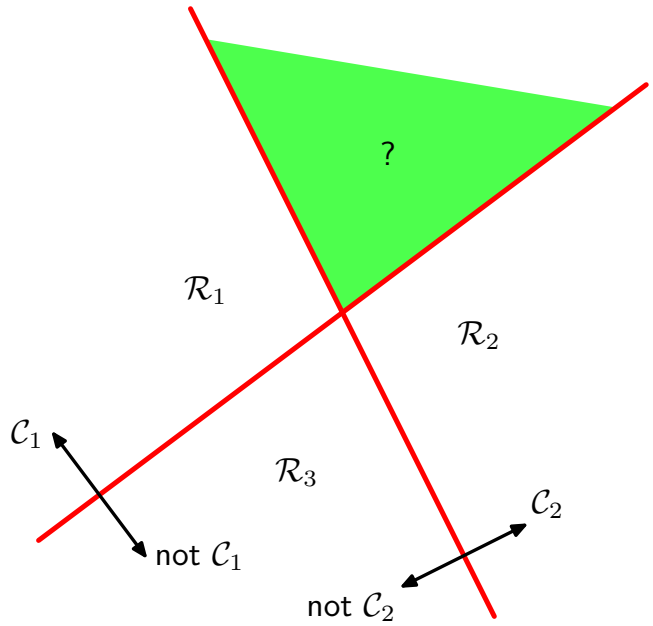
Create one binary classifier for every pair of classes: choose class with highest number of 'votes'

- Variation: use error-correcting output codes (bit strings representing class outcomes), use hamming distance to closest training instances to choose class
- Expensive! (C(C-1)/2 SVMs needed)

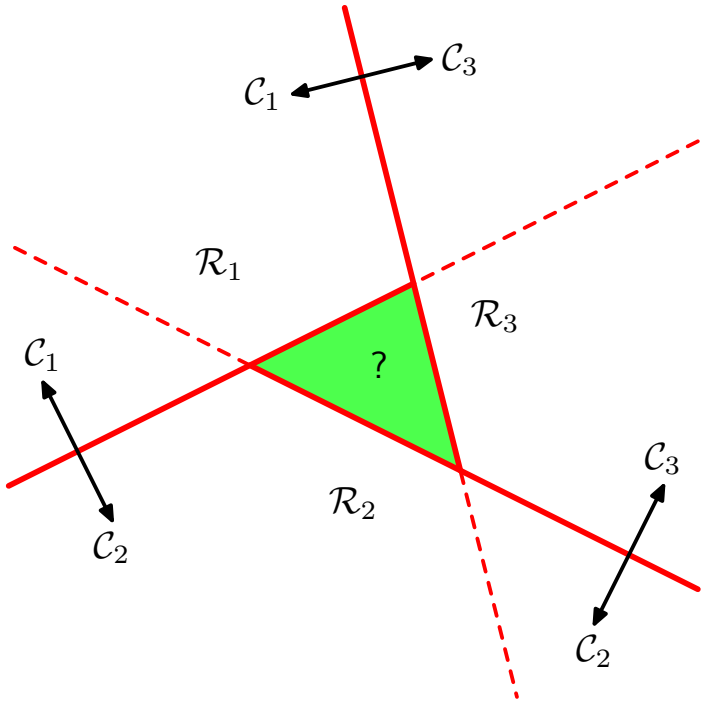
DAGSVM

Organize pair-wise classifiers into a DAG, reduce comparisons

Ambiguous Regions for Combinations of Binary Classifiers



'one vs. all'

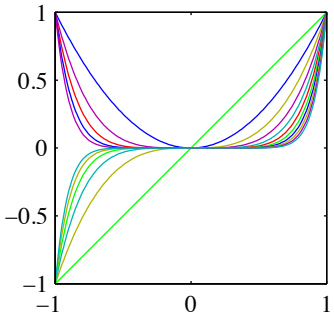


'one vs. one'

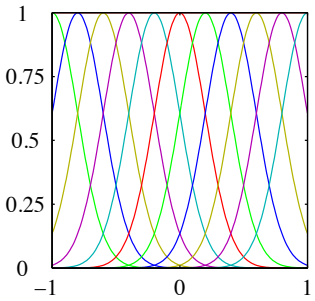
R·I·T *Taken from Bishop, "Pattern Recognition and Machine Learning", p. 183

Kernel Construction: Feature Map Sets

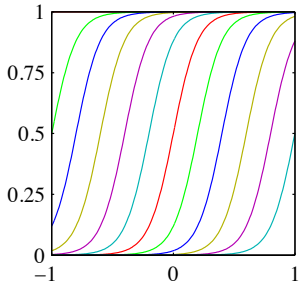
$$k(x, x') = \phi(x)^T \phi(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x')$$ (*M* 1-D mappings)



Polynomials

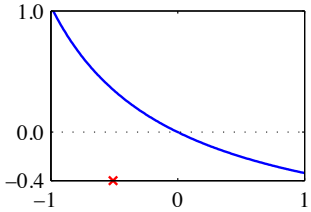


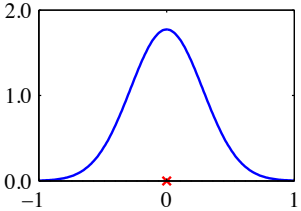
'Gaussians'

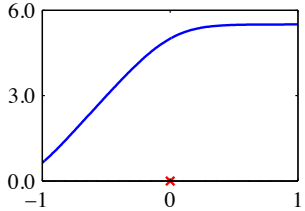


Logistic Sigmoids

(from Bishop, p. 295)







Kernel construction from sets of basis functions used for feature space mappings; red x indicates location of x'

Composing Kernel Functions

$k(x, z) = (x^T z)^2$ *(quadratic (polynomial) kernel)*

Interpretation in Mapped Feature Space:

Note: feature space same as 'circle' example from earlier slide.

$$\begin{aligned} k(x,z) &= (x^T z)^2 = (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2) \\ &= \phi(x)^T \phi(z). \end{aligned}$$

More on Polynomial Kernels

$$k(x, x') = (x^T x' + c)^M, c > 0$$

Polynomial Kernel, containing all terms up to degree M

e.g. for M=2, will add linear terms to the feature mapping.

$$k(x, x')^M$$

(F. mapping contains terms of degree M)
Example: if x, x' are images, produces weighted sum of all possible products of M

Valid Kernel Constructions

(see p. 296 of Bishop text)

‘Gaussian’ Kernel

$$k(x, x') = \exp(-||\mathbf{x} - \mathbf{x}'||^2/2\sigma^2)$$

Not a Probability Distribution

Normalization factor (value of mean) omitted

Demonstration the Kernel is Valid

Through rules for taking exp of a valid kernel; and validity of f(x)k(x,x’)f(x’) for valid kernel k.

$$||x - x'||^2 = x^T x + (x')^T x' - 2x^T x'$$

$$k(x, x') = \exp(-x^t x/2\sigma^2) \exp(x^T x'/\sigma^2) \exp(-(x')^T x'/2\sigma^2)$$

Example of Mapping to an Infinite Feature Space

Kernels for Symbolic Objs.

'Symbolic Objects'

Sets, strings, graphs, text documents

Valid Kernel for Sets Using Intersection

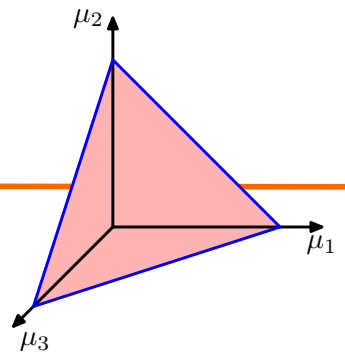
Feature space: set of all subsets of a set A.
Kernel *k* below can be shown to correspond to an inner product in a feature space.

$$A_1, A_2 \in 2^A \quad (power\ set), \qquad k(A_1, A_2) = 2^{|A_1 \cap A_2|}$$

R·I·T

Additional Slides

Linear Programming



Simplex

A bounded linear manifold; generalization of a triangle/tetrahedron in higher dimensions.

Linear Programs

Defined by an objective function and set of linear constraints (variables have scalar coefficients). Note restriction that weights are non-negative; solution region lies within a simplex

Simplex Algorithm

An iterative algorithmic transformation of the 'slack' form of a linear program using a method similar to Gaussian Elimination, but for inequalities.

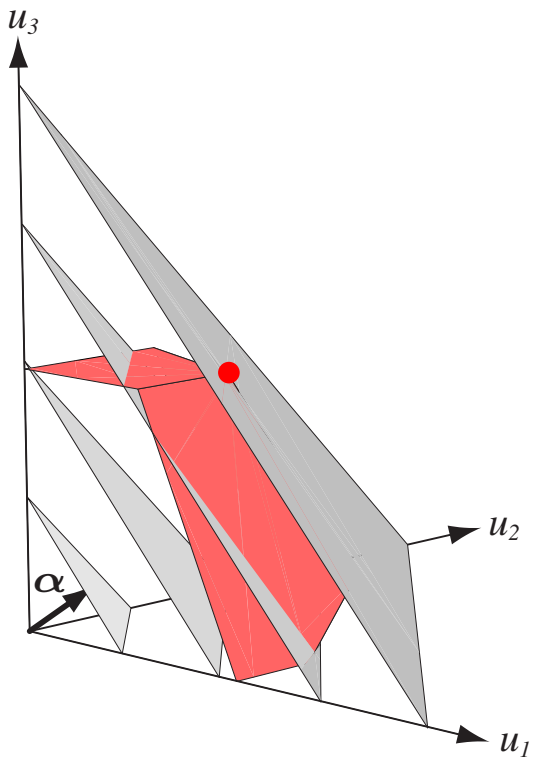


FIGURE 5.18. Surfaces of constant $z = \boldsymbol{\alpha}^t\mathbf{u}$ are shown in gray, while constraints of the form $\mathbf{Au}\boldsymbol{\beta}$ are shown in red. The simplex algorithm finds an extremum of z given the constraints, that is, where the gray plane intersects the red at a single point. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Linear Programming: Cont'd

Suggested Reading

The well-known Cormen et al. algorithms introduces Linear Programming and the Simplex algorithm in Ch. 29 of the 2nd edition:

- Text available to RIT students free online here: http://library.books24x7.com.ezproxy.rit.edu/toc.asp?site=K7ECY&bookid=3444