Machine Learning

Lecture 14
Support Vector Machines

Outline

- A brief history of SVM
- Large-margin linear classifier
  - Linear separable
  - Nonlinear separable
- Creating nonlinear classifiers: kernels
- A simple example
- Discussion on SVM
- Conclusion

History of SVM

- SVM is related to statistical learning theory [3]
- SVM was first introduced in 1992 [1]
- SVM becomes popular because of its success in handwritten digit recognition
  - 1.1% test error rate for SVM. This is the same as the error rates of a carefully constructed neural network, LeNet 4.
    - See Section 5.11 in [2] or the discussion in [3] for details
- SVM is now regarded as an important example of "kernel methods", one of the key area in machine learning

[1] B.E. Boser *et al.* A Training Algorithm for Optimal Margin Classifiers. Proceedings of the Fifth Annual Workshop on Computational Learning Theory 5 144-152, Pittsburgh, 1992.

[2] L. Bottou *et al.* Comparison of classifier methods: a case study in handwritten digit recognition. Proceedings of the 12th IAPR International Conference on Pattern Recognition, vol. 2, pp. 77-82.

[3] V. Vapnik. The Nature of Statistical Learning Theory. 2nd edition, Springer, 1999.

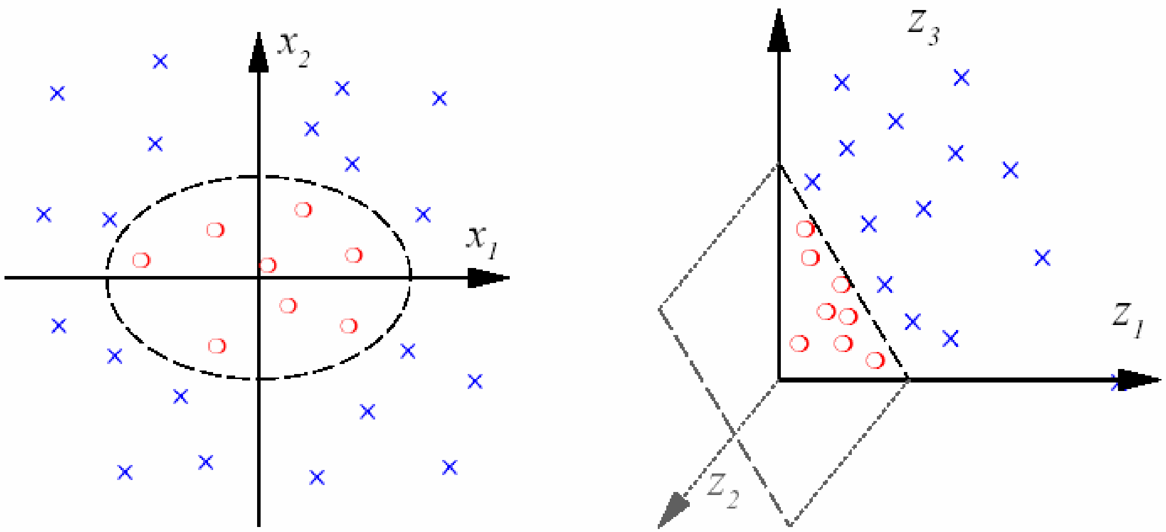
History of SVM (Cont.)

- SVMs introduced in COLT-92 by Boser, Guyon, Vapnik. Greatly developed ever since.
- Initially popularized in the NIPS community, now an important and active field of all Machine Learning research.
- Special issues of Machine Learning Journal, and Journal of Machine Learning Research.
- Kernel Machines: large class of learning algorithms, SVMs a particular instance.

High Dimension Mapping (1)

- Motivation: Linear inseparable problem becomes linear separable in higher dimension space.

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

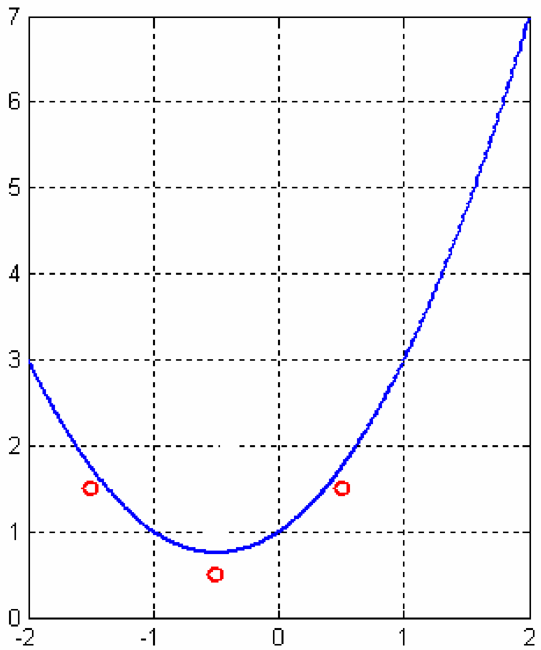


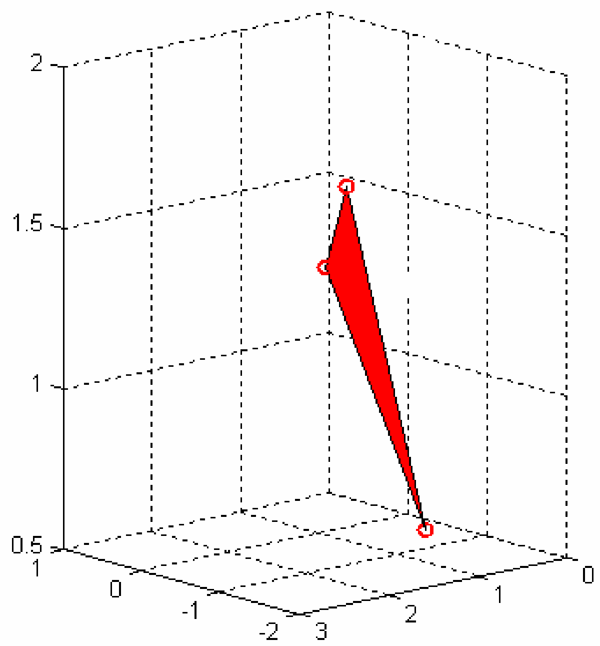
B. Schölkopf, NIPS, 3 December 2001

[2]

High Dimension Mapping (2)

- 2D Blue curve: y=x2+x+1 transfers to 3D Red plane where z'=y, y'=x2 , and x'=x.





Preliminaries

- Task of this class of algorithms: detect and exploit complex patterns in data (e.g.: by clustering, classifying, ranking, cleaning, etc. the data)
- Typical problems: how to represent complex patterns; and how to exclude spurious (unstable) patterns (= overfitting)
- The first is a computational problem; the
- second a statistical problem.

Very Informal Reasoning

- The class of kernel methods implicitly defines the class of possible patterns by introducing a notion of similarity between data
- Example: similarity between documents
  - By length
  - By topic
  - By language ...
- Choice of similarity -> Choice of relevant features

More formal reasoning

- Kernel methods exploit information about the inner products between data items
- Many standard algorithms can be rewritten so that they only require inner products between data (inputs)
- Kernel functions = inner products in some feature space (potentially very complex)
- If kernel given, no need to specify what features of the data are being used

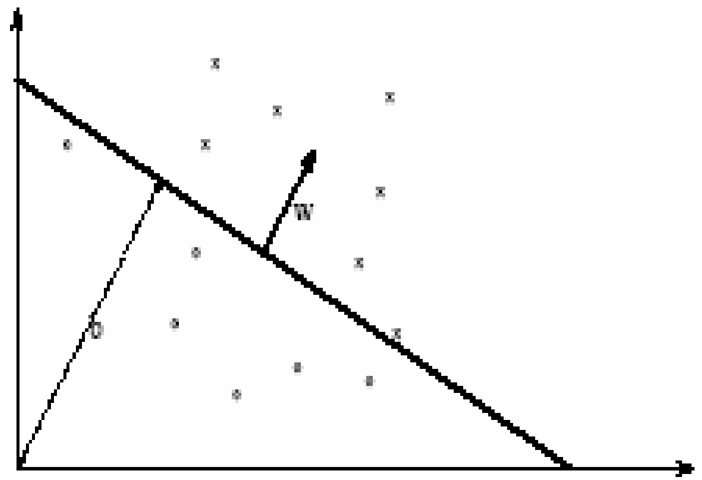
Some definitions

- Inner product between vectors

$$\langle \bar{x}, \bar{z} \rangle = \sum_i x_i z_i$$

- Hyperplane:

$$\langle w, x \rangle + b = 0$$

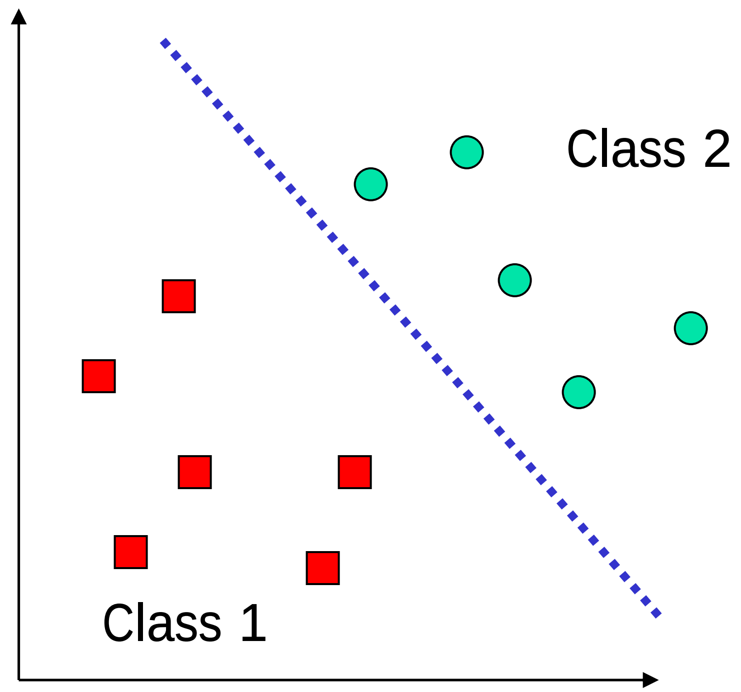


Modularity

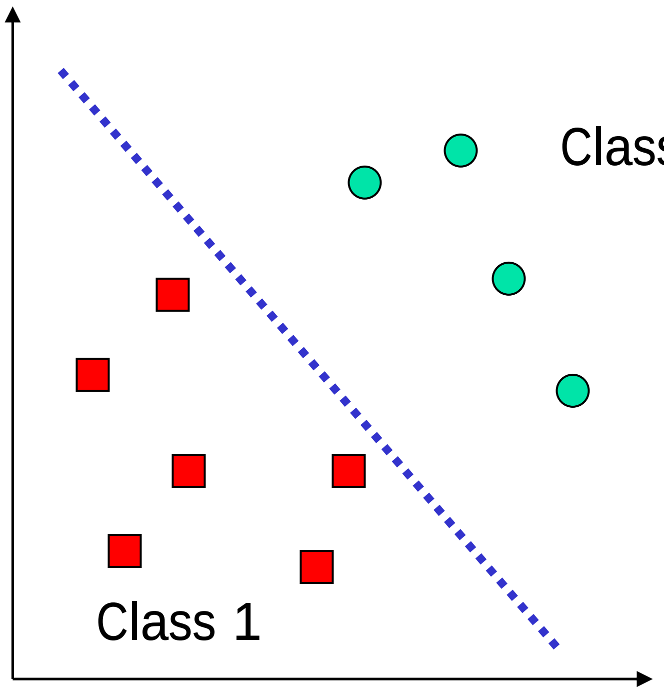
- Any kernel-based learning algorithm composed of two modules:
  – A general purpose learning machine
  – A problem specific kernel function
- Any K-B algorithm can be fitted with any kernel
- Kernels themselves can be constructed in a modular way
- Great for software engineering (and for analysis)

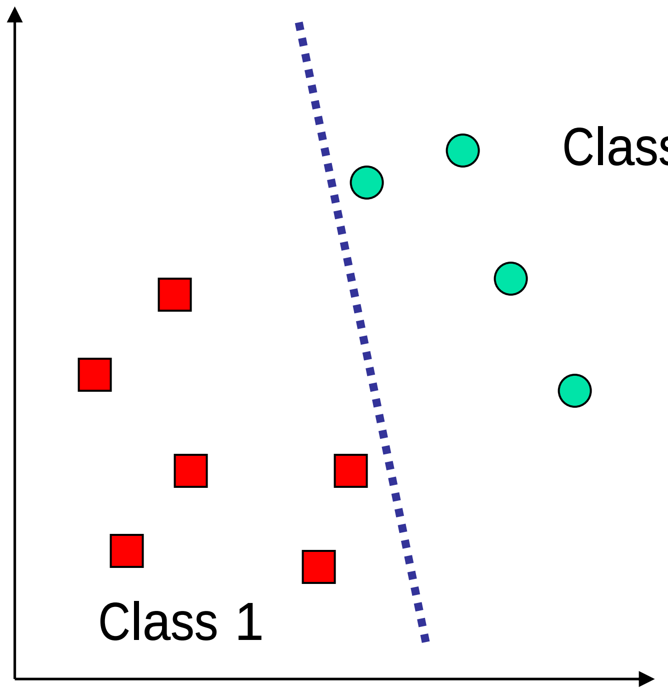
What is a good Decision Boundary?

- Consider a two-class, linearly separable classification problem
- Many decision boundaries!
  - The Perceptron algorithm can be used to find such a boundary
  - Different algorithms have been proposed
  - Are all decision boundaries equally good?

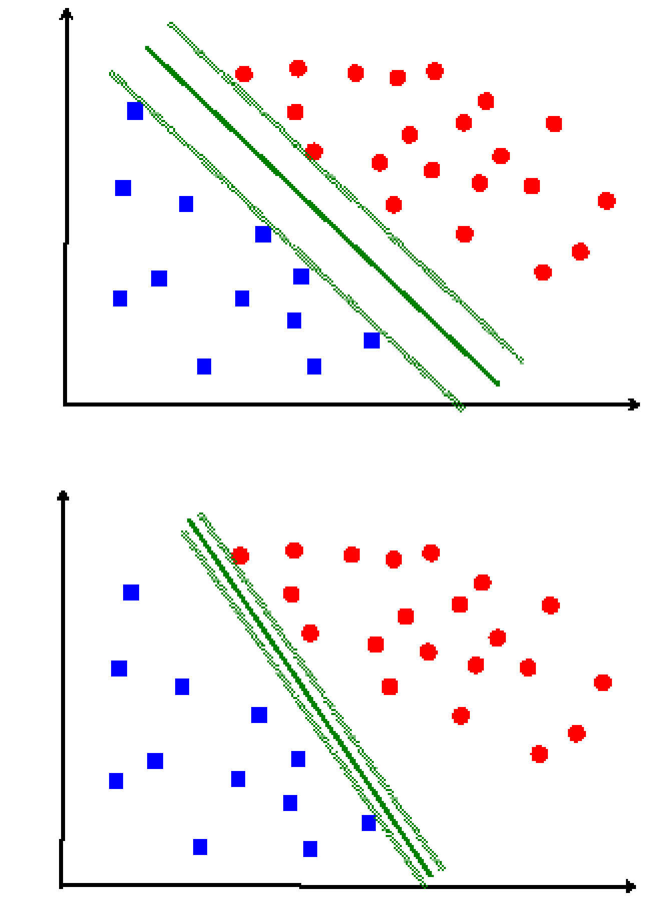


Examples of Bad Decision Boundaries





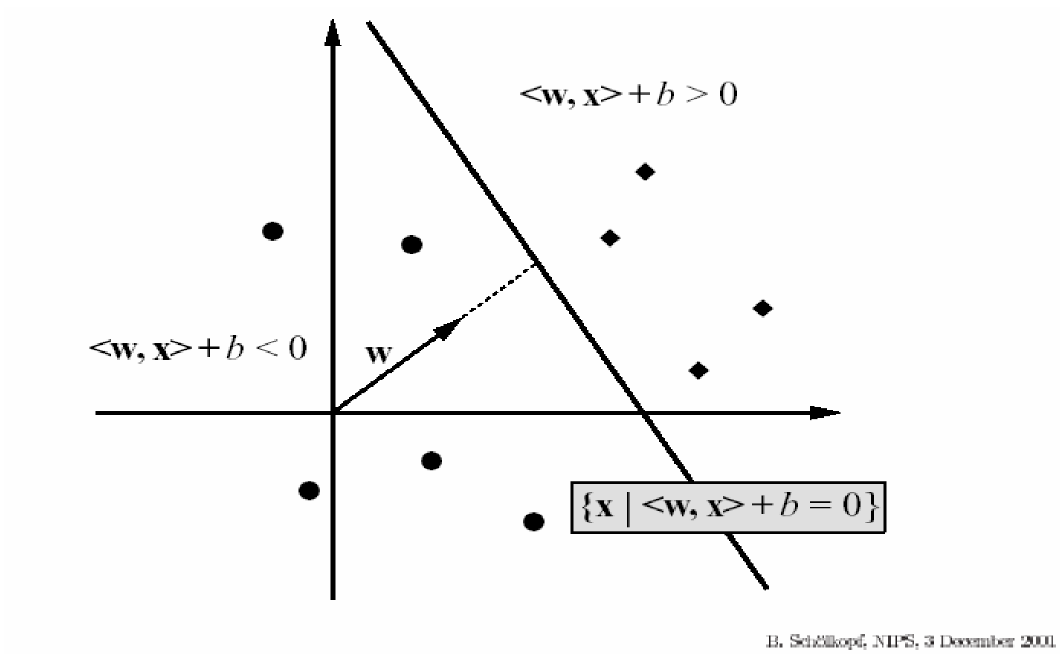
Optimal Separating Hyperplane (1)



- Only consider classification for now;
- the optimal separating hyper plane is the one which has the maximal margin.

[3]

Optimal Separating Hyperplane (2)

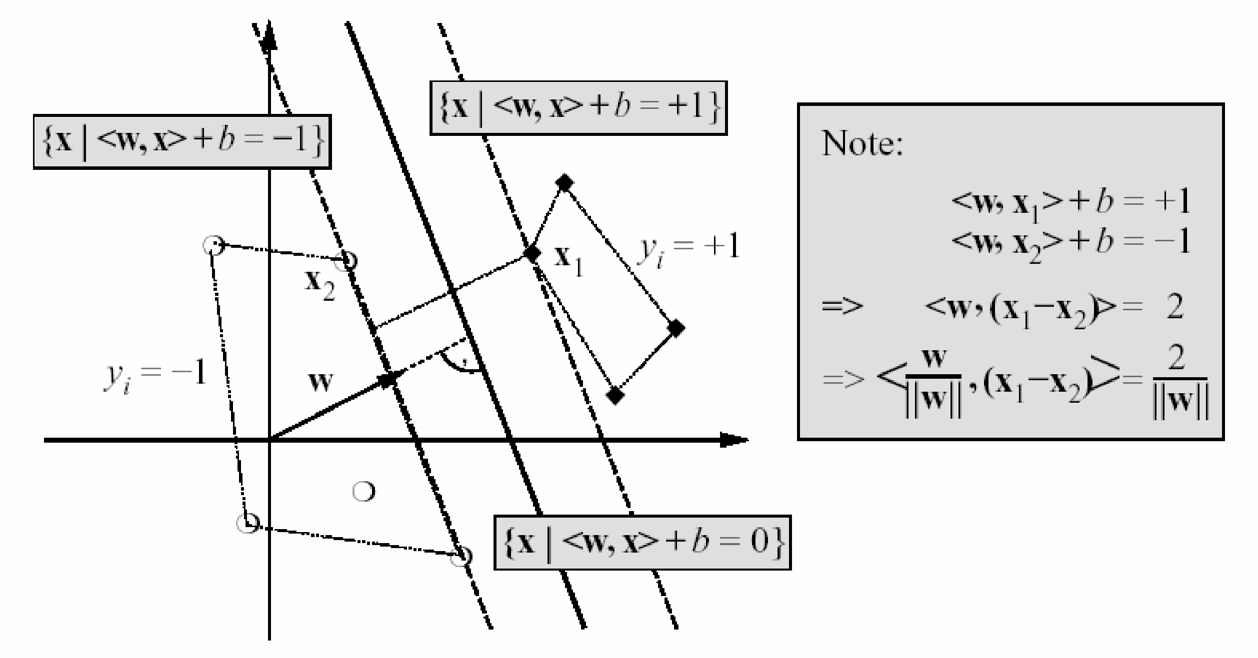


[2]

Hyperplane: H(w,b)={x|<w,x>+b=0};

- Distance from the hyperplane to origin= -b/ ||w||;
- Distance from an arbitrary point X' to the hyperplane = (<w,x'>+b)/ ||w||;

Optimal Separating Hyperplane (3)



Note: if $c \neq 0$, then

$$\{\mathbf{x} | \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\} = \{\mathbf{x} | \langle c\mathbf{w}, \mathbf{x} \rangle + cb = 0\}.$$

Hence $(c\mathbf{w}, cb)$ describes the same hyperplane as $(\mathbf{w}, b)$.

Definition: The hyperplane is in *canonical* form w.r.t. $X^* = \{\mathbf{x}_1, \ldots, \mathbf{x}_r\}$ if $\min_{\mathbf{x}_i \in X} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$. [3]

Optimal Separating Hyperplane (4)

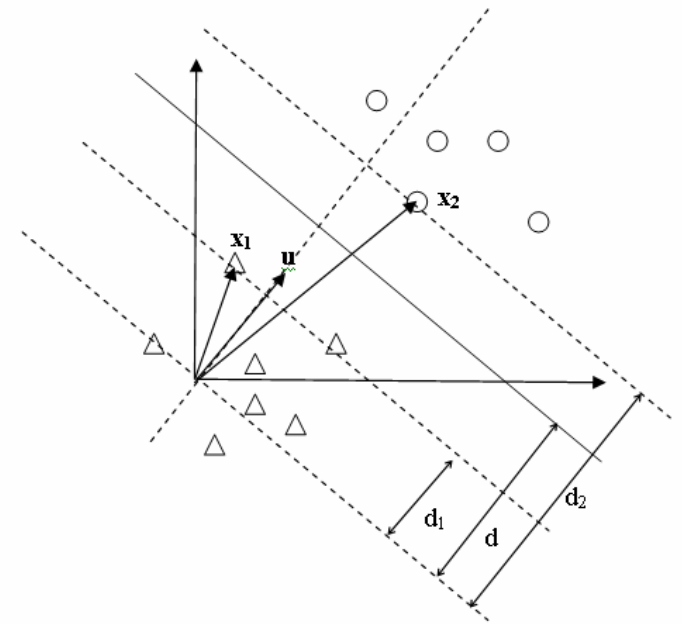
More about canonical form:

To show how canonical form is achieved, let us redefine the hyperplane as below,

$<\boldsymbol{u},\boldsymbol{x}> - d = 0$,

where u is a unit vector, and d is the distance of the hyperplane to origin;

- Note that the same hyperplane is also defined by $<c\boldsymbol{u},\boldsymbol{x}> - cd = 0$, where c is an arbitrary positive real number.
- The criterion of optimal separating hyperplane suggests that we are looking for $\boldsymbol{u}$, and d such tha $d=(d1+d2)/2$, and $(d2-d1)$ is maximized, where $d1=<\boldsymbol{x}1, u>$, $d2=<\boldsymbol{x}2, u>$, and $\boldsymbol{x}1$ and $\boldsymbol{x}2$ are th closest point to the hyperplane for each of the tw classes.



- Let $f(\boldsymbol{x})=<c\boldsymbol{u},\boldsymbol{x}>-cd;$
  Then $f(\boldsymbol{x}_1)=<c\boldsymbol{u},\boldsymbol{x}_1>-cd=cd_1-cd=c(d_1-d_2)/2;$
  $f(\boldsymbol{x}_2)=<c\boldsymbol{u},\boldsymbol{x}_2>-cd=cd_2-cd=c(d_2-d_1)/2;$
- let $c(d_2-d_1)/2=1$, then $f(x_1)=-1$, and $f(x_2)=1$;
  Hence in canonical form, maximize (d_2-d_1) is equivalent to minimize $c/2$,
- If again we define $f(\boldsymbol{x})=<\boldsymbol{w},\boldsymbol{x}>+b;$ where $w = cu$, and $b=-cd;$ note that $c=||w||$.
  Then in canonical form, maximize (d2-d1) is equivalent to minimize $||w||/2$.

Optimal Separating Hyperplane (5)

- Optimal Separating Hyperplane turns out to an optimization problem of the following form:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle$$
$$\text{s.t.} \quad y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \qquad i = 1, 2, \ldots, n$$

1. **Convex quadratic** program
2. Linear inequality constraints (many!)
3. $d+1$ parameters, n constraints

[3]

- It is call

Lagrange multipliers and KKT theorem

Introduce Lagrange multipliers $\alpha_i \geq 0$ and a Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i \cdot [\langle \mathbf{w}, \mathbf{x}_i \rangle + b] - 1\right).$$

- KKT theorem states, a solution to the primal problem must satisfy the following,

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \quad \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0,$$

i.e.

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \qquad \sum_{i=1}^{m} \alpha_i y_i = 0$$

and

$$\sum_{i=1}^{m} \alpha_i \left(y_i \cdot [\langle \mathbf{w}, \mathbf{x}_i \rangle + b] - 1\right) = 0$$

Kuhn-Tucker Theorem

Properties of the solution:

- Duality: can use kernels
- KKT conditions: $\alpha_i \left[y_i (\langle w, x_i \rangle + b) - 1 \right] = 0$ $\forall i$
- Sparseness: only the points nearest to the hyperplane (margin = 1) have positive weight

$$w = \sum \alpha_i y_i x_i$$

- They are called support vectors

Dual Problem(1)

Substitute both into L to get the *dual problem*

Dual: maximize

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

subject to

$$\alpha_i \geq 0, \quad i = 1, \ldots, m, \quad \text{and} \quad \sum_{i=1}^{m} \alpha_i y_i = 0.$$

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$

where for all $i = 1, \ldots, m$ either

$y_i \cdot [\langle \mathbf{w}, \mathbf{x}_i \rangle + b] > 1 \quad \Longrightarrow \alpha_i = 0 \longrightarrow \mathbf{x}_i$ irrelevant

or

$y_i \cdot [\langle \mathbf{w}, \mathbf{x}_i \rangle + b] = 1$ (*on* the margin) $\longrightarrow \mathbf{x}_i$ "Support Vector"

The solution is determined by the examples on the margin.

Thus

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sgn}\left(\langle \mathbf{x}, \mathbf{w} \rangle + b\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b\right). \end{aligned}$$

Dual Problem(2)

- Advantages of Dual Form
  - with simpler constraints, and convex quadratic program algorithms could be applied;
  - the dimension of input space is replaced by the number of input patterns;
  - both the final decision function and the function be maximized are expressed in dot products, which could be computed by a kernel in high dimension space.

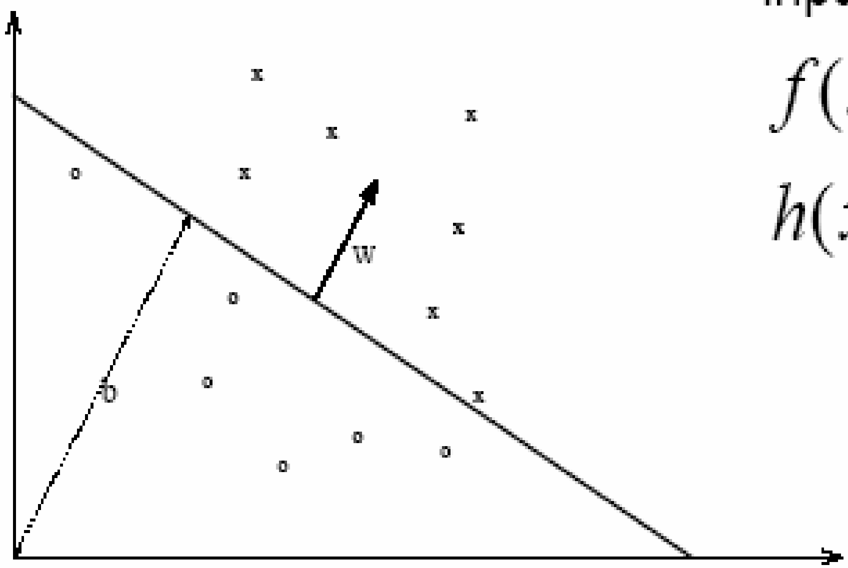
1-Linear Learning Machines

- Simplest case: classification. Decision function is a hyperplane in input space
- The Perceptron Algorithm (Rosenblatt, 1957)
- Useful to analyze the Perceptron algorithm, before looking at SVMs and Kernel Methods in general

Basic Notation

- Input space $x \in X$
- Output space $y \in Y = \{-1, +1\}$
- Hypothesis $h \in H$
- Real-valued: $f : X \rightarrow \mathrm{R}$
- Training Set $S = \{(x_1, y_1), \ldots, (x_i, y_i), \ldots\}$
- Test error ε
- Dot product $\langle x, z \rangle$

Perceptron



- Linear Separation of the input space

$$f(x) = \langle w, x \rangle + b$$

$$h(x) = sign(f(x))$$

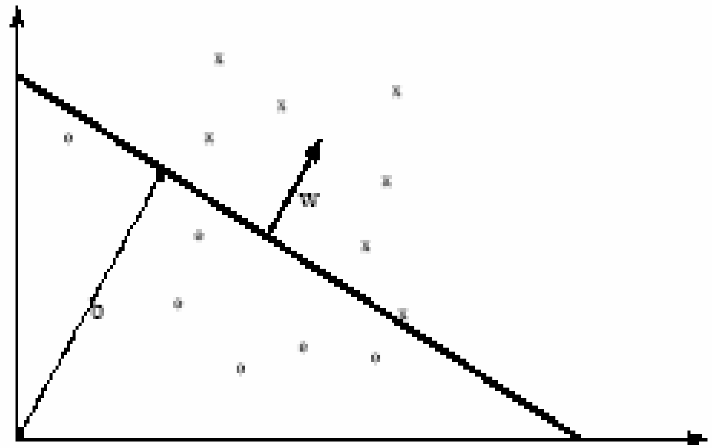
Perceptron Algorithm

Update rule
(ignoring threshold):

- if $y_i(\langle w_k, x_i \rangle) \leq 0$ then

$$w_{k+1} \leftarrow w_k + \eta y_i x_i$$

$$k \leftarrow k + 1$$



Observations

- Solution is a linear combination of training points $w = \sum \alpha_i y_i x_i$

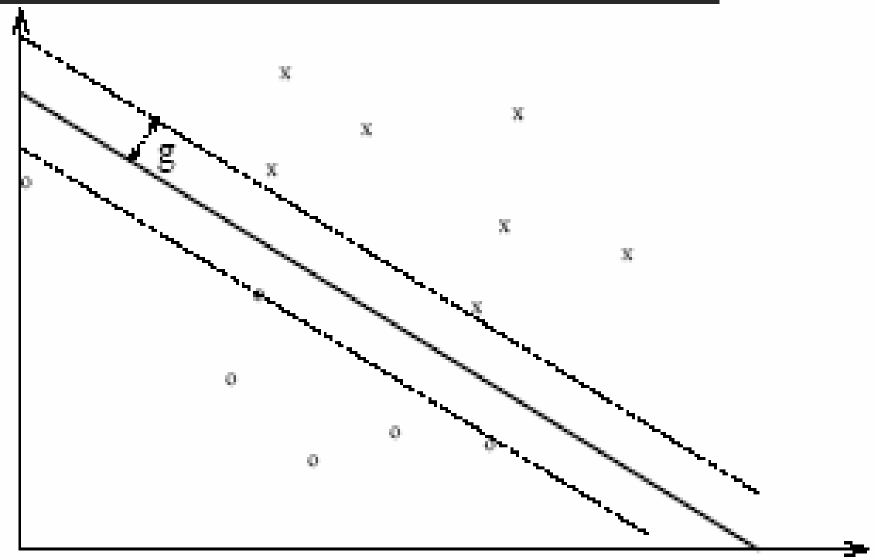
  $\alpha_i \geq 0$

- Only used informative points (mistake driven)
- The coefficient of a point in combination reflects its 'difficulty'

Observations - 2

- Mistake bound:

$$M \leq \left(\frac{R}{\gamma}\right)^2$$



- coefficients are non-negative
- possible to rewrite the algorithm using this alternative representation

Dual representation

The decision function can be re-written as follows:

$$f(x) = \langle w, x \rangle + b = \sum \alpha_i y_i \langle x_i, x \rangle + b$$

$$w = \sum \alpha_i y_i x_i$$

- And also the update rule can be rewritten as follows:
- if $y_i \left(\sum \alpha_j y_j \langle x_j, x_i \rangle + b \right) \leq 0$ then $\alpha_i \leftarrow \alpha_i + \eta$
- Note: in dual representation, data appears only inside dot products

Duality: First Property of SVMs

- DUALITY is the first feature of Support Vector Machines
- SVMs are Linear Learning Machines represented in a dual fashion

$$f(x) = \langle w, x \rangle + b = \sum \alpha_i y_i \langle x_i, x \rangle + b$$

- Data appear only within dot products (in decision function and in training algorithm)

Limitations of LLMs

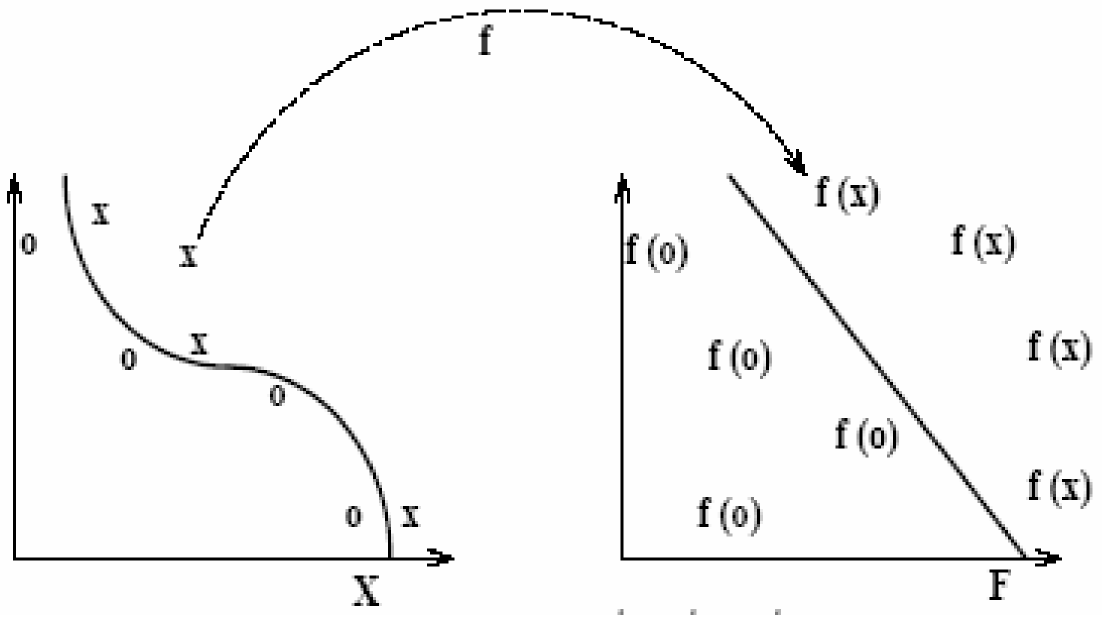
- Linear classifiers cannot deal with
  - Non-linearly separable data
  - Noisy data
- This formulation only deals with vectorial data

Non-Linear Classifiers

- One solution: creating a net of simple linear classifiers (neurons): a Neural Network (problems: local minima; many parameters; heuristics needed to train; etc)
- Other solution: map data into a richer feature space including non-linear features, then use a linear classifier

Learning in the Feature Space

- Map data into a feature space where they are linearly separable $x \rightarrow \phi(x)$



Problems with Feature Space

- Working in high dimensional feature spaces solves the problem of expressing complex functions
- BUT:
- There is a computational problem (working with very large vectors)
- And a generalization theory problem (curse of dimensionality)

Implicit Mapping to Feature Space

- We will introduce Kernels:
- Solve the computational problem of working with many dimensions
- Can make it possible to use infinite dimensions
  – efficiently in time / space
- Other advantages, both practical and conceptual

Kernel-Induced Feature Spaces

- In the dual representation, the data points only appear inside dot products:

$$f(x) = \sum \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle + b$$

- The dimensionality of space F not necessarily important. May not even know the map ϕ

Kernel Trick

- Feature space Mapping

Preprocess the data with

$$\Phi : \mathcal{X} \to \mathcal{H}$$
$$x \mapsto \Phi(x),$$

where $\mathcal{H}$ is a dot product space, and learn the mapping from $\Phi(x)$ to y.

- Example: all 2 degree Monomials

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

$$\begin{aligned}\langle \Phi(x), \Phi(x') \rangle &= (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)(x'^2_1, \sqrt{2}\, x'_1 x'_2, x'^2_2)^\top \\ &= \langle x, x' \rangle^2 \\ &= : k(x, x')\end{aligned}$$

$\longrightarrow$ the dot product in $\mathcal{H}$ can be computed in $\mathbb{R}^2$

Kernels

- A function that returns the value of the dot product between the images of the two arguments

$$K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$$

- Given a function K, it is possible to verify that it is a kernel

- One can use LLMs in a feature space by simply rewriting it in dual representation and replacing dot products with kernels:

$$\langle x_1, x_2 \rangle \leftarrow K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$$

The kernel matrix

- (aka the Gram matrix):

K=

K(1,1)	K(1,2)	K(1,3)	…	K(1,m)
K(2,1)	K(2,2)	K(2,3)	…	K(2,m)
…	…	…	…	…
K(m,1)	K(m,2)	K(m,3)	…	K(m,m)

- The central structure in kernel machines
- Information ‘bottleneck’: contains all necessary information for the learning algorithm
- Fuses information about the data AND the kernel
- Many interesting properties:

Mercer's Theorem

- The kernel matrix is Symmetric Positive Definite
- Any symmetric positive definite matrix can be regarded as a kernel matrix, that is as an inner product matrix in some space

More formal Mercer's Theorem

- Every (semi) positive definite, symmetric function is a kernel: i.e. there exists a mapping

$$\phi$$

such that it is possible to write:

$$K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$$

Pos. Def. $\int K(x,z) f(x) f(z) dx dz \geq 0$

$\forall f \in L_2$

- Eigenvalues expansion of Mercer's Kernels:

$$K(x_1, x_2) = \sum_i \lambda_i \phi_i(x_1) \phi_i(x_2)$$

- That is: the eigenfunctions act as features !

Examples of Kernels

- Simple examples of kernels are:

$$K(x,z) = \langle x,z \rangle^d$$

$$K(x,z) = e^{-\|x-z\|^2/2\sigma}$$

Polynomial kernels

$$x = (x_1, x_2);$$
$$z = (z_1, z_2);$$

$$\langle x,z \rangle^2 = (x_1 z_1 + x_2 z_2)^2 =$$
$$= x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 z_1 x_2 z_2 =$$
$$= \langle (x_1^2, x_2^2, \sqrt{2} x_1 x_2), (z_1^2, z_2^2, \sqrt{2} z_1 z_2) \rangle =$$
$$= \langle \phi(x), \phi(z) \rangle$$

Examples of Kernel Functions

- Polynomial kernel with degree *d*

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$$

- Radial basis function kernel with width s

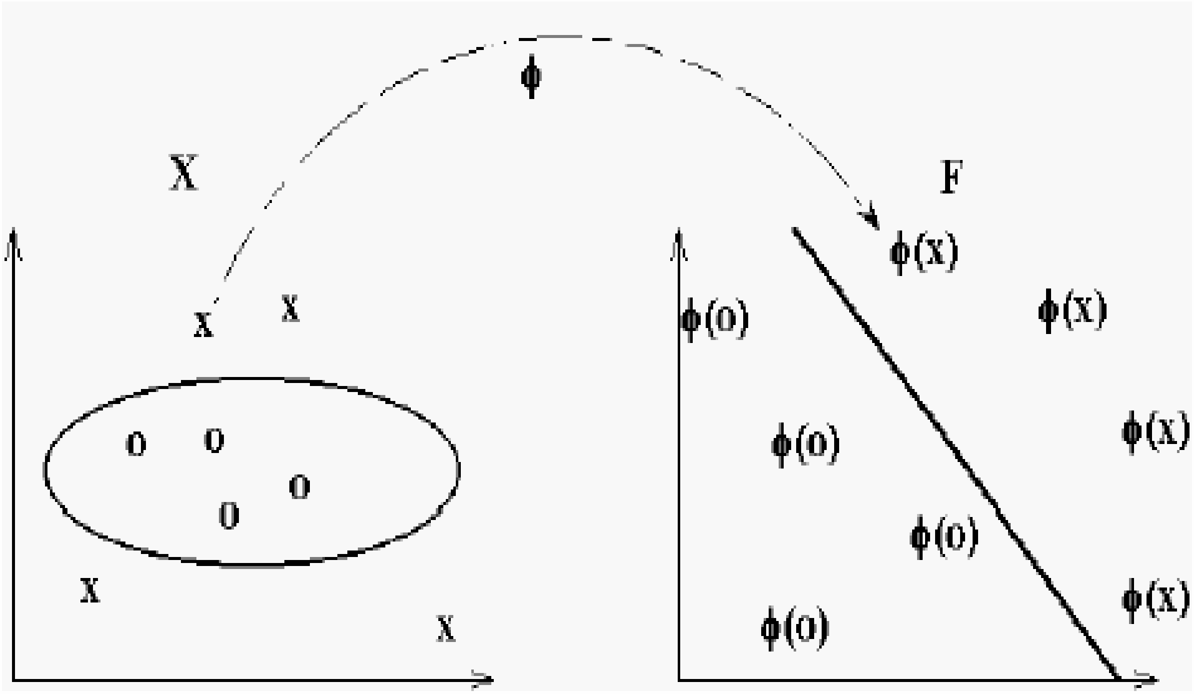
$$K(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||^2/(2\sigma^2))$$

  - Closely related to radial basis function neural networks
  - The feature space is infinite-dimensional
- Sigmoid with parameter k and q

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x}^T \mathbf{y} + \theta)$$

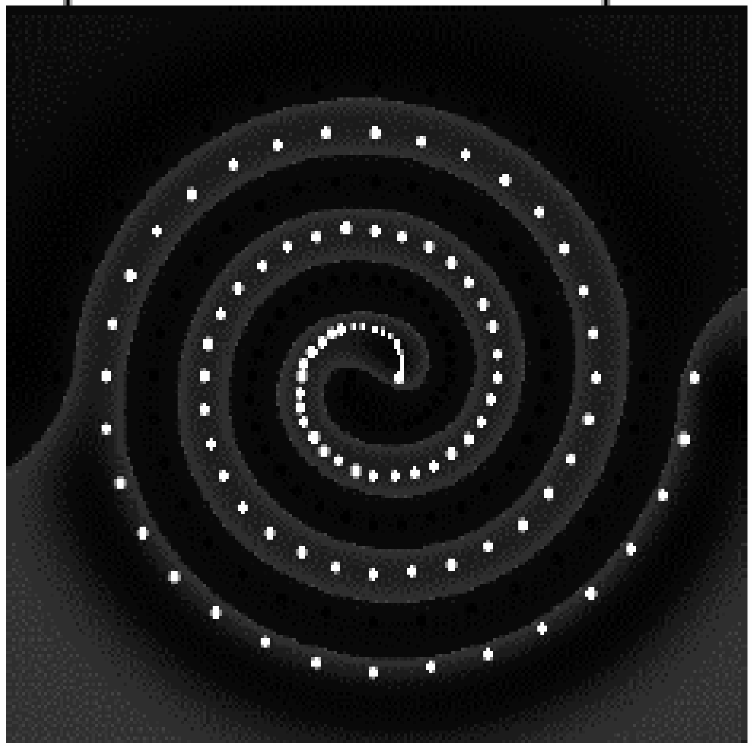
  - It does not satisfy the Mercer condition on all k and q

Polynomial kernels



Example: the two spirals

- Separated by a hyperplane in feature space (gaussian kernels)



Making kernels

- The set of kernels is <u>closed under some operations</u>. If K, K’ are kernels, then:
- K+K’ is a kernel
- cK is a kernel, if c>0
- aK+bK’ is a kernel, for a,b >0
- Etc etc etc……
- can make complex kernels from simple ones: modularity !

Kernel Functions

- In practical use of SVM, the user specifies the kernel function; the transformation f(.) is not explicitly stated
- Given a kernel function $K(\mathbf{x}i, \mathbf{x}j)$, the transformation f(.) is given by its eigenfunctions (a concept in functional analysis)
  - Eigenfunctions can be difficult to construct explicitly
  - This is why people only specify the kernel function without worrying about the exact transformation
- Another view: kernel function, being an inner product, is really a similarity measure between the objects

More on Kernel Functions

- Since the training of SVM only requires the value of *K*(**x**i, **x**j), there is no restriction of the form of **x**i and **x**j
  - **x**i can be a sequence or a tree, instead of a feature vector
- *K*(**x**i, **x**j) is just a similarity measure comparing **x**i and **x**j
- For a test object **z**, the discriminat function essentially is a weighted sum of the similarity between z and a pre-selected set of objects (the support vectors)

$$f(\mathbf{z}) = \sum_{\mathbf{x}_i \in \mathcal{S}} \alpha_i y_i K(\mathbf{z}, \mathbf{x}_i) + b$$

$\mathcal{S}$: the set of support vectors

More on Kernel Functions

- Not all similarity measure can be used as kernel function, however
  - The kernel function needs to satisfy the Mercer function, i.e., the function is “positive-definite”
  - This implies that the *n* by *n* kernel matrix, in which the (i,j)-th entry is the $K(\mathbf{x}i, \mathbf{x}j)$, is always positive definite
  - This also means that the QP is convex and can be solved in polynomial time

Second property of SVMs

SVMs are Linear Learning Machines, that

- Use a dual representation

AND

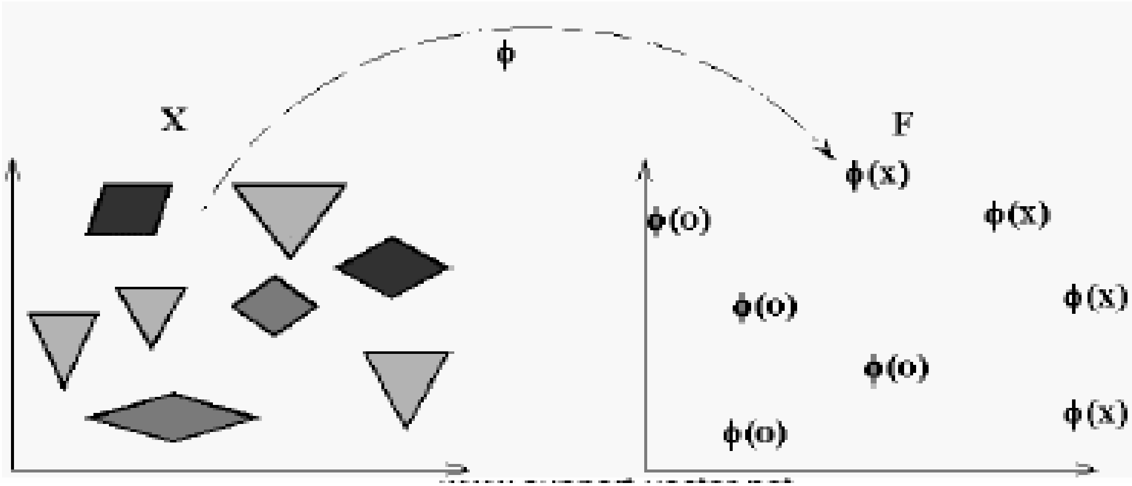
- Operate in a kernel induced feature space

(that is: $f(x) = \sum \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle + b$

is a linear function in the feature space implicitely defined by K)

Kernels over General Structures

- Haussler, Watkins, etc: kernels over sets, over sequences, over trees, etc.
- Applied in text categorization, bioinformatics,



A bad kernel …

- … would be a kernel whose kernel matrix is mostly diagonal: all points orthogonal to each other, no clusters, no structure …

1	0	0	…	0
0	1	0	…	0
		1		
…	…	…	…	…
0	0	0	…	1

Example

- Suppose we have 5 1D data points
  - x1=1, x2=2, x3=4, x4=5, x5=6, with 1, 2, 6 as class 1 and 4, 5 as class 2 ⇒ y1=1, y2=1, y3=-1, y4=-1, y5=1
- We use the polynomial kernel of degree 2
  - K(x,y) = (xy+1)2
  - C is set to 100
- We first find ai (*i*=1, ..., 5) by

$$\text{max.} \quad \sum_{i=1}^{5} \alpha_i - \frac{1}{2} \sum_{i=1}^{5} \sum_{j=1}^{5} \alpha_i \alpha_j y_i y_j (x_i x_j + 1)^2$$

$$\text{subject to } 100 \geq \alpha_i \geq 0, \sum_{i=1}^{5} \alpha_i y_i = 0$$

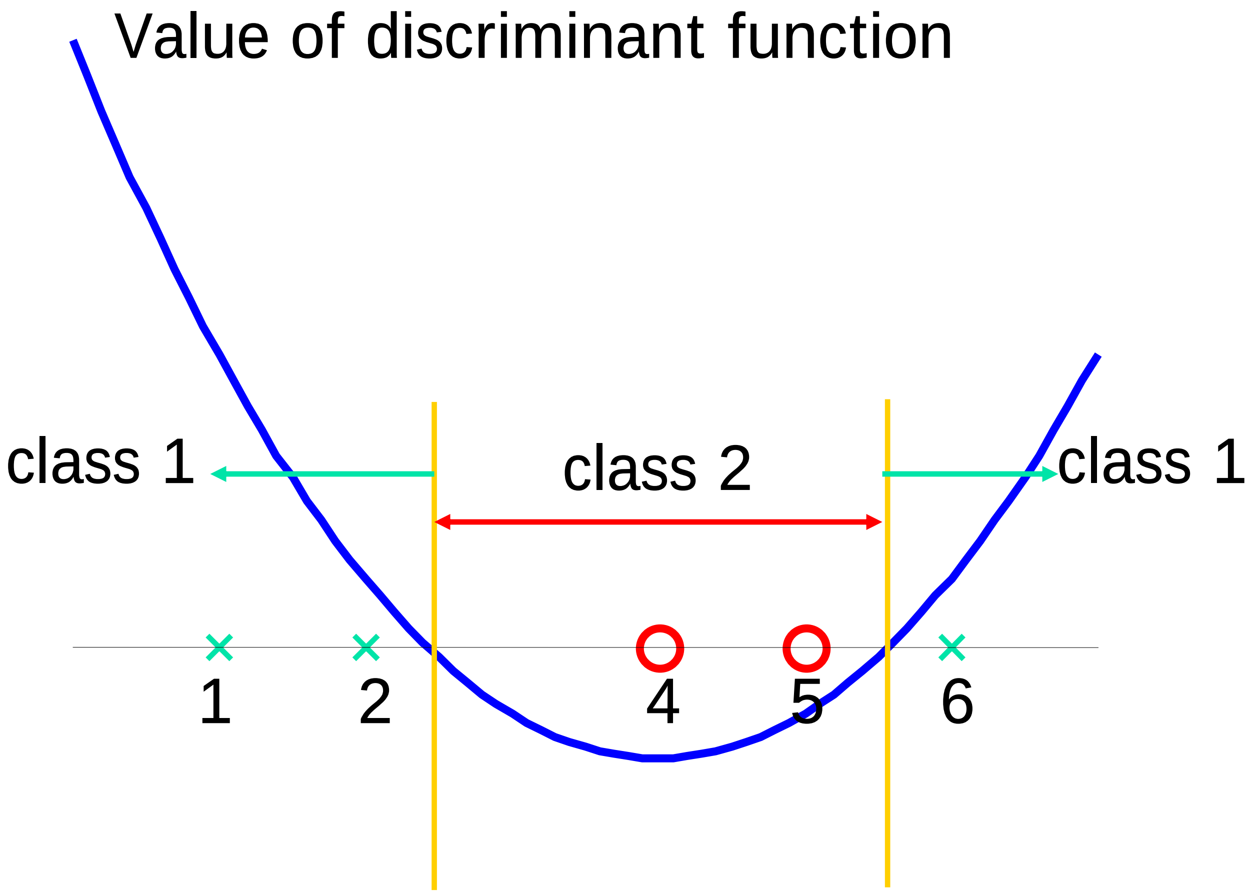
Example

- By using a QP solver, we get
  - a1=0, a2=2.5, a3=0, a4=7.333, a5=4.833
  - Note that the constraints are indeed satisfied
  - The support vectors are {x2=2, x4=5, x5=6}
- The discriminant function is

$$\begin{aligned} & f(z) \\ &= 2.5(1)(2z+1)^2 + 7.333(-1)(5z+1)^2 + \underbrace{4.833}_{\alpha_5}\underbrace{(1)}_{y_5}\underbrace{(6z+1)^2}_{K(z,x_5)} + b \\ &= 0.6667z^2 - 5.333z + b \end{aligned}$$

- *b* is recovered by solving f(2)=1 or by f(5)=-1 or by f(6)=1, as x2 and x5 lie on $\phi(\mathbf{w})^T\phi(\mathbf{x}) + b = 1$ and x4 lies on t$\phi(\mathbf{w})^T\phi(\mathbf{x}) + b = -1$
- All three give b=9 ⟹ $f(z) = 0.6667z^2 - 5.333z + 9$

Example



Value of discriminant function
class 1
class 2
class 1
1
2
4
5
6

No Free Kernel

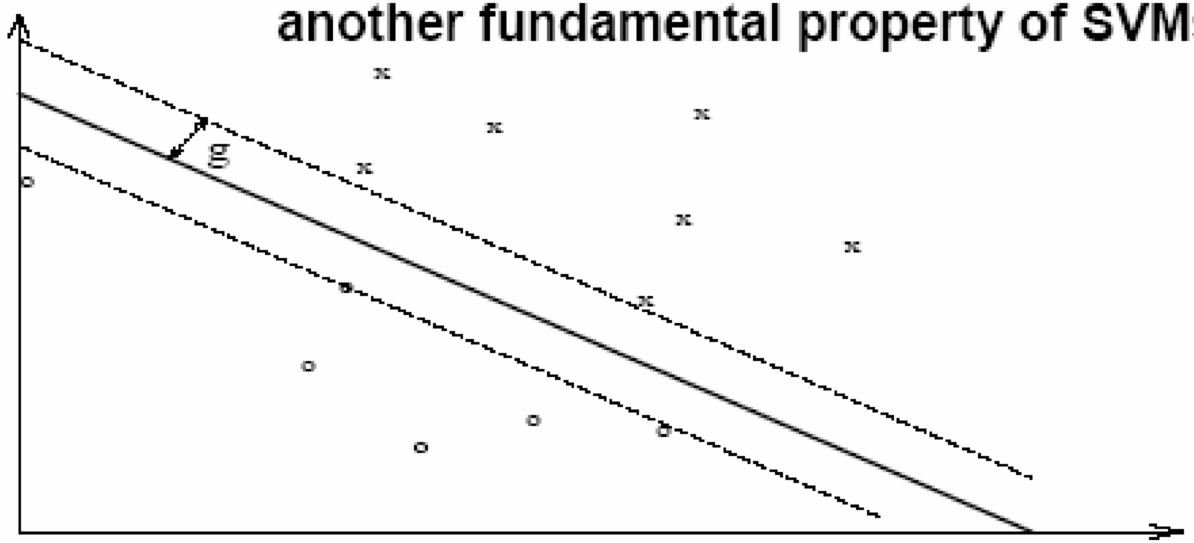
- If mapping in a space with too many irrelevant features, kernel matrix becomes diagonal
- Need some prior knowledge of target so choose a good kernel

Convexity

- This is a Quadratic Optimization problem: convex, no local minima (second effect of Mercer's conditions)
- Solvable in polynomial time ...
- (convexity is another fundamental property of SVMs)

KKT Conditions Imply Sparseness

Sparseness:
another fundamental property of SVMs



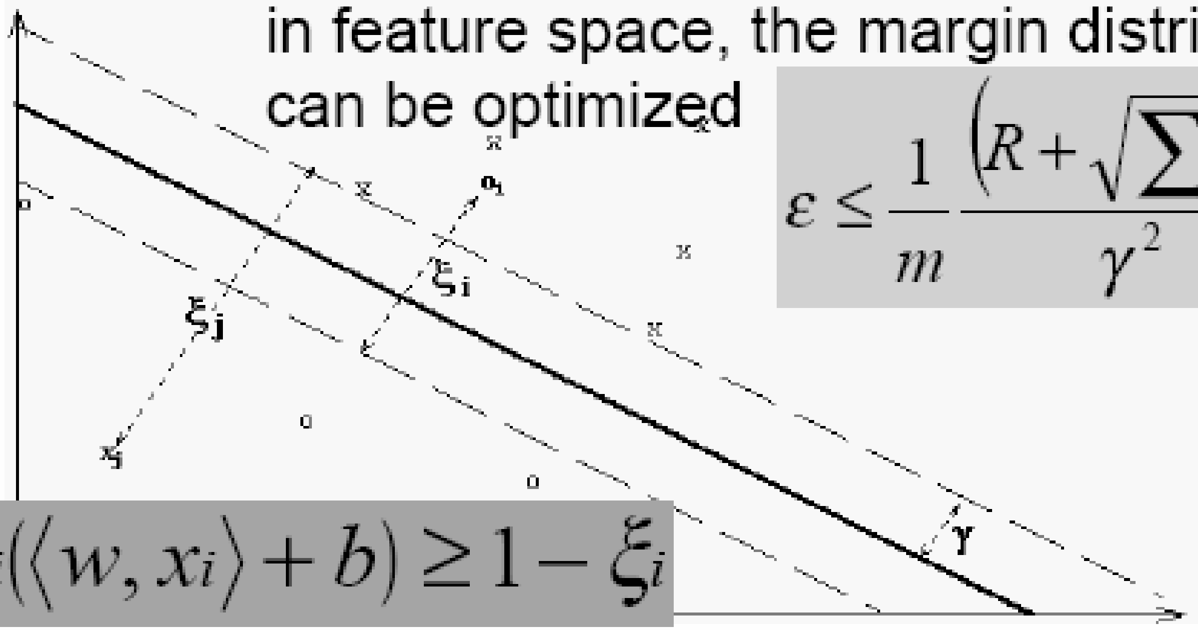
Properties of SVMs - Summary

- ✓ Duality
- ✓ Kernels
- ✓ Margin
- ✓ Convexity
- ✓ Sparseness

Dealing with noise

In the case of non-separable data in feature space, the margin distribution can be optimized

$$\varepsilon \leq \frac{1}{m}\frac{\left(R+\sqrt{\sum \xi^2}\right)^2}{\gamma^2}$$



$$y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i$$

The Soft-Margin Classifier

Minimize: $$\frac{1}{2}\langle w, w\rangle + C\sum_i \xi_i$$

Or: $$\frac{1}{2}\langle w, w\rangle + C\sum_i \xi_i{}^2$$

Subject to: $$y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i$$

Applications of SVMs

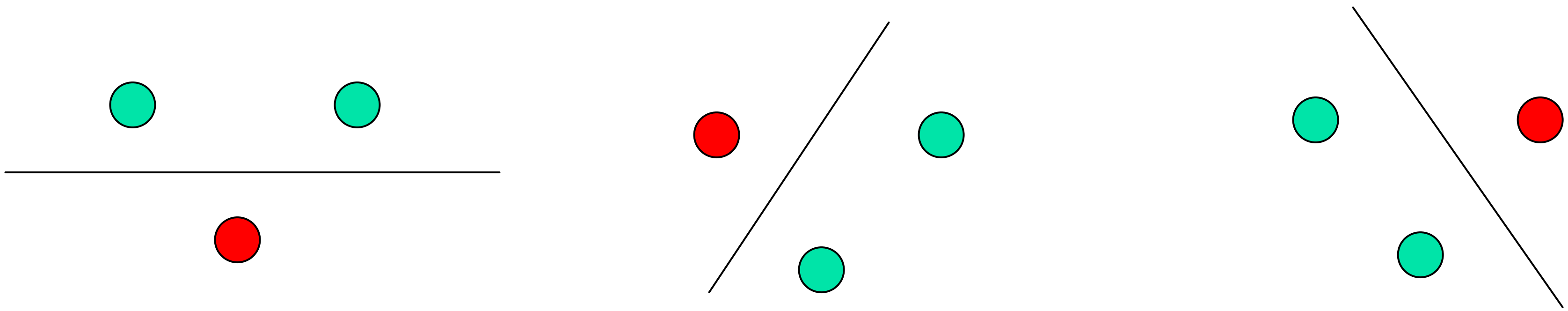
- Bioinformatics
- Machine Vision
- Text Categorization
- Handwritten Character Recognition
- Time series analysis

Why SVM Work?

- The feature space is often very high dimensional. Why don't we have the curse of dimensionality?
- A classifier in a high-dimensional space has many parameters and is hard to estimate
- Vapnik argues that the fundamental problem is not the number of parameters to be estimated. Rather, the problem is about the flexibility of a classifier
- Typically, a classifier with many parameters is very flexible, but there are also exceptions
  - Let $x_i=10^i$ where i ranges from 1 to n. The classifier $y = \text{sign}\big(\sin(\alpha x)\big)$ can classify all x_i correctly for all possible combination of class labels on x_i
  - This 1-parameter classifier is very flexible

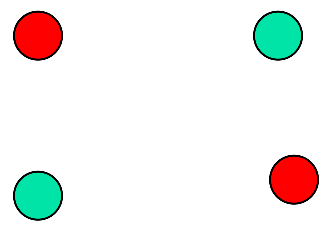
Why SVM works?

- Vapnik argues that the flexibility of a classifier should not be characterized by the number of parameters, but by the flexibility (capacity) of a classifier
  - This is formalized by the "VC-dimension" of a classifier
- Consider a linear classifier in two-dimensional space
- If we have three training data points, no matter how those points are labeled, we can classify them perfectly



VC-dimension

- However, if we have four points, we can find a labeling such that the linear classifier fails to be perfect



- We can see that 3 is the critical number
- The VC-dimension of a linear classifier in a 2D space is 3 because, if we have 3 points in the training set, perfect classification is always possible irrespective of the labeling, whereas for 4 points, perfect classification can be impossible

VC-dimension

- The VC-dimension of the nearest neighbor classifier is infinity, because no matter how many points you have, you get perfect classification on training data
- The higher the VC-dimension, the more flexible a classifier is
- VC-dimension, however, is a theoretical concept; the VC-dimension of most classifiers, in practice, is difficult to be computed exactly
  - Qualitatively, if we think a classifier is flexible, it probably has a high VC-dimension

Choosing the Kernel Function

- Probably the most tricky part of using SVM.
- The kernel function is important because it creates the kernel matrix, which summarizes all the data
- Many principles have been proposed (diffusion kernel, Fisher kernel, string kernel, ...)
- There is even research to estimate the kernel matrix from available information

- In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try
- Note that SVM with RBF kernel is closely related to RBF neural networks, with the centers of the radial basis functions automatically chosen for SVM

Software

- A list of SVM implementation can be found at http://www.kernel-machines.org/software.html

Summary: Steps for Classification

- Prepare the pattern matrix
- Select the kernel function to use
- Select the parameter of the kernel function and the value of *C*
  - You can use the values suggested by the SVM software, or you can set apart a validation set to determine the values of the parameter
- Execute the training algorithm and obtain the α_i
- Unseen data can be classified using the α_i and the support vectors

Strengths and Weaknesses of SVM

- Strengths
  - Training is relatively easy
    - No local optimal, unlike in neural networks
  - It scales relatively well to high dimensional data
  - Tradeoff between classifier complexity and error can be controlled explicitly
  - Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors
- Weaknesses
  - Need to choose a “good” kernel function.

Conclusion

- SVM is a useful alternative to neural networks
- Two key concepts of SVM: maximize the margin and the kernel trick
- Many SVM implementations are available on the web for you to try on your data set!

Resources

- http://www.kernel-machines.org/
- http://www.support-vector.net/
- http://www.support-vector.net/icml-tutorial.pdf
- http://www.kernel-machines.org/papers/tutorial-nips.ps.gz
- http://www.clopinet.com/isabelle/Projects/SVM/applist.html