Support Vector Machines

SVM's

What is an SVM?

- An SVM is a type of neural network.
- It generalize both perceptrons and RBF networks.
- A novel training method is used.
- It has origins in a “different” field: statistical learning theory.
- Originally the work was done at AT&T Bell Labs, and it was called “Support Vector Network” (by Corrina Cortes and Vladimir Vapnik).

Abstract of the Original Paper, 1995

Machine Learning, 20, 273–297 (1995)

Support-Vector Networks

CORINNA CORTES
VLADIMIR VAPNIK
AT&T Bell Labs., Holmdel, NJ 07733, USA

corinna@neural.att.com
vlad@neural.att.com

C. Cortes

Editor: Lorenza Saitta

Abstract. The *support-vector network* is a new learning machine for two-group classification problems. The machine conceptually implements the following idea: input vectors are non-linearly mapped to a very high-dimension feature space. In this feature space a linear decision surface is constructed. Special properties of the decision surface ensures high generalization ability of the learning machine. The idea behind the support-vector network was previously implemented for the restricted case where the training data can be separated without errors. We here extend this result to non-separable training data.

High generalization ability of support-vector networks utilizing polynomial input transformations is demonstrated. We also compare the performance of the support-vector network to various classical learning algorithms that all took part in a benchmark study of Optical Character Recognition.

V. Vapnik

Keywords: pattern recognition, efficient learning algorithms, neural networks, radial basis function classifiers, polynomial classifiers.

SVM Uses

- Classification problems
- Data mining
- Gene analysis
- Scene analysis
- Text analysis
- Regression (function approximation)

The Main Idea

- As we know, many classification problems are not linearly separable.
- However, mapping the data to a higher-dimension space may render it linearly separable.
- SVM's provide a way to map **any** functional data set to a sufficiently high-dimensional space so as to render it linearly separable in that space. [Caution: This may *over-fit* the data.]
- Furthermore, they provide an efficient means of computing the classification in this high-dimensional space without explicitly transforming the data.

Example: xor

- We know that the classification:

  c(0, 0) = 0
  c(1, 0) = 1
  c(0, 1) = 1
  c(1, 1) = 0

  is not linearly separable in 2-space.

- Furthermore, intuitively, no **linear** transformation is going to render it separable.

Example: xor

- We can map the problem to 3-space using a **non-linear** transformation, such as:
  - f(x, y) = (x, y, x*y) x*y = product of x and y
  - f(0, 0) = (0, 0, 0), d(0, 0, 0) = 0
  - f(0, 1) = (0, 1, 0), d(0, 1, 0) = 1
  - f(1, 0) = (1, 0, 0), d(1, 0, 0) = 1
  - f(1, 1) = (1, 1, 1), d(1, 1, 1) = 0

- The corresponding function d ***is*** linearly separable:

  d(x, y, z) = (x + y -z) > 0

- In a sense, we have added another “layer”, but the parameters of this layer can be computed rather than learned.

Feature Space

- The space into which the inputs are mapped is called “feature space” because certain components of it can be identified with features in the input.

- When the separating hyperplane in feature space is mapped back to the input space, it will show as a **non-linear** surface.

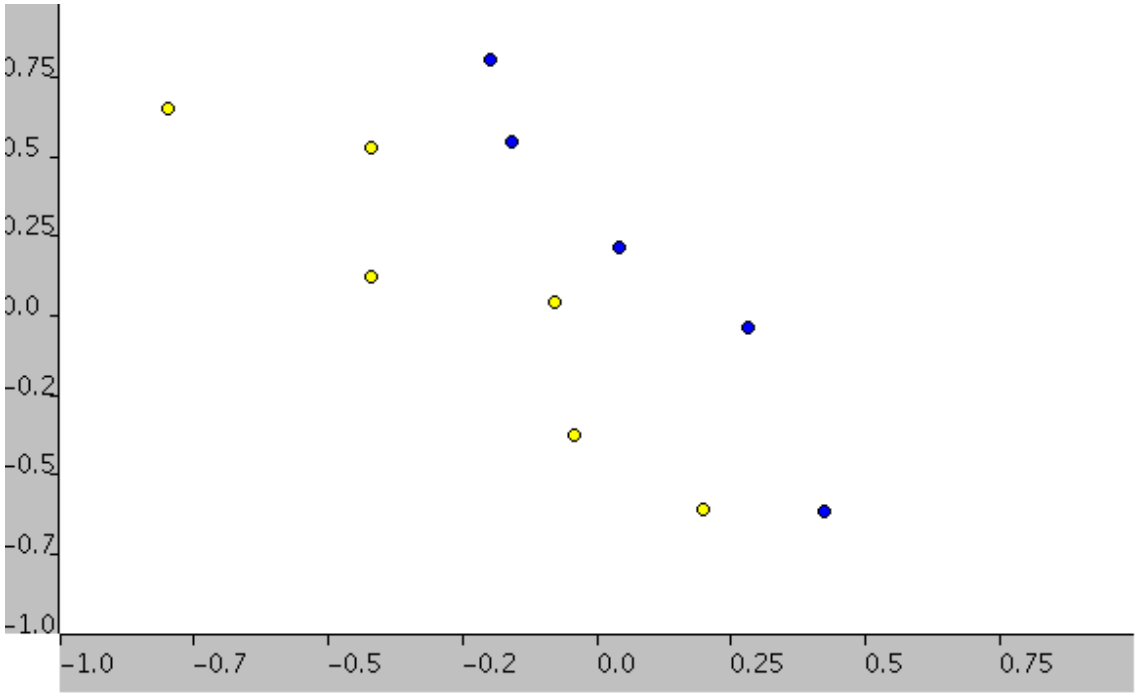
Transformations

- The transformations are selected by the user in advance.
- They are determined by a set of functions called “kernel functions”.
- Once the kernel function and “cost” parameter are set, training is deterministic.
- One possible kernel is the simple inner product: the SVM will perform **optimally** for a linear classifier.

Example Applet (AT&T)

(http://svm.dcs.rhbnc.ac.uk/pagesnew/GPat.shtml)

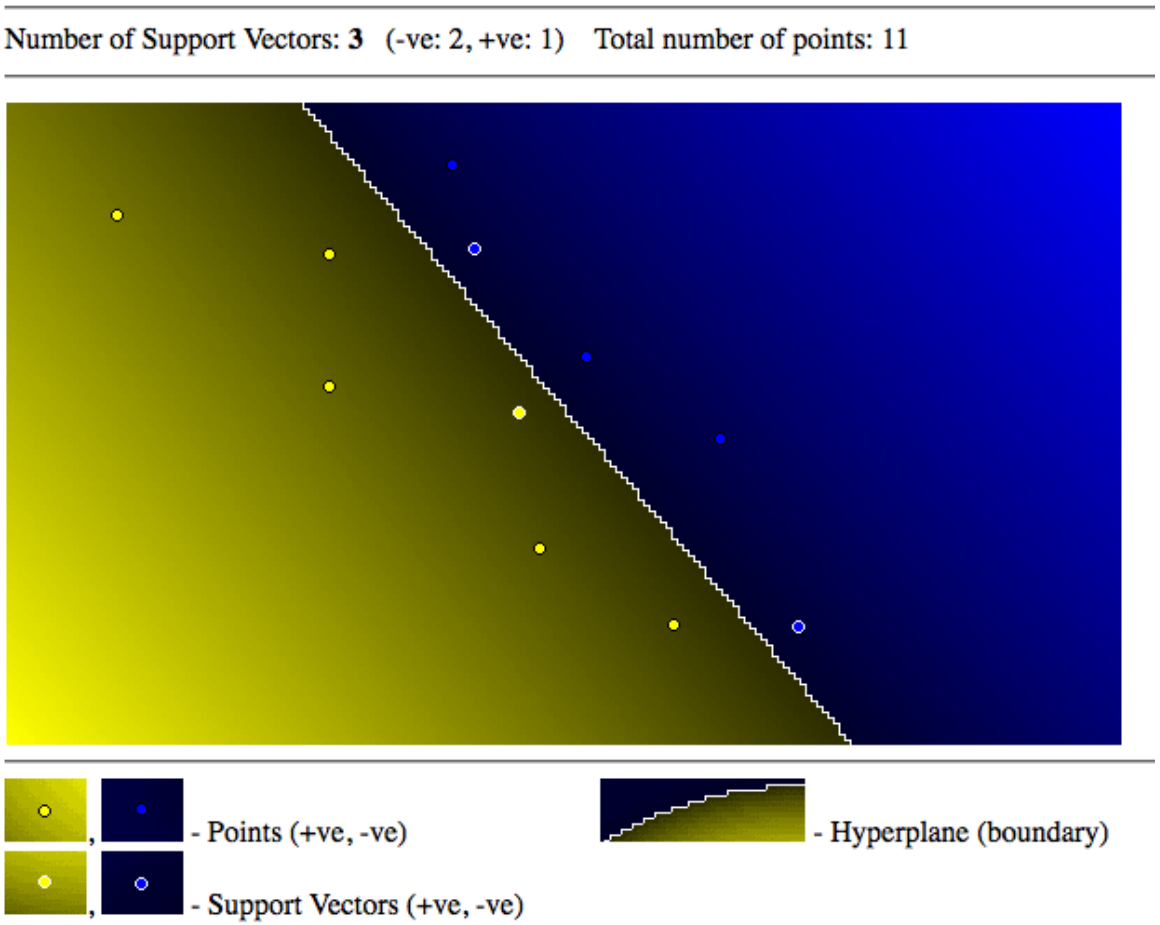
Linearly Separable Case



Example Applet (AT&T)

(http://svm.dcs.rhbnc.ac.uk/pagesnew/GPat.shtml)

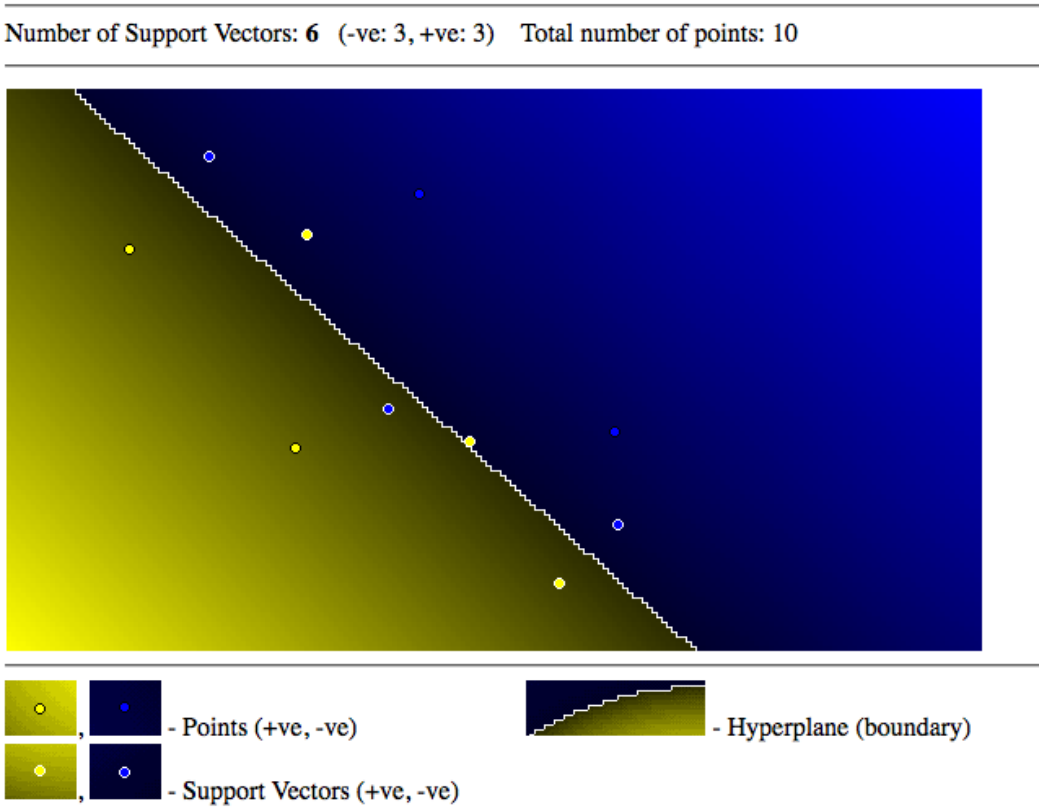
Linearly Separable Case



Example Applet (AT&T)

(http://svm.dcs.rhbnc.ac.uk/pagesnew/GPat.shtml)

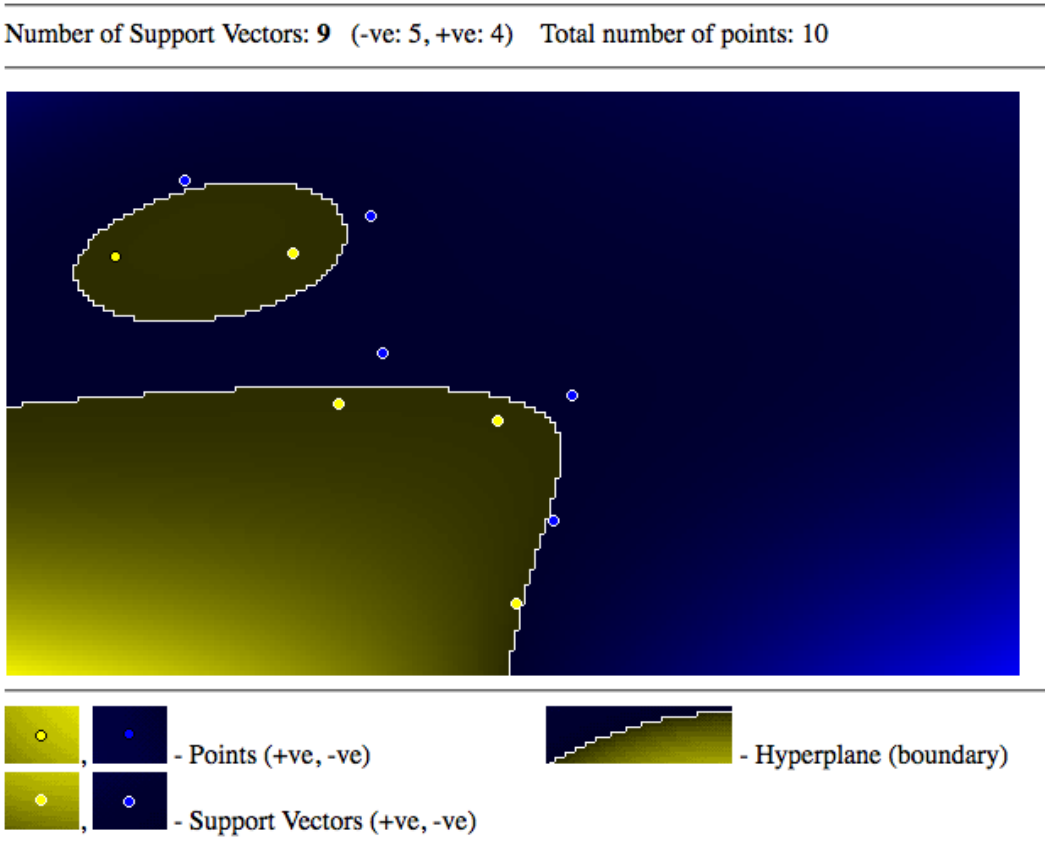
Non-Linearly Separable Case with linear classifier



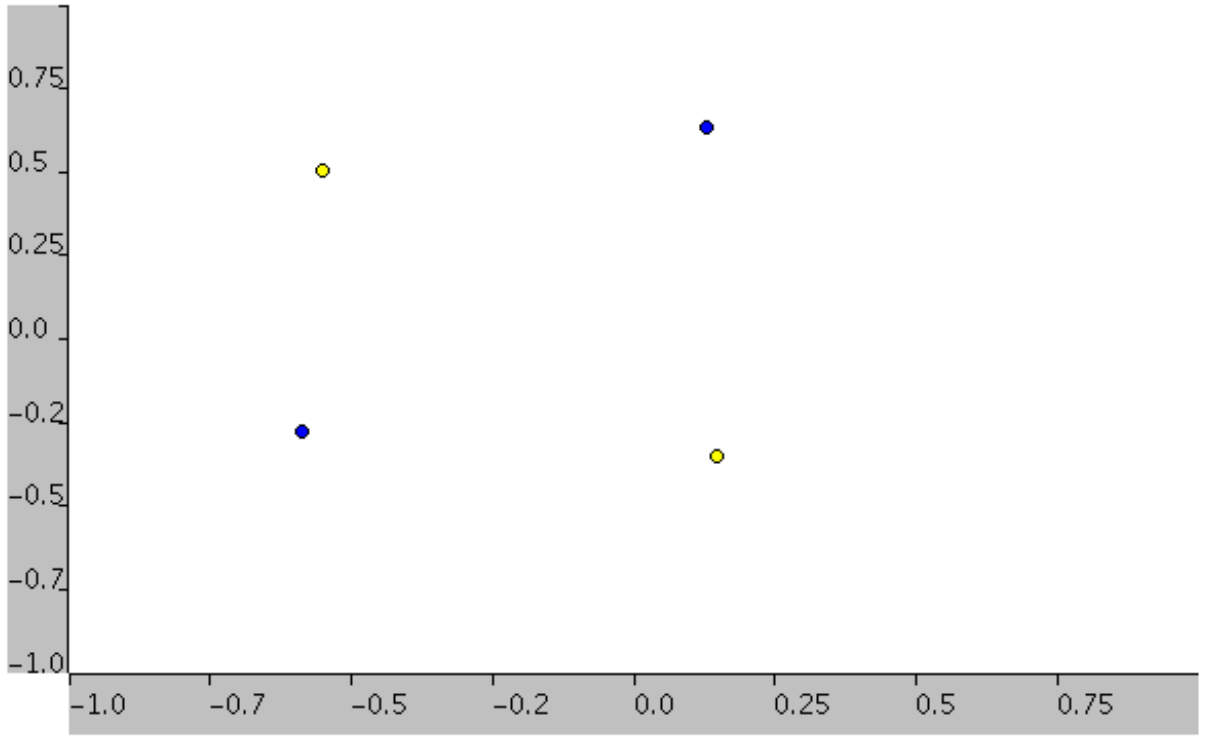
Example Applet (AT&T)

(http://svm.dcs.rhbnc.ac.uk/pagesnew/GPat.shtml)

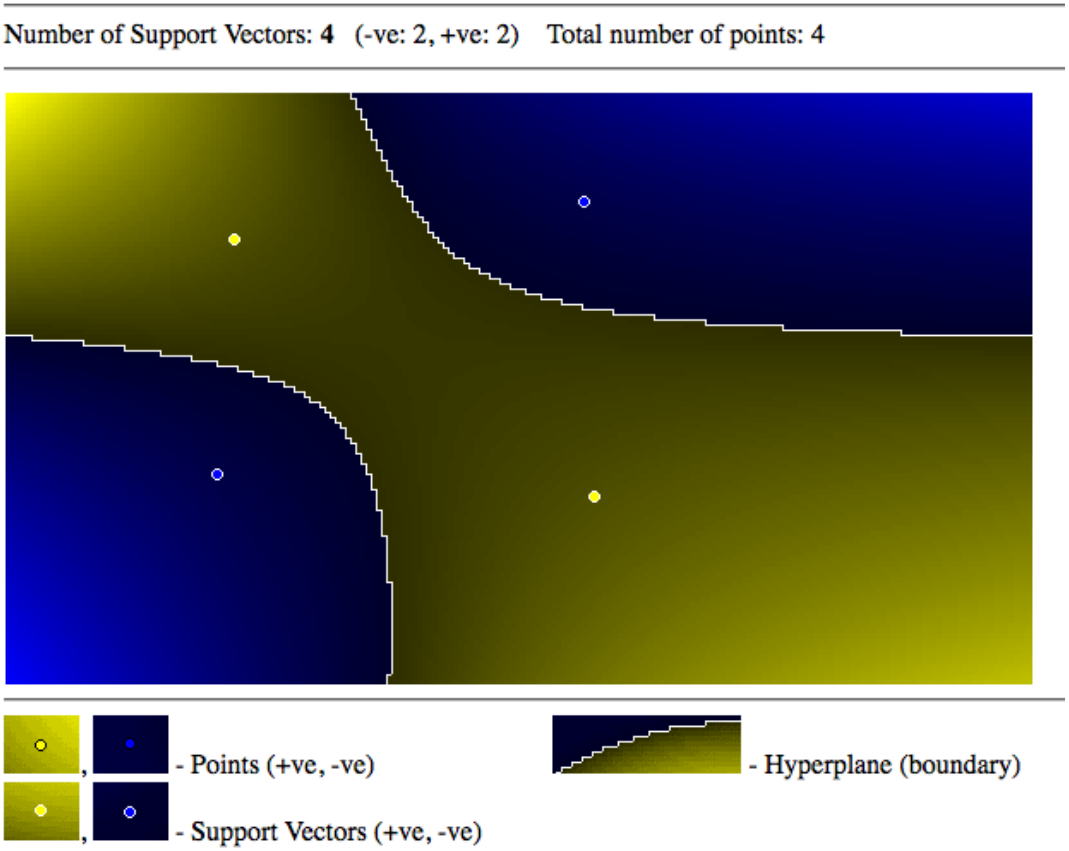
Non-Linearly Separable Case
with non-linear classifier (degree 5 polynomial)



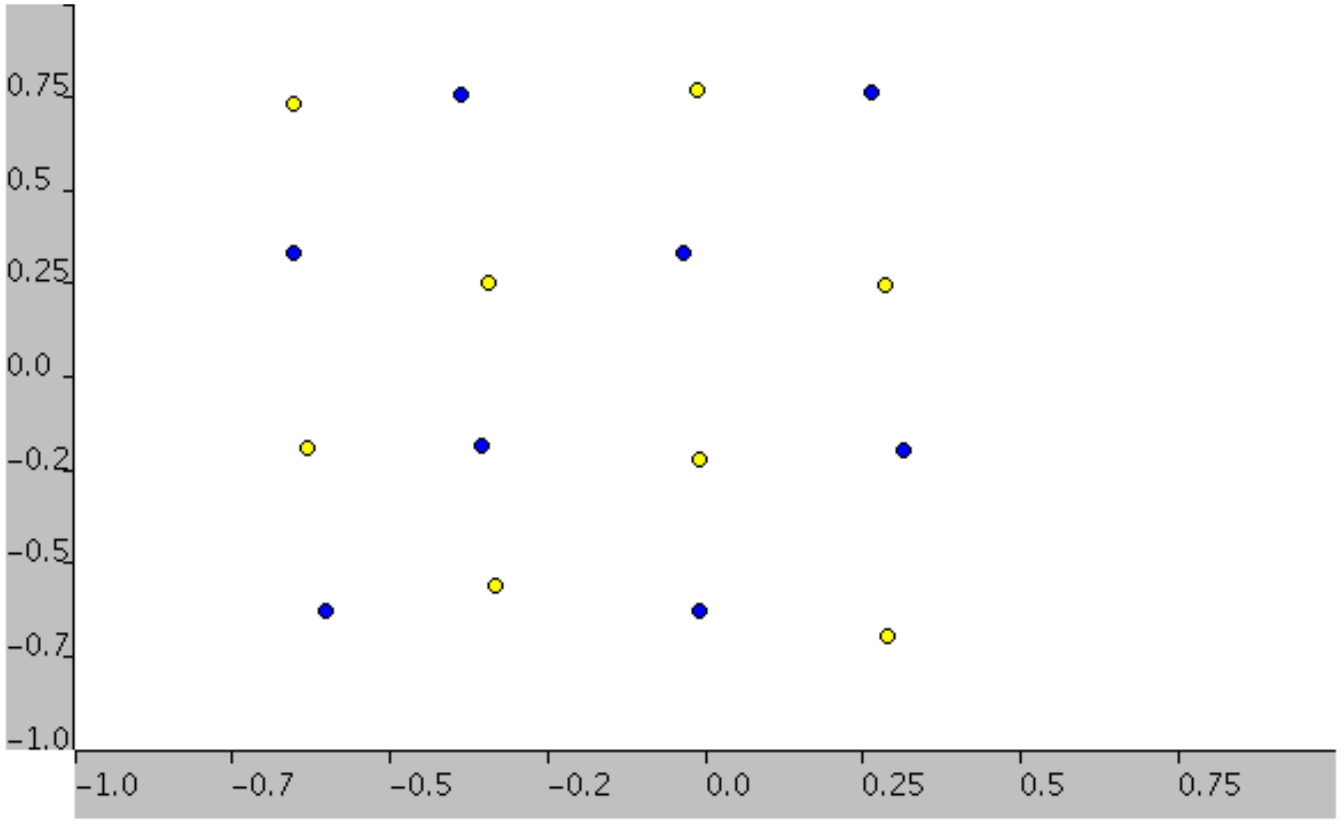
An XOR-like Problem



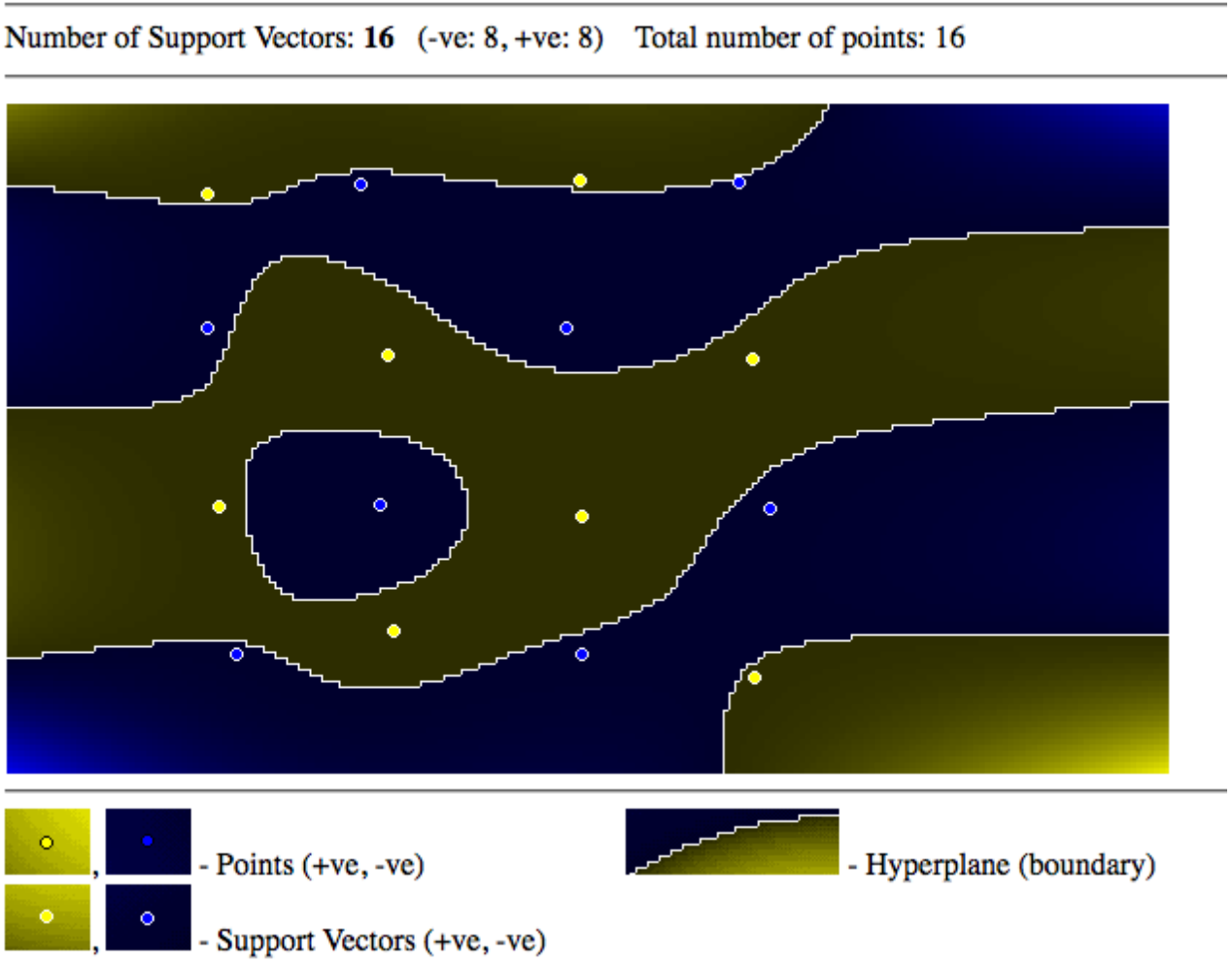
An XOR-like Problem



An Checkerboard Problem



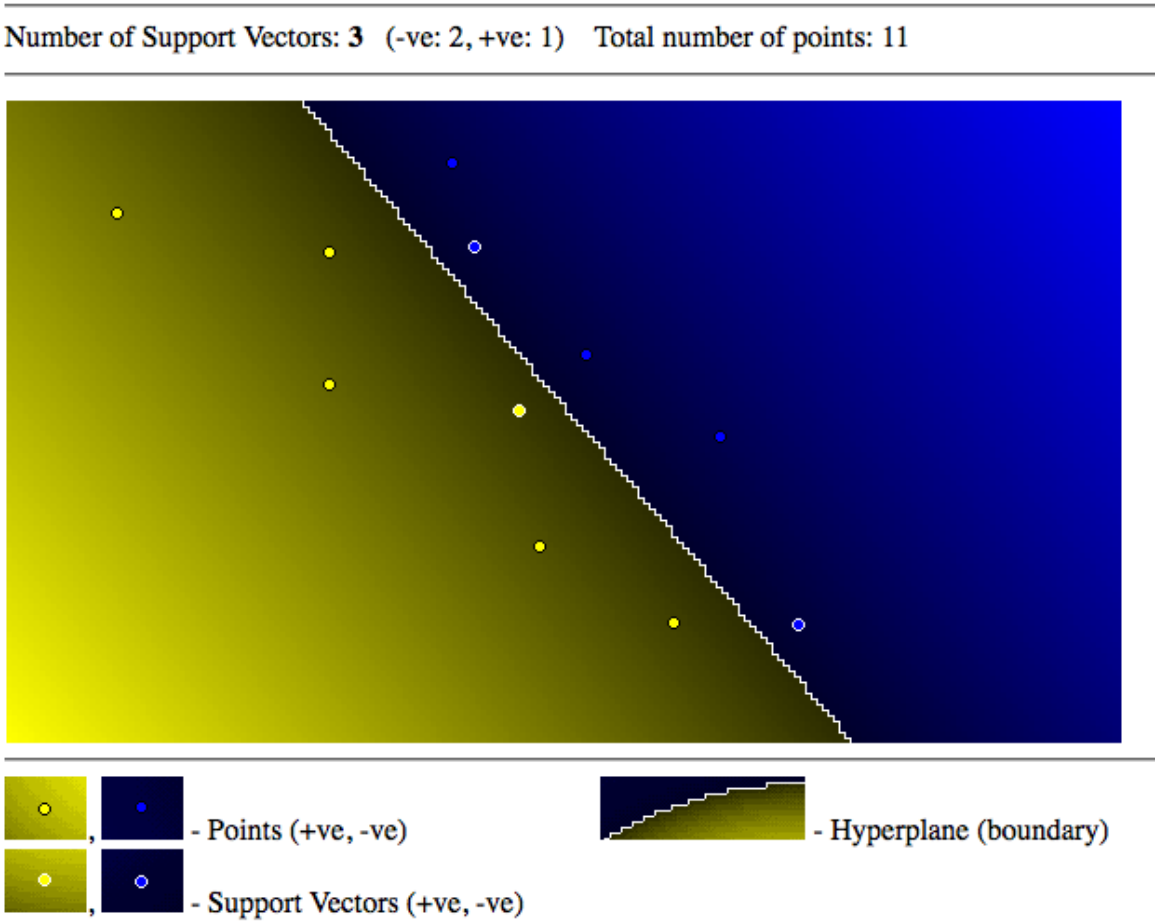
An Checkerboard Problem



What are Support Vectors?

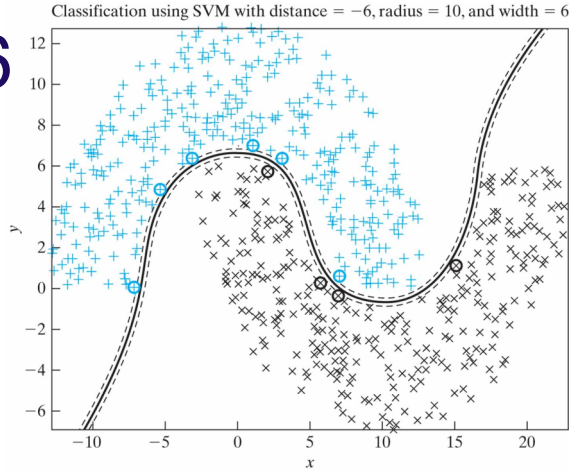
- Support vectors are the points that really count in determining the boundary.
- Support vectors are implicitly computed by the SVM algorithm.
- Other points are, in some sense, superflous.

Support Vectors (3) Circled

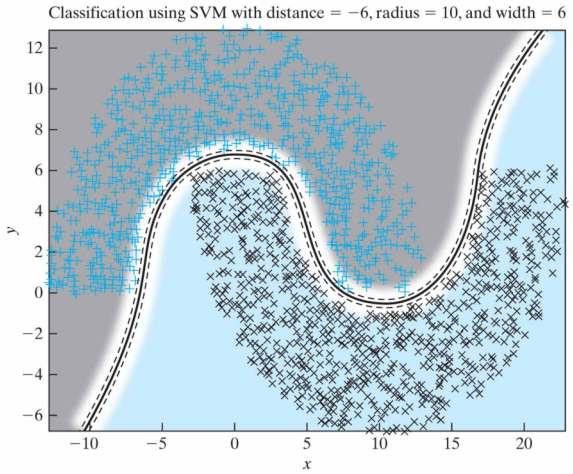


SV's of Double-Moons Problems

d = -6

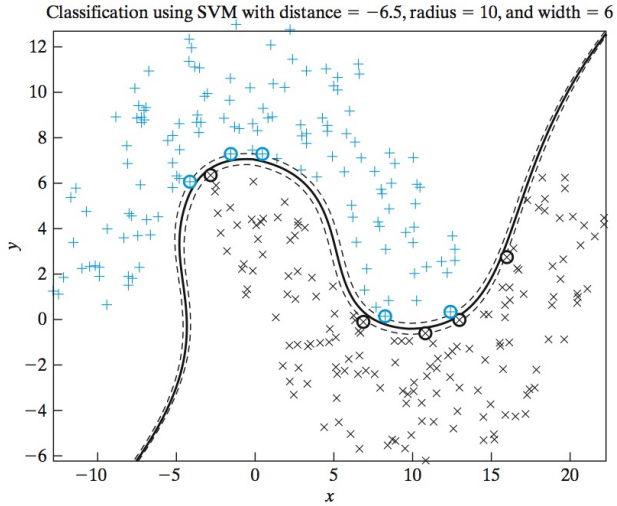


(a) Training result

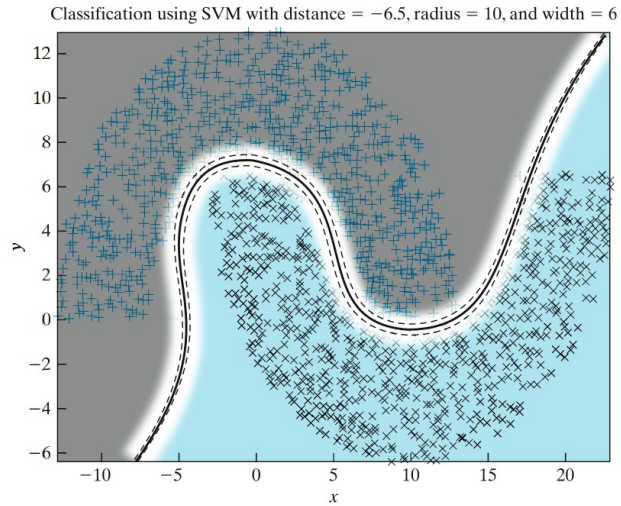


(b) Testing result

d = -6.5



(a) Training result



(b) Testing result

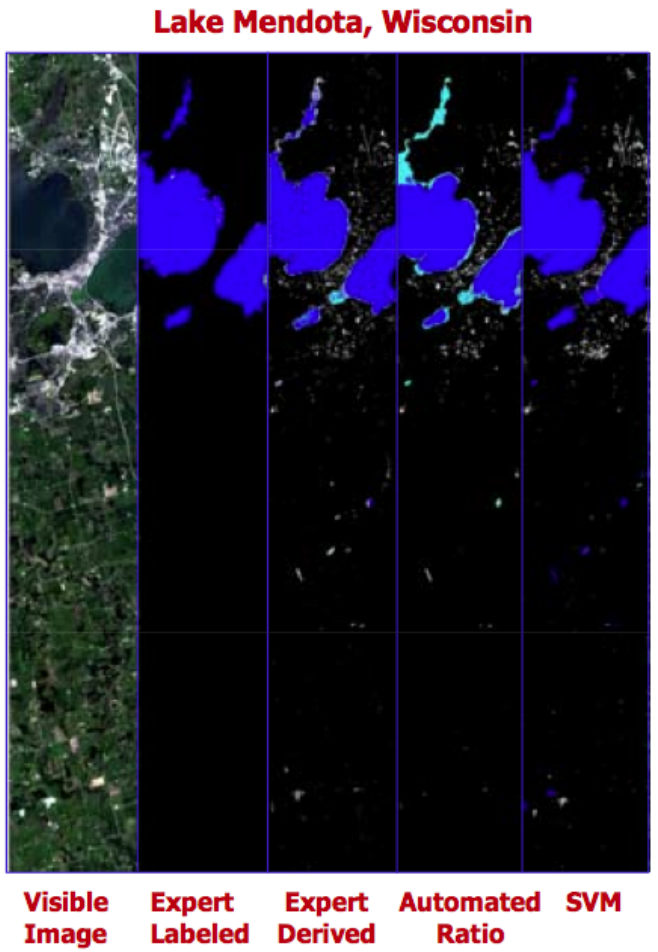
Real-Life Application: Identifying features in MISR satellite images

- Identify areas of land cover (land, ice, water, snow) in a scene
- Two methods:
  - Scientist manually derived
  - Support Vector Machine (SVM)

accuracy

Classifier	Expert Derived	SVM
cloud	45.7%	58.5%
ice	60.1%	80.4%
land	93.6%	94.0%
snow	63.5%	71.6%
water	84.2%	89.1%
unclassified	45.7%	

Courtesy of Steve Chien of NASA/JPL



Music Classification Based on Spectra

Ming-jen Wang & Chia-Jiu Wang

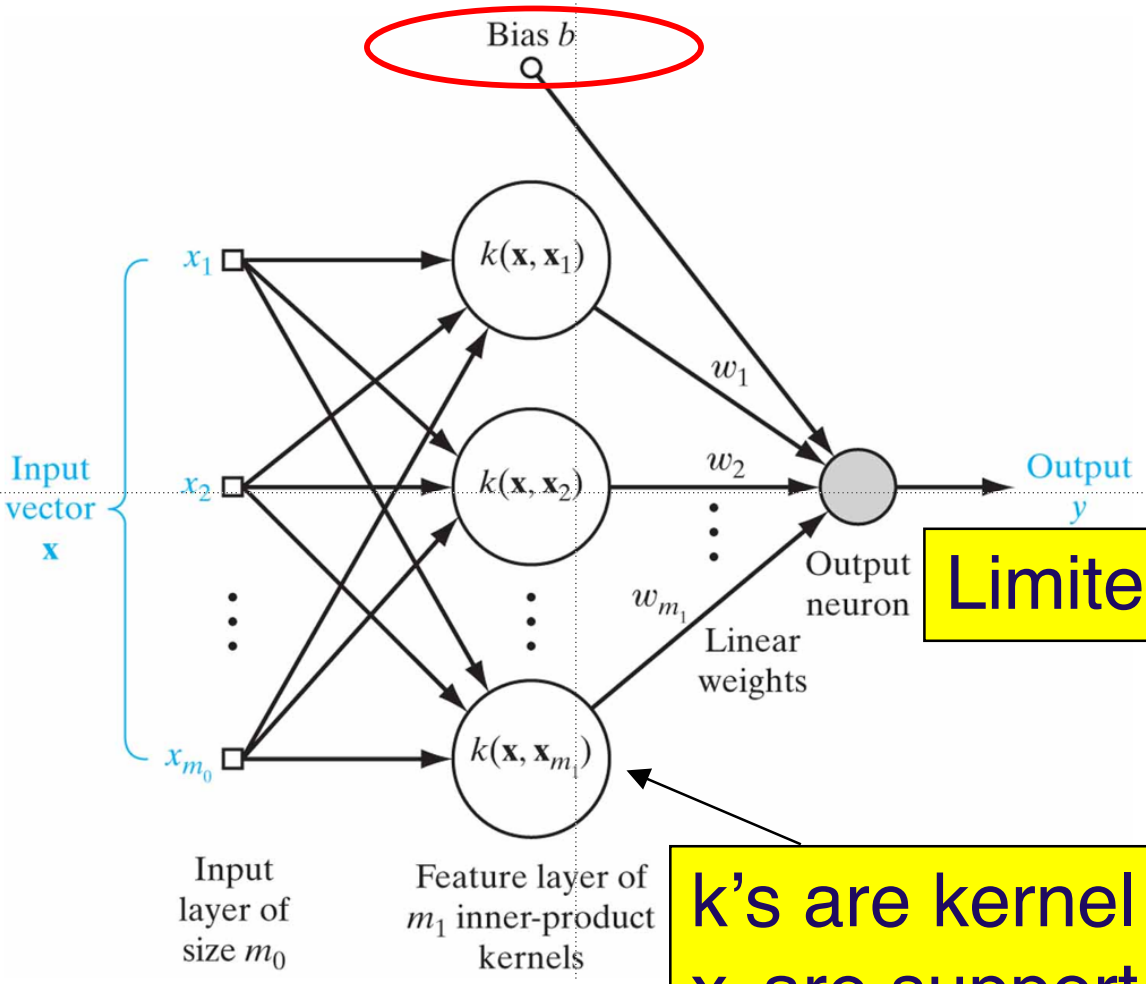
10,000 training samples, 30,000 testing

Accuracy	Classical	Jazz	Rock
Gaussian Classifier [7]	86%	38%	49%
Neural Network [8]	97%	n/a	93%
SVM (individual sample)	98%	79.5%	95%
SVM (sample-set)	99%	96%	94%

Use of Support Vectors in the Generated Machine

- Thinking back to the perceptron algorithm, starting with weights 0, the final weights are a **combination of training inputs**.
- The same is true of SVM, except that the **combination is based only on support vectors**.
- Which vectors are support vectors depends on the chosen kernel functions.

SVM Diagram



Limiter

k's are kernel functions
x_i are support vectors

Some Kernel Function Possibilities

TABLE 6.1 Summary of Mercer Kernels

Type of support vector machine	Mercer kernel $k(\mathbf{x}, \mathbf{x}_i), i = 1, 2, ..., N$	Comments
Polynomial learning machine	$(\mathbf{x}^T\mathbf{x}_i + 1)^p$	Power p is specified *a priori* by the user
Radial-basis-function network	$\exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \mathbf{x}_i\|^2\right)$	The width σ^2, common to all the kernels, is specified *a priori* by the user
Two-layer perceptron	$\tanh(\beta_0\mathbf{x}^T\mathbf{x}_i + \beta_1)$	Mercer's theorem is satisfied only for some values of β_0 and β_1

Why the +1 in polynomial case?
This provides a rich collection of cross products between individual vector components.

Mercer Kernels

- A Mercer Kernel k is a kernel function that is factorizable into an **inner product** of the form

$$k(x, x_i) = \Phi(x)^T \cdot \Phi(x_i)$$

also written $\langle \Phi(x), \Phi(x_i) \rangle$

where Φ maps the input space into some appropriate vectors space having a distance metric (a Hilbert space), **the feature space**.

Dimensionality of Feature Space

- For polynomials, the feature space is finite-dimensional, being determined by the degree of the polynomial and dimensionality of the input space.

- For radial basis functions, the feature space is infinite-dimensional (think Taylor's series expansion).

As shown in the original paper

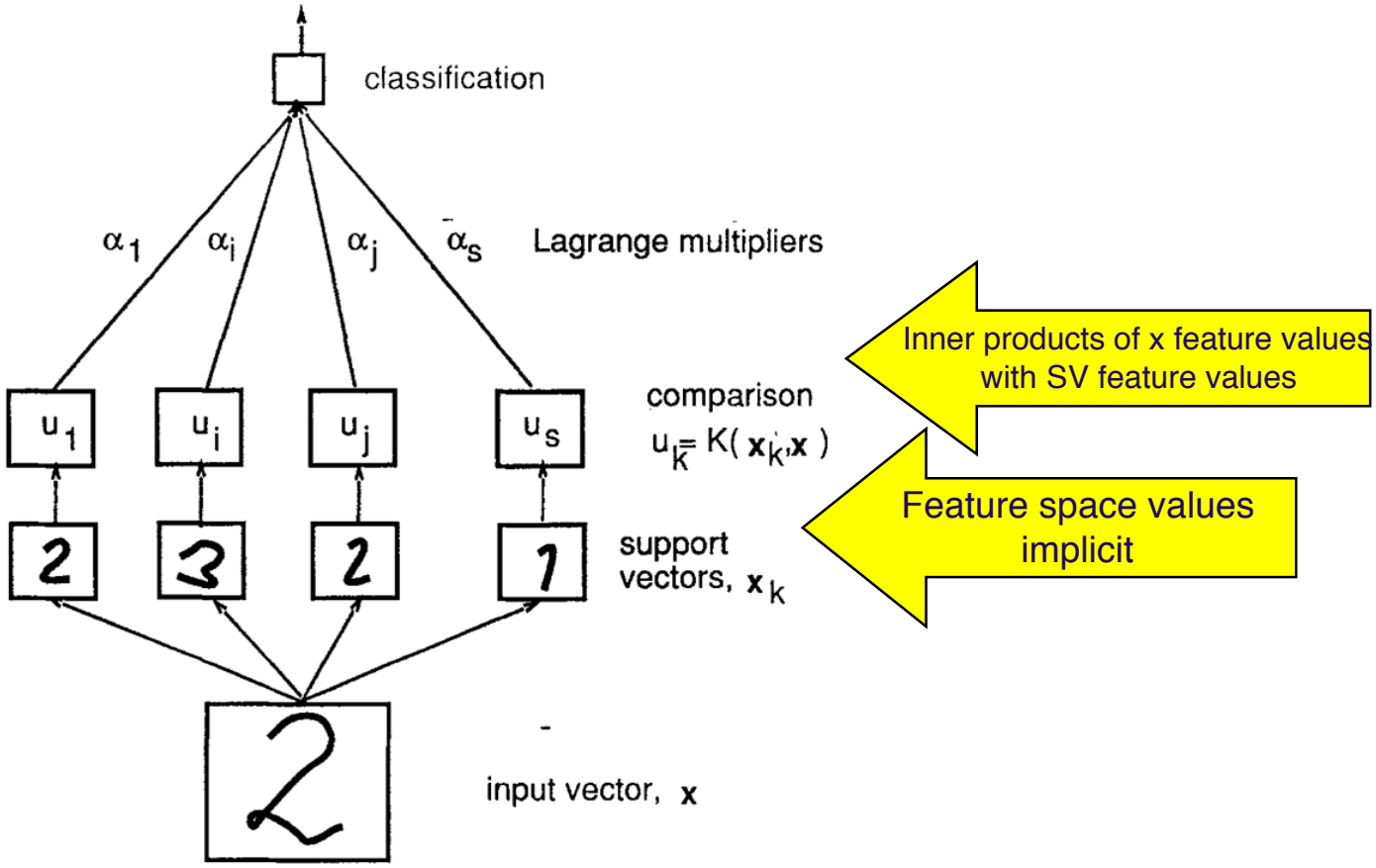


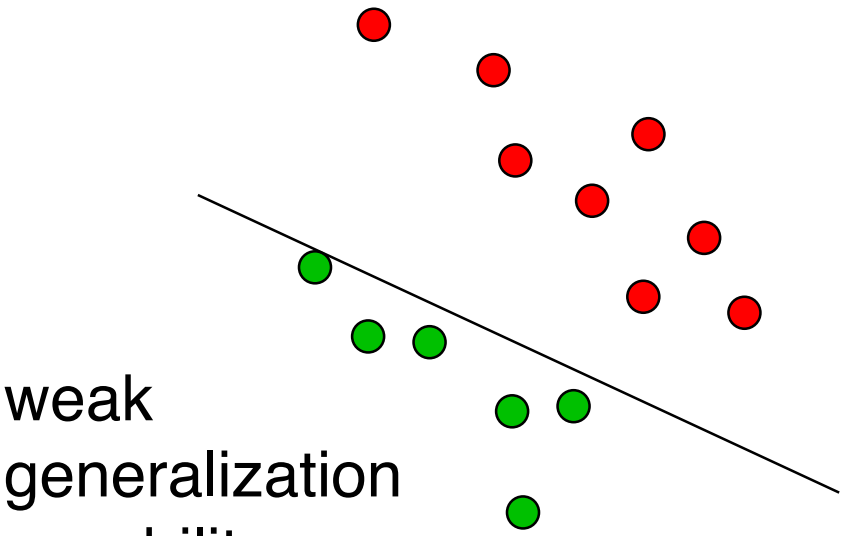
Figure 4. Classification of an unknown pattern by a support-vector network. The pattern is in input space compared to support vectors. The resulting values are non-linearly transformed. A linear function of these transformed values determine the output of the classifier.

Optimality Criteria

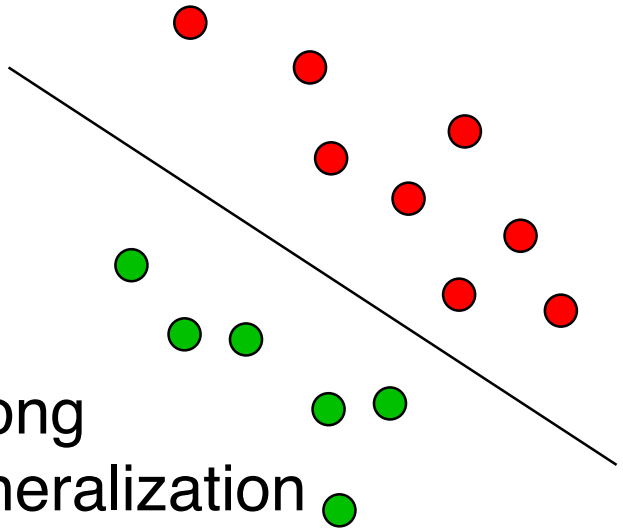
- An SVM uses linear classification in the feature space.
- So we look to linear classifiers to provide the definition of **optimality**.

Optimality?

- Not all separating hyperplanes are equal; some generalize better than others.



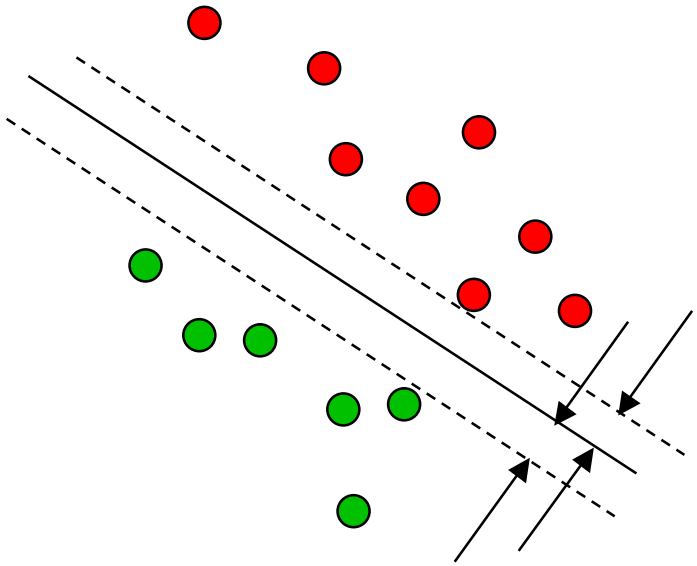
weak generalization capability



strong generalization capability

Optimality?

- An SVM tries to ensure the "**widest margins**":



Expressing Margins

- The weight vector **w** is the vector normal to the separating hyperplane. The equation for the hyperplane is **w x** = b, where the bias b determines the displacement of the hyperplane along the normal.

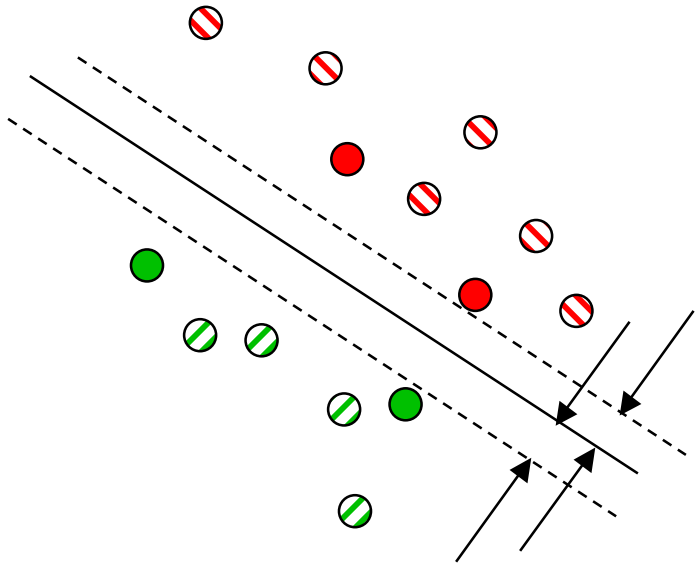
- To ensure the greatest margin, we'd like to **maximize** the sum of the distances to the separating hyperplane of all data input values x_j, subject to **constraints** of the form

$$y_j [x_j \mathbf{w} + b] \geq 1$$

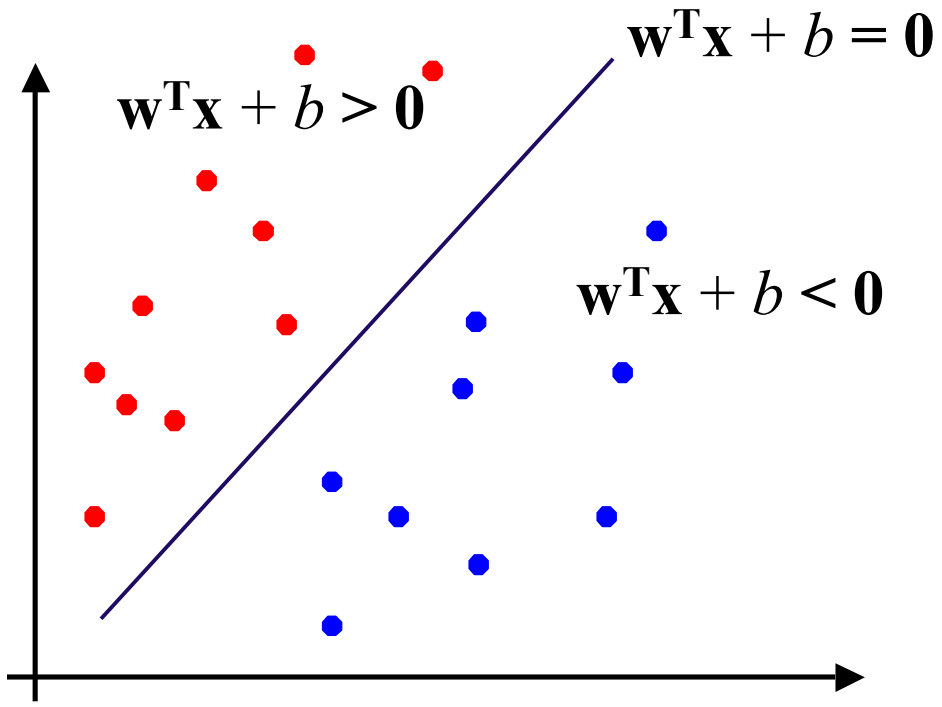
Here $y_j \in \{+1, -1\}$ is the **desired category** of the point x_j.

Support Vectors Again

- To **maximize** the distance of the points to the hyperplane, we don't need to worry about all points, but only the ***boundary*** points.
- These points are called "**support vectors**", shown as solid below.



Linear Classification

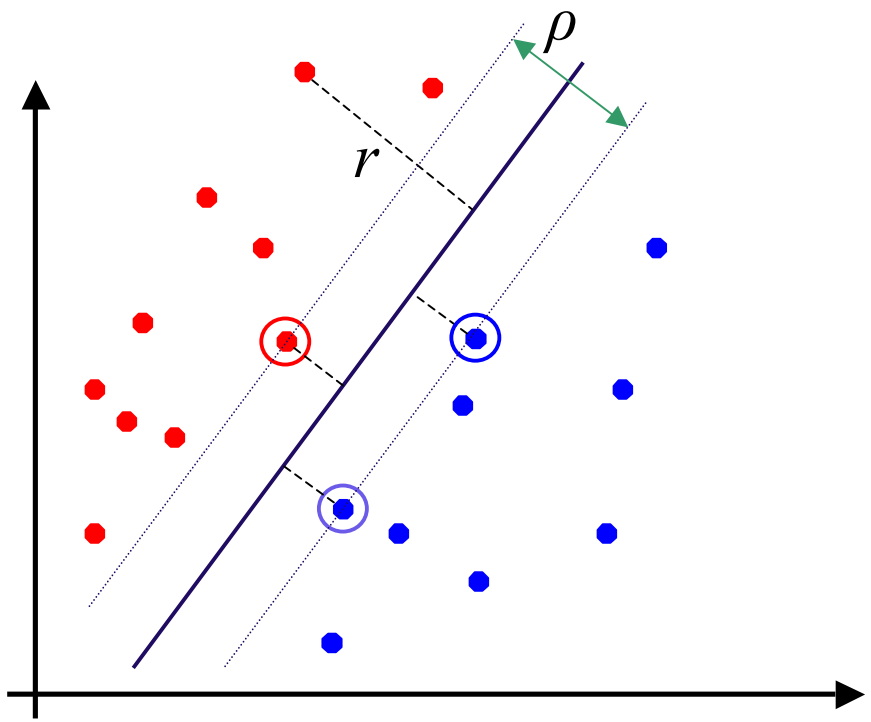


Classifier

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$$

Classification Margin

- Distance from example $\mathbf{x}_i$ to the separator is $r = \dfrac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are ***support vectors***.
- ***Margin*** ρ (TBD) of the separator is the distance between support vectors.



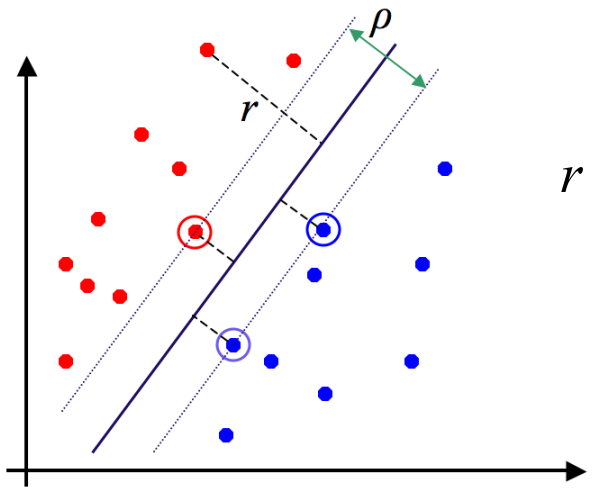
Hard vs. Soft Margin

- In the separable case, no points lie within the margin on either side of the hyperplane.
- In the non-separable case, points can lie within, or on the “wrong” side of the hyperplane. This “soft margin” case is deferred to later.

Classification by Distance

- Let training set $\{(\mathbf{x}_i, y_i)\}_{i=1..n}$, $\mathbf{x}_i \in \mathbf{R}^d$, $y_i \in \{-1, 1\}$ be separated by a hyperplane with **margin** ρ.
- Then for each training example $(\mathbf{x}_i, y_i)$:

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_i + b &\leq -\rho/2 \quad \text{if } y_i = -1 \\ \mathbf{w}^T\mathbf{x}_i + b &\geq \rho/2 \quad \text{if } y_i = 1 \end{aligned} \quad \Leftrightarrow \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq \rho/2$$



$$r = \frac{\mathbf{w}^T\mathbf{x}_i + b}{\|\mathbf{w}\|}$$ distance from $\mathbf{x}_i$ to separator

Margin in Terms of Weights

- For the special case of **support vectors**, the inequality becomes an **equality**:

$$y_i(\mathbf{w}^{\mathbf{T}}\mathbf{x}_s + b) = \rho/2$$

- After rescaling **w** and *b* by *ρ/2* in the equality, we obtain that distance between each support vector $\mathbf{x}_s$ and the hyperplane is

$$r = \frac{y_s(\mathbf{w}^T\mathbf{x}_s + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

- Then the margin can be expressed through (rescaled) **w** and b as: $\rho = 2r = \frac{2}{\|\mathbf{w}\|}$

Margin Maximization

- To maximize margin ρ:

Find $\mathbf{w}$ and b such that

$$\rho = \frac{2}{\|\mathbf{w}\|} \text{ is } \textbf{\textit{maximized}}$$

subject to for all $(\mathbf{x}_i, y_i)$, i=1..n : $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Equivalently:

Find $\mathbf{w}$ and b such that

$\mathbf{P}(\mathbf{w}) = \|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w}$ is ***minimized***

subject to for all $(\mathbf{x}_i, y_i)$, i=1..n : $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Convex Optimization Problem

- The problem of finding the weights has been reduced to that of minimizing a **quadratic function** subject to linear inequality constraints.

Find $\mathbf{w}$ and b such that
$\mathbf{P}(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i\,(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- There exist solver programs for this class of problems, known as “quadratic program” solvers.

Solving by Dualization

- The **dual** problem of the original is:

dual

Find $\alpha_1 \ldots \alpha_n$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is **maximized** and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

- The α_i are known as **Lagrange multipliers**.

primal

Find $\mathbf{w}$ and b such that
$\boldsymbol{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Dual to Primal

- Given a solution $\alpha_1 \ldots \alpha_n$ to the dual problem, solution to the primal is determined by:

$$\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x}_i \qquad b = y_k - \Sigma \alpha_i y_i \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_k \quad \text{for any } \alpha_k > 0$$

- Each **non-zero** α_i indicates that corresponding $\mathbf{x}_i$ is a **support vector**.
- Then the classifying function is:

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

- This says we don't need to compute **w** explicitly (called the "kernel trick").
- Notice that *f* relies on an ***inner product*** between the test point **x** and the support vectors $\mathbf{x}_i$. (In the general case, this inner product occurs in feature space, where $\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$ is replaced with $k(\mathbf{x}_i^{\mathrm{T}}, \mathbf{x}_j)$.)

- Solving the dual optimization problem involved computing the inner products $\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$ between all training points, called the **Gram matrix**.

Non-Linear Case

- Dual problem formulation, where k are the kernel functions:

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

- The classifier then uses the k values in place of inner products:

$$f(\mathbf{x}) = \Sigma\alpha_i y_i\ k(\mathbf{x}_i, \mathbf{x}_j) + b$$

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

Gram Matrix

- The values of $k(\mathbf{x}_i, \mathbf{x}_j)$ for each i, j are tabulated in a matrix called the “Gram matrix”. (For the linear case, k is just the ordinary inner product.)
- [This was the matrix Φ in the case of RBF networks.]
- The Gram matrix is thus the main input to the quadratic optimizer.

Worked Example: xor (see Haykin, sec. 6.6)

$\mathbf{x}_i$	y_i
(-1, -1)	-1
(-1, 1)	1
(1, -1)	1
(1, 1)	-1

Chosen kernel: 2nd degree polynomial:

$$k(x, x_i) = (1 + x^T x_i)^2$$

$$= 1 + x_1^2 x_{i1}^2 + 2 x_1 x_{i1} x_2 x_{i2} + x_2^2 x_{i2}^2 + 2 x_1 x_{i1} + 2 x_2 x_{i2}$$

The feature space thus has dimension 6.

Factored kernel:

$$\varphi(x) = [1, x_1^2, \sqrt{2} x_1 x_2, x_2^2, \sqrt{2} x_1, \sqrt{2} x_2]^T$$

xor: Primal and Dual Problems

Primal: maximize $P(w) = \frac{1}{2} w^T w$ subject to constraints

Dual: minimize $Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$ subject to constraints

Substituting from xor table: Maximize

$$Q(\alpha) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \tfrac{1}{2}(9\alpha_1^2 - 2\alpha_1\alpha_2 - 2\alpha_1\alpha_3 + 2\alpha_1 + 9\alpha_2^2 + 2\alpha_2\alpha_3 - 2\alpha_2\alpha_4 + 9\alpha_3^2 - 2\alpha_3\alpha_4 + 9\alpha_2^4)$$

Optimizing $Q(\alpha)$ gives 4 equations in unknowns $\alpha_1, \alpha_2, \alpha_3, \alpha_4$:

$$\begin{aligned} 9\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4 &= 1 \\ -\alpha_1 + 9\alpha_2 + \alpha_3 - \alpha_4 &= 1 \\ -\alpha_1 + \alpha_2 + 9\alpha_3 - \alpha_4 &= 1 \\ \alpha_1 - \alpha_2 - \alpha_3 + 9\alpha_4 &= 1 \end{aligned}$$

which has the solution:

$$\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \frac{1}{8}$$

$$Q(\alpha) = \frac{1}{4}$$

(so all input vectors are support vectors)

xor: Weight Parameters

solution:

$$\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \frac{1}{8}$$

$$Q(\alpha) = \frac{1}{4}$$

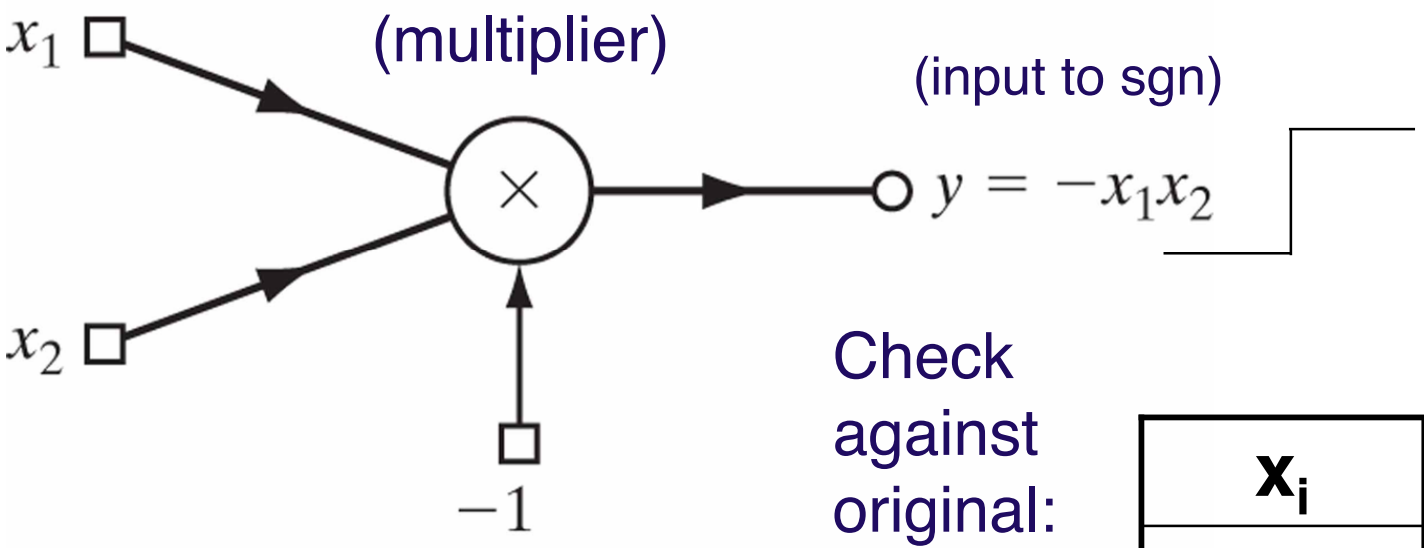
$$w_o = \sum_{i=1}^{N} \alpha_i y_i \varphi(x_i) \quad \text{where} \quad \varphi(x) = [1, x_1^2, \sqrt{2}x_1 x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2]^T$$

Then substitute for the values α_i, y_i, x_i i = 1..4, to get w = [0, 0, -1, 0, 0, 0].

The first element, which multiplies 1, is the bias term (0).

The classifier is thus sgn(- $x_1 x_2$).

Final SVM for xor (Haykin Figure 6.6a)



Check against original:

$\mathbf{x_i}$	y_i
(-1, -1)	-1
(-1, 1)	1
(1, -1)	1
(1, 1)	-1

Kernel Trick, applied to xor

$$k(x, x_i) = (1 + x^T x_i)^2$$

$$= 1 + x_1^2 x_{i1}^2 + 2 x_1 x_{i1} x_2 x_{i2} + x_2^2 x_{i2}^2 + 2 x_1 x_{i1} + 2 x_2 x_{i2}$$

$$f(\mathbf{x}) = \Sigma \alpha_i y_i \, k(\mathbf{x}_i, \mathbf{x}_j) + b$$

$$\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \tfrac{1}{8}$$

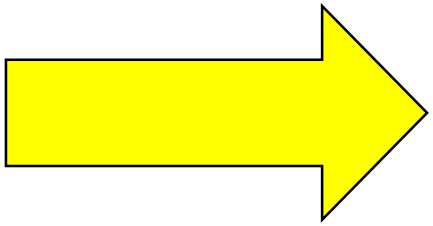
$$y_1 = y_4 = -1, \; y_2 = y_3 = 1$$

Gram Matrix

$$k(x, x_i) = (1 + x^T x_i)^2$$

$$= 1 + x_1^2 x_{i1}^2 + 2 x_1 x_{i1} x_2 x_{i2} + x_2^2 x_{i2}^2 + 2 x_1 x_{i1} + 2 x_2 x_{i2}$$

$\mathbf{x}_i$
(-1, -1)
(-1, 1)
(1, -1)
(1, 1)



9	1	1	1
1	9	1	1
1	1	9	1
1	1	1	9

Check (with b = 0): $f(\mathbf{x}) = \Sigma \alpha_i y_i \, k(\mathbf{x}_i, \mathbf{x}_j) + b$

Comparisons from original paper Digit recognition problem

6.2. Experiments with Digit Recognition

Our experiments for constructing support-vector networks make use of two different databases for bit-mapped digit recognition, a small and a large database. The small one is a US Postal Service database that contains 7,300 training patterns and 2,000 test patterns. The resolution of the database is 16×16 pixels, and some typical examples are shown in Fig. 6. On this database we report experimental research with polynomials of various degree.

The large database consists of 60,000 training and 10,000 test patterns, and is a 50-50 mixture of the NIST[7] training and test sets. The resolution of these patterns is 28×28 yielding an input dimensionality of 784. On this database we have only constructed a 4th degree polynomial classifier. The performance of this classifier is compared to other types of learning machines that took part in a benchmark study (Bottou, 1994).

In all our experiments ten separators, one for each class, are constructed. Each hypersurface makes use of the same dot product and pre-processing of the data. Classification of an unknown patterns is done according to the maximum output of these ten classifiers.

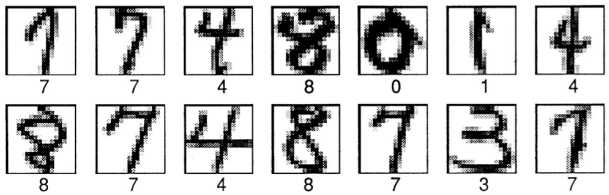


Figure 6. Examples of patterns with labels from the US Postal Service digit database.

Comparisons from original paper Digit recognition problem

Classifier	Raw error, %
Human performance	2.5
Decision tree, CART	17
Decision tree, C4.5	16
Best 2 layer neural network	6.6
Special architecture 5 layer network	5.1

Table 2. Results obtained for dot products of polynomials of various degree. The number of "support vectors" is a mean value per classifier.

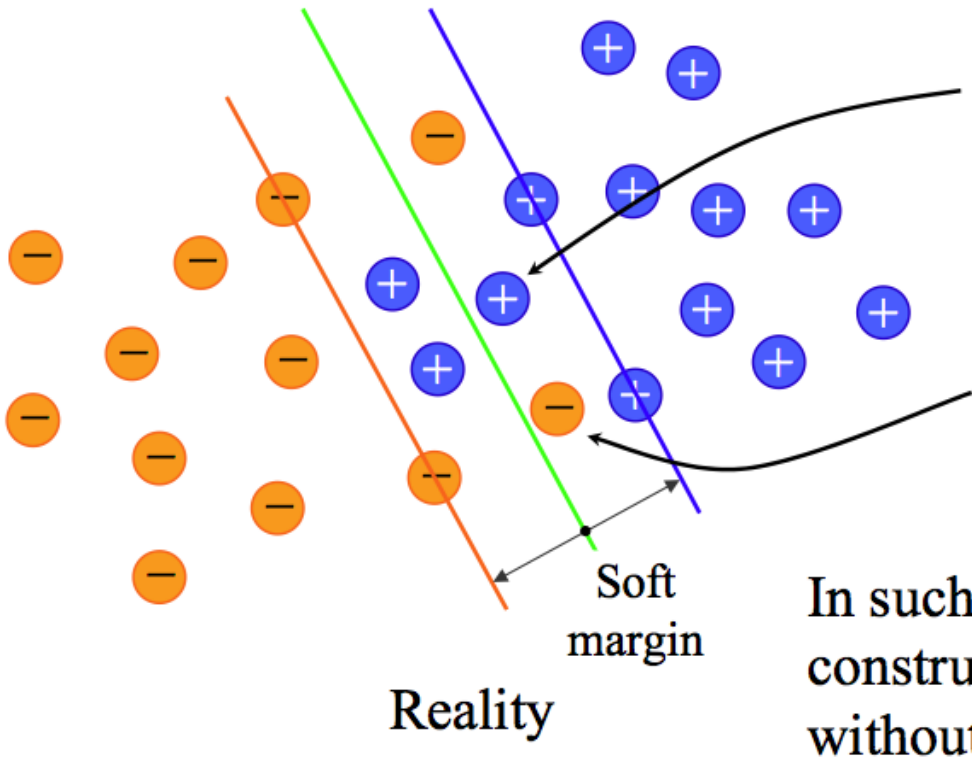
Degree of polynomial	Raw error, %	Support vectors	Dimensionality of feature space
1	12.0	200	256
2	4.7	127	~33000
3	4.4	148	$\sim 1 \times 10^6$
4	4.3	165	$\sim 1 \times 10^9$
5	4.3	175	$\sim 1 \times 10^{12}$
6	4.2	185	$\sim 1 \times 10^{14}$
7	4.3	190	$\sim 1 \times 10^{16}$

Soft-Margin Case

- There is no a priori guarantee that a separating hyperplane exists for a given data set and choice of kernel functions.

- Thus a robust SVM algorithm must be able to accommodate this possibility **without knowing in advance** whether the data will be separable in feature space.

Two Possibilities wrt Margin



(a) A data point falls inside the region of separation but on the right side of the decision surface.

(b) A data point falls on the wrong side of the decision surface.

In such a case, it is impossible to construct a separating hyperplane without misclassifications

Constraints

- Separable Case: Positive and negative points lie outside the margin on the correct sides.

$$y_i(\vec{w}_0^T \vec{x}_i + b_0) \geq 1 \qquad \text{for } i = 1,2,\ldots,N$$

- Soft-Margin Case: Introduce Slack Variables $\xi_i \geq 0$

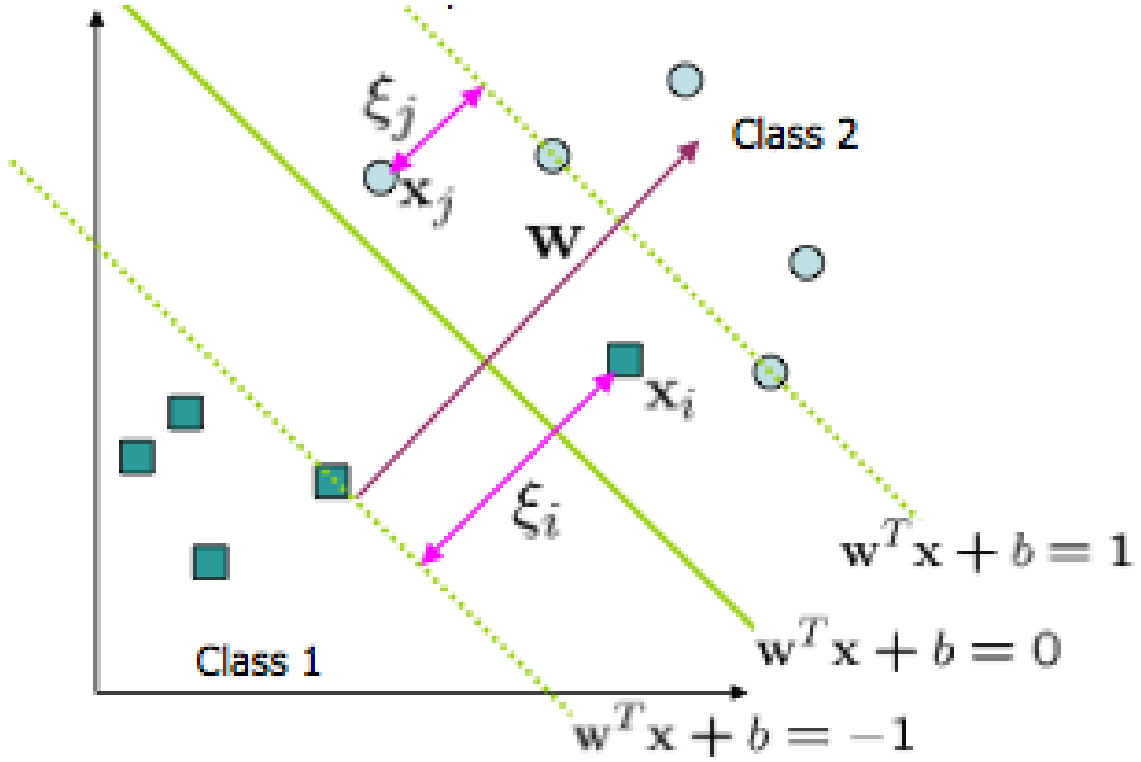
$$y_i(\vec{w}^T \vec{x}_i + b) \geq 1 - \xi_i \qquad \text{for } i = 1,\ldots,N$$

ξ_i is called a slack variable measuring the deviation of a data point from the ideal separation

Slack Variables

$$y_i(\vec{w}^T\vec{x}_i + b) \geq 1 - \xi_i$$

$\xi_i \leq 0$	implies correct, outside margin
$0 < \xi_i \leq 1$	implies correct, inside margin
$\xi_i > 1$	implies incorrect



Indicator Function I

$\xi_i \leq 0$	implies correct, outside margin
$0 < \xi_i \leq 1$	implies correct, inside margin
$\xi_i > 1$	implies incorrect classification

- Define $I(\xi) = 0$ if $\xi \leq 0$, 1 otherwise
- Misclassification error **for the entire training sample** is

$$\Phi(\xi) = \Sigma\ I(\xi_i - 1) \quad \text{(summed over i = 1 .. N)}$$

- We want to minimize $\Phi(\xi)$ ***in addition to*** the margin $\mathbf{w}^T\mathbf{w}$.

Minimization Heuristic

- The joint minimization of Φ and $\mathbf{w}^T\mathbf{w}$ has been shown to be NP-complete (i.e. very time-consuming).

- So instead, an approximate Φ is used:

  $\Phi(\xi) = \Sigma\xi_i$ instead of $\Phi(\xi) = \Sigma\ \mathtt{I}(\xi_i - 1)$

- The new objective to be minimized is then

  $\Phi(w, \xi) = \mathbf{w}^T\mathbf{w} + C\ \Sigma\xi_i$

  where C is user-specified weighting constant.

New Optimization Problem

- Minimize

$$\Phi(w, \xi) = \mathbf{w}^T\mathbf{w} + C \Sigma\xi_i$$

wrt w and ξ, subject to constraints:

$$\xi_i \geq 0$$

$$y_i(\vec{w}^T \vec{x}_i + b) \geq 1 - \xi_i \qquad \text{for } i = 1,\ldots,N$$

Solving the Optimization Problem

- As before, the optimization is solved by converting to the dual.

Dual: minimize $$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

- But now there are more constraints:
  - $\Sigma\ \alpha_i\, y_i = 0$
  - $0 \leq \alpha_i \leq C$
    (before there was no upper bound on α_i)

C Tradeoff

- Minimize

$$\Phi(w, \xi) = \mathbf{w}^T\mathbf{w} + C \Sigma\xi_i$$

- The larger C is, the more the algorithm tries to fit the data without error (smaller margin).

- Smaller C allows more misclassifications.