Support Vector Machines (SVMs), Part II

CSCI-P556 Applied Machine Learning
Lecture 19

D.S. Williamson

Agenda and Learning Outcomes

Today's Topics

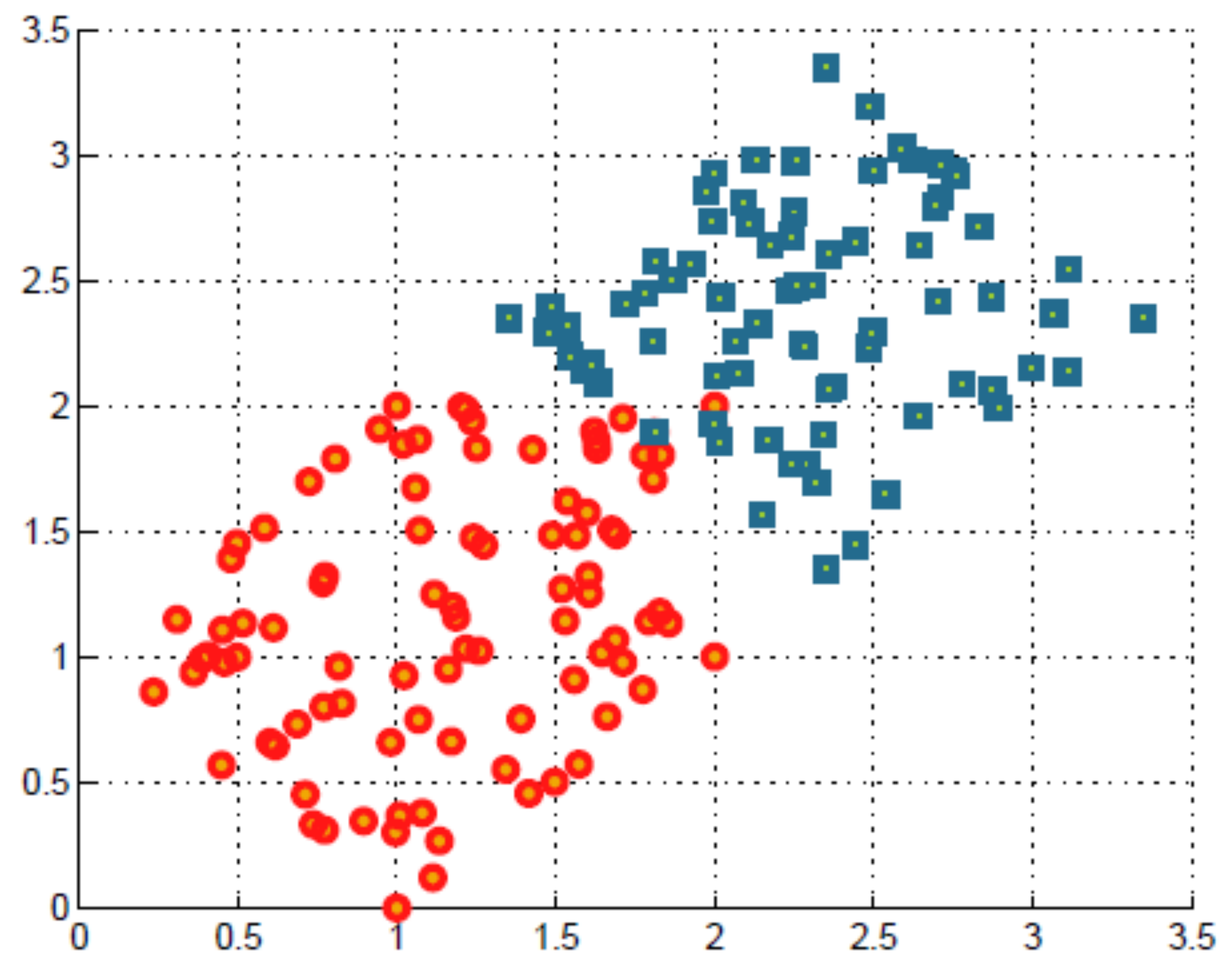
- **Topics**:
  - Support Vector Machines
    - Soft margins
    - Non-linearly separable data
- **Announcements**
  - Quiz #2 today
  - Project proposal comments released
  - HW#3 posted tonight

So far…

- We demonstrated that we prefer to have linear classifiers with large margin.
- We formulated the problem of ***finding the maximum margin linear classifier*** as a ***quadratic optimization*** problem
- This problem can be solved by solving its ***dual problem***, and efficient QP algorithms are available.

Inseparable data set

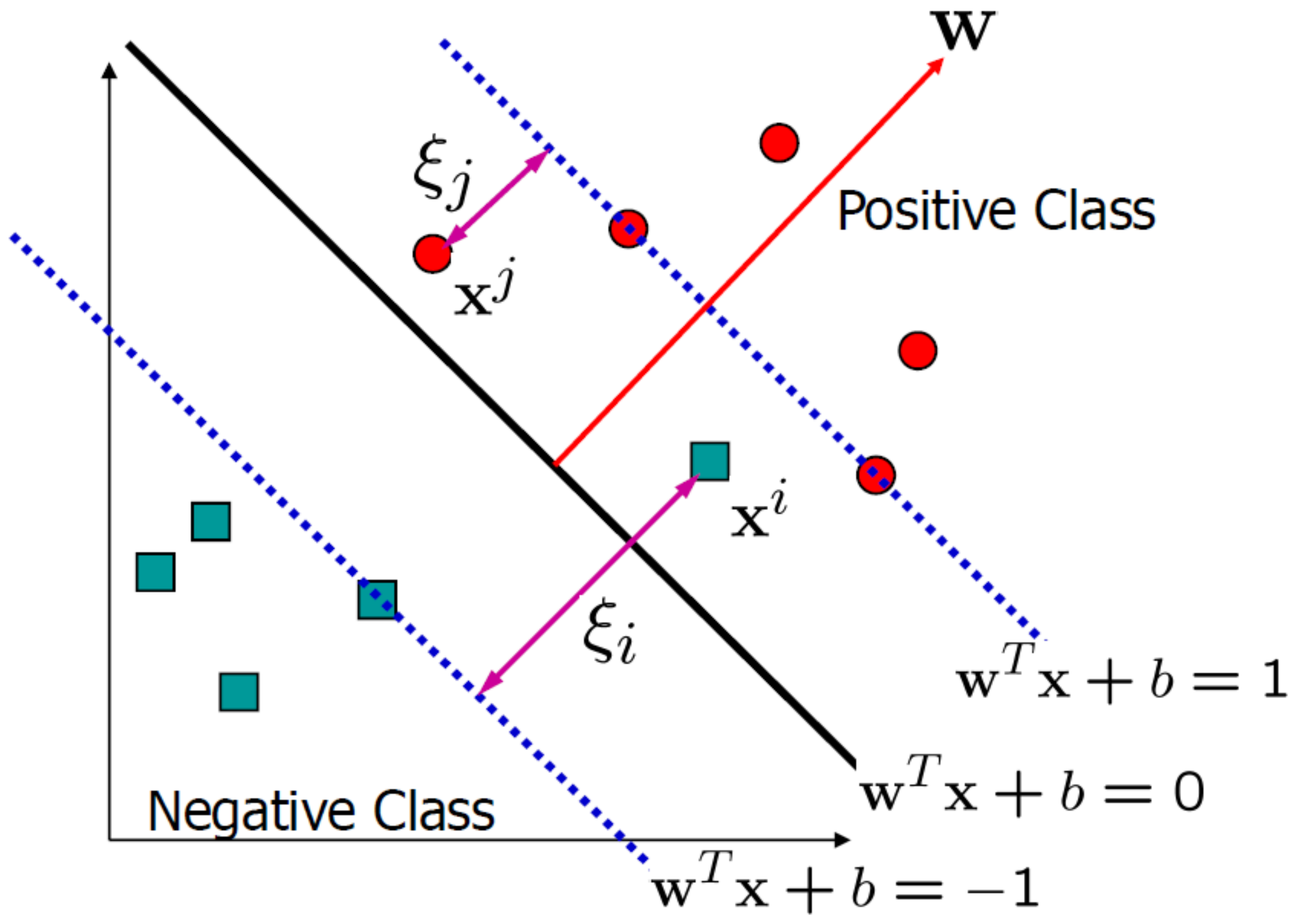
- So far we have assumed that the data is ***linearly separable*** and the formulation derived is the ***hard-margin SVM.***
- But what if there are ***a few-bad examples?***



This is an example of an *inseparable* data set.

Soft Margin - SVMs

- Modify the objective to ***minimize error of misclassified*** points with respect to the *margin*
  - In other words, give some ***slack*** to the margin for the points that cross the boundary
  - Allow functional margins to be less than 1
  - But will charge a penalty, based on distance between point and preferred boundary (e.g. slack)


$\mathbf{w}$
ξ_j
$\mathbf{x}^j$
Positive Class
$\mathbf{x}^i$
ξ_i
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = 0$
Negative Class
$\mathbf{w}^T\mathbf{x} + b = -1$

Soft-Margin Maximization

- Introduce **slack variables** to allow functional margins to be smaller than 1
- Parameter *c* controls the tradeoff between maximizing the margin and fitting the training example
  - c is chosen by the user
  - High values indicate high confidence in the quality of the training sample
  - Small values, training data is considered noisy

Slack Variables

$$\min_{\mathbf{w},b,\xi_i} \frac{1}{2}\|\mathbf{w}\|^2 + c\sum_i \xi_i$$

Subject to: $y^i(\mathbf{w}\cdot\mathbf{x}^i + b) \geq 1 - \xi_i, i = 1,\ldots,N$

$$\xi_i \geq 0, i = 1,\ldots, N$$

Dual Formulation - Soft Margin

$$max \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i \alpha_j y^i y^j < \mathbf{x}^i \cdot \mathbf{x}^j >$$

Subject to: $\sum_{i=1}^{N} \alpha_i y^i = 0$

$0 \leq \alpha_i \boxed{\leq c}$ $\quad i=1,\dots, N$

- For the dual problem, the only difference is that the alpha’s are upper-bounded by c.
- We now also have support vectors for data that have functional margin less than one (in addition to those that equal 1), but there α_i’s will only equal c

<u>Solution</u>: Solve problem as before but with new constraint

support vectors (α_i>0)

$c > \alpha_i > 0$:	$y^i(w \cdot x^i + b) = 1$, i.e., $\xi_i = 0$
$\alpha_i = c$:	$y^i(w \cdot x^i + b) \leq 1$, i.e., $\xi_i \geq 0$

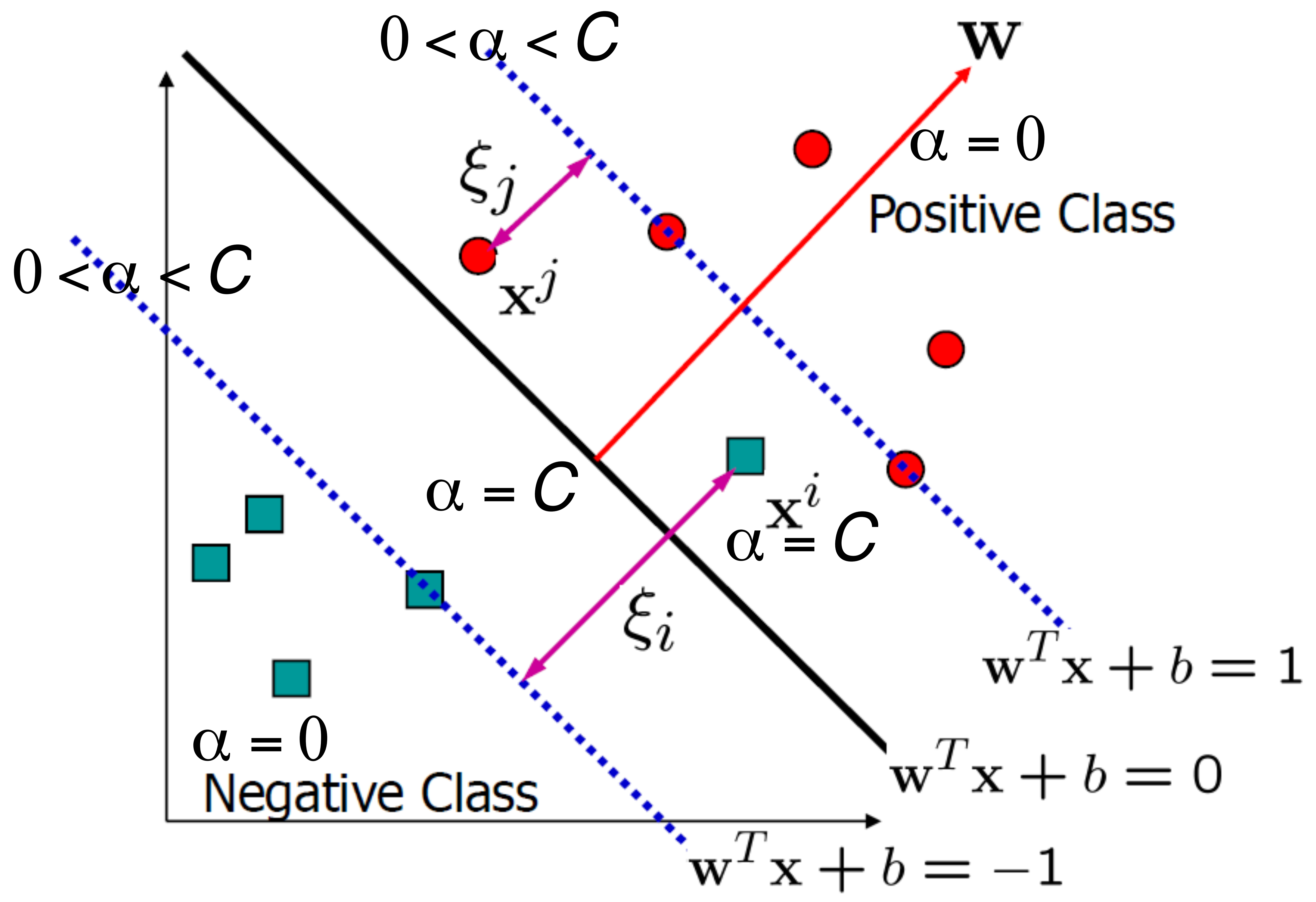
Samples within the margin

The optimal **w** can then be computed:

$$\mathbf{w} = \sum \alpha_i y^i \mathbf{x}^i$$

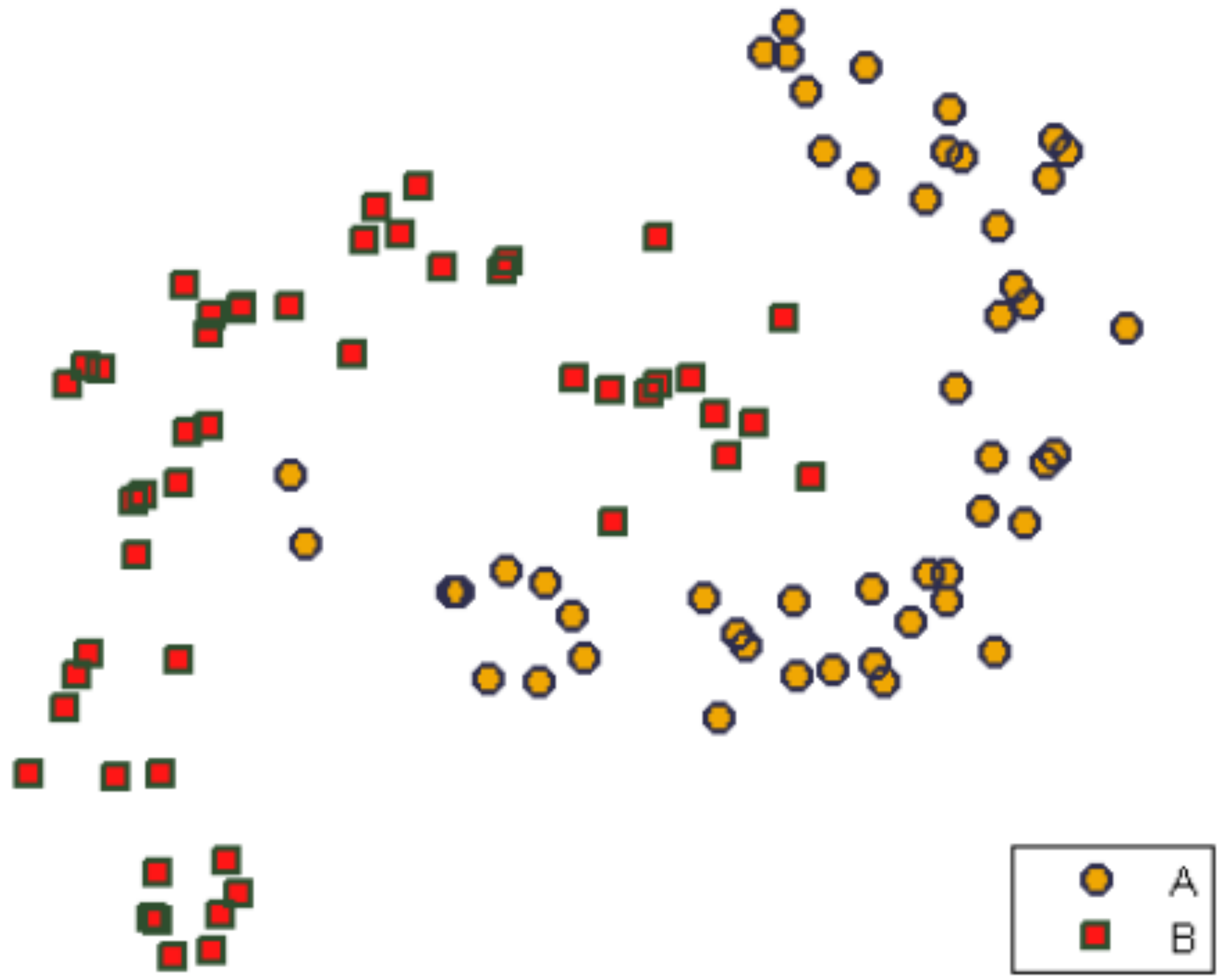
Soft Margin – Geometric View

Summarizing alpha values



Non Linear data?

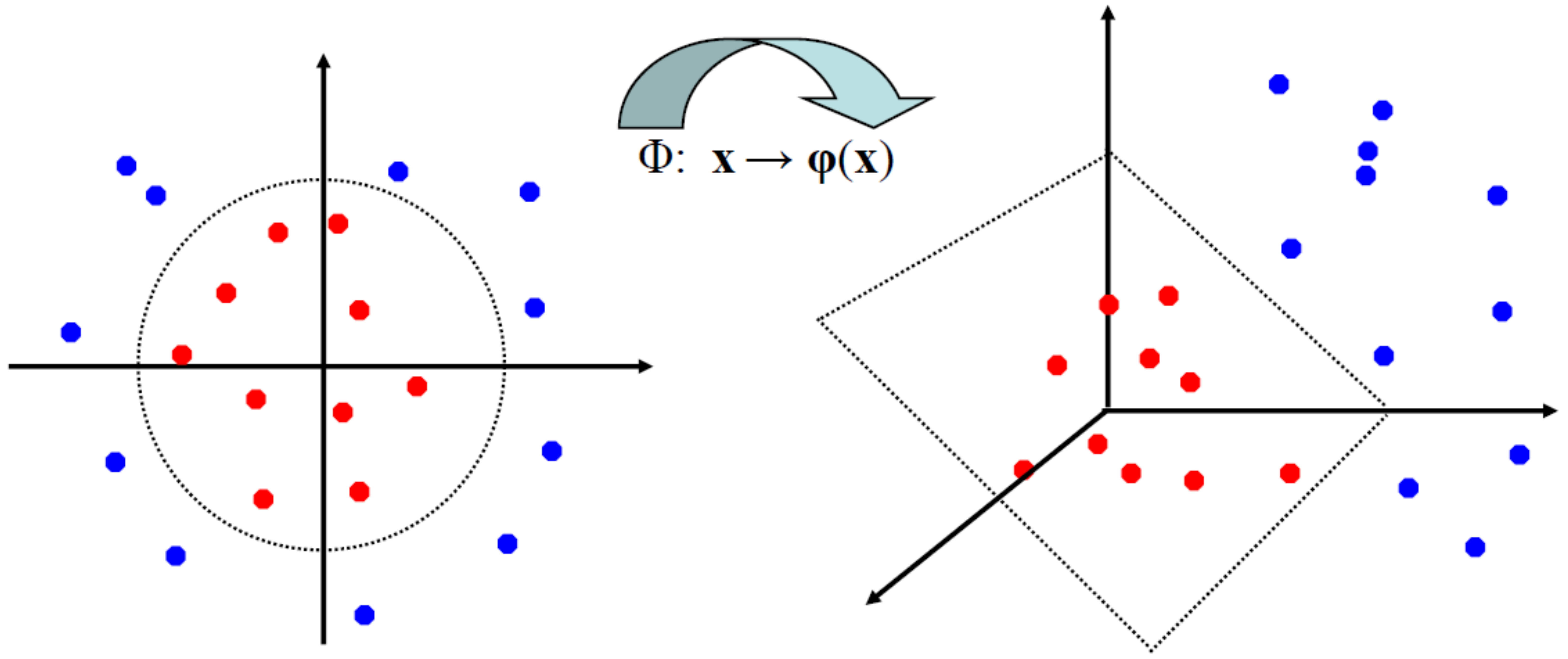
- So far, we have assumed ***linearly separable data.*** What if the data is non-linearly separable (e.g. linearly inseparable)?



Non-Linear SVMs: Feature spaces

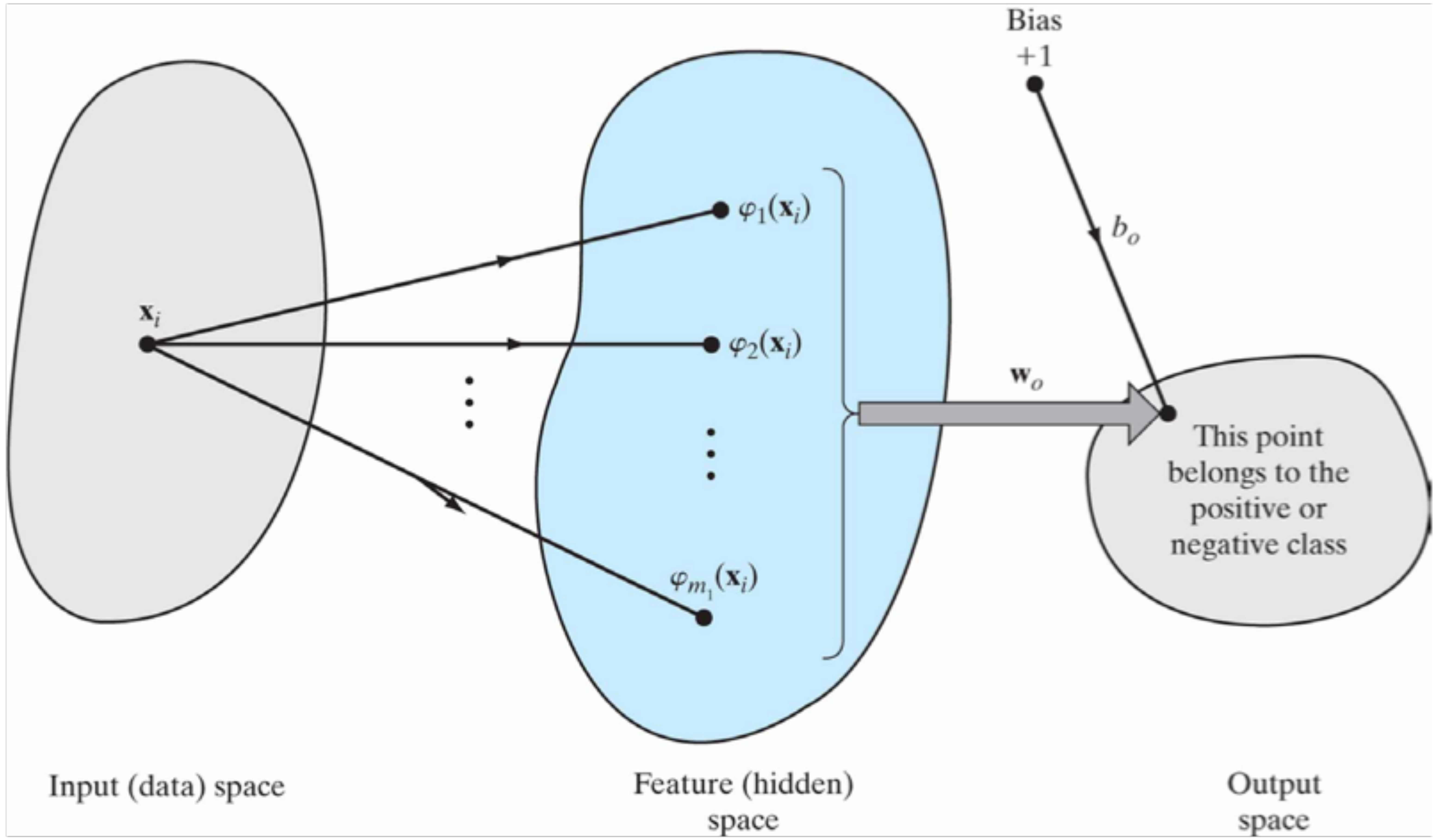
How do we learn in Non-linear spaces?

- Main Idea: For ***<u>any</u>*** data set, the original input space can always be mapped to some higher-dimensional feature space such that the data is linearly separable (e.g., **Cover's Theorem**)



Kernel Machine Illustration

- Once projected, optimal weights can be learned in this new feature space.



Bias
+1
b_o
$\mathbf{x}_i$
$\varphi_1(\mathbf{x}_i)$
$\varphi_2(\mathbf{x}_i)$
$\varphi_{m_1}(\mathbf{x}_i)$
$\mathbf{w}_o$
This point belongs to the positive or negative class
Input (data) space
Feature (hidden) space
Output space

Kernel Function

- The **linear classifier** relies on inner product between vector $K(\mathbf{x}^i,\mathbf{x}^j)=<\mathbf{x}^i \cdot \mathbf{x}^j>$

- If every data point is mapped into high-dimensional space via some transformation:, the inner product becomes $\Phi\text{:}\ \ \mathbf{x} \rightarrow \phi(\mathbf{x}),$

$$K(\mathbf{x}_i,\mathbf{x}_j)= <\phi(\mathbf{x}^i) \cdot \phi(\mathbf{x}^j)>$$

- A ***kernel function*** is a function that is equivalent to an inner product in some feature space. Nonlinear SVM relies on the this kernel.

- Example: we can define a kernel as

$$K(\mathrm{x}^i, \mathrm{x}^j) = (\mathrm{x}^i \cdot \mathrm{x}^j + 1)^2$$

This is equivalent to mapping to the quadratic space!

Example: Quadratic Kernel

- Consider a 2-d input space: (generalizes to n-d)

$$
\begin{aligned}
K(\mathrm{x}^i, \mathrm{x}^j) &= (\mathrm{x}^i \cdot \mathrm{x}^j + 1)^2 \\
&= (x_1^i x_1^j + x_2^i x_2^j + 1)^2 \\
&= {x_1^i}^2 {x_1^j}^2 + 2 x_1^i x_2^i x_1^j x_2^j + {x_2^i}^2 {x_2^j}^2 + 2 x_1^i x_1^j + 2 x_2^i x_2^j + 1 \\
&= ({x_1^i}^2, \sqrt{2}\, x_1^i x_2^i,\ {x_2^i}^2, \sqrt{2}\, x_1^i, \sqrt{2}\, x_2^i, 1) \cdot \\
&\quad\ ({x_1^j}^2, \sqrt{2}\, x_1^j x_2^j,\ {x_2^j}^2, \sqrt{2}\, x_1^j, \sqrt{2}\, x_2^j, 1) \\
&= \Phi(\mathrm{x}^i) \cdot \Phi(\mathrm{x}^j)
\end{aligned}
$$

nonlinear mapping of $\mathbf{x}^i$ and $\mathbf{x}^j$ to quadratic space

A kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\phi(\mathbf{x})$ explicitly).

Computing inner product of quadratic features is O(m^2) time vs. O(m) time for kernel

Non-Linear SVMs

- Remember the soft-margin SVM dual?

$$\max \quad -\frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\alpha_i\alpha_j y_i y_j \mathbf{x}_i'\mathbf{x}_j + \sum_{i=1}^{\ell}\alpha_i$$

$$\text{s.t.} \quad \sum_{i=1}^{\ell}\alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C \quad \forall i = 1 \ldots \ell$$

Optimization techniques for finding α_i's remain the same

- Replace $\mathbf{x}_i'\mathbf{x}_j$ with a kernel $K(\mathbf{x}_i, \mathbf{x}_j)$

This shows why the dual formulation is very useful

$$\max \quad -\frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{\ell}\alpha_i$$

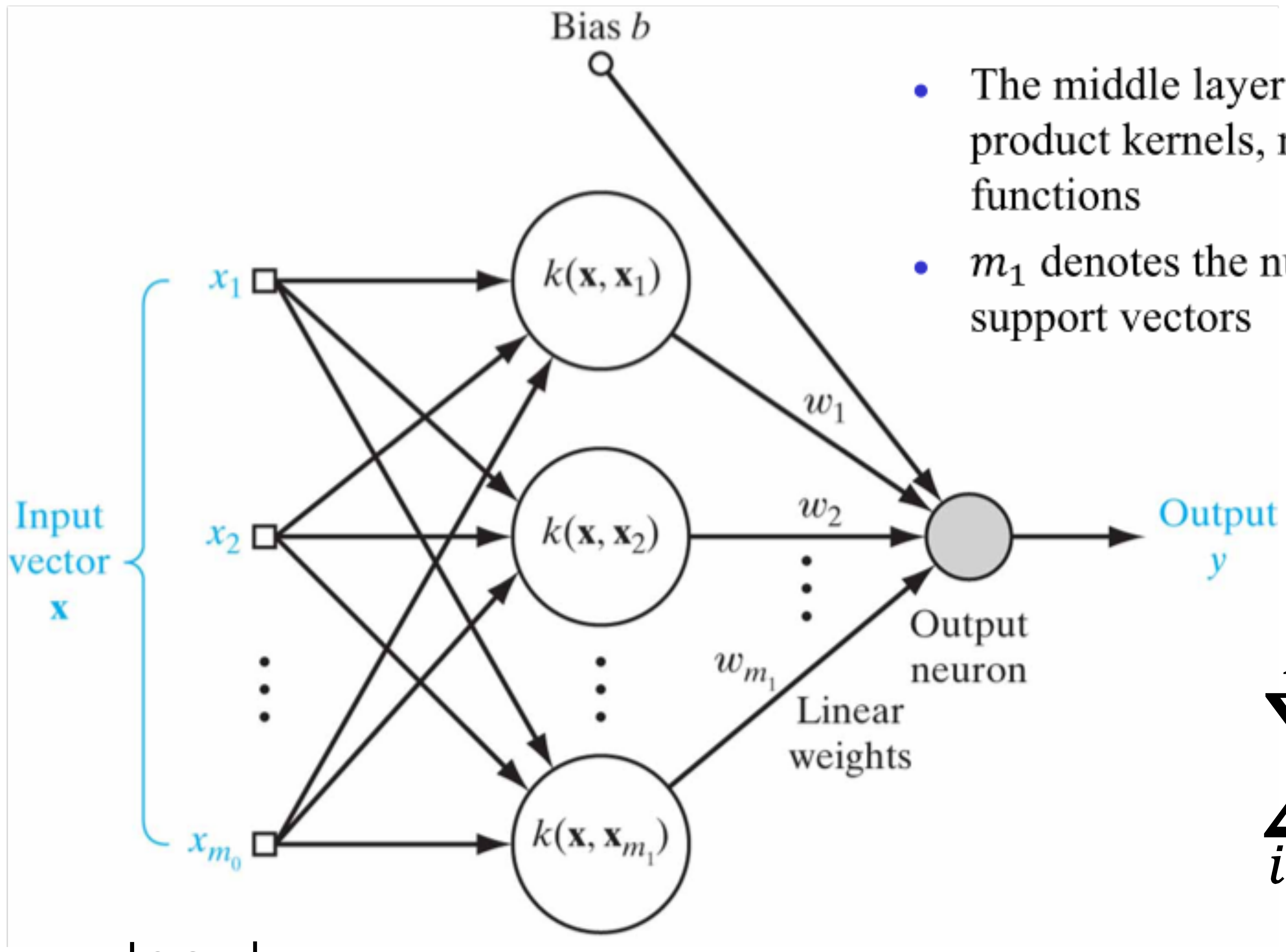
$$\text{s.t.} \quad \sum_{i=1}^{\ell}\alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C \quad \forall i = 1 \ldots \ell$$

Examples

- Some popular kernels
  - Linear kernel: $\kappa(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle$
  - Polynomial kernel: $\kappa(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + c)^d$, $c, d \geq 0$
  - Gaussian kernel: $\kappa(\mathbf{x}, \mathbf{z}) = e^{-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{\sigma}}$, $\sigma > 0$
  - Sigmoid kernel: $\kappa(\mathbf{x}, \mathbf{z}) = \tanh^{-1} \eta \langle \mathbf{x}, \mathbf{z} \rangle + \theta$
- Kernels can also be constructed from other kernels
  - Conical (not linear) combinations, $\kappa(\mathbf{x}, \mathbf{z}) = a_1 \kappa_1(\mathbf{x}, \mathbf{z}) + a_2 \kappa_2(\mathbf{x}, \mathbf{z})$
  - Products of kernels, $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) \kappa_2(\mathbf{x}, \mathbf{z})$
  - Products of functions, $\kappa(\mathbf{x}, \mathbf{z}) = f_1(\mathbf{x}) f_2(\mathbf{z})$, f_1, f_2 are real valued functions.

Kernel SVM as a Network


Bias b
x_1
x_2
x_{m_0}
Input vector $\mathbf{x}$
$k(\mathbf{x}, \mathbf{x}_1)$
$k(\mathbf{x}, \mathbf{x}_2)$
$k(\mathbf{x}, \mathbf{x}_{m_1})$
w_1
w_2
w_{m_1}
Linear weights
Output neuron
Output y

- The middle layer depicts inner-product kernels, not mapping functions
- m_1 denotes the number of support vectors

Input layer

Feature layer

$$k(\mathbf{x}, \mathbf{x}_i) = \boldsymbol{\phi}^T(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x})$$

$$\sum_{i=1}^{N_s} \alpha_i d_i k(\mathbf{x}, \mathbf{x}_i) = 0$$

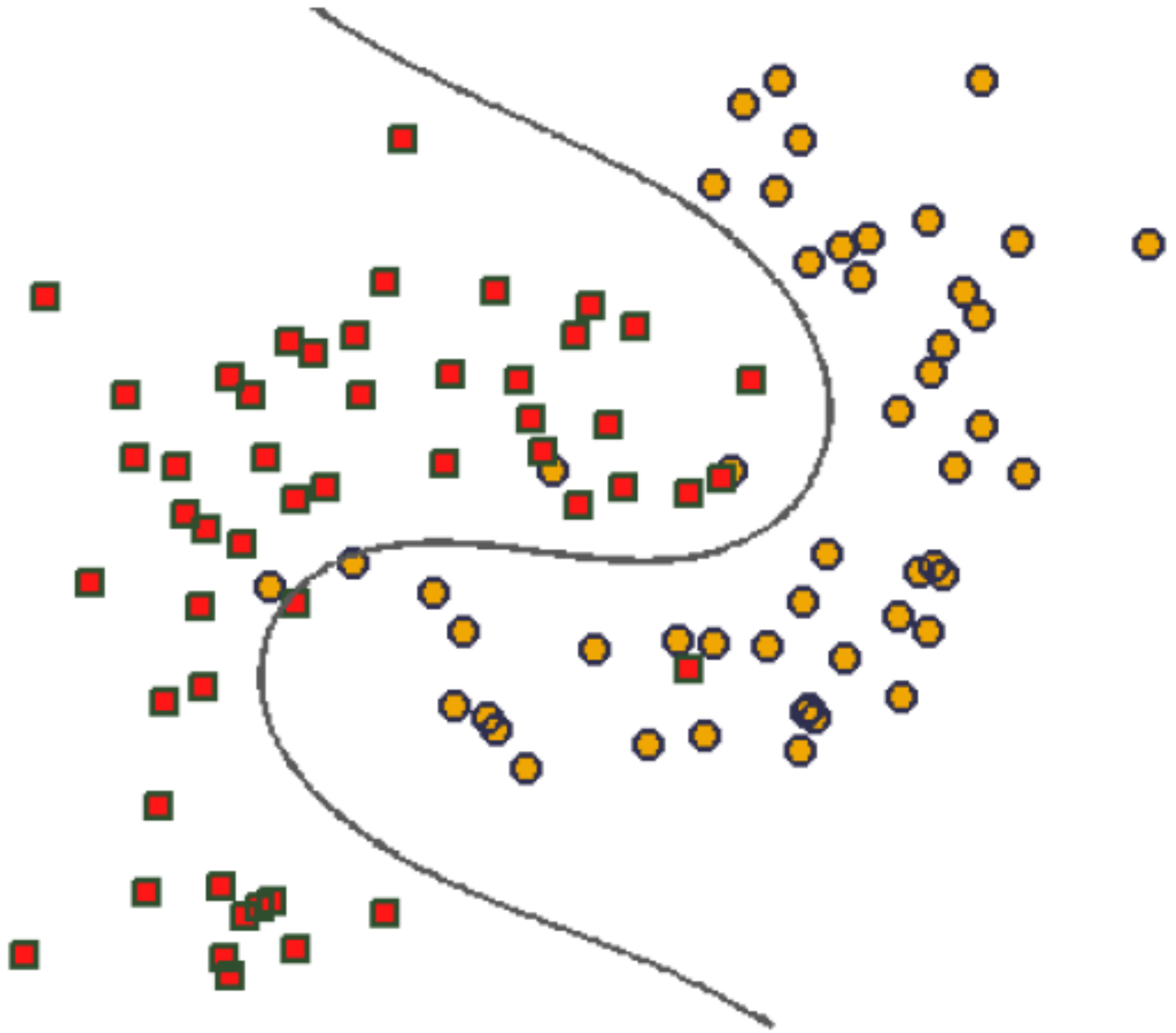
Optimal Hyperplane

$$\mathbf{w} = \sum_{i=1}^{N_s} \alpha_i d_i \boldsymbol{\phi}(\mathbf{x}_i)$$

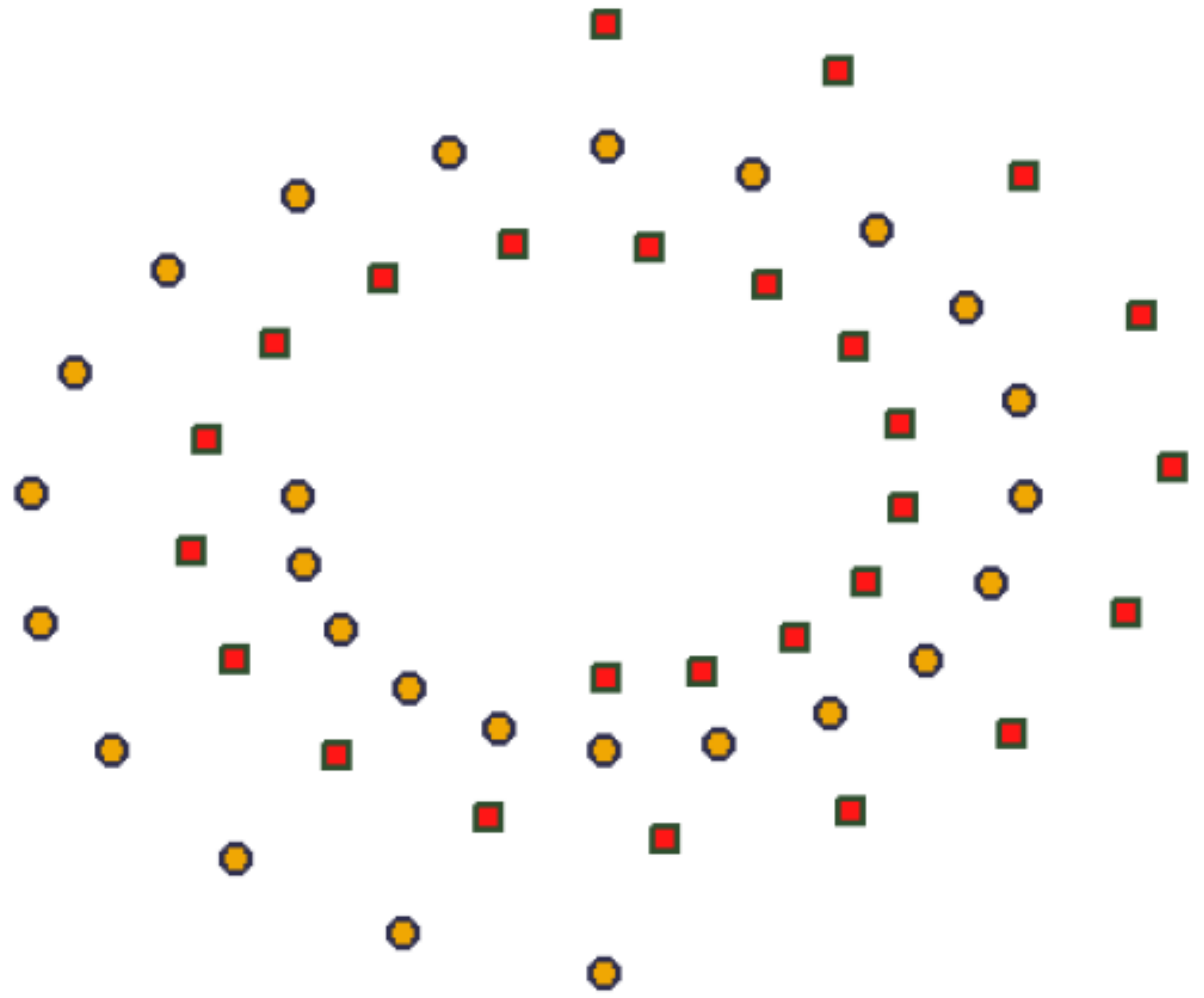
Optimal Weights

kernel of degree 3:

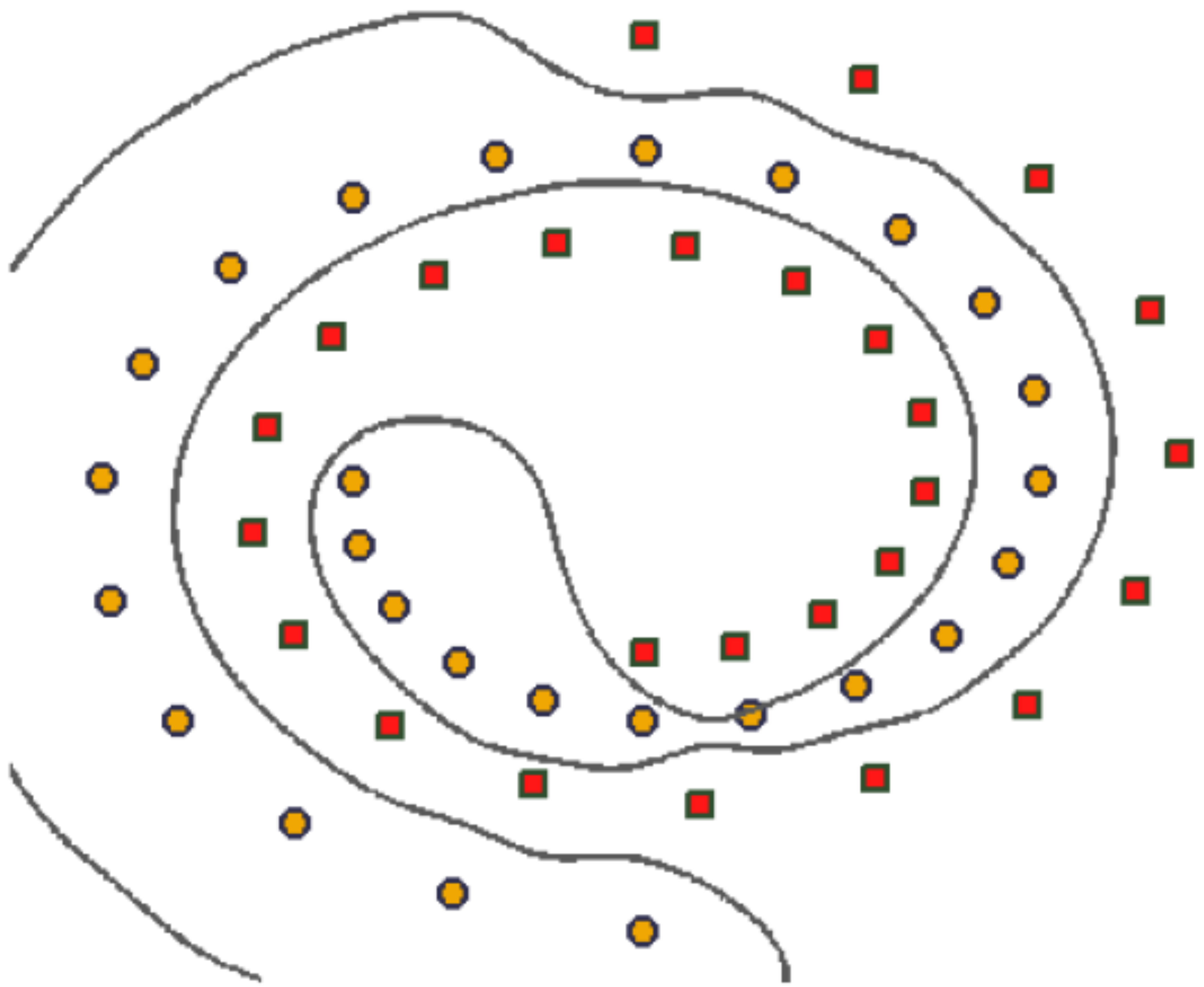
Polynomial kernel, p = 3



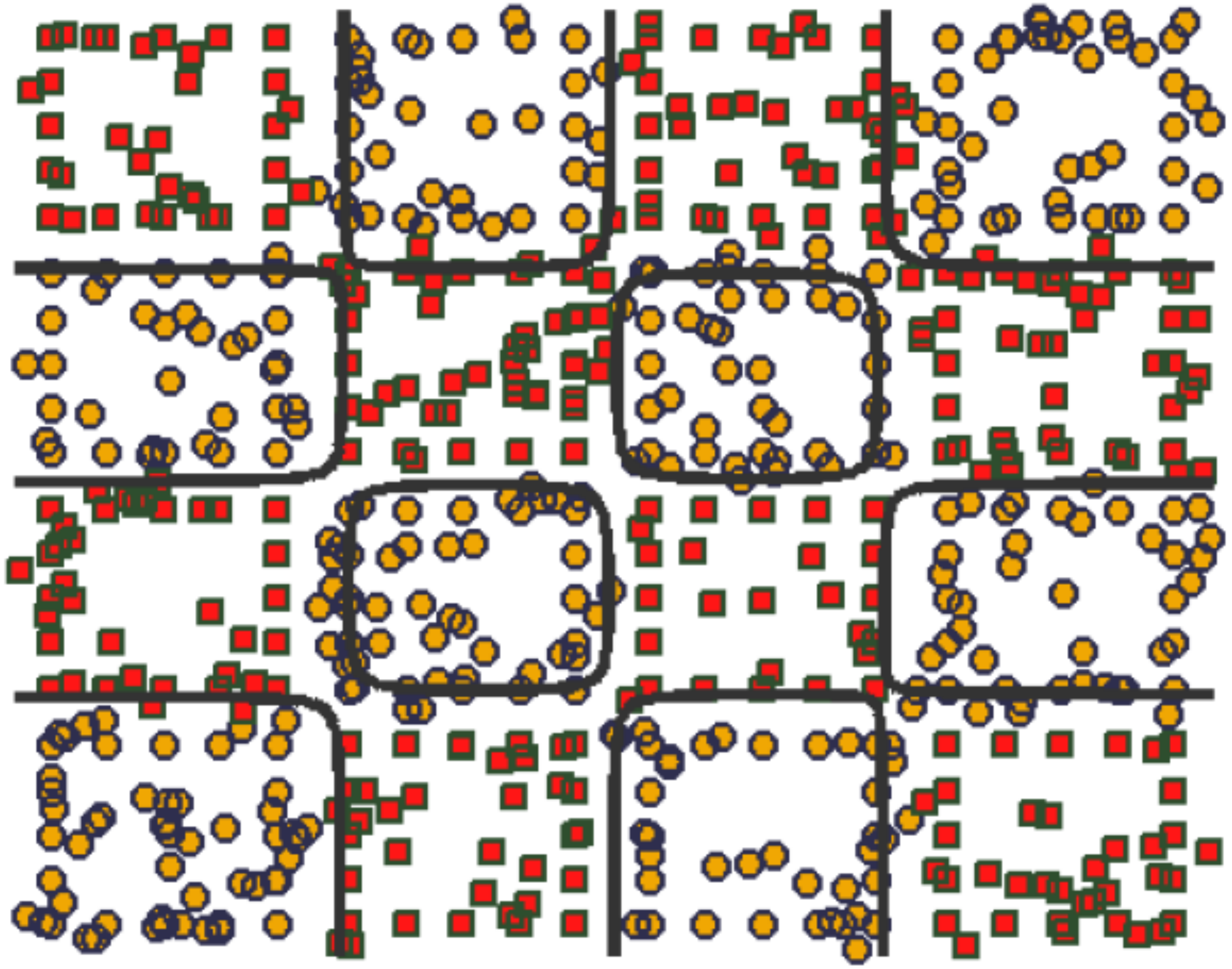
What about something like this?



Gaussian kernels transform the input space to an infinite-dimensional feature space!



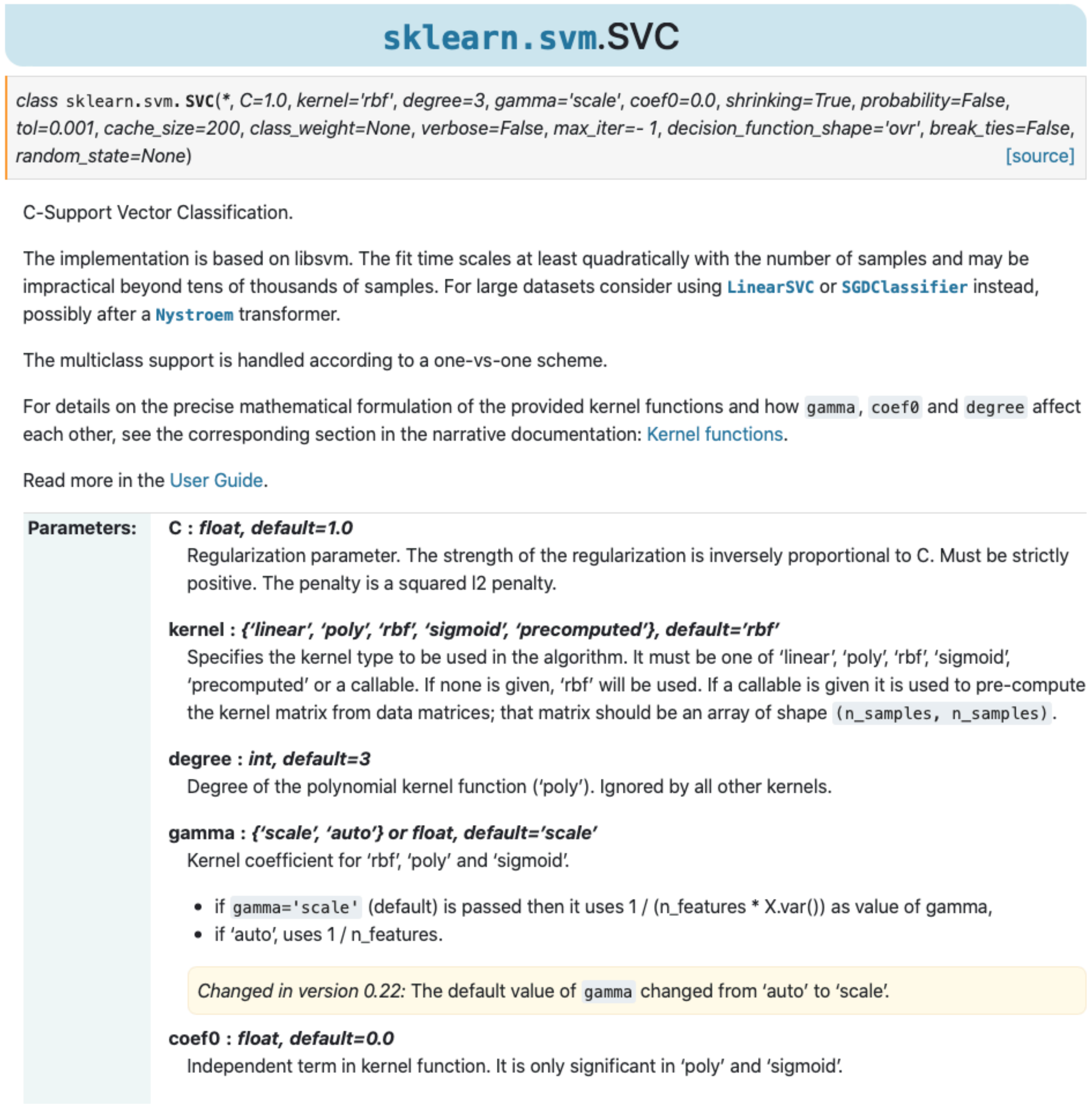
SVMs can fit “anything” with the appropriate choice of parameters and kernel.



SVMs Summary

- Advantages of SVMs
  - polynomial-time exact optimization rather than approximate methods
    - unlike decision trees and neural networks
  - Kernels allow very flexible hypotheses
  - Can be applied to very complex data types, e.g., graphs, sequences
- Disadvantages of SVMs
  - Must choose a good kernel and kernel parameters
  - Very large problems are computationally intractable
    - quadratic in number of examples
    - problems with more than 20k examples are very difficult to solve exactly

Next Class



sklearn.svm.SVC

class sklearn.svm.**SVC**(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using `LinearSVC` or `SGDClassifier` instead, possibly after a `Nystroem` transformer.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Read more in the User Guide.

Parameters:

C : *float, default=1.0*
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'*
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, default=3*
Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features.

Changed in version 0.22: The default value of `gamma` changed from 'auto' to 'scale'.

coef0 : *float, default=0.0*
Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.