Support Vector Machines

The “real” SVMs

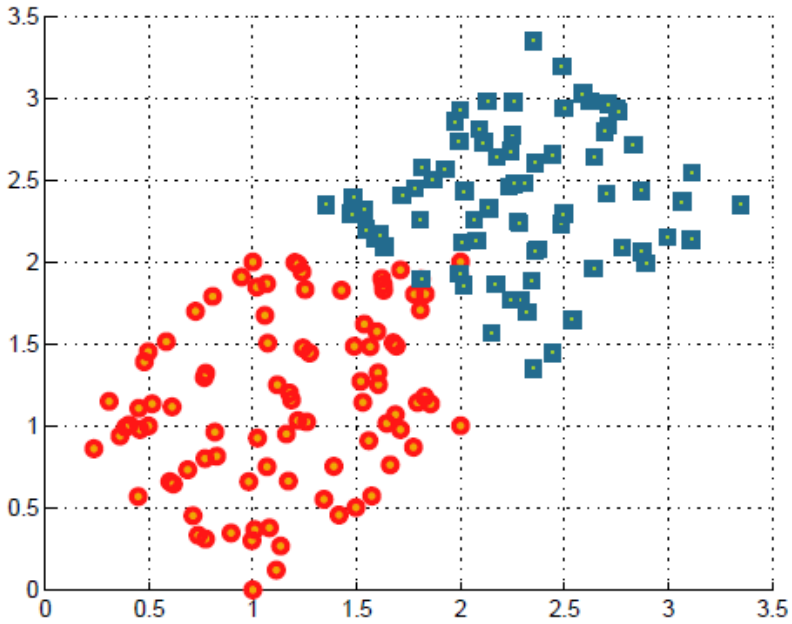
So far…

- 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.
- Problem solved?

Inseparable data set

So far we have assumed that the data was *linearly separable* and the formulation derived is the *hard-margin SVM*.

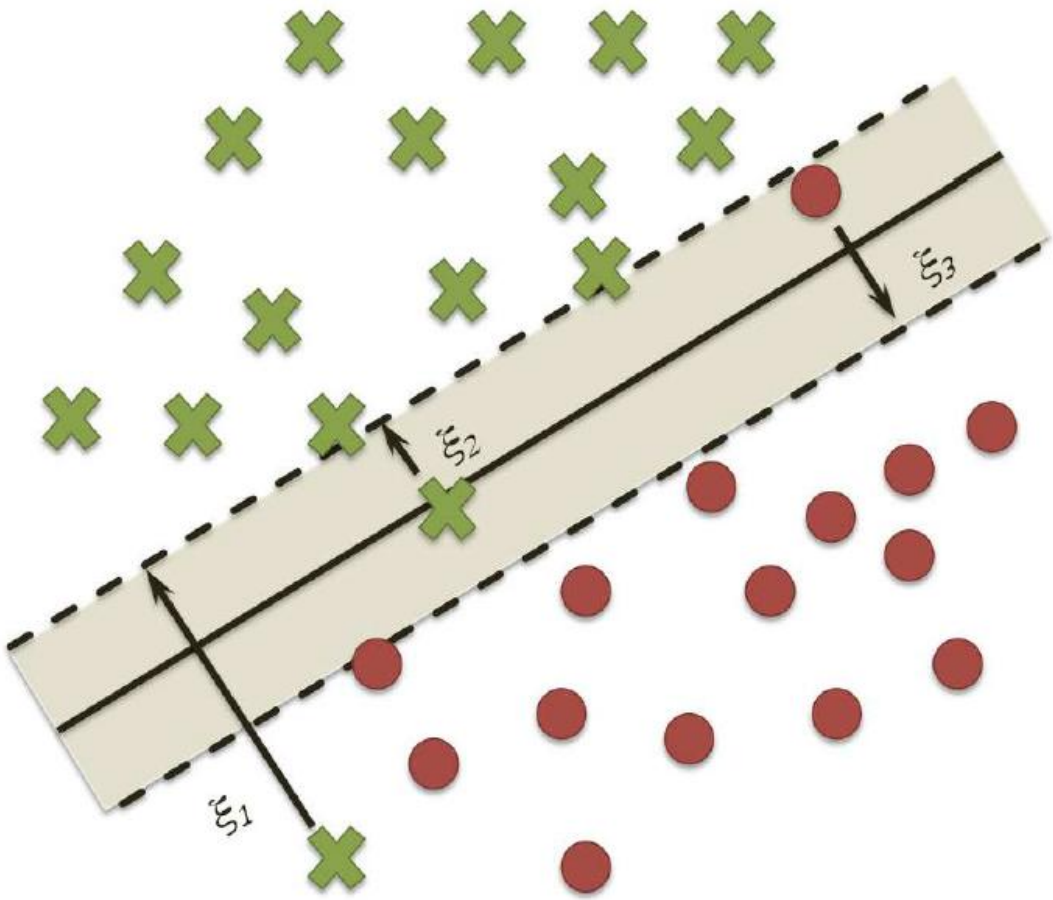
But what if there is *one bad example*?



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

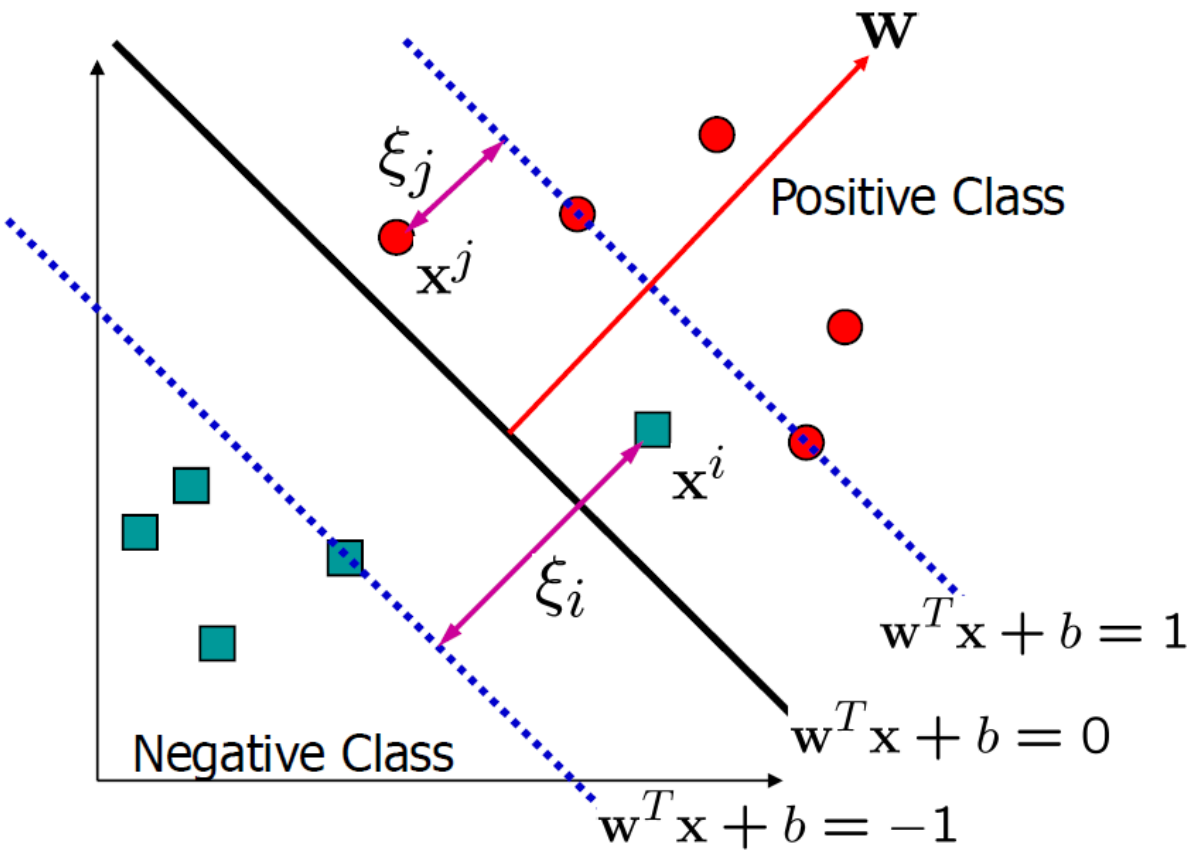
Soft Margin SVMs

Modify objective to minimize error of misclassified points with respect to the *margin*



Soft Margin

- Allow functional margins to be less than 1
  - But will charge a penalty



Soft-Margin Maximization

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

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

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

- Introduce **slack variables** ξ_i to allow functional margins to be smaller than 1
- Parameter *C* controls the tradeoff between maximizing the margin and fitting the training example

Dual Formulation of softmargin

$$\begin{aligned} \max \quad & -\tfrac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\alpha_i\alpha_j y_i y_j \mathbf{x}_i'\mathbf{x}_j + \sum_{i=1}^{\ell}\alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{\ell}\alpha_i y_i = 0 \\ & 0 \le \alpha_i \le C, \quad \forall i = 1 \ldots \ell \end{aligned}$$

For the dual problem, the only difference between hard and soft margin cases is that *the α_i's are upper-bounded by C*.

The effect of C is to put box constraints on α, the weights of support vectors

Limits the influence of outliers/noise

Dual Formulation

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

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

$0 \leq \alpha_i \leq c$ $\quad$ $i=1, \ldots, N$

- We now have 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

support vectors (α_i>0)

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

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

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

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

Linear SVMs

- So far our classifier is a *separating hyperplane.*
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points $\mathbf{x}^i$ are support vectors with non-zero Lagrange multipliers α_i.
- For both training and classification, we see training data appear only inside inner products:

Find $\alpha_1 \ldots \alpha_N$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y^i y^j \boxed{<\mathbf{x}^i \cdot \mathbf{x}^j>}$ is maximized

and

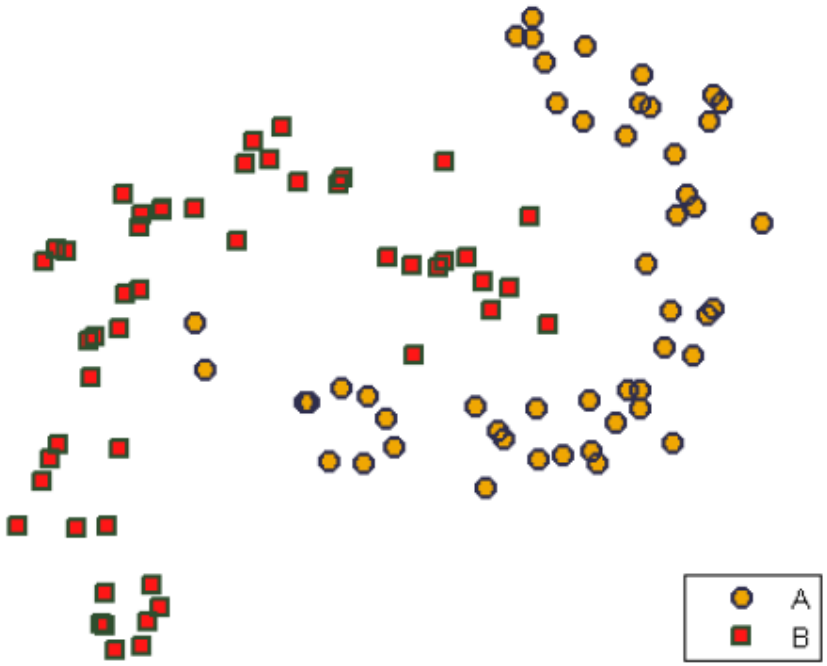
(1) $\Sigma\alpha_i y^i = 0$

(2) $0 \leq \alpha_i \leq c$ for all α_i

$$f(\mathbf{x}) = \Sigma\alpha_i y^i \boxed{<\mathbf{x}^i \cdot \mathbf{x}>} + b$$

Non Linear data?

So far, we have assumed *linearly* separable and inseparable data. Gives us the *linear* hard and soft-margin SVMs i.e., the classification surface is a linear hyperplane.



How do we learn in Non-linear spaces?

How do we handle this case? Naive approach: Explicit transformation.

- Transformed space is very high-dimensional.
- Linear algorithms break down in nonlinear spaces.
- Cannot guarantee *convexity*.

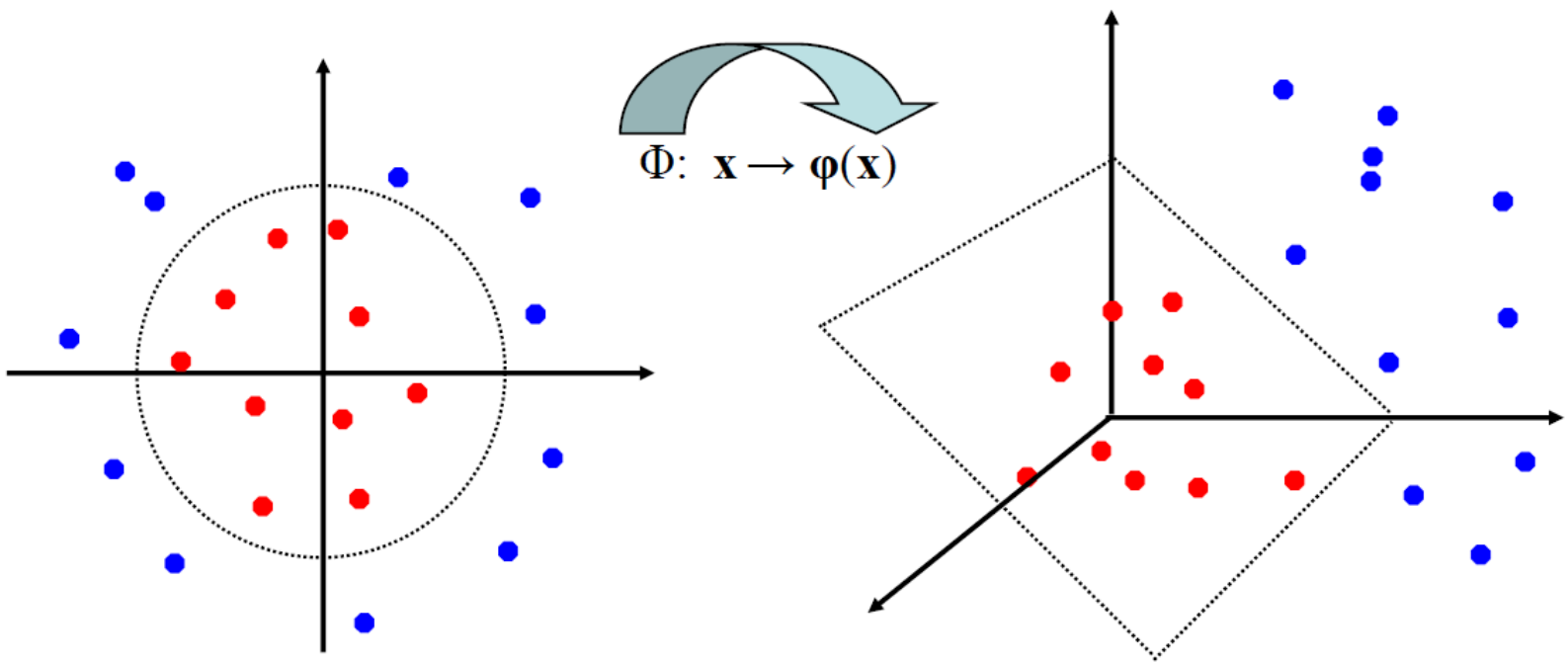
But we already have a linear version which is

- well understood.
- computationally efficient.
- convex

How can we do the *nonlinear transformations* and keep the *linear algorithms*?

Non-Linear SVMs: Feature 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



Quadratic Example

- Assume m input dimensions

$$\mathbf{x} = (x_1, x_2, \cdots, x_m)$$

- Number of quadratic terms:

(m+2)-choose-2 $= \frac{(m+2)(m+1)}{2}$

- The number of dimensions increase rapidly - expensive to compute!

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \vdots \\ \sqrt{2}x_m \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_m^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \sqrt{2}x_2x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \vdots \\ \sqrt{2}x_{m-1}x_m \end{pmatrix}$$

Constant Term

Linear Terms

Pure Quadratic Terms

Quadratic Cross-Terms

You may be wondering about the $\sqrt{2}$'s

You will find out why they are there soon!

Kernel Function

- The linear classifier relies on inner product between vectors $K(\mathbf{x}^i,\mathbf{x}^j)=<\mathbf{x}^i \cdot \mathbf{x}^j>$
- If every data point is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes:

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

- A ***kernel function*** is a function that is equivalent to an inner product in some feature space.
- 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)

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

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

Kernel Trick

We have a mapping ϕ that can

- map nonlinear, low-dimensional data to a linear, high-dimensional (feature) space. Primal linear methods can be directly applied.
- inner-products in feature space can easily be computed. Dual linear methods can be directly applied by modifying how inner products are represented.

A kernel is a function, that for all $\mathbf{x}$, $\mathbf{z}$ *satisfies* $\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$, *where* ϕ *is a mapping from an input space to a feature space.*

If we use algorithms that only depend on inner product information, then we never have to compute (or even know) the actual features. We just need $\kappa(\mathbf{x}, \mathbf{z})$.

Non-Linear SVMs

Remember the soft-margin SVM dual?

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

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

$$0 \le \alpha_i \le C \quad \forall i = 1 \dots \ell$$

Optimization techniques for finding α_i's remain the same

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

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

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

$$0 \le \alpha_i \le C \quad \forall i = 1 \dots \ell$$

This shows why the dual formulation is very useful

Kernel Functions

- In practical, the user specifies the kernel function K, without explicitly stating the transformation $\phi(\cdot)$
- Given a kernel function, finding its corresponding transformation can be very cumbersome
  - This is why people only specify the kernel function without worrying about the exact transformation
- Another view: a kernel function computes some kind of measure of similarity between objects
- If you have a reasonable measure of similarity for your application, can we use it as the kernel in an SVM?

Examples

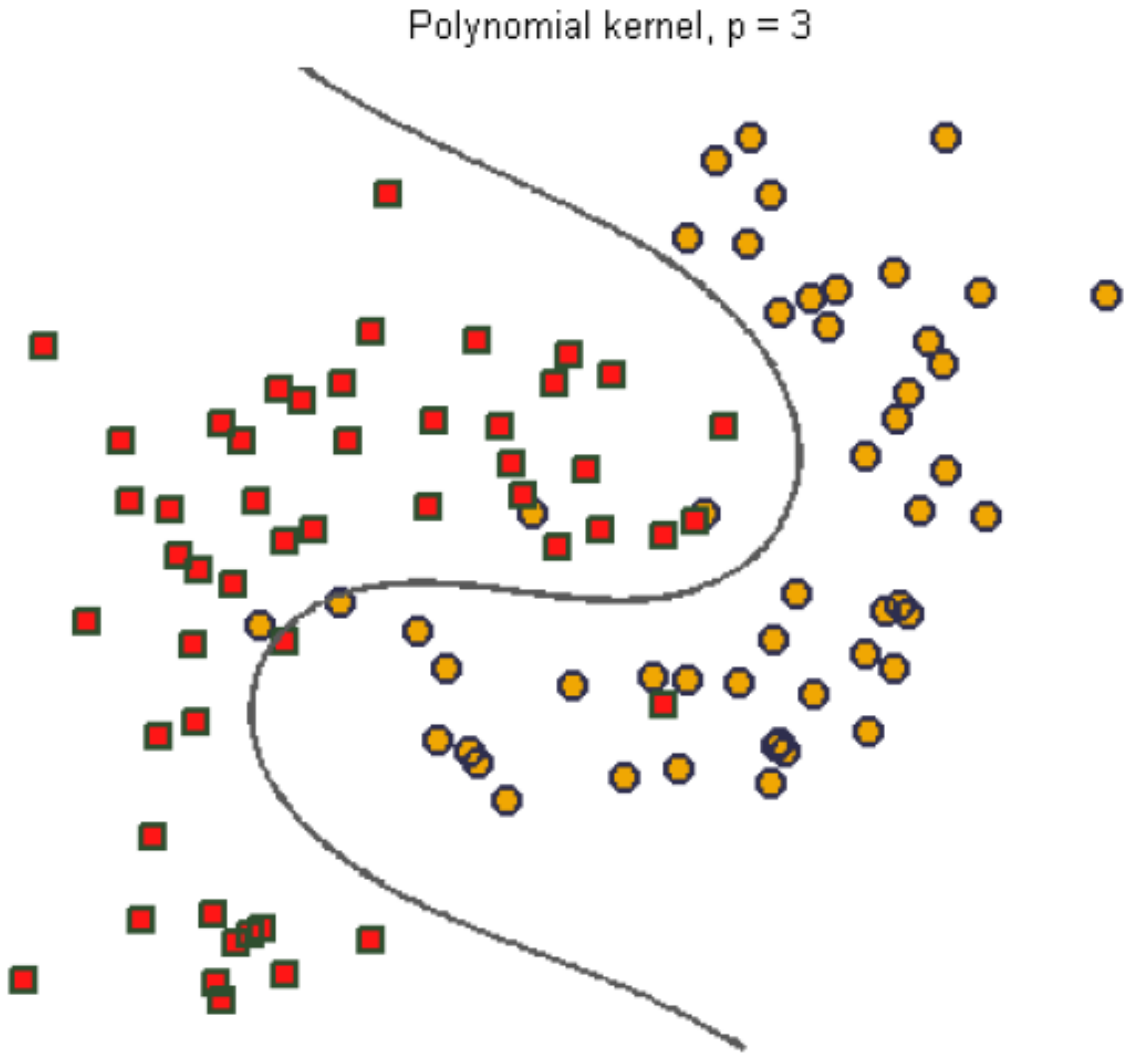
Some popular kernels

- 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: $\kappa(\mathbf{x}, \mathbf{z}) = \tanh^{-1} \eta \langle \mathbf{x}, \mathbf{z} \rangle + \theta$

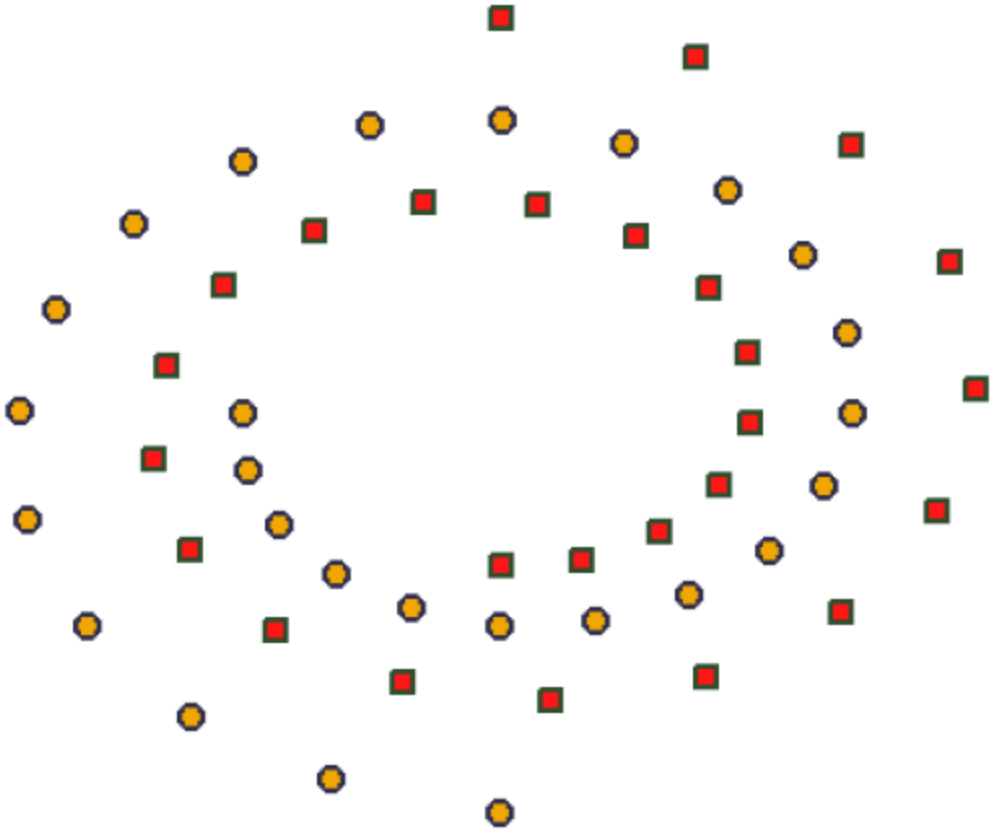
Kernels can also be constructed from other kernels:

- 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.

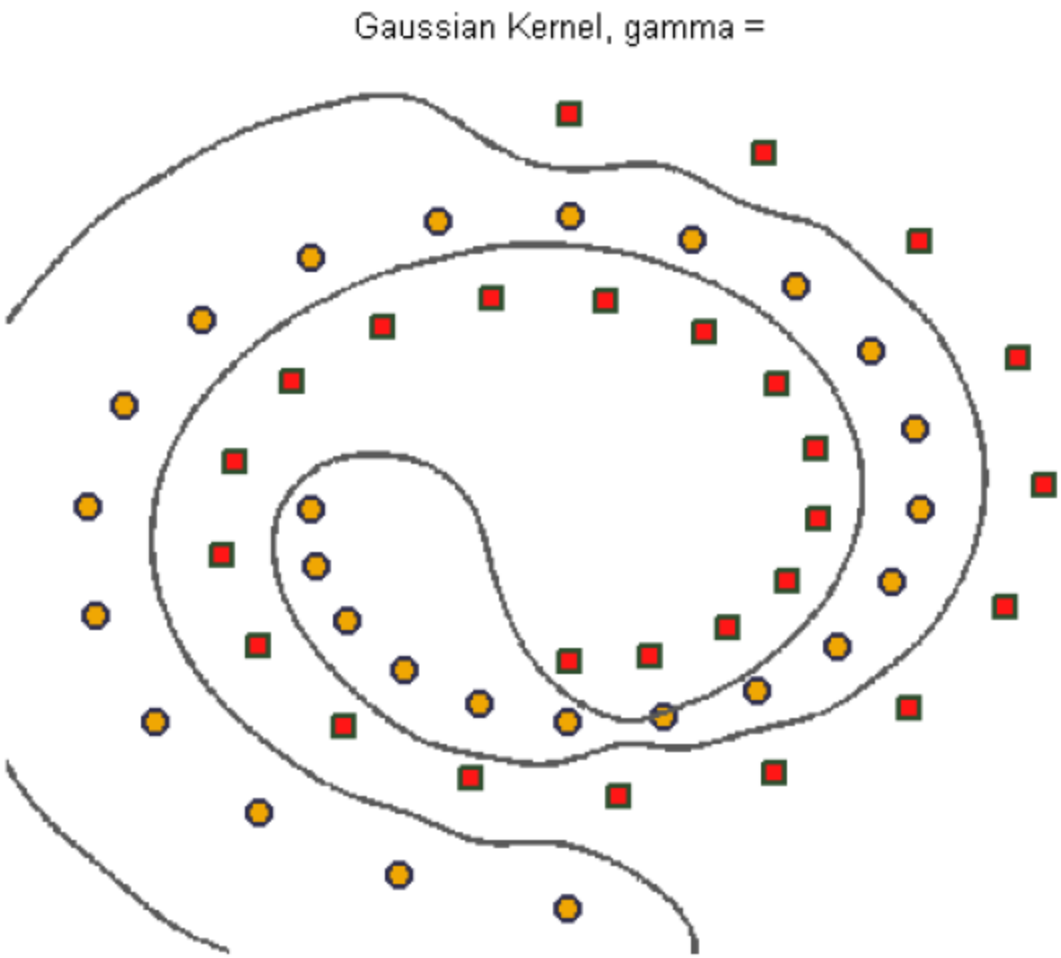
kernel of degree 3:



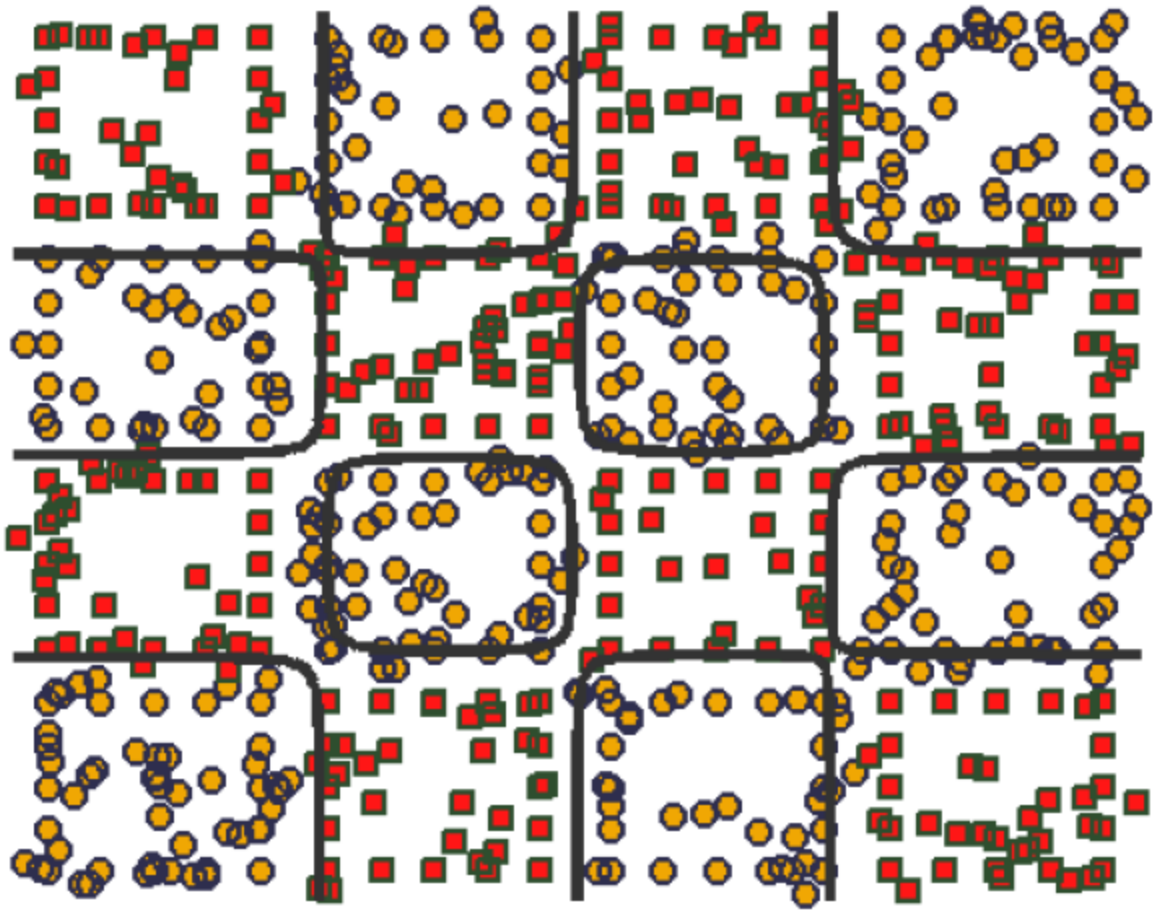
What about something like this?



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

- Select the kernel function to use (important but often trickiest part of SVM)
  - In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try and usually support by off-the-shelf software
- Select the parameter of the kernel function and the value of ***c***
  - You can use the values suggested by the SVM software

    see www.kernel-machines.org/software.html for a list of available software
  - You can set apart a validation set to determine the values of the parameter

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