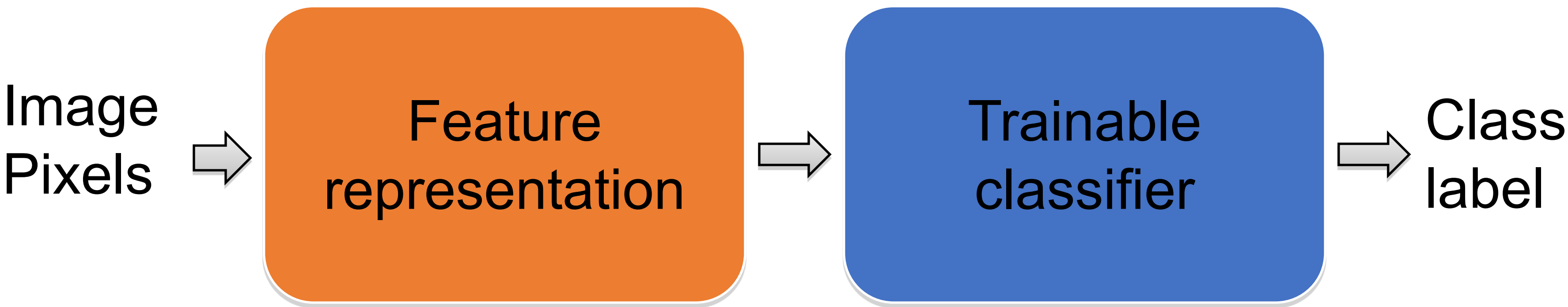
Principles of Safe Autonomy
Lecture 8: Linear Classifiers
Support Vector Machines

Sayan Mitra

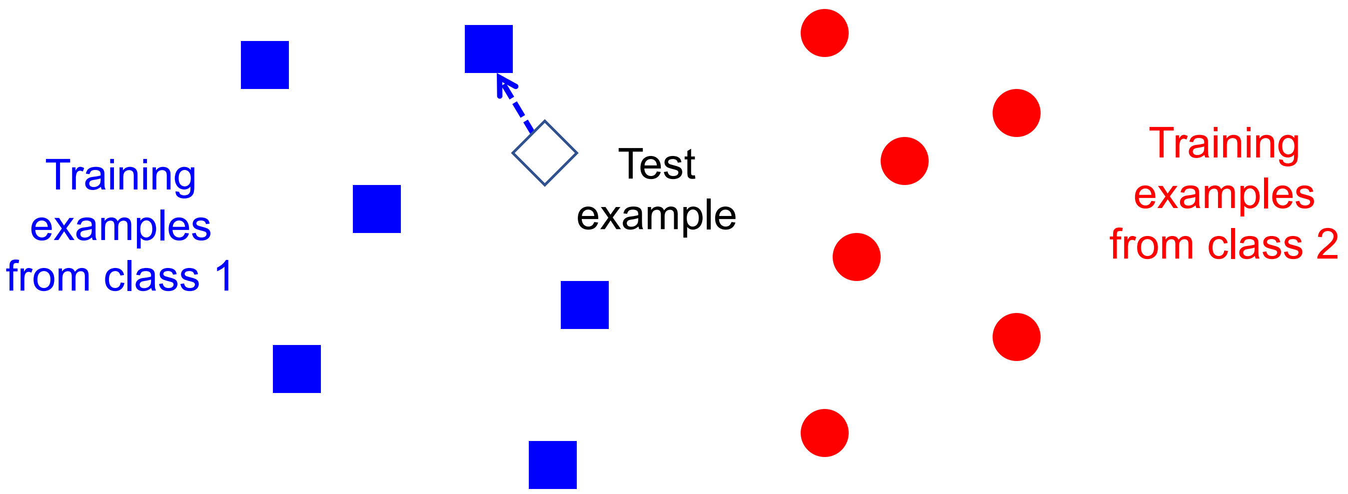
Feb 18 2019

“Classic” recognition pipeline



- Hand-crafted feature representation
- Off-the-shelf trainable classifier

Classifiers: Nearest neighbor

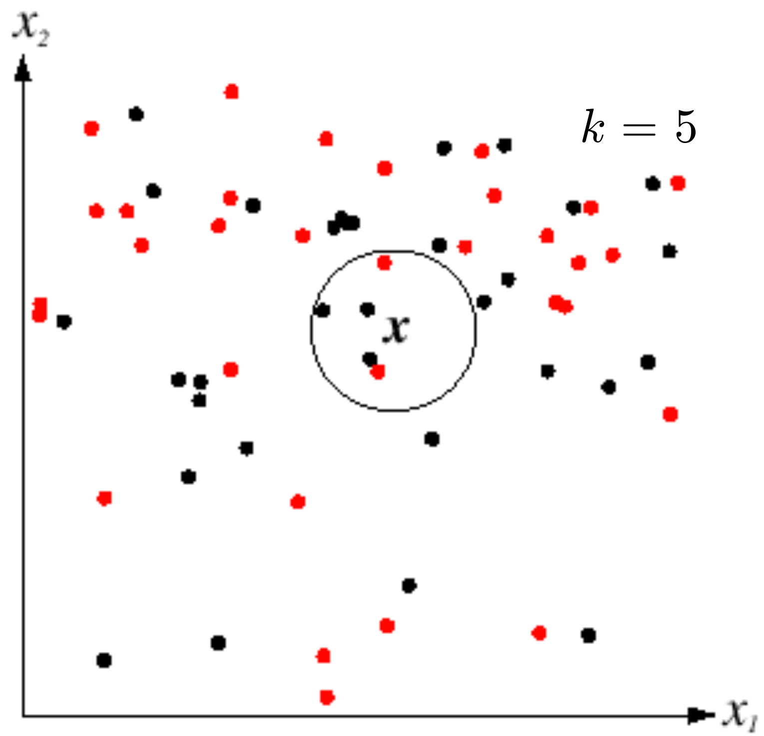


f(**x**) = label of the training example nearest to **x**

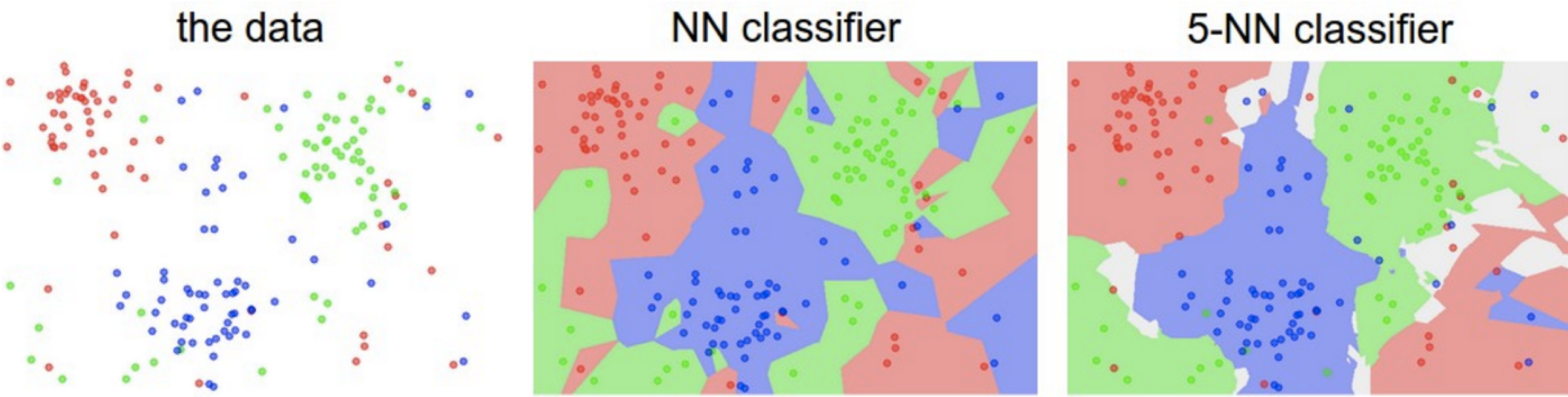
- All we need is a distance or similarity function for our inputs
- No training required!

K-nearest neighbor classifier

- For a new point, find the k closest points from training data
- Vote for class label with labels of the k points



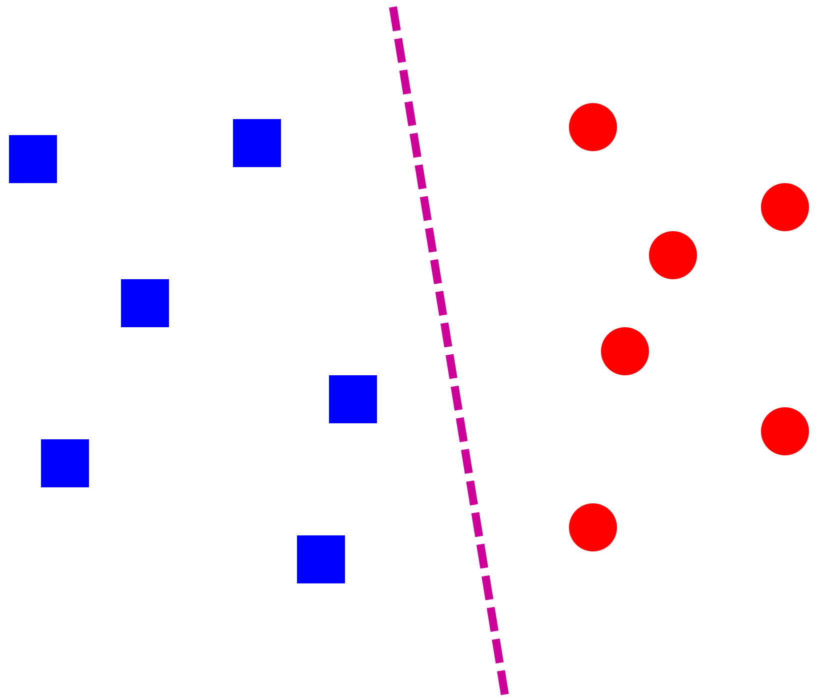
K-nearest neighbor classifier



- 2d points, and 3 classes. White regions are “ambiguous”
- Which classifier is more robust to *outliers*?

Credit: Andrej Karpathy, http://cs231n.github.io/classification/

Linear classifiers



- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

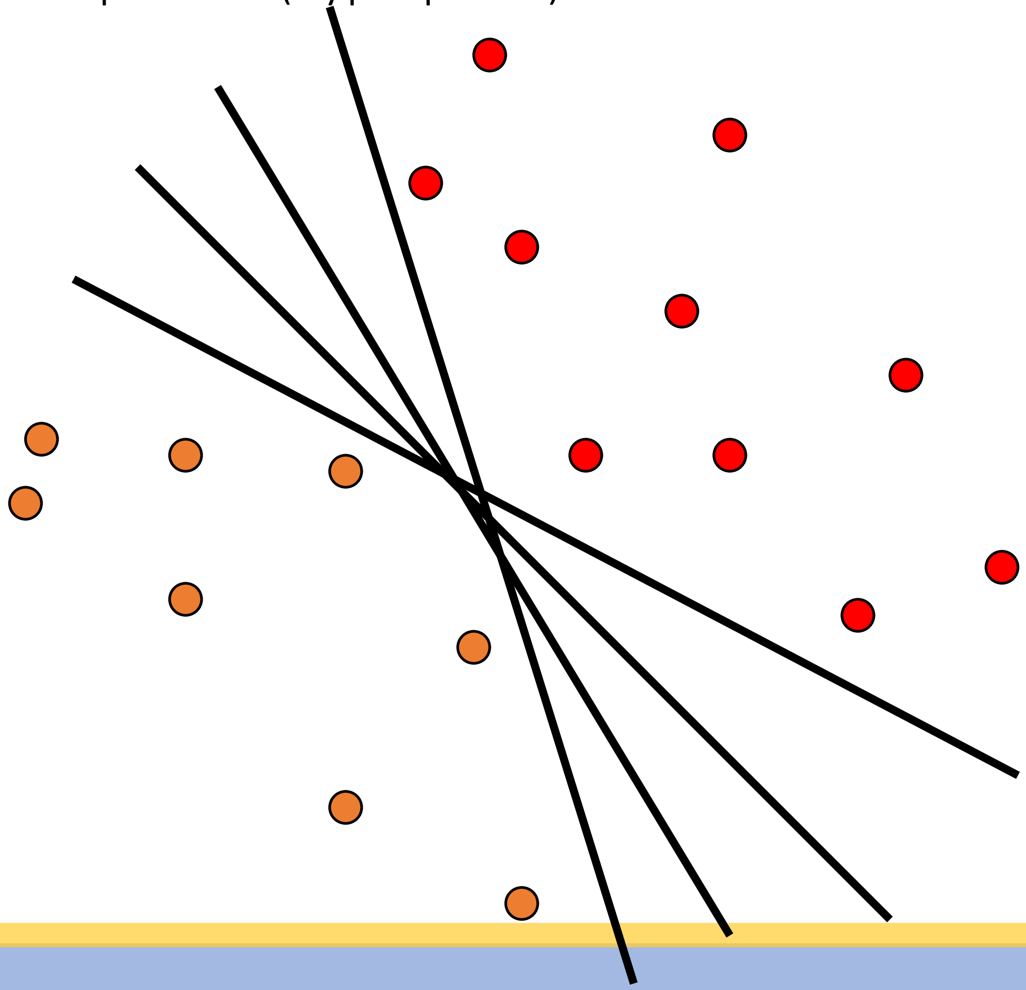
Nearest neighbor vs. linear classifiers

- NN pros:
  - Simple to implement
  - Decision boundaries not necessarily linear
  - Works for any number of classes
  - *Nonparametric* method
- NN cons:
  - Need good distance function
  - Slow at test time
- Linear pros:
  - Low-dimensional *parametric* representation
  - Very fast at test time
- Linear cons:
  - Works for two classes
  - How to train the linear function?
  - What if data is not linearly separable?

SVMs

Linear classifiers

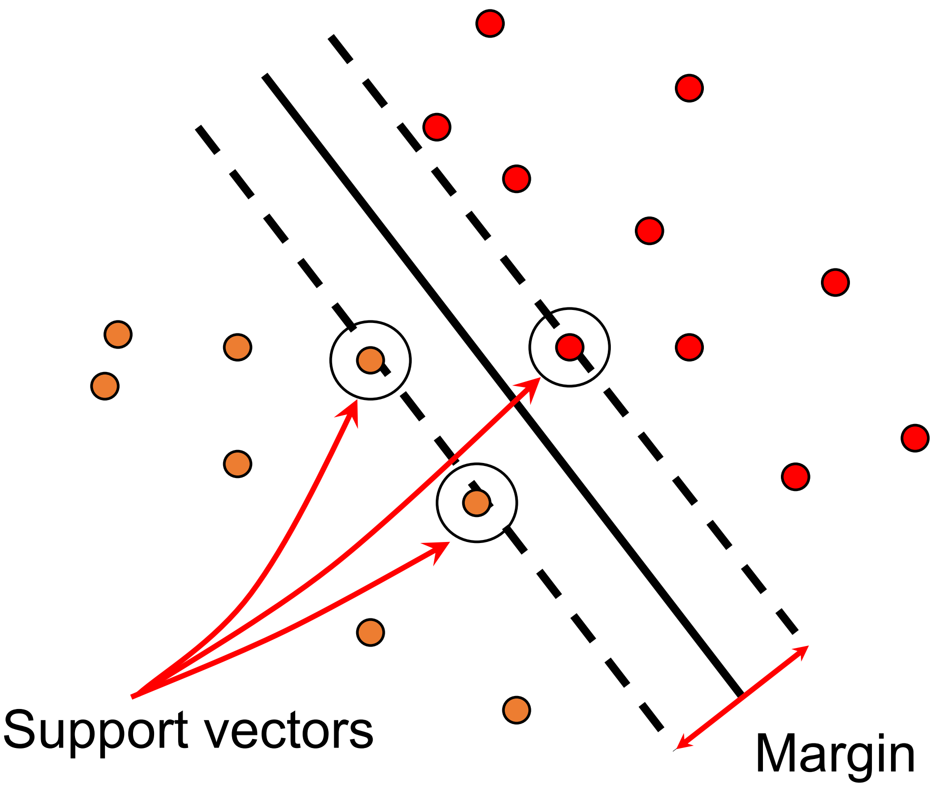
- When the data is linearly separable, there may be more than one separator (hyperplane)



Which separator is best?

Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



$\mathbf{x}_i$ positive $(y_i = 1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$ negative $(y_i = -1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support vectors, $\quad \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Distance between point and hyperplane: $\quad \dfrac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$

Therefore, the margin is $2 / \|\mathbf{w}\|$

C. Burges, [A Tutorial on Support Vector Machines for Pattern Recognition](), Data Mining and Knowledge Discovery, 1998

- https://cs.stanford.edu/people/karpathy/svmjs/demo/

Finding the maximum margin hyperplane

1. Maximize margin $2 / ||\mathbf{w}||$
2. Correctly classify all training data:

$$\mathbf{x}_i \text{ positive } (y_i = 1): \qquad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \qquad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

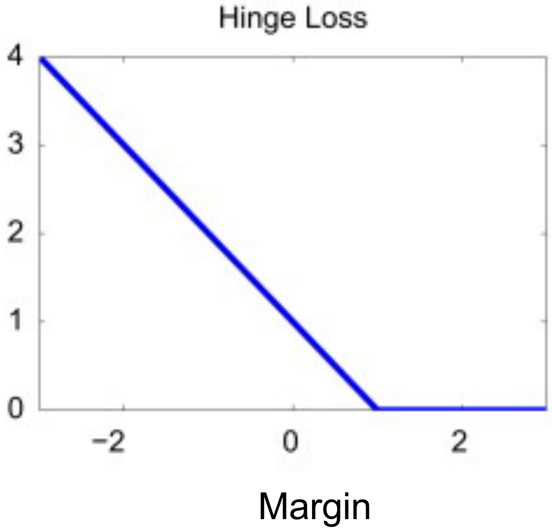
- *Quadratic optimization problem*:
-

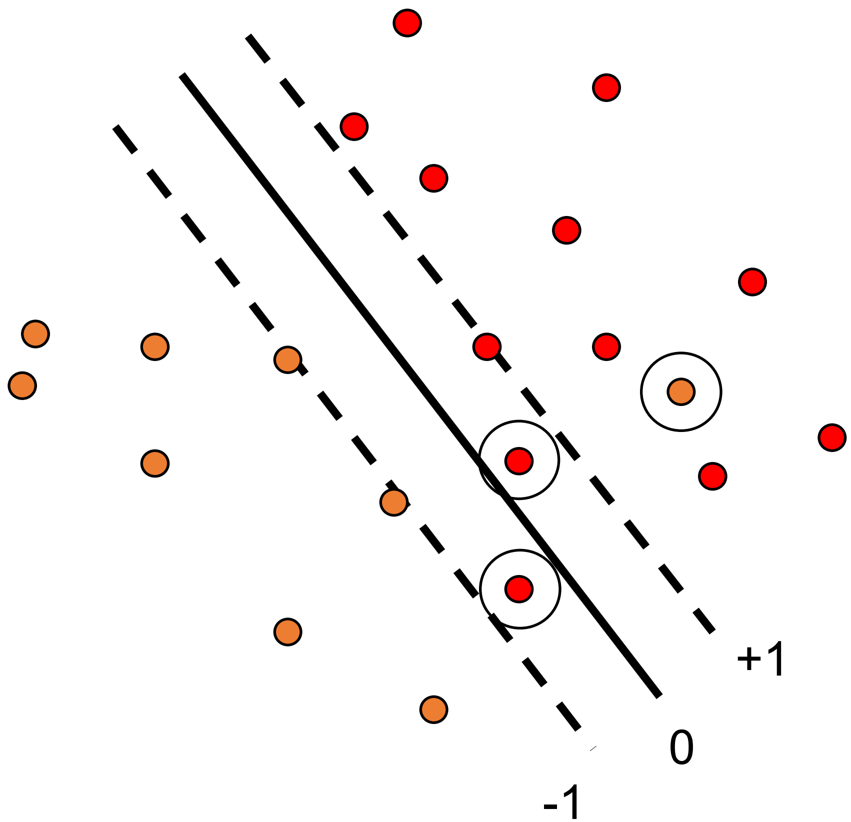
$$\boxed{\min_{w,b} \frac{1}{2} ||w||^2 - \sum \alpha_i [y_i (w.x_i + b) - 1]}$$

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

SVM parameter learning

$$\min_{w,b} \frac{1}{2}||w||^2 - \sum \alpha_i [y_i (w.x_i + b) - 1]$$





- Demo: http://cs.stanford.edu/people/karpathy/svmjs/demo

Nonlinear SVMs

General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable

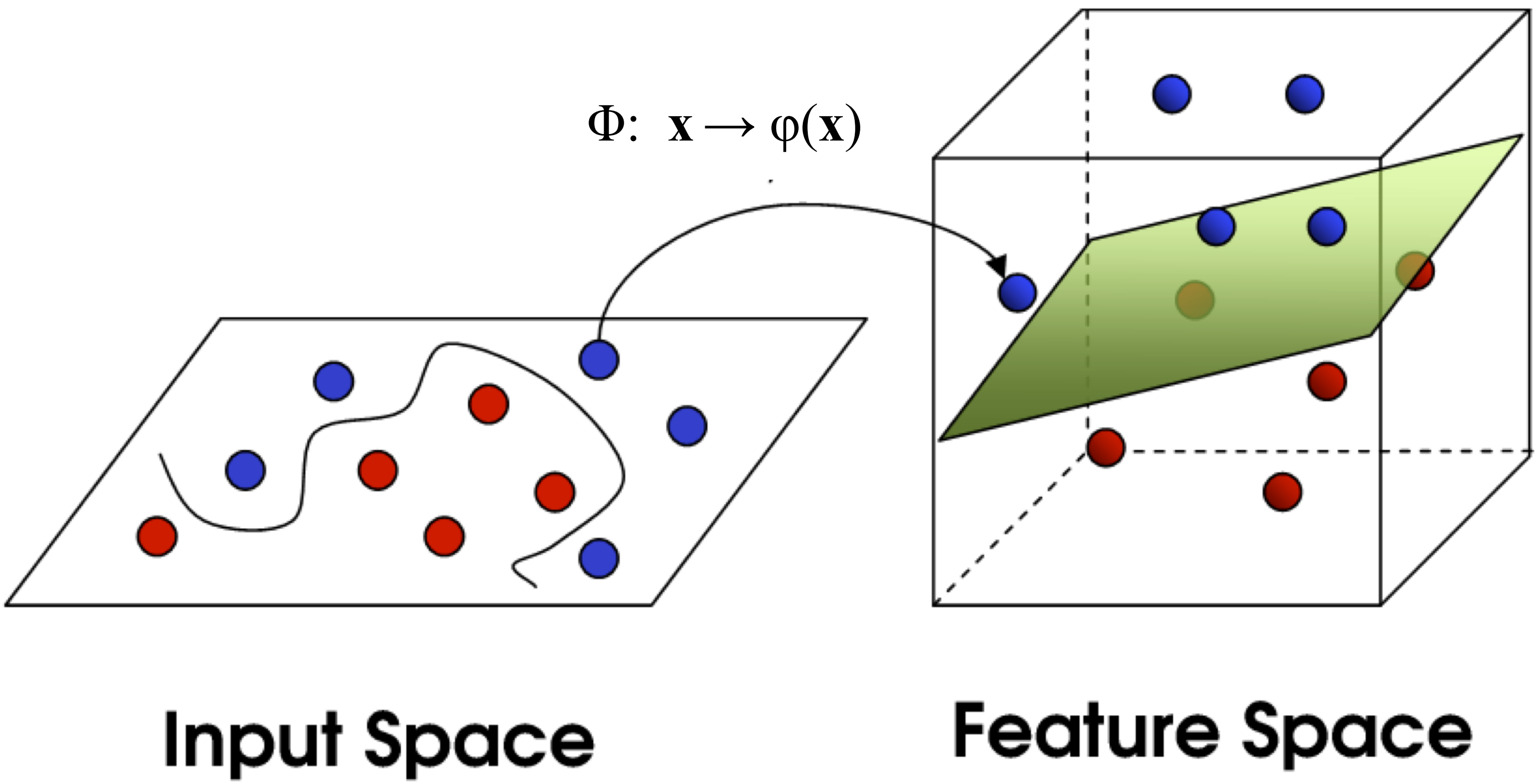
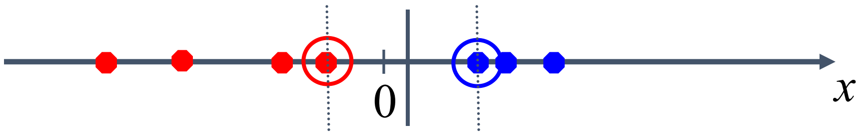


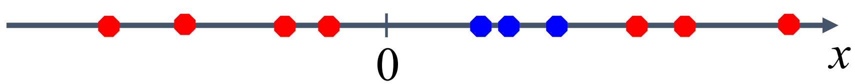
Image source

Nonlinear SVMs

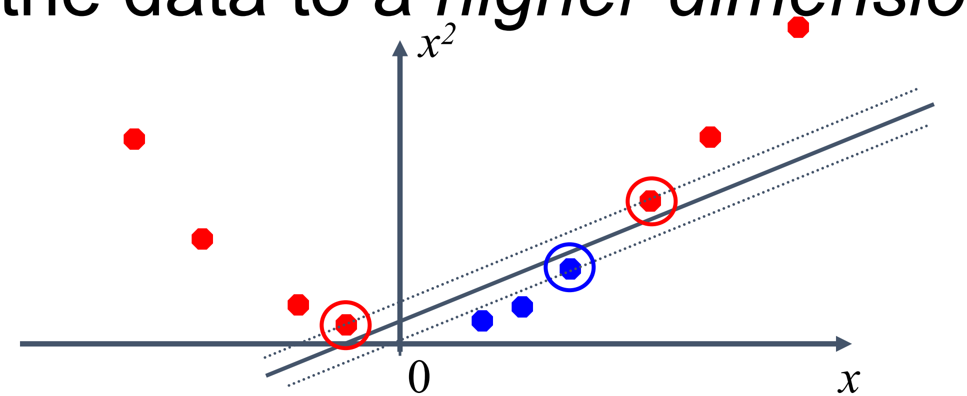
Linearly separable dataset in 1D:



Non-separable dataset in 1D:



We can map the data to a *higher-dimensional space*:



The kernel trick

General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable

The kernel trick: instead of explicitly computing the lifting transformation $\boldsymbol{\varphi}(\mathbf{x})$, define a kernel function K such that

$$K(\mathbf{x}, \mathbf{y}) = \boldsymbol{\varphi}(\mathbf{x}) \cdot \boldsymbol{\varphi}(\mathbf{y})$$

(to be valid, the kernel function must satisfy *Mercer's condition*)

The kernel trick

- Linear SVM decision function:

$$\mathbf{w}\cdot\mathbf{x}+b=\sum_i \alpha_i y_i \mathbf{x}_i\cdot\mathbf{x}+b$$

learned weight

Support vector

C. Burges, [A Tutorial on Support Vector Machines for Pattern Recognition](), Data Mining and Knowledge Discovery, 1998

The kernel trick

- Linear SVM decision function:

$$\mathbf{w}\cdot\mathbf{x}+b=\sum\nolimits_i \alpha_i y_i \mathbf{x}_i\cdot\mathbf{x}\ +b$$

- Kernel SVM decision function:

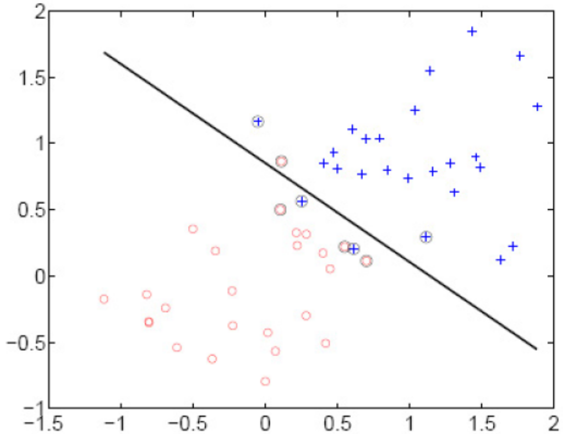
$$\sum_i \alpha_i y_i \varphi(\mathbf{x}_i)\cdot\varphi(\mathbf{x})\ +b=\sum_i \alpha_i y_i K(\mathbf{x}_i,\mathbf{x})\ +b$$

- This gives a nonlinear decision boundary in the original feature space

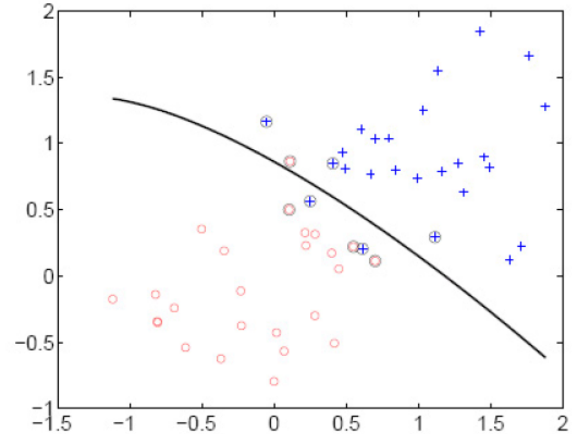
C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

$$K(\mathbf{x},\mathbf{y}) = (c + \mathbf{x}\cdot\mathbf{y})^d$$

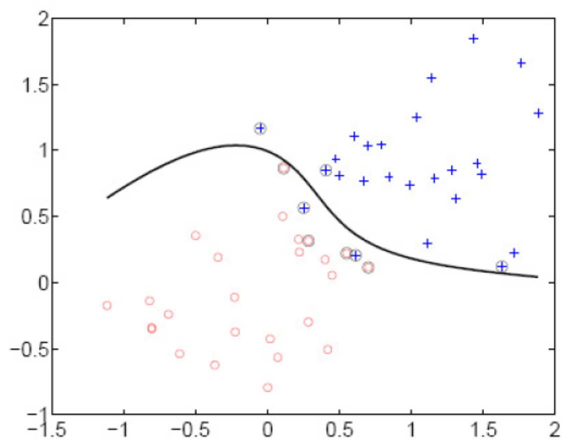
Polynomial kernel:



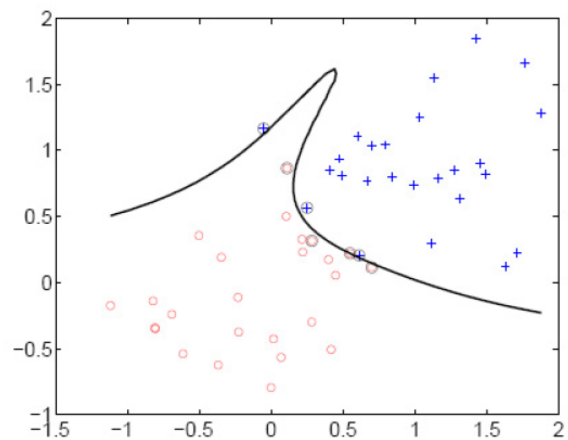
linear



2^{nd} order polynomial



4^{th} order polynomial

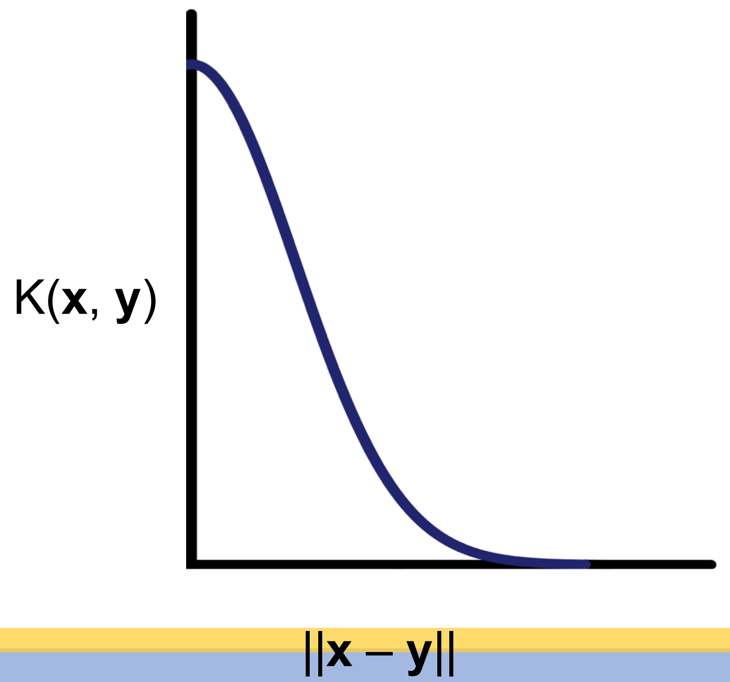


8^{th} order polynomial

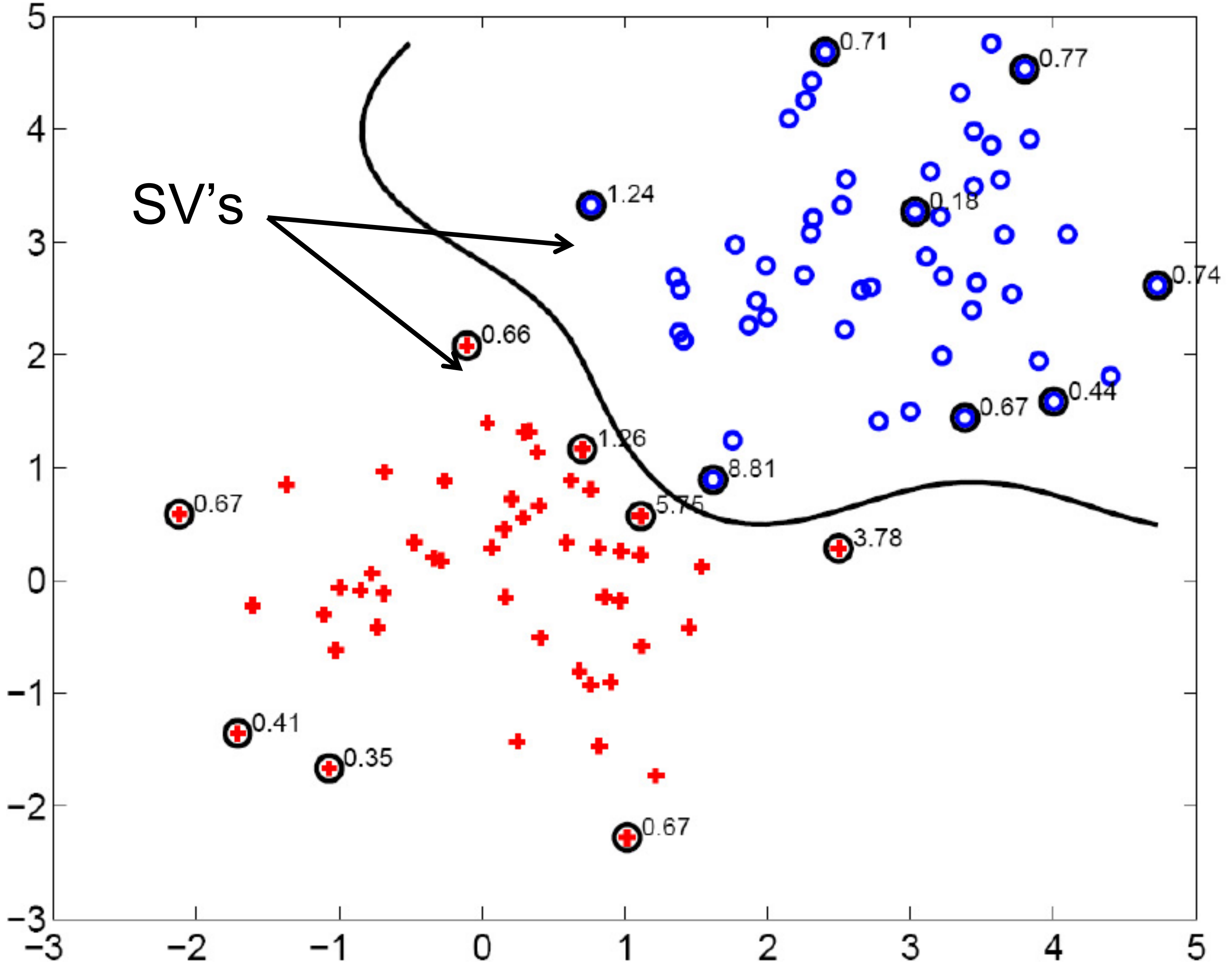
Gaussian kernel

- Also known as the radial basis function (RBF) kernel:

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{1}{\sigma^2} \|\mathbf{x} - \mathbf{y}\|^2 \right)$$



Gaussian kernel



SVMs: Pros and cons

- Pros
  - Kernel-based framework is very powerful, flexible
  - Training is convex optimization, globally optimal solution can be found
  - Amenable to theoretical analysis
  - SVMs work very well in practice, even with very small training sample sizes
- Cons
  - No “direct” multi-class SVM, must combine two-class SVMs (e.g., with one-vs-others)
  - Computation, memory (esp. for nonlinear SVMs)

Kernels for bags of features

- Histogram intersection: $$K(h_1, h_2) = \sum_{i=1}^{N} \min(h_1(i), h_2(i))$$
- Square root (Bhattacharyya kernel): $$K(h_1, h_2) = \sum_{i=1}^{N} \sqrt{h_1(i) h_2(i)}$$
- Generalized Gaussian kernel: $$K(h_1, h_2) = \exp\left(-\frac{1}{A} D(h_1, h_2)^2\right)$$
  - D can be L1 distance, Euclidean distance, χ^2 distance, etc.