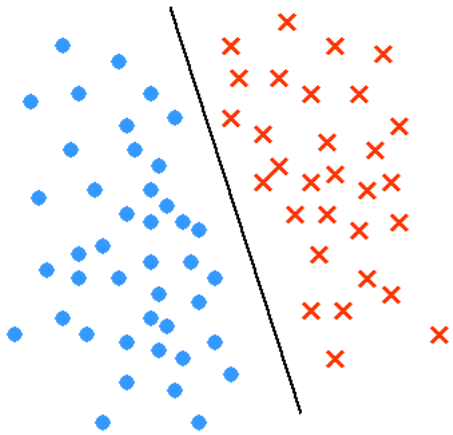
Introduction to

Support Vector Machines

Shivani Agarwal

Support Vector Machines (SVMs)

- Algorithm for learning linear classifiers
- Motivated by idea of maximizing margin
- Efficient extension to non-linear SVMs through use of kernels



Problem Setting

Instances:	$\mathbf{x} \in X$
Class labels:	$y \in Y = \{+1, -1\}$
Target function:	$g : X \rightarrow Y$
Unknown distribution:	$\mathcal{D}$ (on X)

Training data: $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}, \quad y_i = g(\mathbf{x}_i)$

Objective: Given new $\mathbf{x}$, predict y so that probability of error is minimal

Problem Setting (continued)

Hypothesis space: $\mathcal{H} = \{h : X \to Y\}$

Error of h on training set S: $err_S(h) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{I}_{\{h(\mathbf{x}_i) \neq y_i\}}$

Probability of error on new $\mathbf{x}$: $err(h) = E_{\mathcal{D}}[\mathbf{I}_{h(\mathbf{x}) \neq g(\mathbf{x})}]$

More precise objective: Find $h \in \mathcal{H}$ such that $err(h)$ is minimal

Linear Classifiers

Instance space: $X = \mathbb{R}^n$

Set of class labels: $Y = \{+1, -1\}$

Training data: $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$

Hypothesis space: $\mathcal{H}_{lin(n)} = \{h : \mathbb{R}^n \rightarrow Y \mid h(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b), \mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}\}$

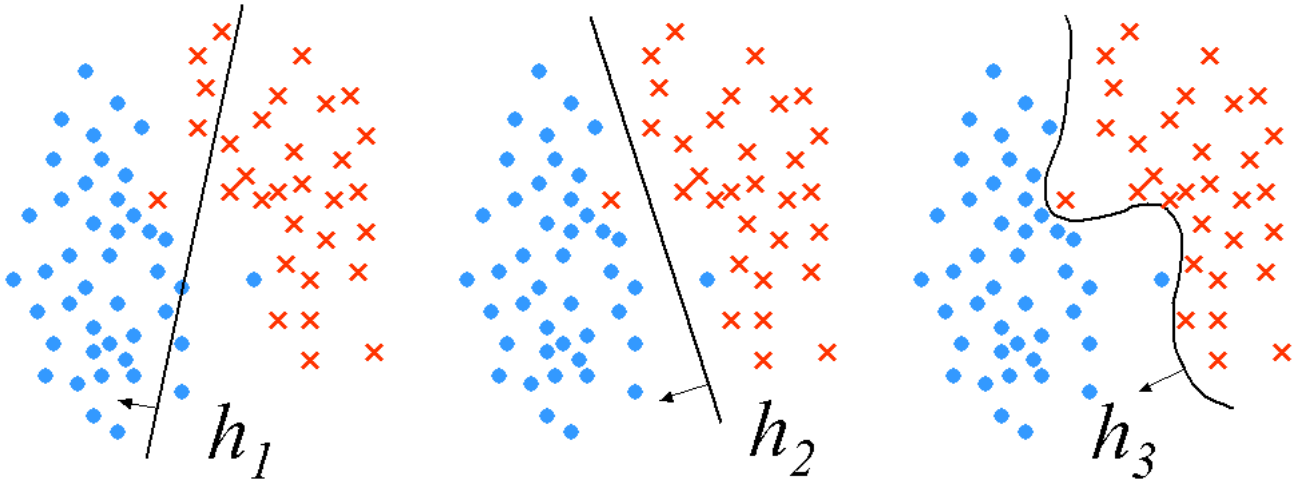
$$\text{sign}(\mathbf{w} \cdot \mathbf{x} + b) = \begin{cases} +1 & \text{if } (\mathbf{w} \cdot \mathbf{x} + b) > 0 \\ -1 & \text{otherwise} \end{cases}$$

Thus the goal of a learning algorithm that learns a linear classifier is to find a hypothesis $h \in \mathcal{H}_{lin(n)}$ with minimal *err*(h).

Example 1

$X = \mathbb{R}^2$
$Y = \{+1(\bullet), -1(\mathbf{x})\}$

Which classifier would you expect to generalize better?



Remember: want to minimize probability of error on future instances!

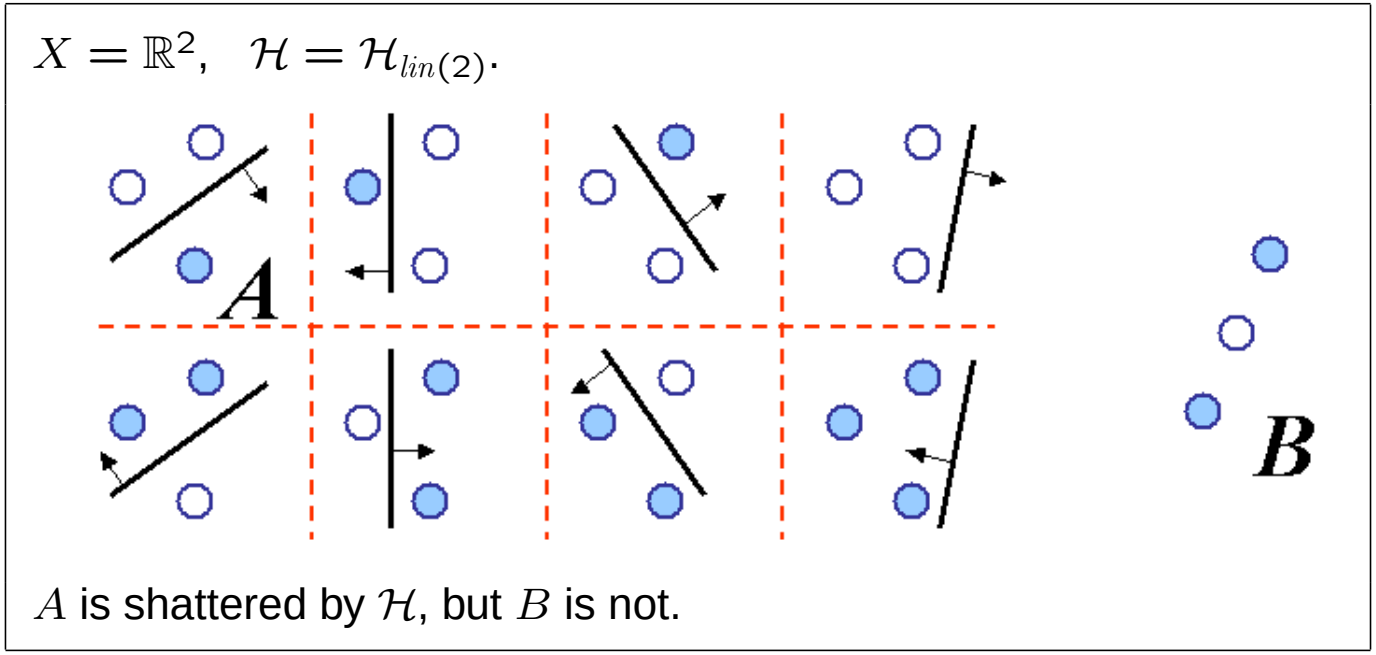
Intuitive Justification

- We expect the error on the training set, $err_S(h)$, to give some indication of the probability of error on a new instance, $err(h)$.
- We expect "simple" hypotheses to generalize better than more "complex" hypotheses.

Can we quantify the above ideas?

Complexity of a Hypothesis Space

A hypothesis space $\mathcal{H}$ over instances in X is said to **shatter** a set $A \subseteq X$ if *all possible dichotomies of* A (all +/- labelings of elements of A) can be represented by some hypothesis in $\mathcal{H}$.



A is shattered by $\mathcal{H}$, but B is not.

Complexity of a Hypothesis Space (continued)

The **VC dimension** of a hypothesis space $\mathcal{H}$, denoted by $\mathrm{VC}(\mathcal{H})$, is defined as the size of the *largest* subset of X that can be shattered by $\mathcal{H}$.

We saw that there exists a set of 3 points in $\mathbb{R}^2$ that can be shattered by $\mathcal{H}_{lin(2)}$.
No set of 4 points in $\mathbb{R}^2$ can be shattered by $\mathcal{H}_{lin(2)}$:

Thus, $\mathrm{VC}(\mathcal{H}_{lin(2)}) = 3$. In general, $\mathrm{VC}(\mathcal{H}_{lin(n)}) = n + 1$.

The VC dimension of $\mathcal{H}$ is a measure of the complexity of $\mathcal{H}$.

Generalization Bound from Learning Theory

Let $0 < \delta < 1$.

Given a training set S of size m and a classifier $h \in \mathcal{H}$, with probability $1 - \delta$ (over the choice of S) the following holds:

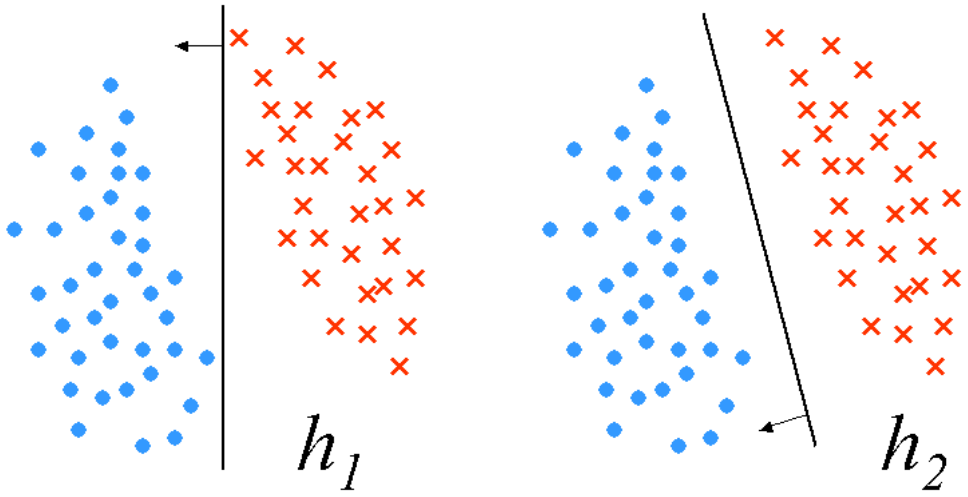
$$err(h) \leq err_S(h) + \sqrt{\frac{\mathrm{VC}(\mathcal{H})\left(\log(\frac{2m}{\mathrm{VC}(\mathcal{H})}) + 1\right) + \log(4/\delta)}{m}}$$

Example 2

$X = \mathbb{R}^2$

$Y = \{+1(\bullet), -1(\mathbf{x})\}$

Which of these classifiers would be likely to generalize better?



Example 2 (continued)

Recall the VC-based generalization bound:

$$err(h) \leq err_S(h) + \sqrt{\frac{\mathrm{VC}(\mathcal{H})\left(\log(\frac{2m}{\mathrm{VC}(\mathcal{H})}) + 1\right) + \log(4/\delta)}{m}}$$

In this case, we get the same bound for both classifiers:

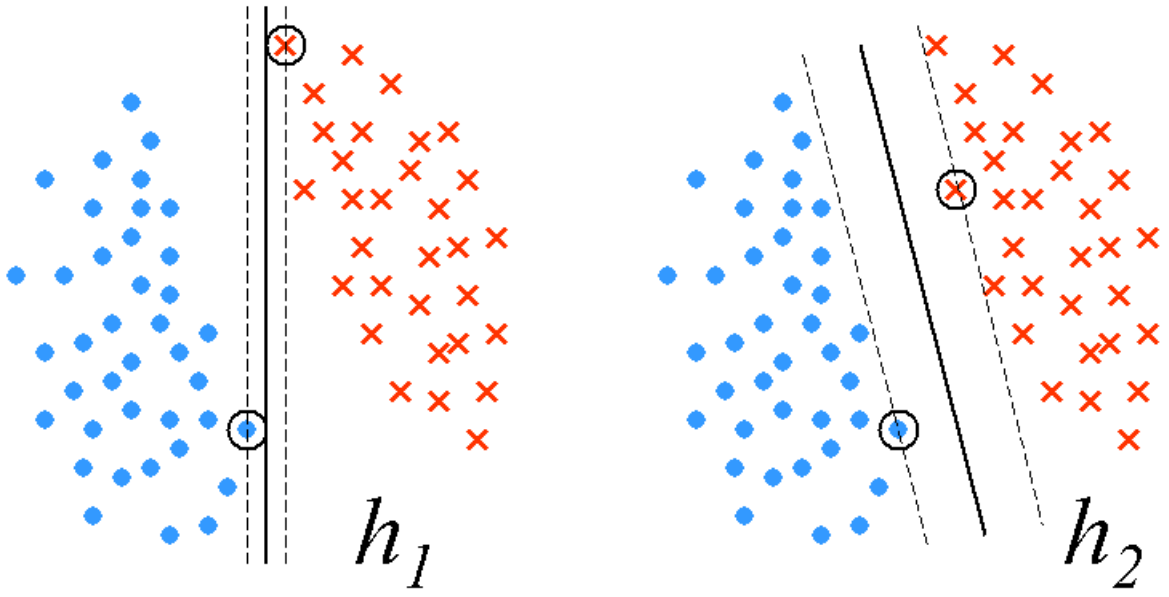
$$err_S(h_1) = err_S(h_2) = 0$$

$$h_1, h_2 \in \mathcal{H}_{lin(2)}, \quad \mathrm{VC}(\mathcal{H}_{lin(2)}) = 3$$

How, then, can we explain our intuition that h_2 should give better generalization than h_1?

Example 2 (continued)

Although both classifiers separate the data, the distance with which the separation is achieved is different:



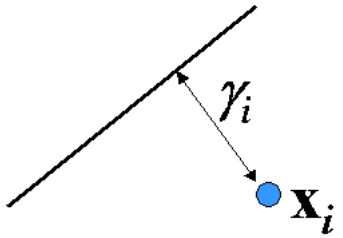
Concept of Margin

The **margin** γ_i of a point $\mathbf{x}_i \in \mathbb{R}^n$ with respect to a linear classifier $h(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$ is defined as the distance of $\mathbf{x}_i$ from the hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$:

$$\gamma_i = \left| \frac{\mathbf{w} \cdot \mathbf{x}_i + b}{\|\mathbf{w}\|} \right|$$

The margin of a set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ is defined as the margin of the point closest to the hyperplane:

$$\gamma = \min_{1 \leq i \leq m} \gamma_i = \min_{1 \leq i \leq m} \left| \frac{\mathbf{w} \cdot \mathbf{x}_i + b}{\|\mathbf{w}\|} \right|$$



Margin-Based Generalization Bound

If $\mathcal{H}$ is the space of all linear classifiers in $\mathbb{R}^n$ that separate the training data with margin at least γ, then

$$\mathrm{VC}(\mathcal{H}) \leq \min\left(\left\lceil \frac{R^2}{\gamma^2} \right\rceil, n\right) + 1,$$

where R is the radius of the smallest sphere (in $\mathbb{R}^n$) that contains the data.

Thus for such classifiers, we get a bound of the form

$$err(h) \leq err_S(h) + \sqrt{\frac{\mathcal{O}\left(\frac{R^2}{\gamma^2}\right) + \log(4/\delta)}{m}}$$

The SVM Algorithm (Linearly Separable Case)

Given training data $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^n$, the SVM algorithm finds a linear classifier that separates the data with maximal margin.

Without loss of generality, we can represent any linear classifier in $\mathbb{R}^n$ by some $\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}$ such that

$$\min_{1 \leq i \leq m} |\mathbf{w} \cdot \mathbf{x}_i + b| = 1. \tag{1}$$

The margin of the data with respect to the classifier is then given by

$$\gamma = \min_{1 \leq i \leq m} \left| \frac{\mathbf{w} \cdot \mathbf{x}_i + b}{\|\mathbf{w}\|} \right| = \frac{1}{\|\mathbf{w}\|}.$$

Maximizing the margin is therefore equivalent to minimizing the norm $\|\mathbf{w}\|$ of the classifier, subject to the constraint in Eq. (1) above.

Optimization Problem

$$\begin{array}{ll} \text{Minimize} & f(\mathbf{w}, b) \equiv \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} & y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, m \end{array}$$

This is an optimization problem in $(n + 1)$ variables, with m linear inequality constraints.

Introducing Lagrange multipliers α_i, $i = 1, \ldots, m$ for the inequality constraints above gives the primal Lagrangian:

$$\begin{array}{ll} \text{Minimize} & L_P(\mathbf{w}, b, \boldsymbol{\alpha}) \equiv \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] \\ \text{subject to} & \alpha_i \geq 0, \quad i = 1, \ldots, m \end{array}$$

Optimization Problem (continued)

Setting the gradients of L_P with respect to $\mathbf{w}, b$ equal to zero gives:

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \qquad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{i=1}^{m} \alpha_i y_i = 0$$

Substituting the above in the primal gives the following dual problem:

Maximize $$L_D(\boldsymbol{\alpha}) \equiv \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

subject to $$\sum_{i=1}^{m} \alpha_i y_i = 0; \quad \alpha_i \geq 0, \;\; i = 1, \ldots, m$$

This is a convex quadratic programming problem in $\boldsymbol{\alpha}$.

Solution

The parameters $\mathbf{w}, b$ of the maximal margin classifier are determined by the solution $\boldsymbol{\alpha}$ to the dual problem:

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

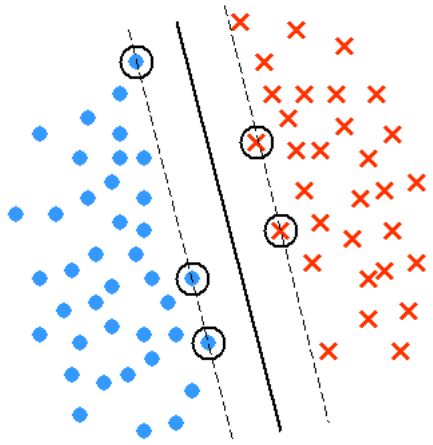
$$b = -\frac{1}{2}\left(\min_{y_i=+1} (\mathbf{w} \cdot \mathbf{x}_i) + \max_{y_i=-1} (\mathbf{w} \cdot \mathbf{x}_i) \right)$$

Support Vectors

Due to certain properties of the solution (known as the Karush-Kuhn-Tucker conditions), the solution α must satisfy

$$\alpha_i[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] = 0, \quad i = 1, \ldots, m.$$

Thus, $\alpha_i > 0$ only for those points x_i that are closest to the classifying hyperplane. These points are called the **support vectors**.



Non-Separable Case

Want to relax the constraints

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1.$$

Can introduce slack variables ξ_i:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i,$$

where $\xi_i \geq 0 \; \forall i$. An error occurs when $\xi_i > 1$.

Thus we can assign an extra cost for errors as follows:

Minimize	$f(\mathbf{w}, b, \boldsymbol{\xi}) \equiv \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \xi_i$
subject to	$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i; \quad \xi_i \geq 0, \quad i = 1, \ldots, m$

Non-Separable Case (continued)

Dual problem:

$$\text{Maximize} \quad L_D(\boldsymbol{\alpha}) \equiv \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{subject to} \quad \sum_{i=1}^{m} \alpha_i y_i = 0; \quad 0 \leq \alpha_i \leq C, \;\; i = 1, \ldots, m$$

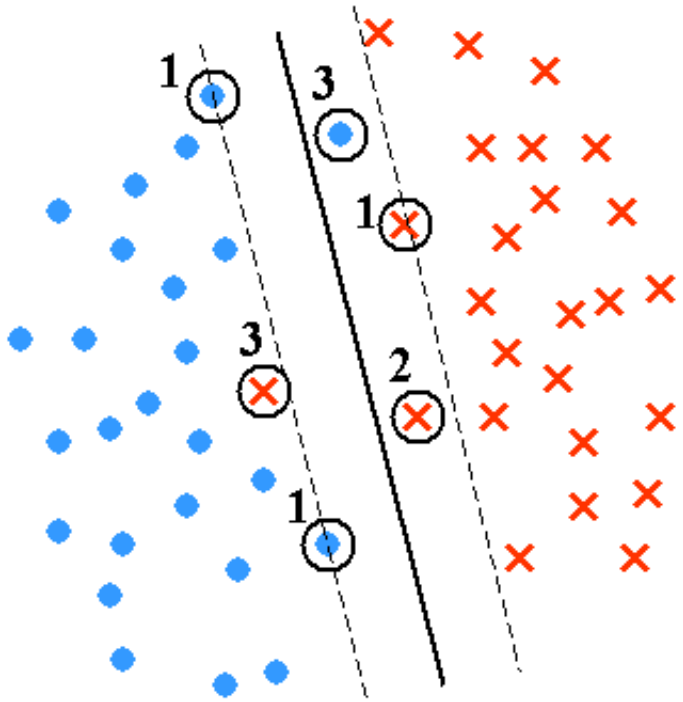
Solution:

The solution for $\mathbf{w}$ is again given by

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

The solution for b is similar to that in the linear case.

Visualizing the Solution in the Non-Separable Case



1.	Margin support vectors	$\xi_i = 0$	Correct
2.	Non-margin support vectors	$\xi_i < 1$	Correct (in margin)
3.	Non-margin support vectors	$\xi_i > 1$	Error

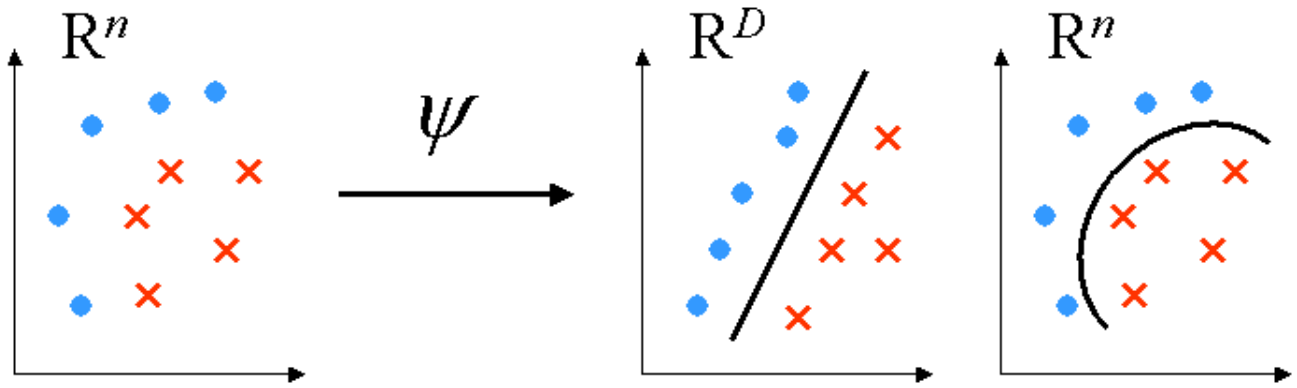
Non-Linear SVMs

Basic idea:

Map the given data to some (high-dimensional) feature space $\mathbb{R}^D$, using a non-linear mapping ψ:

$$\psi : \mathbb{R}^n \rightarrow \mathbb{R}^D.$$

Learn a linear classifier in the new space.



Dot Products and Kernels

Training phase:

Maximize $$L_D(\boldsymbol{\alpha}) \equiv \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{\psi}(\mathbf{x}_i) \cdot \boldsymbol{\psi}(\mathbf{x}_j))$$

subject to $$\sum_{i=1}^{m} \alpha_i y_i = 0; \quad 0 \leq \alpha_i \leq C, \;\; i = 1, \ldots, m$$

Test phase:

$$h(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \boldsymbol{\psi}(\mathbf{x}) + b), \quad \mathbf{w} \in \mathbb{R}^D, b \in \mathbb{R}.$$

Dot Products and Kernels (continued)

Recall the form of the solution:

$$\mathbf{w} = \sum_{i \in \mathrm{SV}} \alpha_i y_i \boldsymbol{\psi}(\mathbf{x}_i).$$

Therefore, the test phase can be written as

$$\begin{aligned} h(\mathbf{x}) &= \mathrm{sign}(\mathbf{w} \cdot \boldsymbol{\psi}(\mathbf{x}) + b) \\ &= \mathrm{sign}\left(\sum_{i \in \mathrm{SV}} \alpha_i y_i (\boldsymbol{\psi}(\mathbf{x}_i) \cdot \boldsymbol{\psi}(\mathbf{x})) + b \right). \end{aligned}$$

Thus both training and test phases use only dot products between images $\boldsymbol{\psi}(\mathbf{x})$ of points $\mathbf{x}$ in $\mathbb{R}^n$.

Dot Products and Kernels (continued)

A **kernel function** is a symmetric function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$.

A **Mercer kernel**, in addition, computes a dot product in some (high-dimensional) space:

$$K(\mathbf{x}, \mathbf{z}) = \boldsymbol{\psi}(\mathbf{x}) \cdot \boldsymbol{\psi}(\mathbf{z}),$$

for some $\boldsymbol{\psi}$, D such that $\boldsymbol{\psi} : \mathbb{R}^n \to \mathbb{R}^D$.

Thus if we can find a Mercer kernel that computes a dot product in the feature space we are interested in, we can use the kernel to replace the dot products in the SVM.

SVMs with Kernels

Training phase:

$$\text{Maximize} \quad L_D(\boldsymbol{\alpha}) \equiv \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to} \quad \sum_{i=1}^{m} \alpha_i y_i = 0; \quad 0 \leq \alpha_i \leq C, \;\; i = 1, \ldots, m$$

Test phase:

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i \in \text{SV}} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \right).$$

Example: Quadratic Kernel

Let $X = \mathbb{R}^2$, and consider learning a quadratic classifier in this space:

$$h(\mathbf{x}) = \text{sign}(w_1 x_1^2 + w_2 x_2^2 + w_3 x_1 x_2 + w_4 x_1 + w_5 x_2 + b)$$

This is equivalent to learning a linear classifier in the feature space $\psi(\mathbb{R}^2)$, where

$$\psi : \mathbb{R}^2 \to \mathbb{R}^5, \qquad \psi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_1 \\ x_2 \end{bmatrix}.$$

Without the use of a kernel, learning such a classifier using an SVM would require computing dot products in $\mathbb{R}^5$.

Example: Quadratic Kernel (continued)

Consider the kernel

$$K : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}, \qquad K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^2$$

It can be verified that

$$K(\mathbf{x}, \mathbf{z}) = \boldsymbol{\psi}'(\mathbf{x}) \cdot \boldsymbol{\psi}'(\mathbf{z}),$$

where

$$\boldsymbol{\psi}' : \mathbb{R}^2 \to \mathbb{R}^6, \qquad \boldsymbol{\psi}'\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ 1 \end{bmatrix}$$

Thus, an SVM with the above kernel can be used to learn a quadratic classifier in $\mathbb{R}^2$ using only dot products in $\mathbb{R}^2$.

Some Commonly Used Kernels

$\mathbf{x}, \mathbf{z} \in X = \mathbb{R}^n$

Polynomial kernels of degree d:

$$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^d$$

Gaussian kernels:

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

Such kernels are used to efficiently learn a linear classifier in a high-dimensional space using computations in only the original, lower-dimensional space. The Gaussian kernel corresponds to a dot product in an infinite dimensional space.

Kernels over Structured Data

Kernels can also be defined for non-vectorial data, i.e.

$$K : X \times X \rightarrow \mathbb{R}$$

where X is a space other than $\mathbb{R}^n$.

A kernel $K(\mathbf{x}, \mathbf{z})$ over $\mathbb{R}^n$ that computes dot products in some space can be viewed as computing some sort of similarity measure between data elements $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$. Therefore an appropriate similarity measure between elements in any space X can be used to define a kernel over X.

Such kernels have been defined, for example, for data represented as trees or strings; SVMs can therefore be used to learn classifiers for such types of data.

Summary

- The SVM algorithm learns a linear classifier that maximizes the margin of the training data.
- Training an SVM consists of solving a quadratic programming problem in m variables, where m is the size of the training set.
- An SVM can learn a non-linear classifier in the original space through the use of a kernel function, which simulates dot products in a (high-dimensional) feature space.

Current Research

- Much work on efficient methods for finding approximate solutions to the quadratic programming problem, especially for large datasets.
- Multitude of new kernels for different types of structured data.
- Work on trying to optimize the margin distribution over all training points, rather than optimizing the margin of only the points closest to the separating hyperplane.