Support Vector Machines for Classification

Charles Nicholson

DSA/ISE 5103

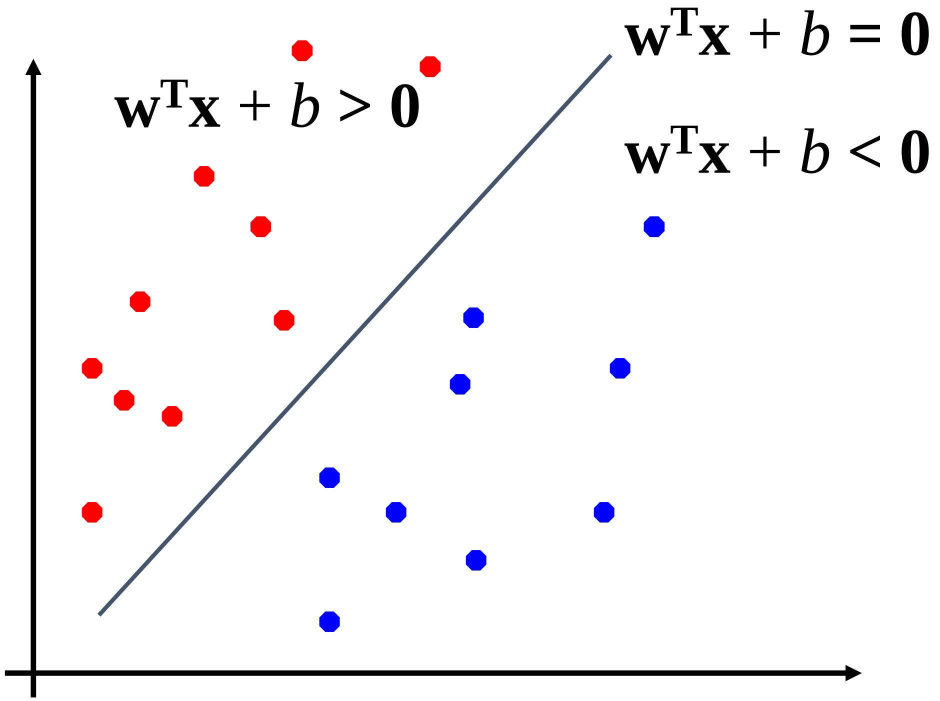
Characteristics of Support Vector Machines

- Can be used for regression or classification
- Linear and non-linear relationships can be learned
- Computationally more taxing than other methods
- Requires the training data to be available (or at least some of it) to make predicts
- Can be VERY good classifier if TUNED well
- SVM has been used successfully in many real-world problems
    - text (and hypertext) categorization
    - image classification
    - bioinformatics (protein classification, cancer classification)
    - hand-written character recognition

$$\boxed{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = 0}$$

Linear Separators

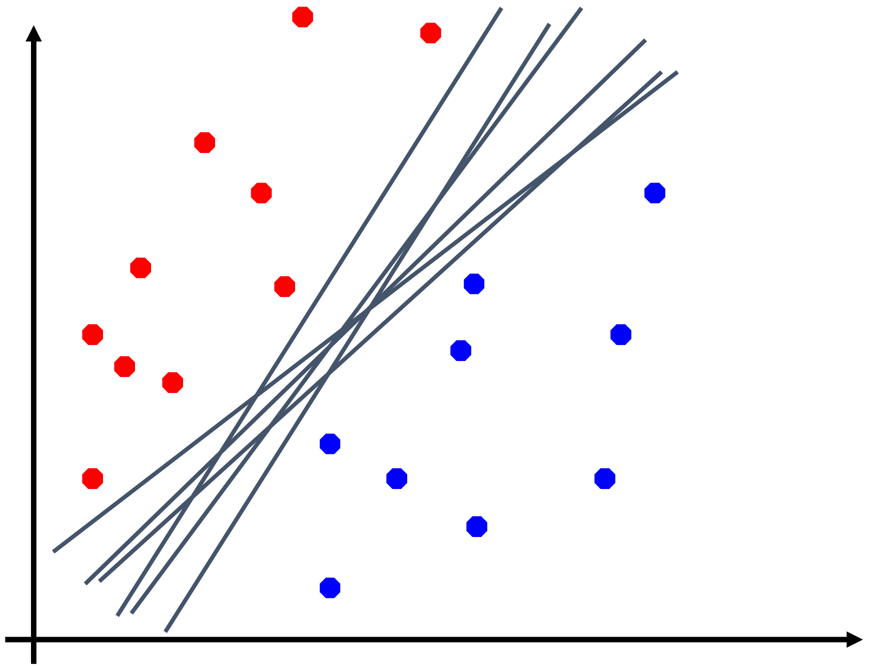
Binary classification can be viewed as the task of separating classes in feature space



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

Linear Separators

Which of the linear separators is optimal?

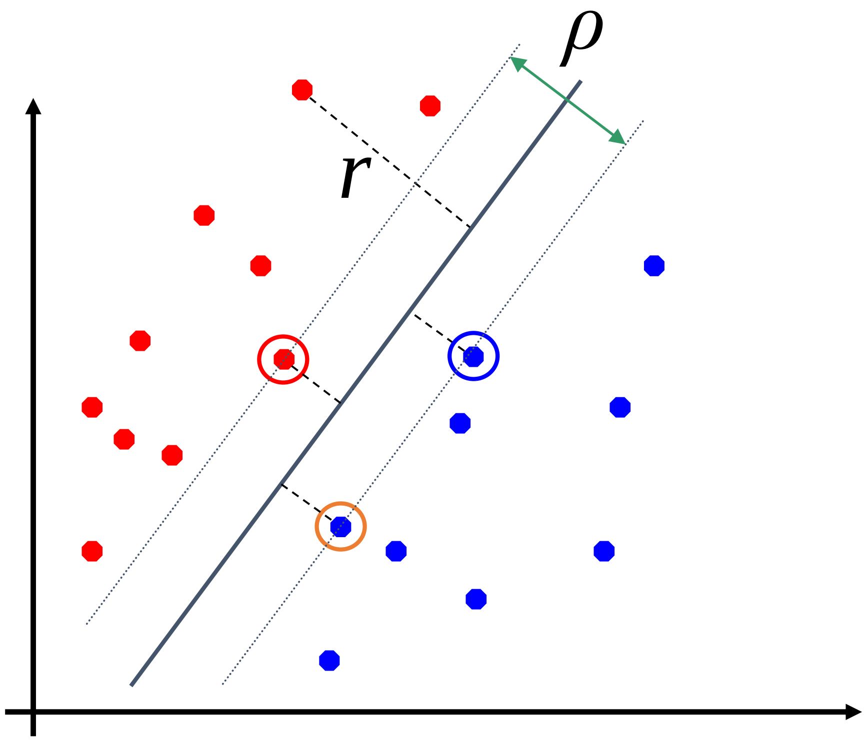


Classification Margin

- Distance from example $\mathbf{x}_i$ to the separator is

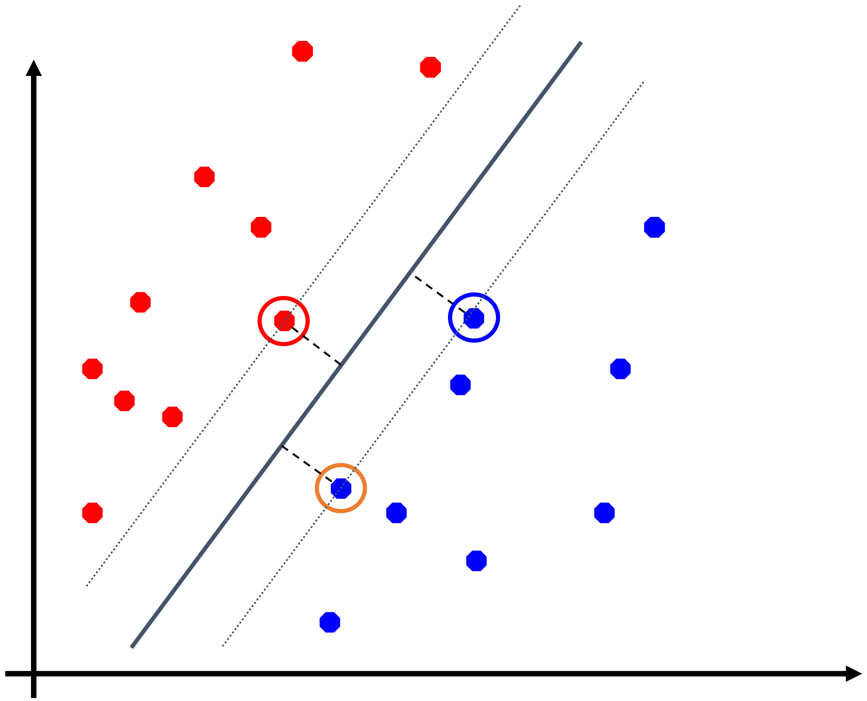
$$r = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$$

- Observations closest to hyperplane are ***support vectors***
- ***Margin*** ρ of the separator is the distance between support vectors



Maximum Margin Classification

- Maximizing the margin is good according to intuition and *probably approximately correct learning* (PAC) theory
- Implies that only support vectors matter; other training examples are ignorable.



Linear SVM Mathematically

- 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 hyperplane with margin ρ
- Then, for each training example $(\mathbf{x}_i, y_i)$:

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

- For every support vector $\mathbf{x}_s$ the above inequality is an equality
- After rescaling $\mathbf{w}$ and b by $\rho/2$ in the equality, we obtain that distance between each $\mathbf{x}_s$ and the hyperplane is

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

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

Linear SVMs Mathematically

Then we can formulate the *quadratic optimization problem:*

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

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

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

Which can be reformulated as:

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

$$\boldsymbol{\Phi}(\boldsymbol{w}) = ||\boldsymbol{w}||^2 = \boldsymbol{w}^T\boldsymbol{w} \text{ is minimized}$$

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

Solving the Optimization Problem

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

- Need to optimize a *quadratic* function subject to *linear* constraints
- Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every inequality constraint in the primal (original) problem:

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

Optimization Problem Solution

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

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

- Each non-zero α_i indicates that corresponding $\mathbf{x}_i$ is a support vector
- Then the classifying function is (note that we don't need **w** explicitly):

$$\boxed{f(\boldsymbol{x}) \;=\; \Sigma \alpha_i y_i \boldsymbol{x}_i^{\boldsymbol{T}} \boldsymbol{x} \;+\; b}$$

- Notice that it relies on an *inner product* between the test point $\mathbf{x}$ (new data) and the support vectors $\mathbf{x}_i$
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j$ between all training points

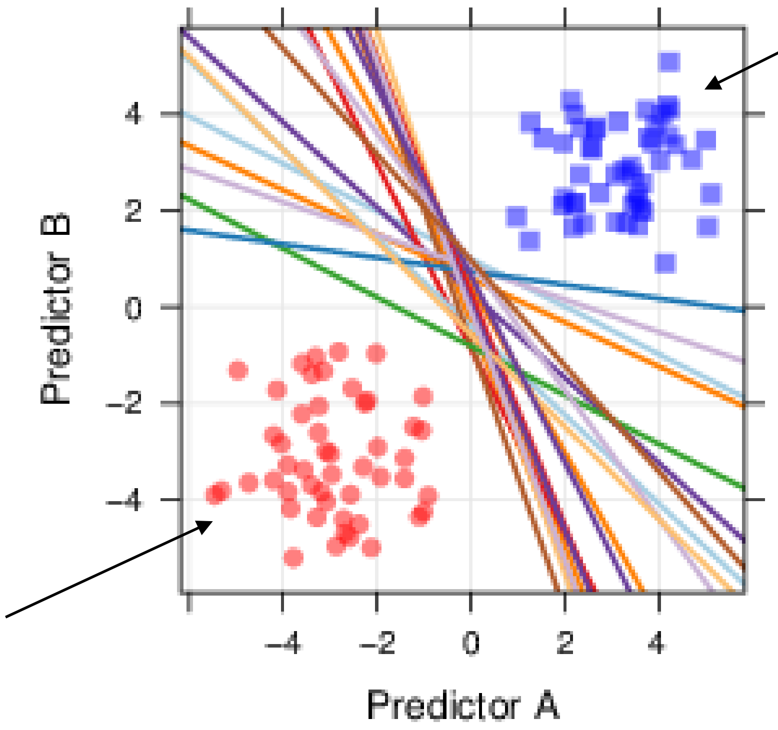
Example from "Applied Predictive Modeling"
(Ch.13)

- Perfectly separable data
- 3 support vectors identified

Note: The actual numeric positions of the SV's were not provided in the textbook, but I've estimated them in such a way to reproduce the math.

Also, the intercept b_0 seems to be incorrect in the text; so we will say that b_0 = -4.326

Class 2
$y_i = -1$



Class 1
$y_i = 1$

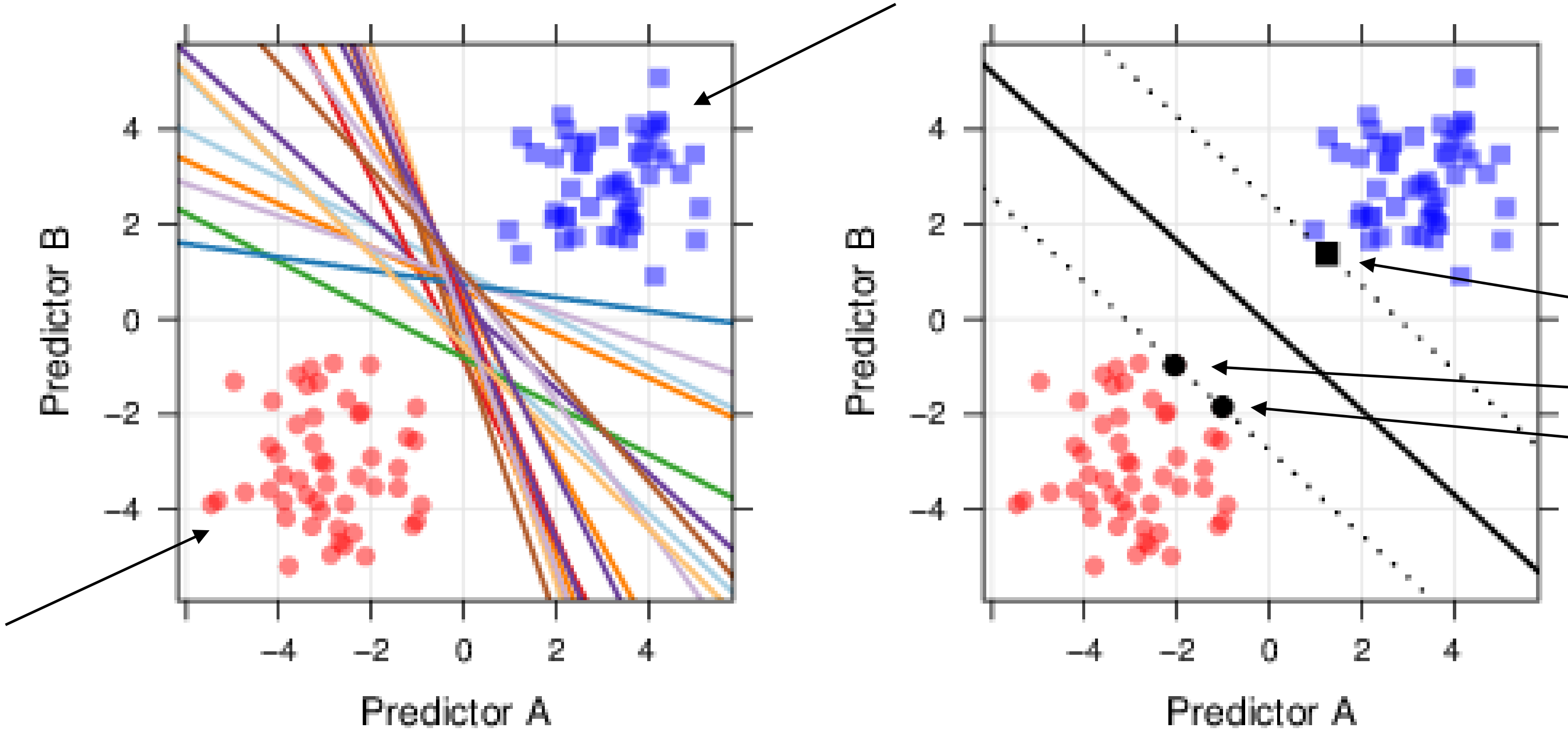
Example from "Applied Predictive Modeling"
(Ch.13)

- Perfectly separable data
- 3 support vectors identified

Note: The actual numeric positions of the SV's were not provided in the textbook, but I've estimated them in such a way to reproduce the math.

Also, the intercept b_0 seems to be incorrect in the text; so we will say that b_0 = -4.326

Class 2
$y_i = -1$



Class 1
$y_i = 1$

SV 1 : (1.3,1.5)

SV 2 : (-2.0,-95)

SV 3: (-1, -1.8)

Example from "Applied Predictive Modeling"

Prediction of new point: (-3,1)

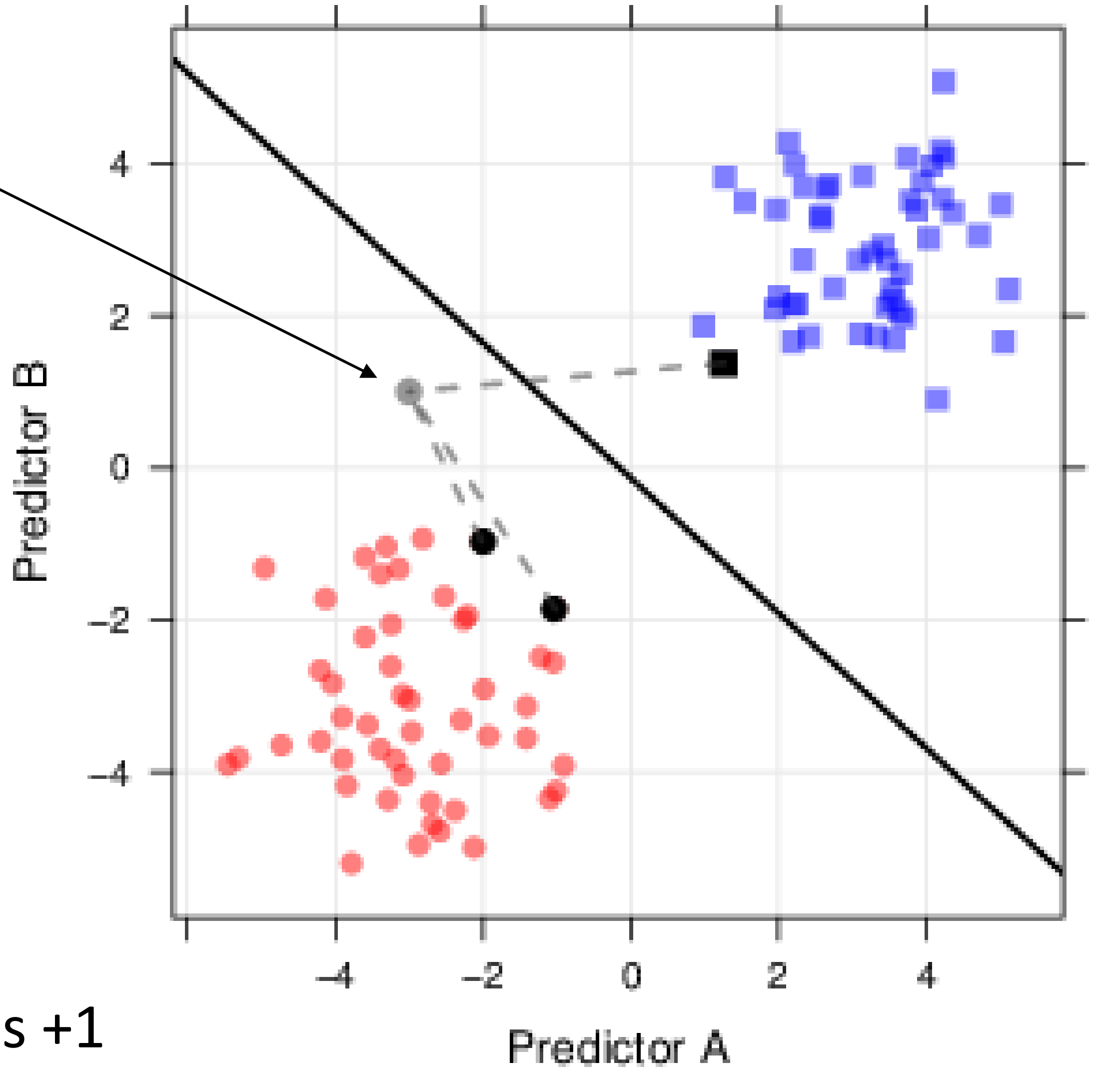
$$f(\boldsymbol{x}) = \Sigma \alpha_i y_i \boldsymbol{x}_i{}^T \boldsymbol{x} + \mathrm{b}_0$$

	α_i	y_i	SV	New Data	Inner Product	Product
SV 1	1.00	-1	(1.3,1.5)	(-3, 1)	-2.4	2.4
SV 2	0.34	1	(-2,-.95)	(-3, 1)	5.05	1.72
SV 3	0.66	1	(-1,-1.8)	(-3, 1)	1.2	0.79

b_0 = -4.326

So: -4.326 - 2.4 + 1.72 + 0.79 = 0.583

Since 0.583 > 0, the new data point is classed as +1

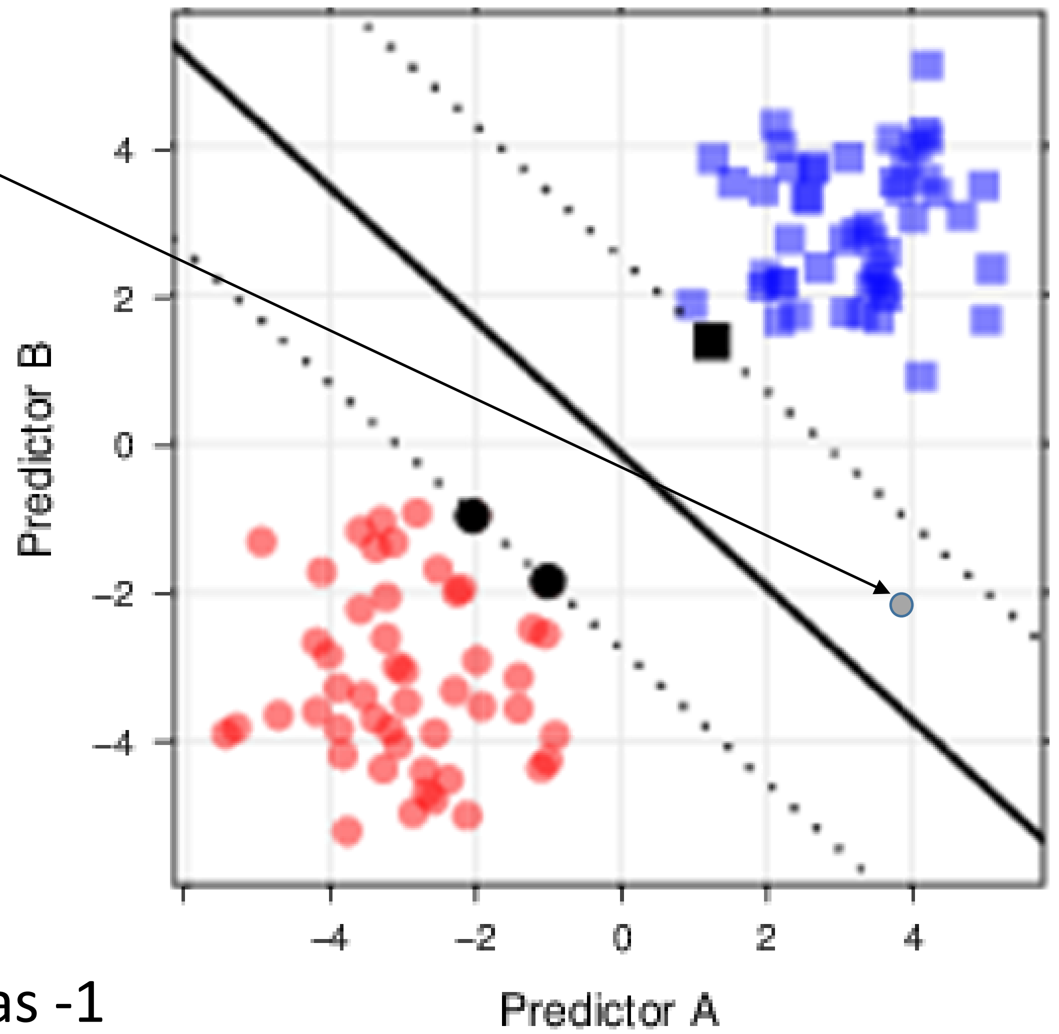


Example from "Applied Predictive Modeling"

Prediction of new point: (4,-2)

$$f(\boldsymbol{x}) = \Sigma \alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{x} + b_0$$

	α_i	y_i	SV	New Data	Inner Product	Product
SV 1	1.00	-1	(1.3,1.5)	(4, -2)	2.2	-2.2
SV 2	0.34	1	(-2,-.95)	(4, -2)	-6.1	-6.1
SV 3	0.66	1	(-1,-1.8)	(4, -2)	-0.4	-0.4



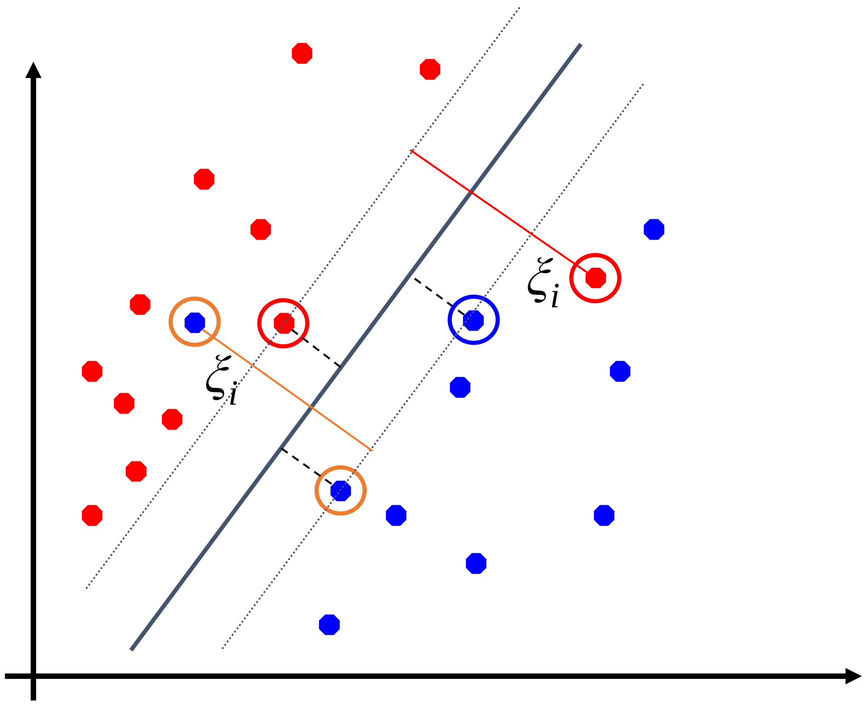
b_0 = -4.326

So: -4.326 -2.2 -6.1 -0.4 = -13.026

Since -13.026 < 0, the new data point is classed as -1

Soft Margin Classification

- What if the training set is not completely separable?
- *Slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples, resulting in a so-called *soft margin*.



Soft Margin Classification Mathematically

- The old formulation:

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

- Modified formulation incorporates slack variables:

Find **w** and b such that
$\boldsymbol{\Phi}(\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{w} + C\Sigma\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, i=1..n : $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_{i,}$, $\xi_i \geq 0$

- Parameter *C* helps control overfitting: it trades off the relative importance of maximizing the margin and fitting the data
- For large C, the optimization will choose a smaller-margin hyperplane if the improves accuracy.
- A small C will result in larger-margin separating hyperplane, even if that incurs misclassifications

Soft Margin Classification – Solution

- Dual problem is identical to separable case

Find $\alpha_1 \ldots \alpha_N$ such that
$\boldsymbol{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - ½\Sigma\Sigma\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^{\boldsymbol{T}}\boldsymbol{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

- Again, $\mathbf{x}_i$ with non-zero α_i will be support vectors.
- Solution to the dual problem is:

$$\boldsymbol{w} = \Sigma\alpha_i y_i \boldsymbol{x}_i$$
$$b = y_k(1 - \xi_k) - \Sigma\alpha_i y_i \boldsymbol{x}_i^{\boldsymbol{T}}\boldsymbol{x}_k \quad \text{for any } k \text{ s.t. } \alpha_k>0$$

Again, we don't need to compute **w** explicitly for classification:

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

Linear SVMs: Overview

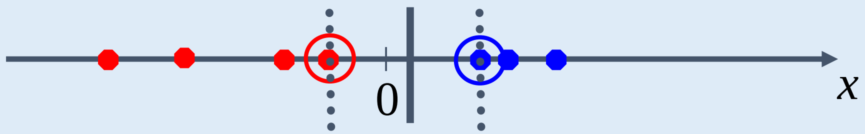
- The classifier is a *separating hyperplane.*
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points $\mathbf{x}_i$ are support vectors with non-zero Lagrangian multipliers α_i.
- Both in the dual formulation of the problem and in the solution training points appear only inside inner products:

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

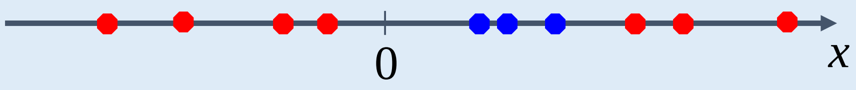
$$f(\mathbf{x}) = \Sigma\alpha_i y_i \boxed{\mathbf{x}_i^{\mathrm{T}}}\mathbf{x} + b$$

Non-linear SVMs

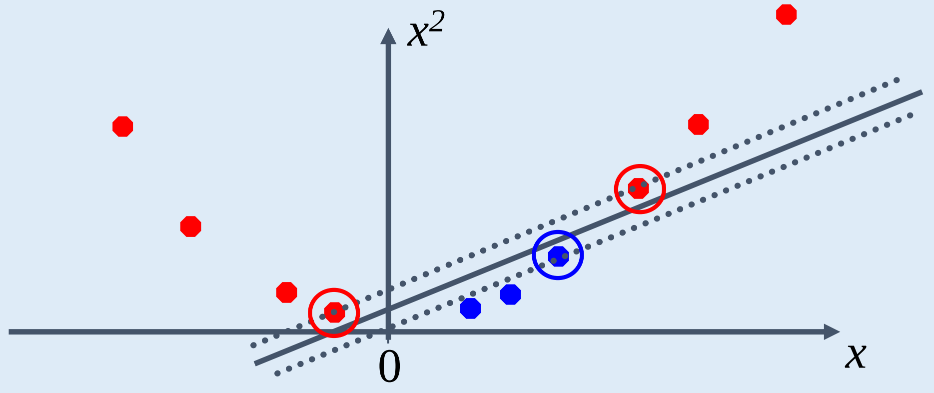
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?

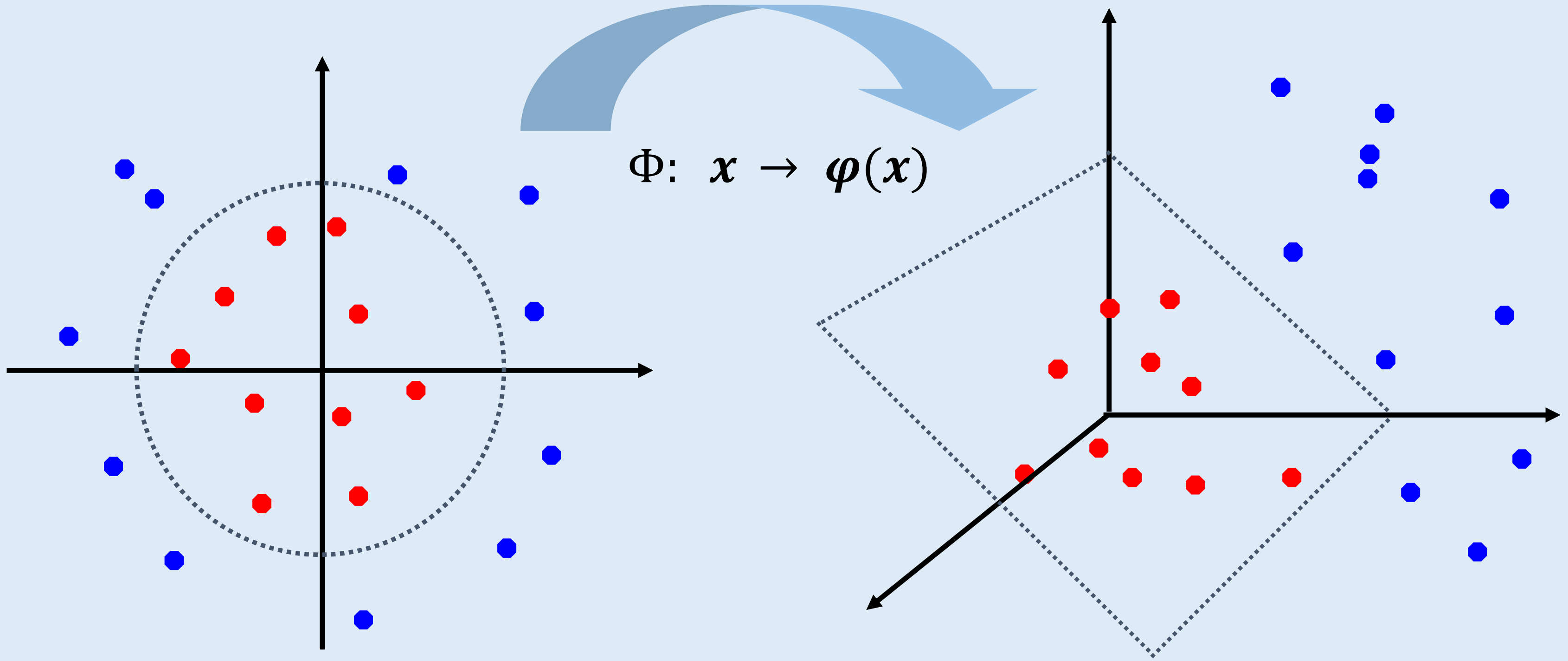


- How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

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



The "Kernel Trick"

- The linear classifier relies on inner product between vectors $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^{\boldsymbol{T}} \boldsymbol{x}_j$

- If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$, the inner product becomes:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j)$$

A *kernel function* is a function that is equivalent to a dot product in some (most likely, higher-dimensional) feature space.

The "Kernel Trick"

Example:

Consider the 2-dimensional vectors $\boldsymbol{a} = [a_1 \;\; a_2]$ and $\boldsymbol{b} = [b_1 \;\; b_2]$

And let's examine the well known quadratic kernel:

$$K(\boldsymbol{a}, \boldsymbol{b}) = (1 + \boldsymbol{a}^T \boldsymbol{b})^2$$

$K(\boldsymbol{a}, \boldsymbol{b})$ can be shown to be the result of a dot-product between a higher dimensional representation of $\boldsymbol{a}$ and $\boldsymbol{b}$

Need to show that $K(\boldsymbol{a}, \boldsymbol{b}) = \varphi(\boldsymbol{a})^T \varphi(\boldsymbol{b})$

The "Kernel Trick"

$$K(\boldsymbol{a},\boldsymbol{b}) = (1+\boldsymbol{a}^T\boldsymbol{b})^2$$
$$= \ 1 + {a_1}^2{b_1}^2 + {a_2}^2{b_2}^2 + 2a_1b_1 \ + \ 2a_2b_2 + 2\,a_1b_1\,a_2b_2$$

This would be the result of the dot product of these two 6d vectors:

$$[1, a_1^2, a_2^2, \sqrt{2}a_1, \sqrt{2}a_2, \sqrt{2}a_1a_2]$$
$$[1, b_1^2, b_2^2, \sqrt{2}b_1, \sqrt{2}b_2, \sqrt{2}b_1b_2]$$

So, $\varphi(\mathbf{x}) = \varphi(x_1, x_2) = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2)$

And thus, $K(\boldsymbol{a},\boldsymbol{b}) = (1+\boldsymbol{a}^T\boldsymbol{b})^2 = \varphi(\boldsymbol{a})^T\varphi(\boldsymbol{b})$

Examples of Kernel Functions

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

- Linear: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^{\boldsymbol{T}} \boldsymbol{x}_j$
  - Mapping Φ: $\boldsymbol{x} \rightarrow \boldsymbol{\varphi}(\boldsymbol{x})$, where $\boldsymbol{\varphi}(\boldsymbol{x})$ is $\boldsymbol{x}$ itself
- Polynomial of power p: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i^{\boldsymbol{T}} \boldsymbol{x}_j)^p$
- Gaussian (radial-basis function): $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right)$
- Sigmoid: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\beta_0 \boldsymbol{x}_i^{\boldsymbol{T}} \boldsymbol{x}_j + \beta_1)$

Non-linear SVMs Mathematically

- Dual problem formulation:

Find $\alpha_1 \ldots \alpha_n$ such that
$\boldsymbol{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - ½\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

- The solution is:

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

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

Non Linear SVMs: Overview

- SVM locates a separating hyperplane in the feature space and classify points in that space
- It does not need to represent the space explicitly, simply by defining a kernel function
- The kernel function plays the role of the dot product in the feature space.

Some Notes/Issues

- Probably should scale data (e.g., z-score scaling) prior to training

- Choice of kernel
  - Gaussian (RBF) or polynomial kernel is default
  - if ineffective, more elaborate kernels are needed

- Choice of kernel parameters
  - e.g., σ in Gaussian kernel
  - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters → You NEED to TUNE!

Packages in R

For SVM:

- e1701
- kernlab
- LiblineaR