Lecture 17: Support Vector Classifer & Kernel Methods

BIOS635

03/24/2020

Some slides taken/modified from other people's lectures online.

Support vector machines

Here we approach the two-class classification problem in a direct way:

> *We try and find a plane that separates the classes in feature space.*

If we cannot, we get creative in two ways:

- We soften what we mean by “separates”, and
- We enrich and enlarge the feature space so that separation is possible.

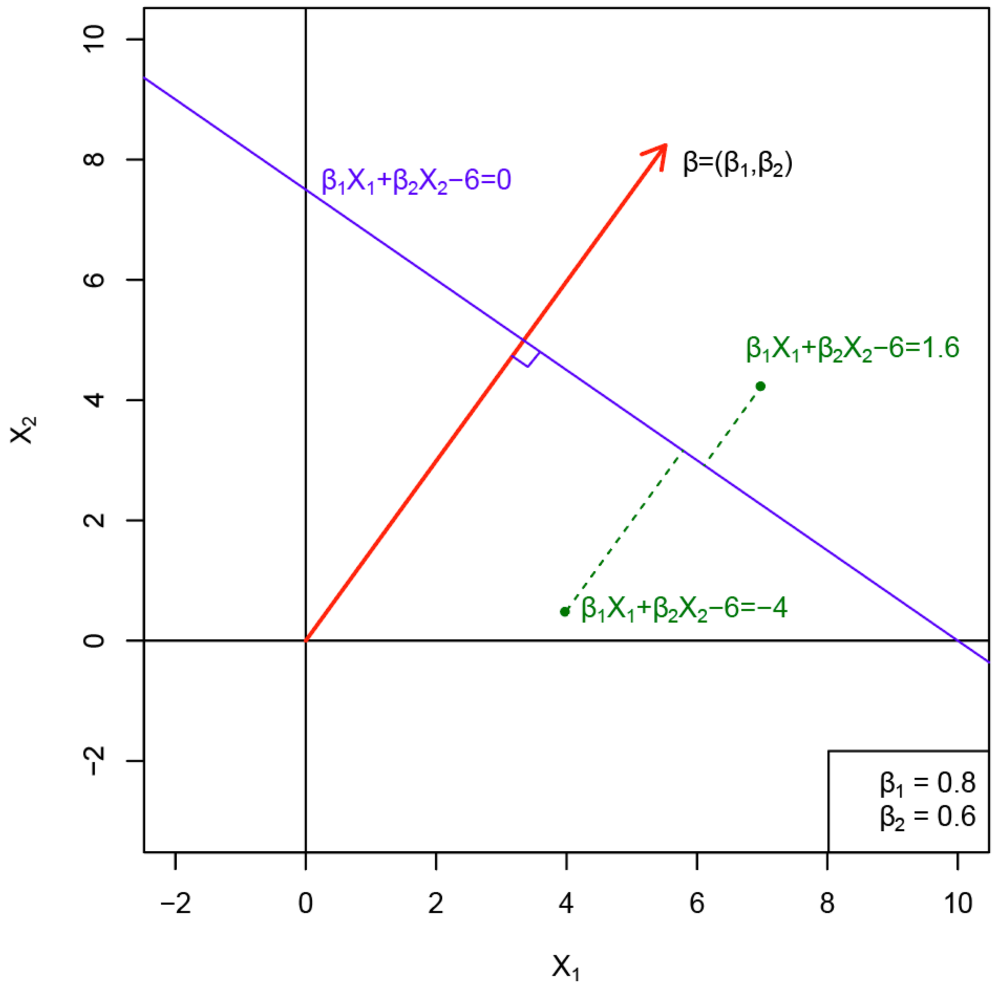
What is a hyperplane?

- A hyperplane in p dimensions is a flat affine subspace of dimension $p-1$.
- In general the equation for a hyperplane has the form

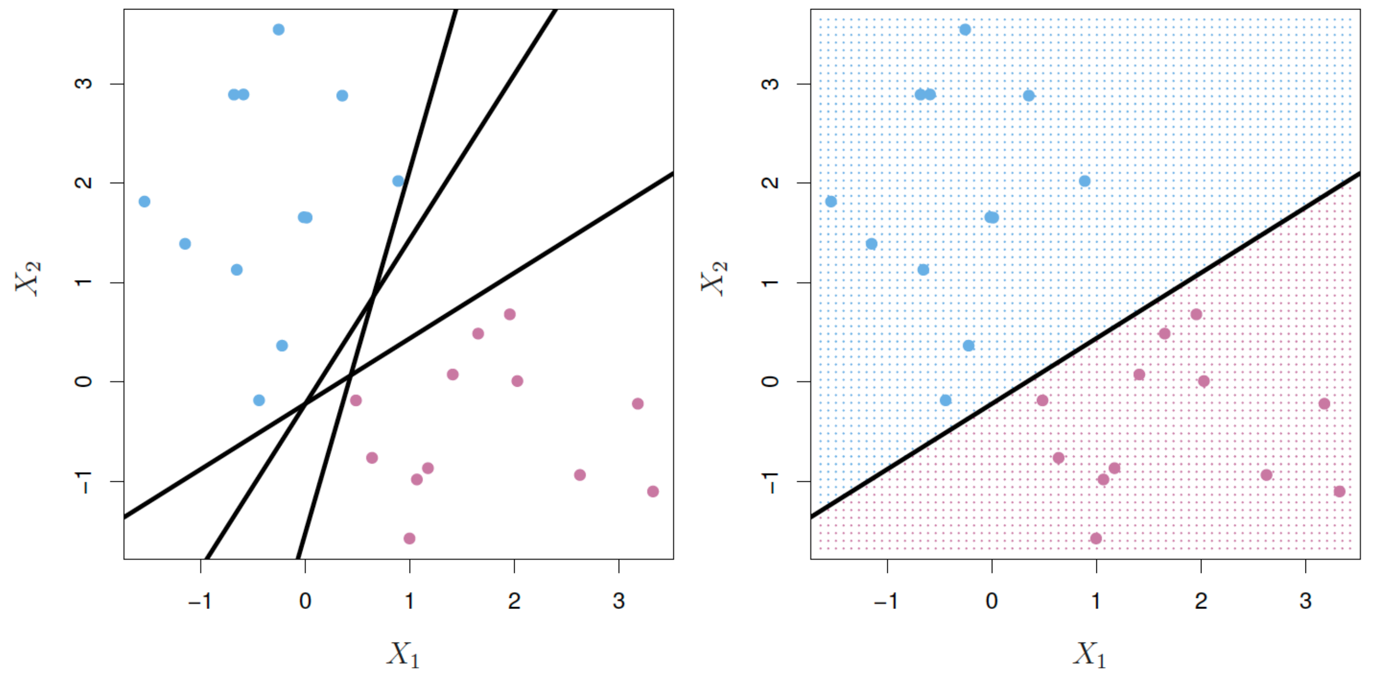
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

- In $p = 2$ dimensions a hyperplane is a line.
- If $\beta_0 = 0$, the hyperplane goes through the origin, otherwise not.
- The vector $\beta = (\beta_1, \beta_2, \cdots, \beta_p)$ is called the normal vector — it points in a direction orthogonal to the surface of a hyperplane.

Hyperplane in 2 dimensions



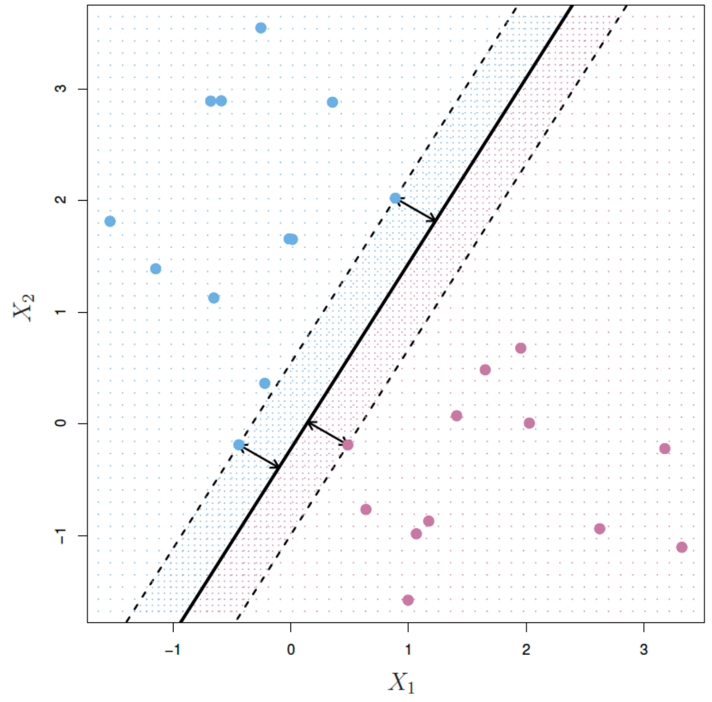
Separating hyperplanes



- If $f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$, then $f(X) > 0$ for points on one side of the hyperplane, and $f(X) < 0$ for points on the other.
- If we code the colored points as $Y_i = +1$ for blue, say, and $Y_i = -1$ for mauve, then if $Y_i \cdot f(X_i) > 0$ for all i, $f(X) = 0$ defines a *separating hyperplane*.

Maximal margin classifier

Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes.



Constrained optimization problem

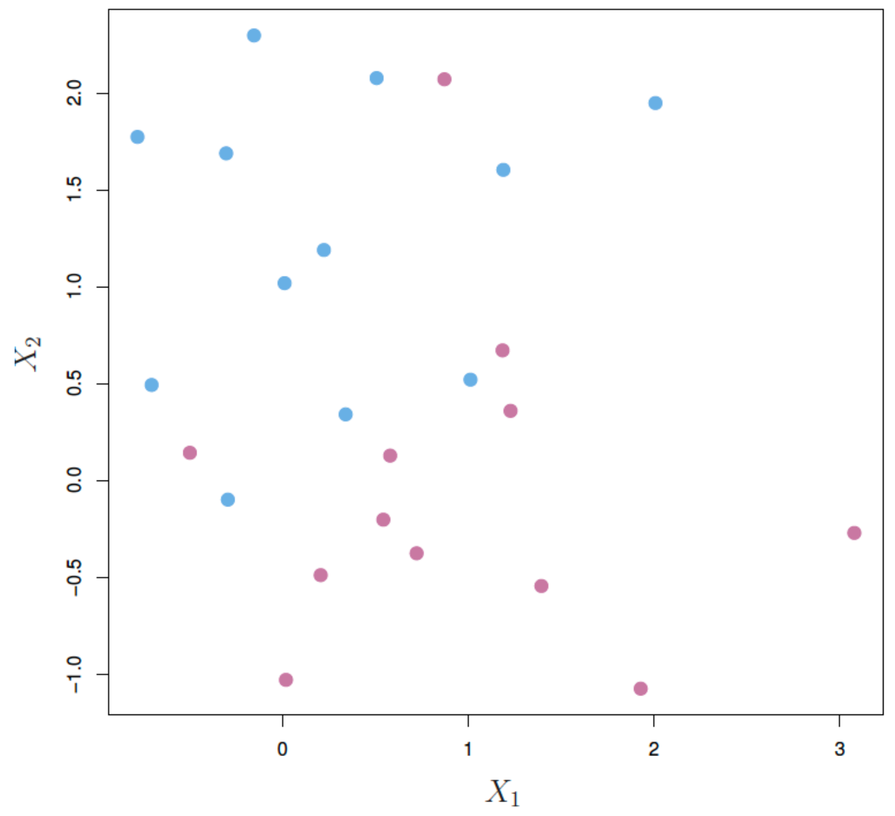
$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\, M$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M$$
$$\text{for all } \; i = 1, \ldots, N.$$

This can be rephrased as a convex quadratic program, and solved efficiently. The function `svm()` in package `e1071` solves this problem efficiently

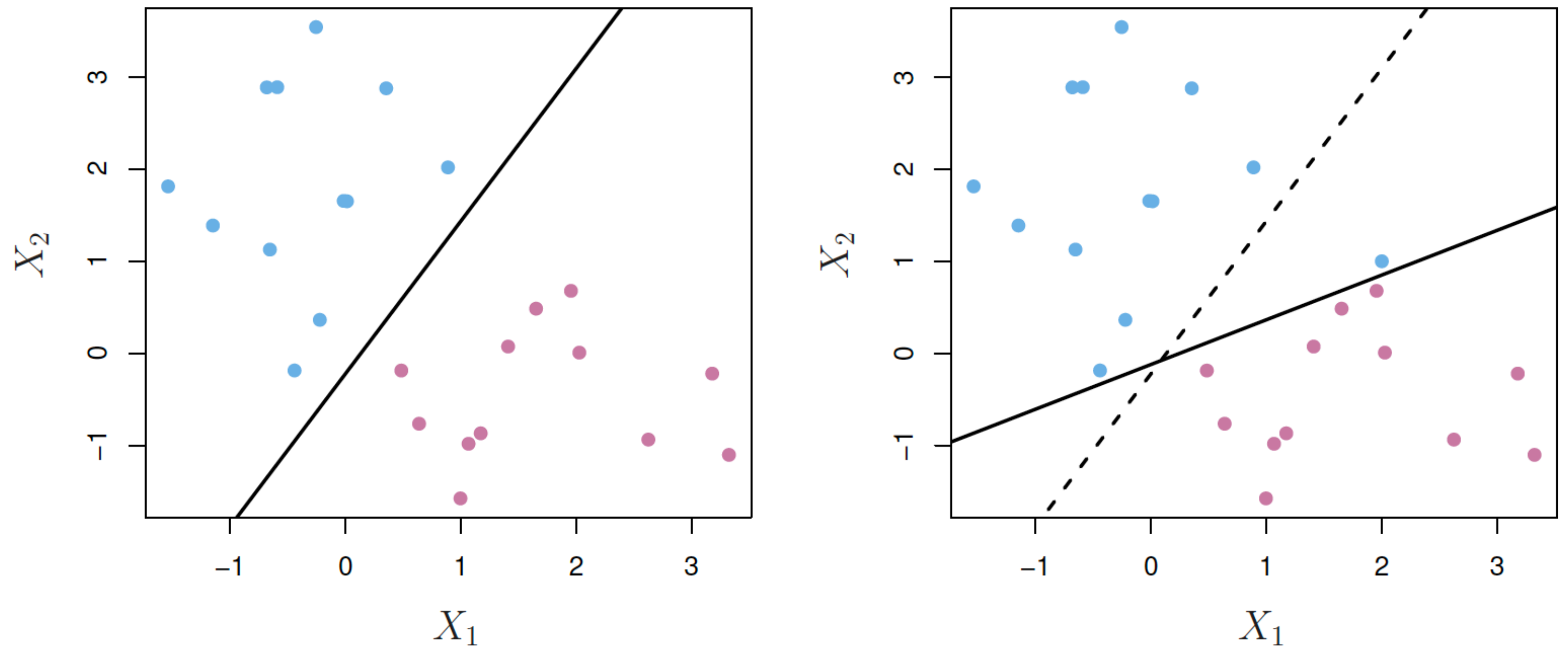
Non-separable data



The data on the left are not separable by a linear boundary.

This is often the case, unless $N < p$.

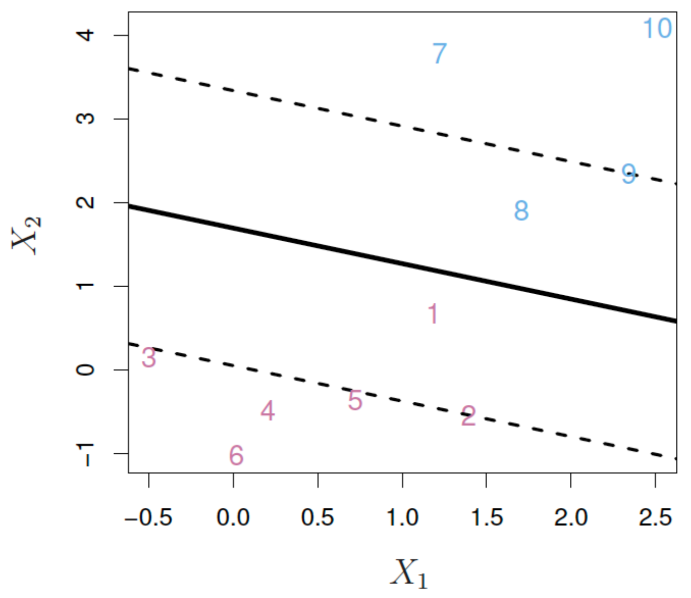
Noisy data

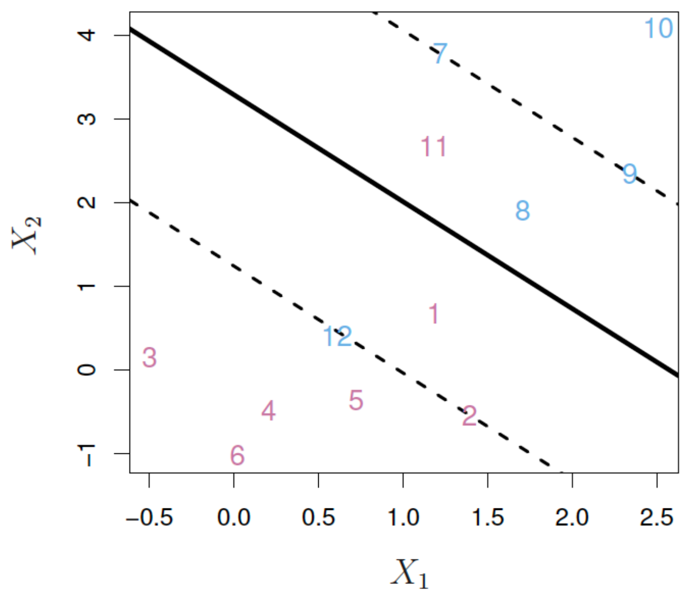


Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier.

The *support vector classifier* maximizes a *soft* margin.

Support vector classifier



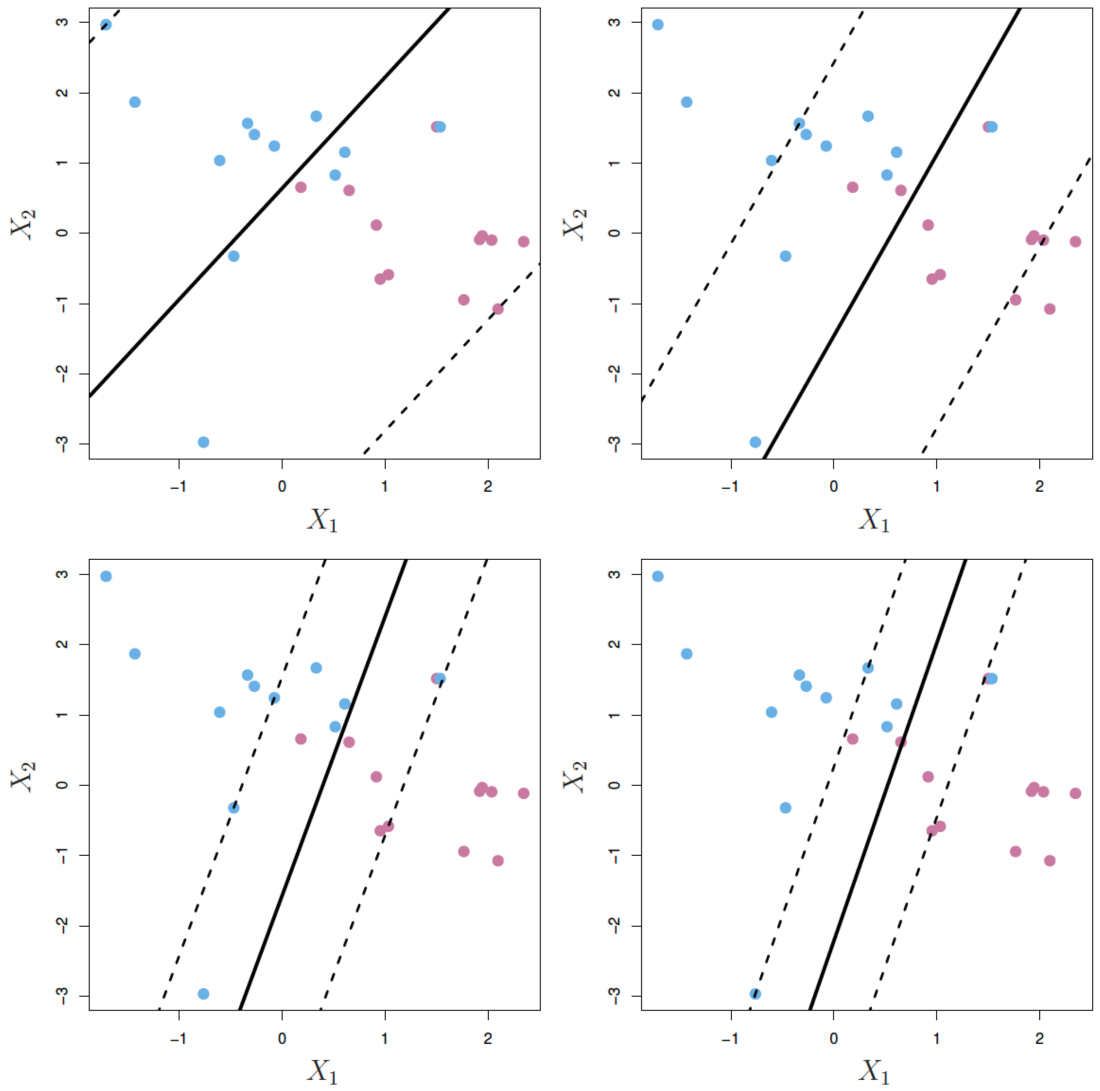


$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

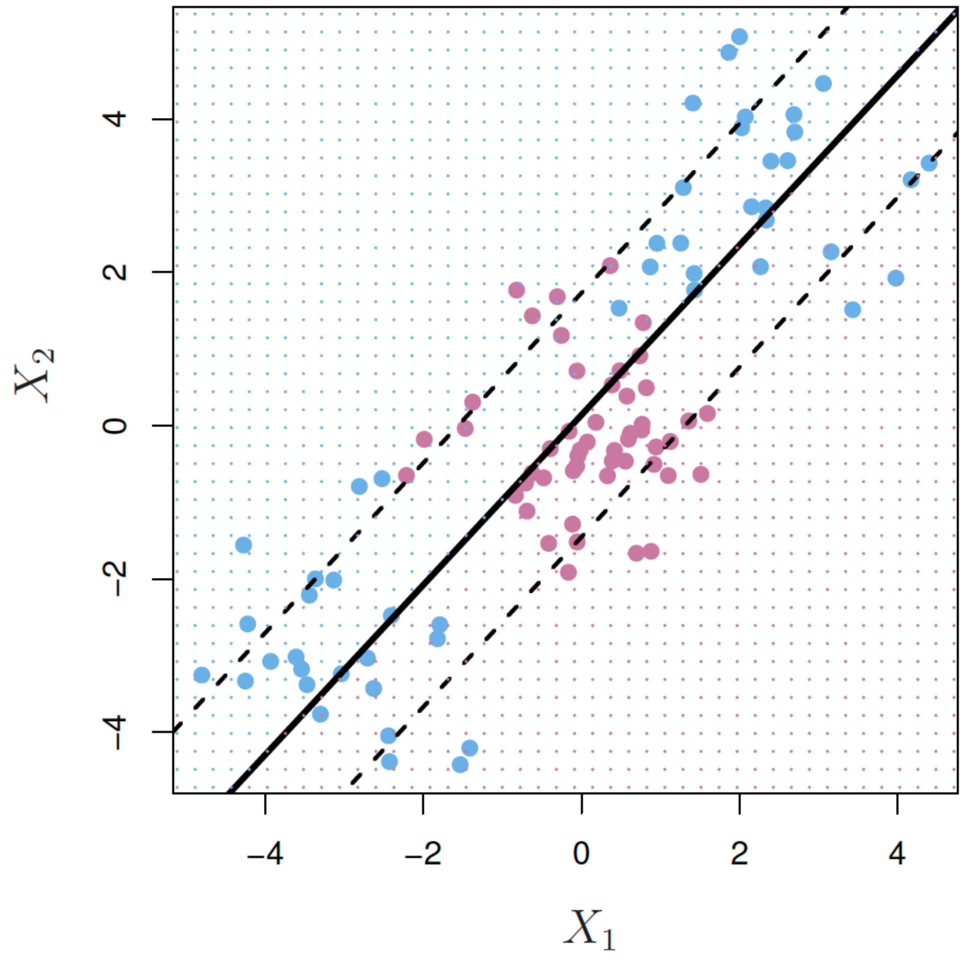
$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

C is a regularization parameter



- Larger C: higher bias, lower variance
- Since Euclidean distance is used, features are usually standardized.

Linear boundary can fail



Sometime a linear boundary simply won't work, no matter what value of C.

The example on the left is such a case.

What to do?

Feature expansion

- Enlarge the space of features by including transformations; e.g. X_1^2, X_1^3, X_1X_2, $X_1X_2^2$,. . .. Hence go from a p-dimensional space to a $M > p$ dimensional space.
- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the original space.

Example: Suppose we use $(X_1,\ X_2,\ X_1^2,\ X_2^2,\ X_1X_2)$ instead of just (X_1, X_2). Then the decision boundary would be of the form

$$\beta_0 + \beta_1X_1 + \beta_2X_2 + \beta_3X_1^2 + \beta_4X_2^2 + \beta_5X_1X_2 = 0$$

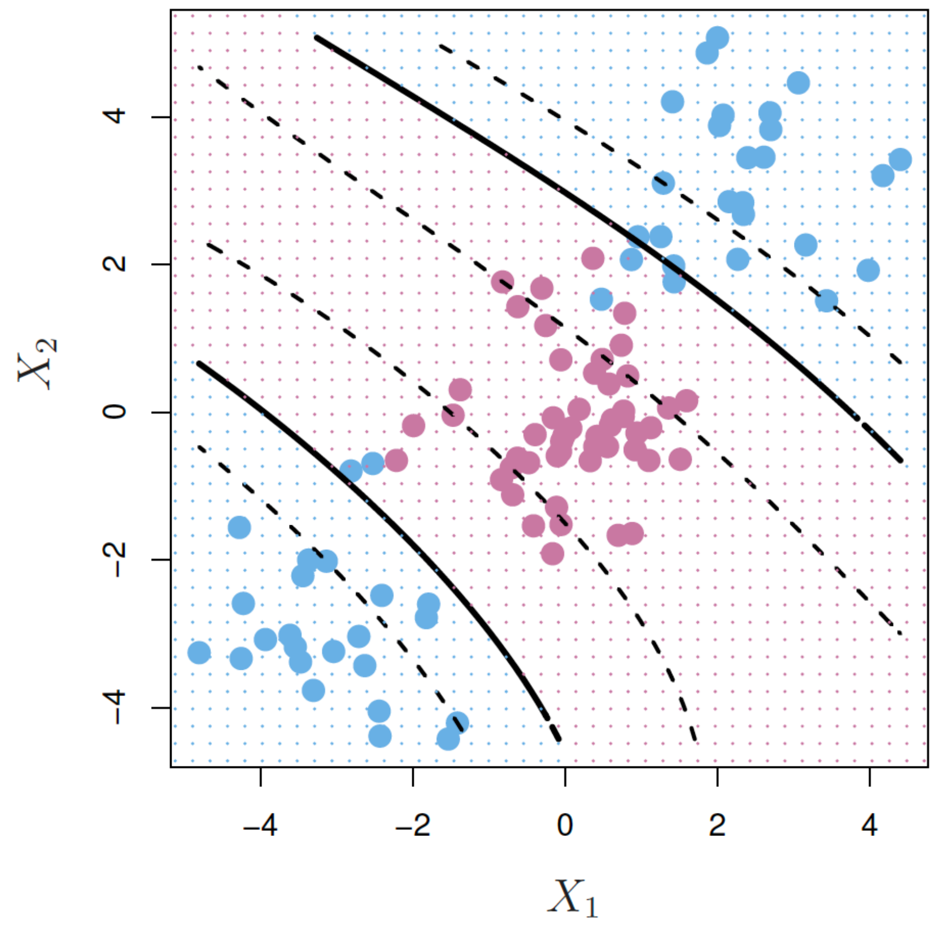
This leads to nonlinear decision boundaries in the original space (quadratic conic sections).

Cubic polynomials

Here we use a basis expansion of cubic polynomials

From 2 variables to 9

The support-vector classifier in the enlarged space solves the problem in the lower-dimensional space

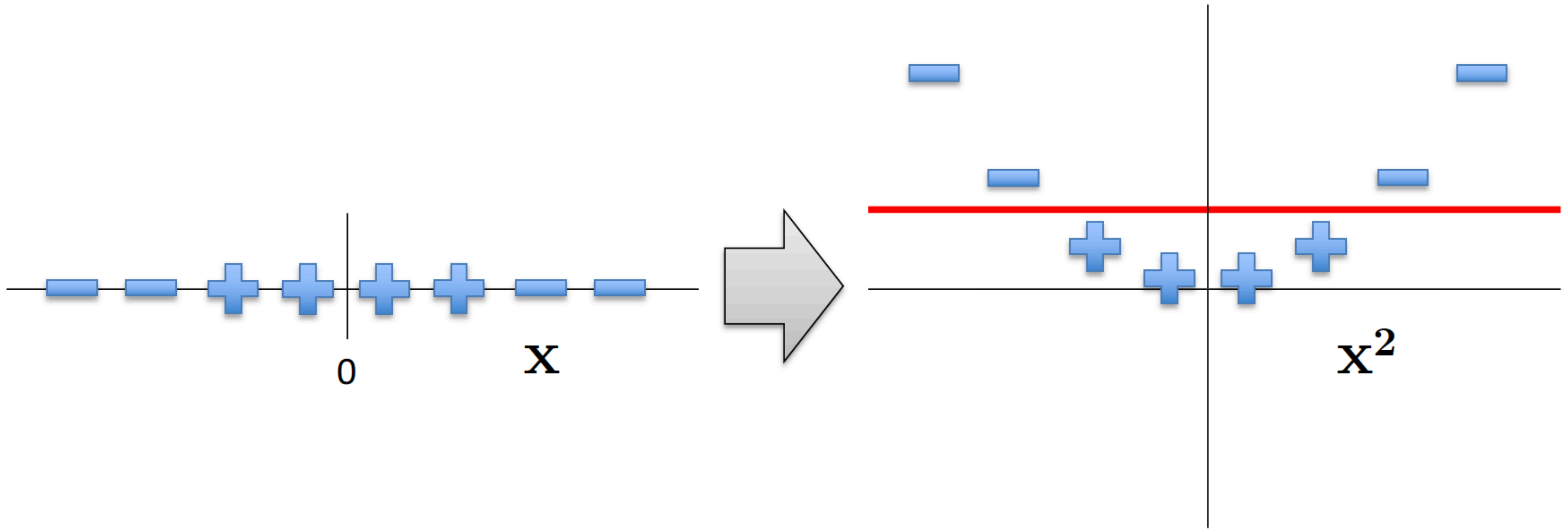


$$\beta_0+\beta_1X_1+\beta_2X_2+\beta_3X_1^2+\beta_4X_2^2+\beta_5X_1X_2+\beta_6X_1^3+\beta_7X_2^3+\beta_8X_1X_2^2+\beta_9X_1^2X_2=0$$

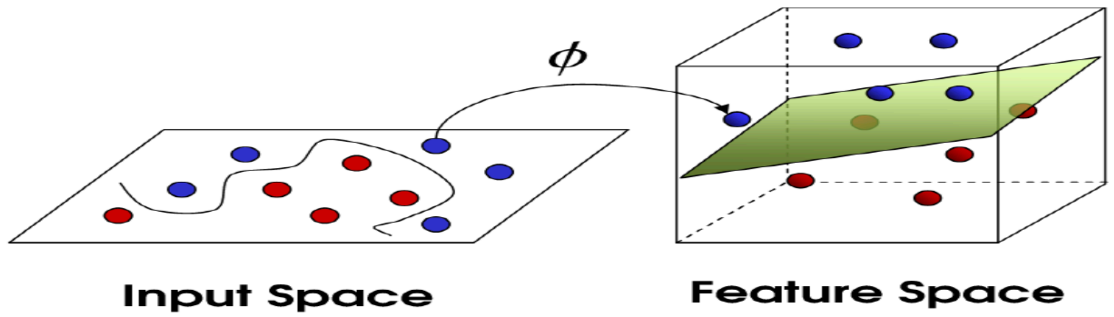
Nonlinearities and kernels

- Polynomials (especially high-dimensional ones) get wild rather fast.
- There is a more elegant and controlled way to introduce nonlinearities in support-vector classifiers — through the use of *kernels*.

When linear separators fail



Map into a new feature space



$$\Phi : \mathcal{X} \mapsto \hat{\mathcal{X}} = \Phi(\mathbf{x})$$

- For example, with $\mathbf{x}_i \in \mathbb{R}^2$

$$\Phi([x_{i1}, x_{i2}]) = [x_{i1}, x_{i2}, x_{i1}x_{i2}, x_{i1}^2, x_{i2}^2]$$

- Rather than run SVM on $\mathbf{x}_i$, run it on $\Phi(\mathbf{x}_i)$
  - Find non-linear separator in input space
- What if $\Phi(\mathbf{x}_i)$ is really big?
- Use kernels to compute it implicitly!

The kernel trick

$\overrightarrow{X_i}, \overrightarrow{X_j} \in \mathbb{R}^N$
$K(\overrightarrow{X_i}, \overrightarrow{X_j}) = \left\langle \phi(\overrightarrow{X_i}), \phi(\overrightarrow{X_j}) \right\rangle_M$,
where $< \cdot, \cdot >_M$ is an inner product of $\mathbb{R}^M (M > N)$.
$\phi(\overrightarrow{X_i})$ transforms $\overrightarrow{X_i}$ from $\mathbb{R}^N$ to $\mathbb{R}^M$.

Kernel trick
We can compute dot products in a high-dimensional space $\mathbb{R}^M$ while remaining in $\mathbb{R}^N$, i.e., compute every element in K without explicitly computing $\phi(x)$.
Transformation is impractical for large dimensions.

Kernel example

$$X = (x_1, x_2)^T, \phi(X) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$$
$$Y = (y_1, y_2)^T, \phi(Y) = (y_1^2, \sqrt{2}y_1y_2, y_2^2)^T$$

$$\begin{aligned} K(X,Y) = \phi(X)^T\phi(Y) &= \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right) \left(\begin{array}{c} y_1^2 \\ \sqrt{2}y_1y_2 \\ y_2^2 \end{array} \right) \\ &= x_1^2y_1^2 + 2x_1y_1x_2y_2 + x_2^2y_2^2 \\ &= (x_1y_1 + x_2y_2)^2 \\ &= (X^TY)^2, \end{aligned}$$

which is the polynomial kernel.

Popular kernels

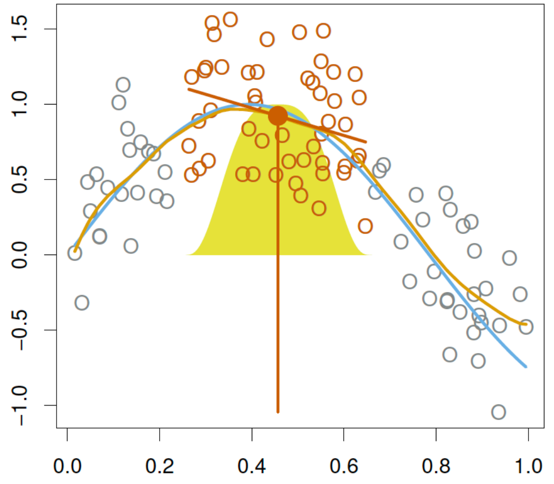
$$K(X,Y) = X^TY + C, \text{ Linear kernel}$$
$$K(X,Y) = (\alpha X^TY + C)^d, \text{ Polynomial kernel}$$
$$K(X,Y) = \exp\left(-\gamma ||X - Y||^2\right), \text{ Gaussian/Radial kernel}$$
$$K(X,Y) = \tanh(\alpha X^TY + C), \text{ Hyperbolic tangent/ Sigmoid kernel}$$

Kernel regression/classification

- Two of the shortcomings of the K-NN method is that all neighbors receive equal weight and the number of neighbors must be chosen globally.
- Kernel regression addresses these issues. Instead of selected nearest neighbors, all neighbors are used, but with different weights. Closer neighbors receive higher weight. The weighting function is called a kernel and it measures similarity (as opposed to distance) between examples.

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

Convert from a distance d(·,·) to a Gaussian kernel K(·,·)



Kernel regression/classification algorithm

Given training data $D = \{\mathbf{x}_i, y_i\}$, Kernel function $K(\cdot, \cdot)$ and input $\mathbf{x}$

- (regression) if $y \in \mathbf{R}$, return weighted average:

$$\hat{y}(\mathbf{x}) = \frac{\sum_{i=1}^{n} K(\mathbf{x},\mathbf{x}_i) y_i}{\sum_{i=1}^{n} K(\mathbf{x},\mathbf{x}_i)}$$

- (classification) if $y \in \pm 1$, return weighted majority:

$$\hat{y}(\mathbf{x}) = sign(\sum_{i=1}^{n} K(\mathbf{x}, \mathbf{x}_i) y_i)$$

Kernel PCA

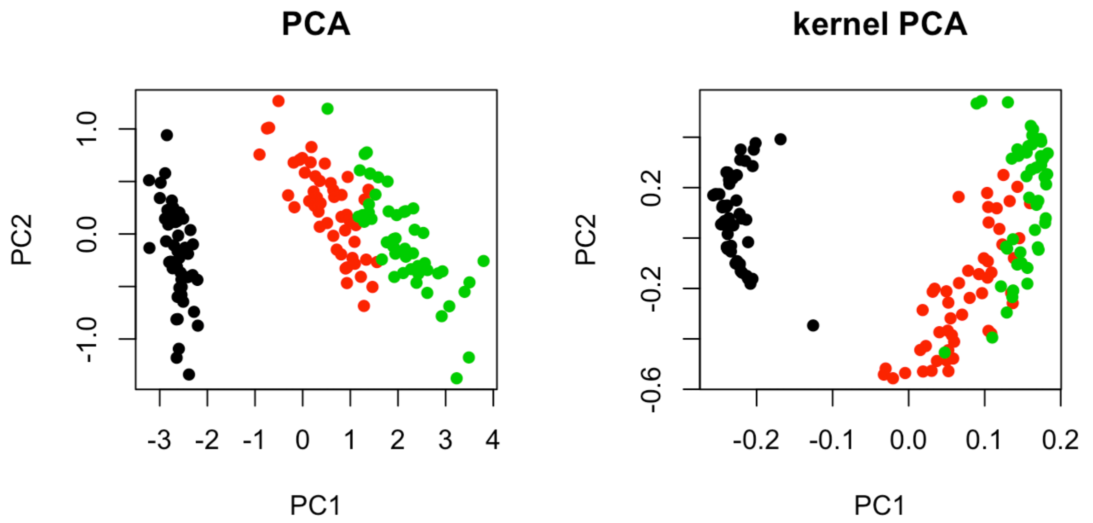
- Map each data point to nonlinear feature space
- Extract PC in the transformed space, which is non-linear in the original data space.

Need to center in the feature space $\tilde{\Phi}(\mathbf{x}_i) = \Phi(\mathbf{x}_i) - \frac{1}{n}\sum_{k=1}^{n} \Phi(\mathbf{x}_k)$.

Construct the centered kernel matrix: $\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_{1/n}\mathbf{K} - \mathbf{K}\mathbf{1}_{1/n} + \mathbf{1}_{1/n}\mathbf{K}\mathbf{1}_{1/n}$.

Solve an eigenvalue problem: $\tilde{\mathbf{K}}\boldsymbol{\alpha}_i = \lambda_i \boldsymbol{\alpha}_i$.

For any data point, we can represent it as $y_j = \sum_{i=1}^{n} \alpha_{ij} \tilde{\mathrm{K}}(\mathbf{x}, \mathbf{x}_i), j = 1, \dots, d$.


PCA
PC2
PC1
kernel PCA
PC2
PC1