Class 8 – Support Vector Machines (SVM)

Classification

Pedram Jahangiry

Fall 2019

JON M.
HUNTSMAN
SCHOOL OF BUSINESS
UtahStateUniversity

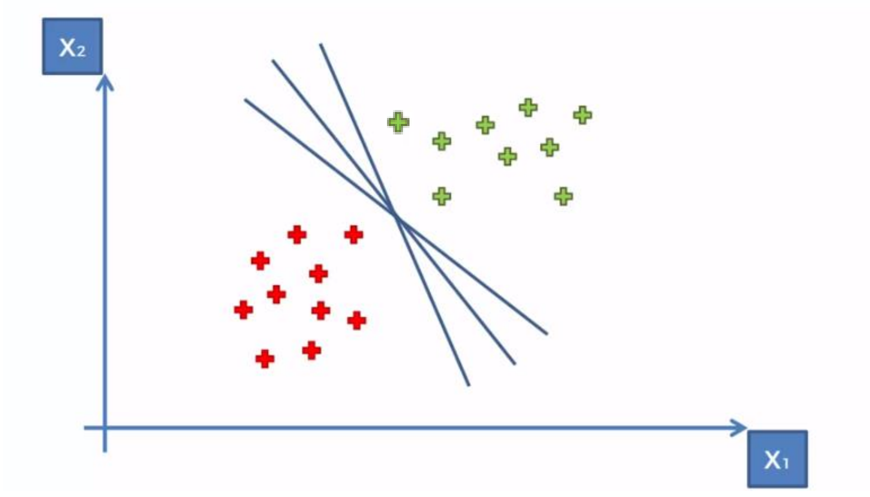
Support Vector Machines

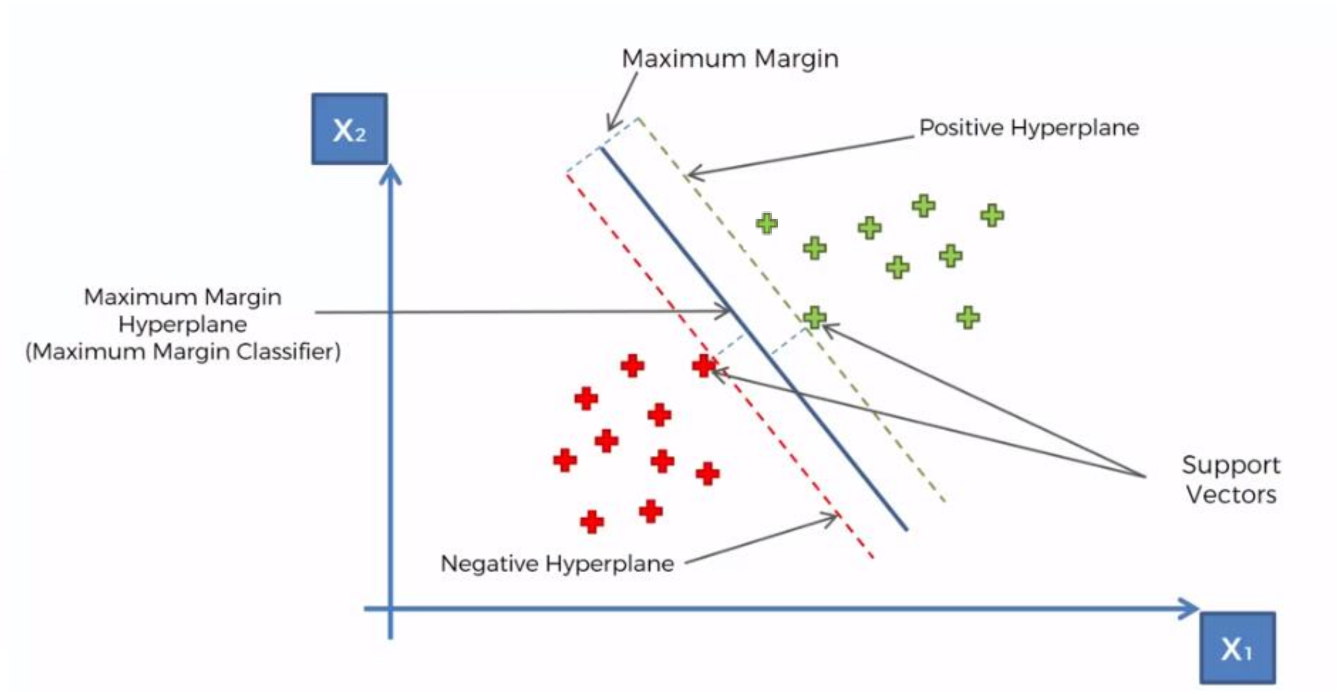
- Here we approach the two-class classification problem in a direct way:

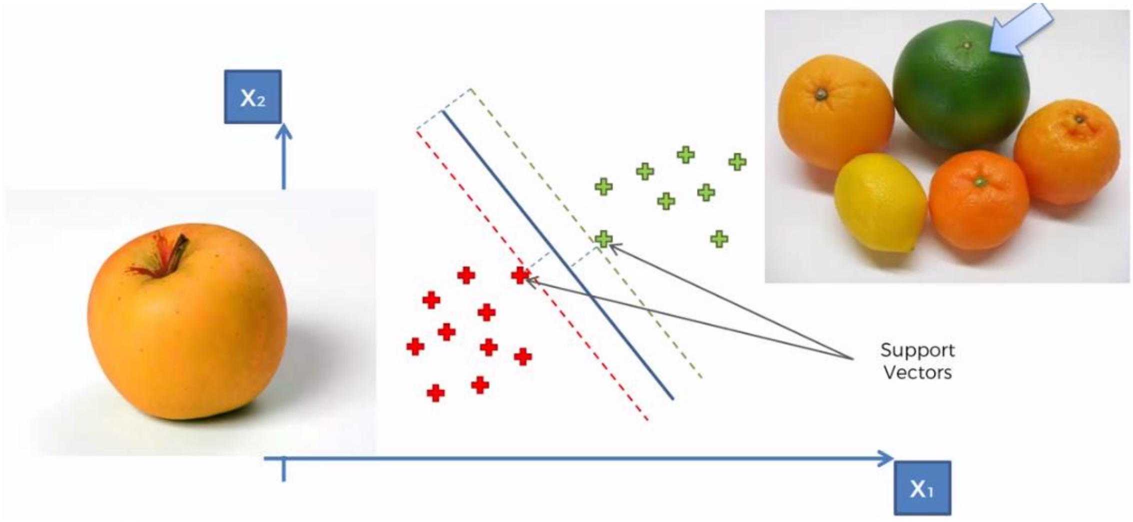
  We try to find a plane that separates the classes in feature space.

- If we cannot, we get creative in two ways:
  1. We soften what we mean by "separates", and
  2. We enrich and enlarge the feature space so that separation is possible.

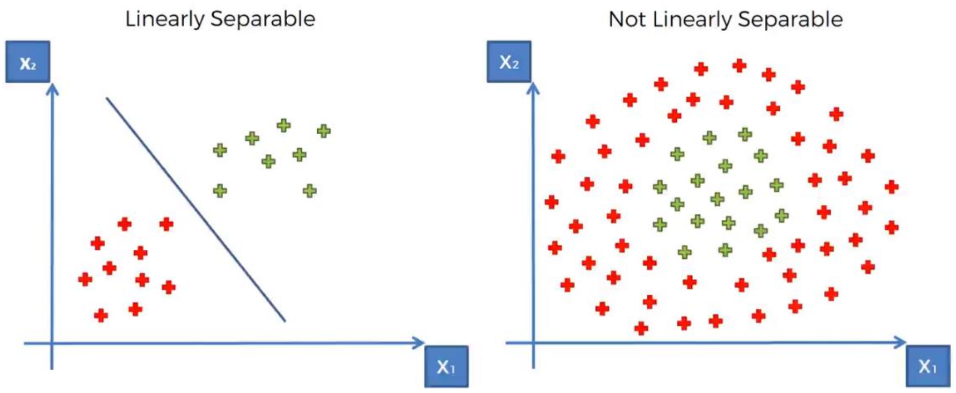
Separable data

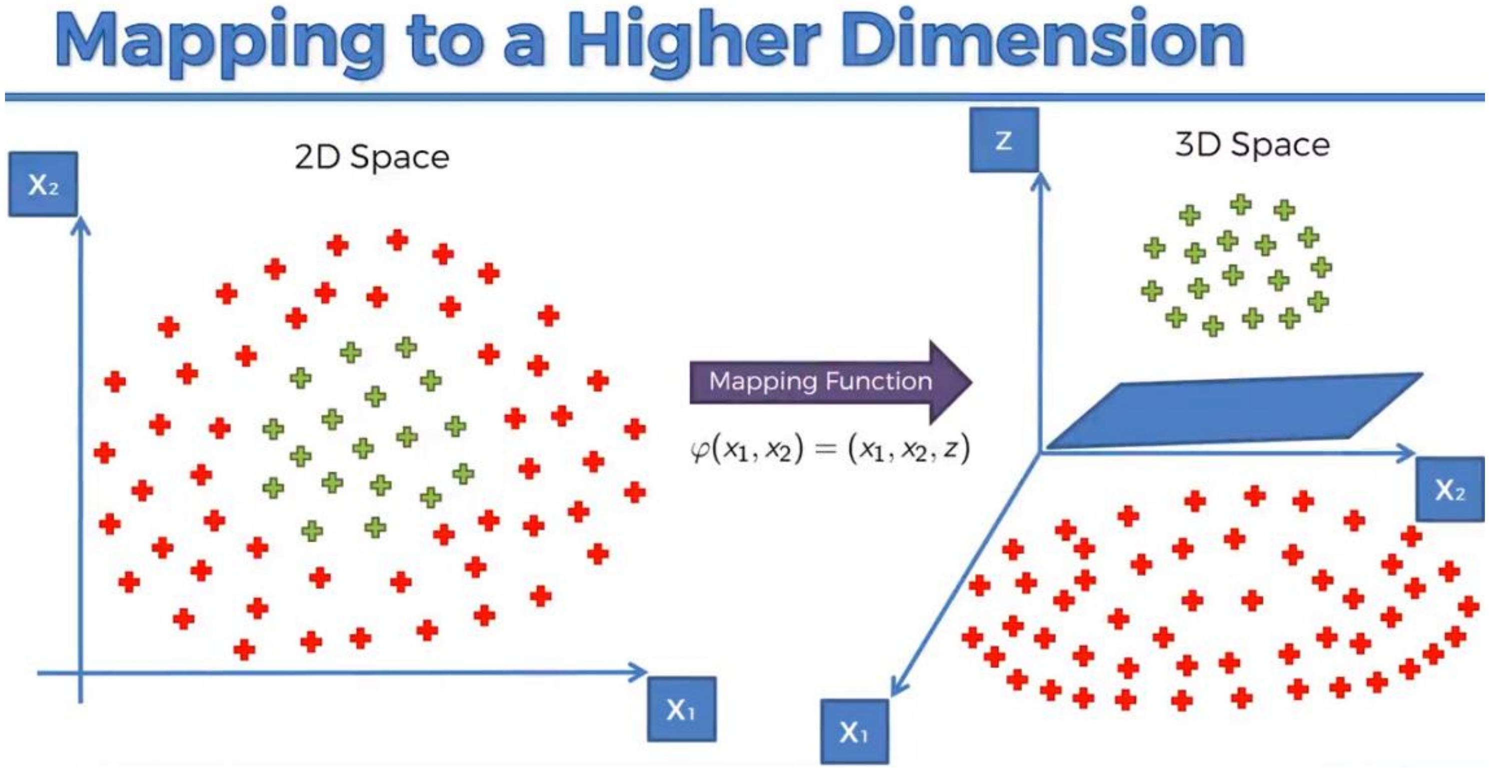




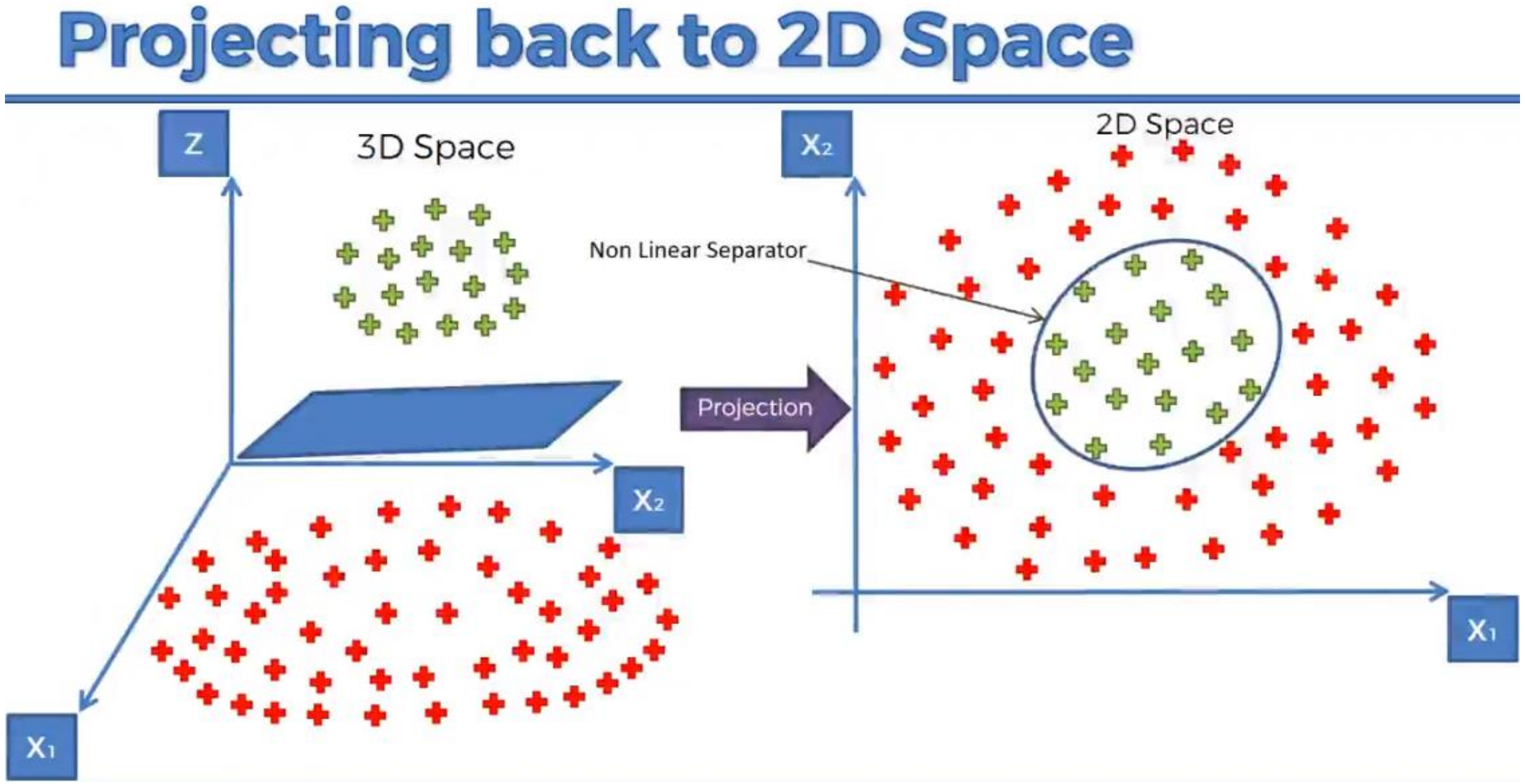


Non-Linearly Separable data



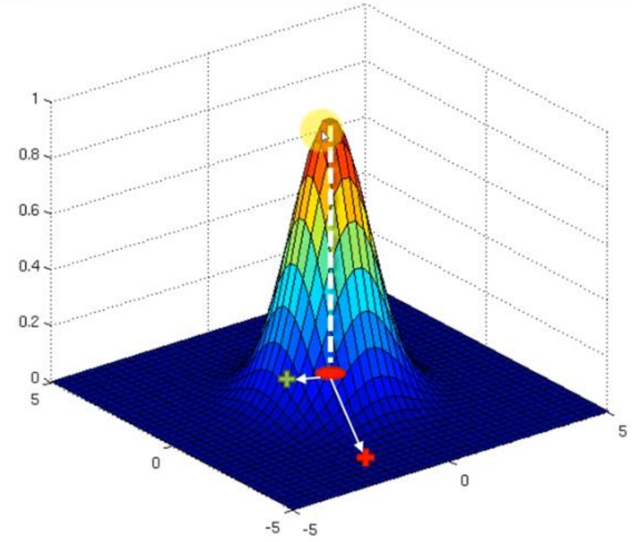


Non-Linearly Separable data (cont’d)

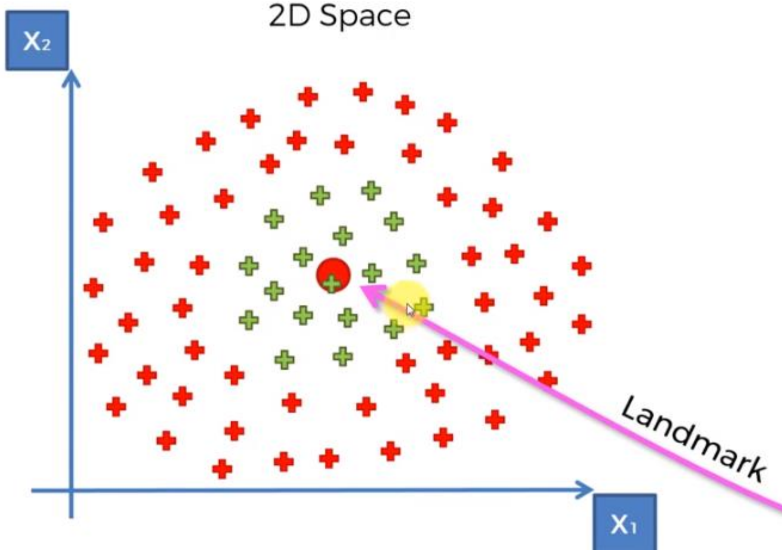


The Gaussian RBF Kernel

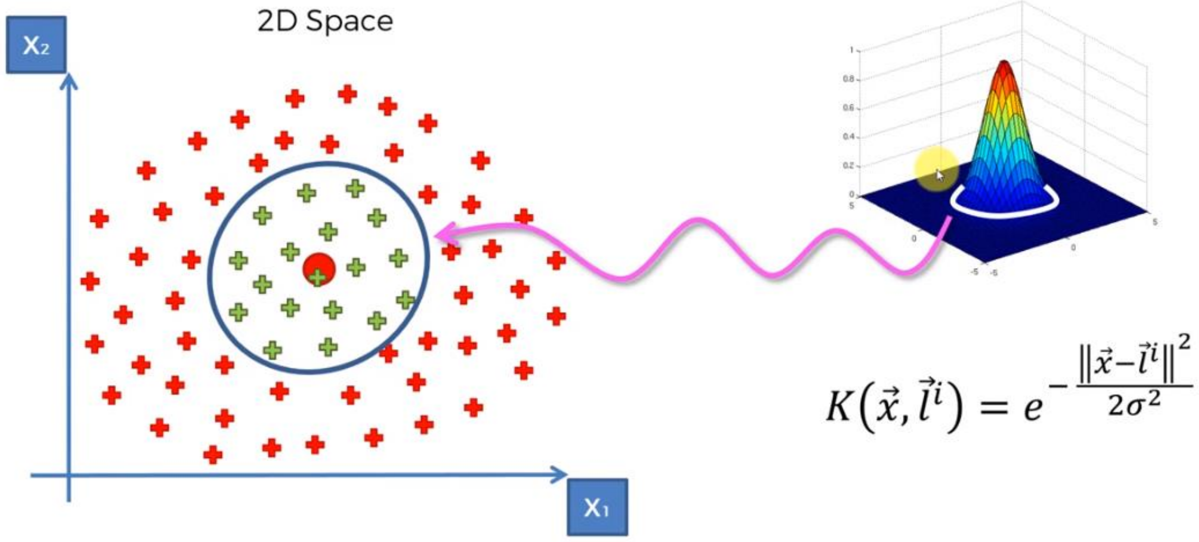
The Kernel trick!



$$K(\vec{x},\vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$

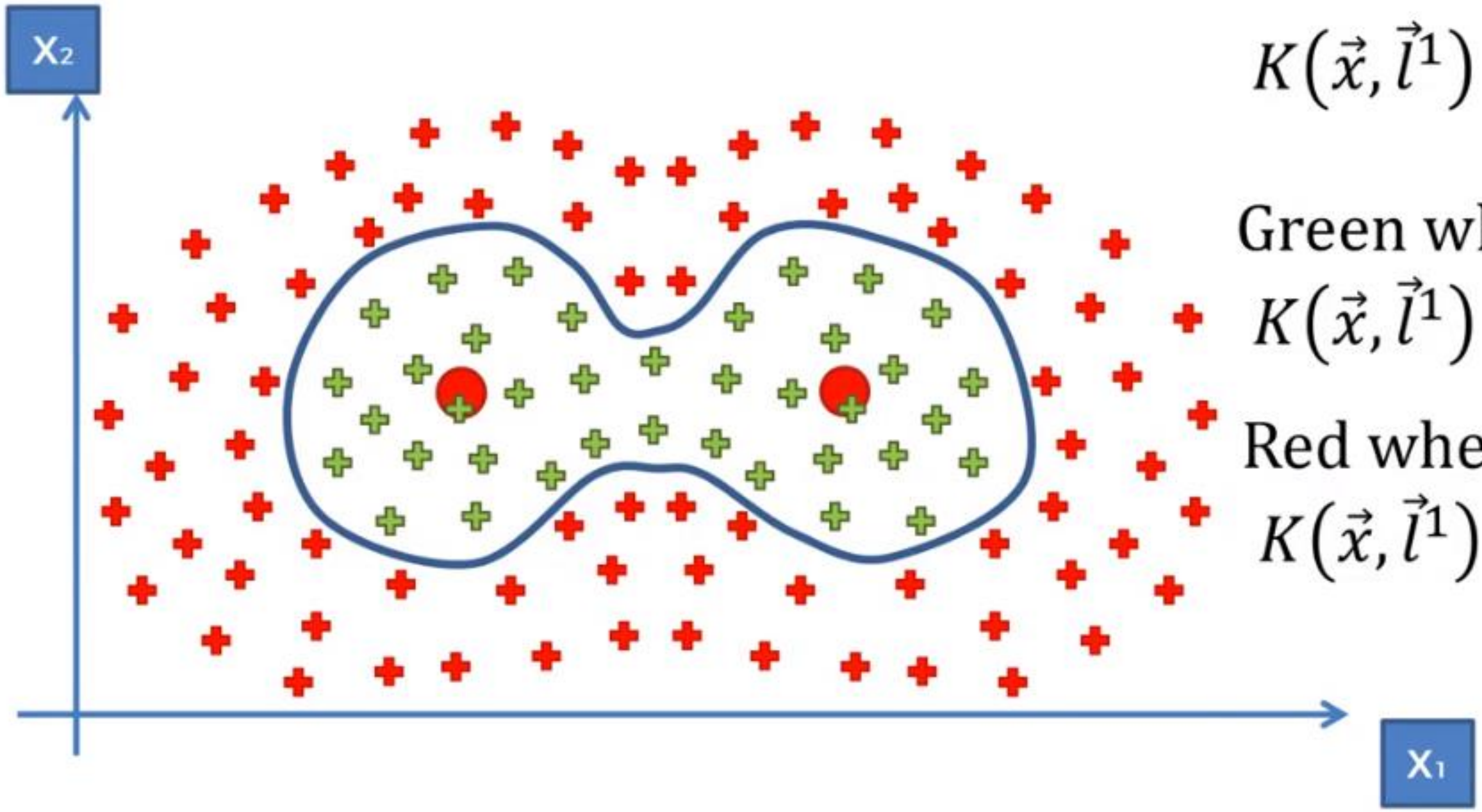


$$K(\vec{x},\vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$



$$K(\vec{x},\vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$

A more complex Kernel function.



$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2)$$

(Simplified Formula)

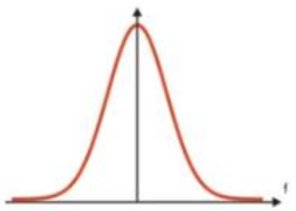
Green when:

$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2) > 0$$

Red when:

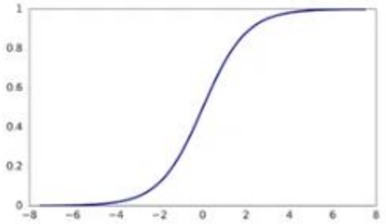
$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2) = 0$$

Types of Kernel Functions



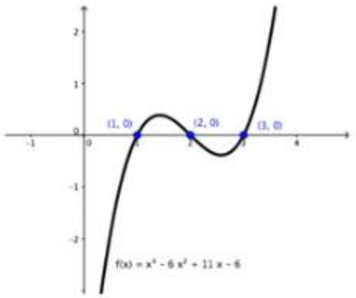
Gaussian RBF Kernel

$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$



Sigmoid Kernel

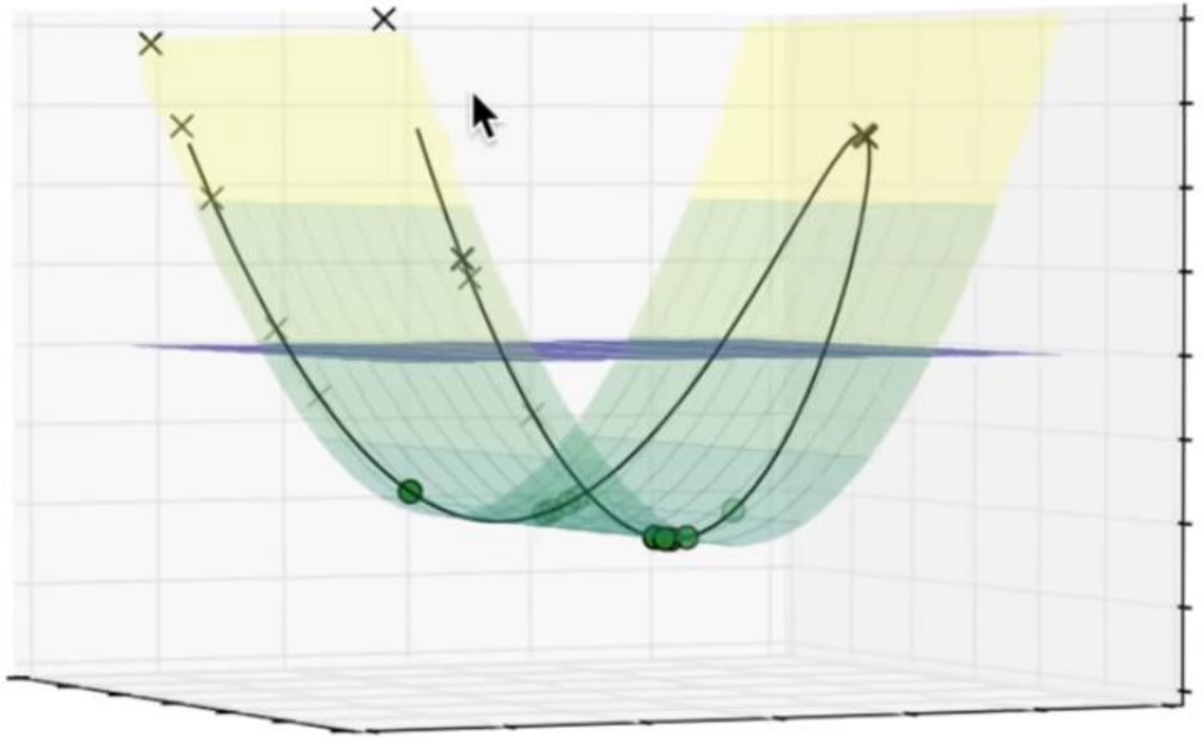
$$K(X, Y) = \tanh(\gamma \cdot X^T Y + r)$$



Polynomial Kernel

$$K(X, Y) = (\gamma \cdot X^T Y + r)^d, \gamma > 0$$

Visualization of SVM



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.

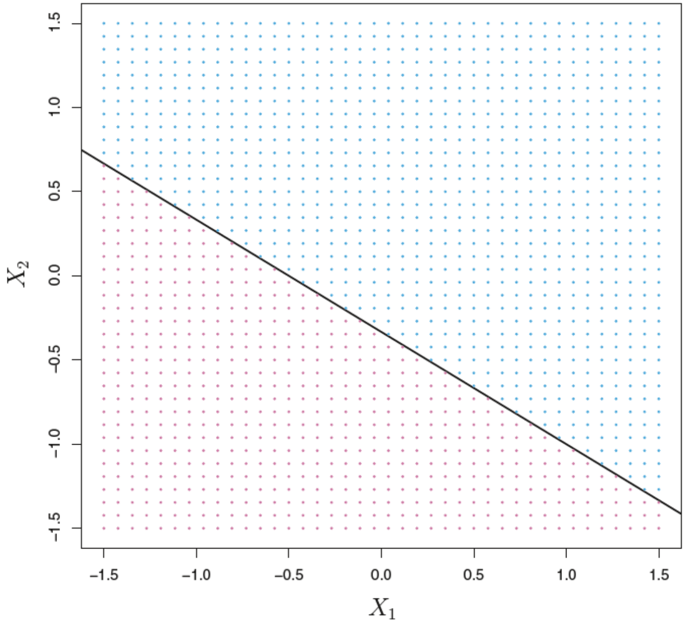
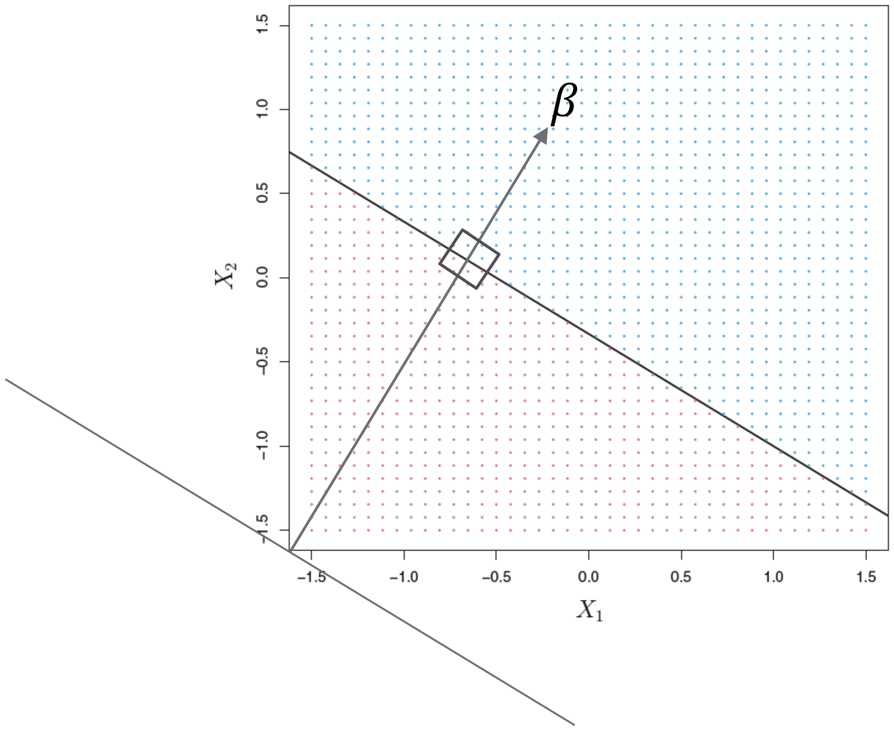


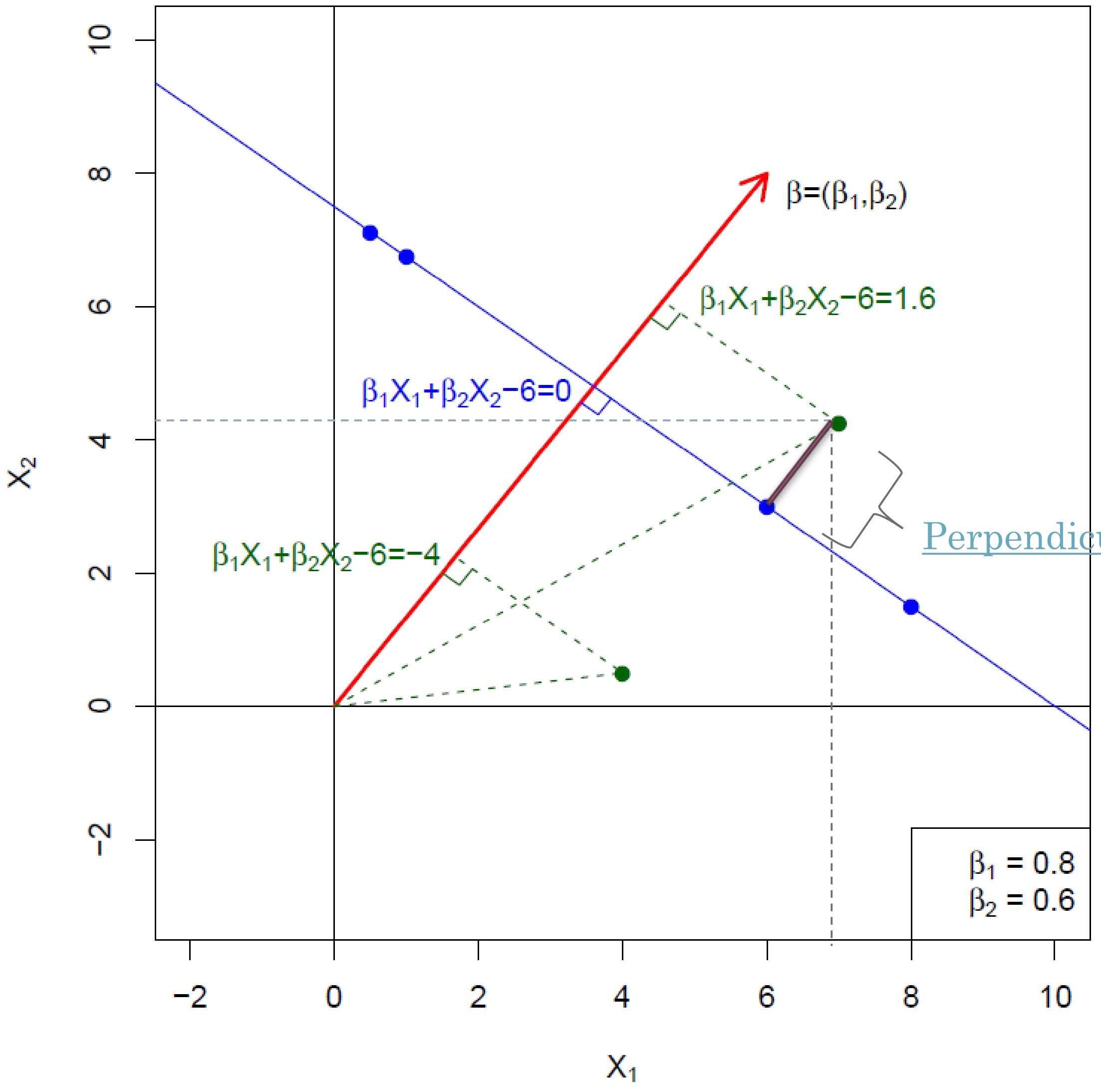
FIGURE 9.1. *The hyperplane* $1 + 2X_1 + 3X_2 = 0$ *is shown. The blue region is the set of points for which* $1 + 2X_1 + 3X_2 > 0$*, and the purple region is the set of points for which* $1 + 2X_1 + 3X_2 < 0$*.*

What is a Hyperplane?

- 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.

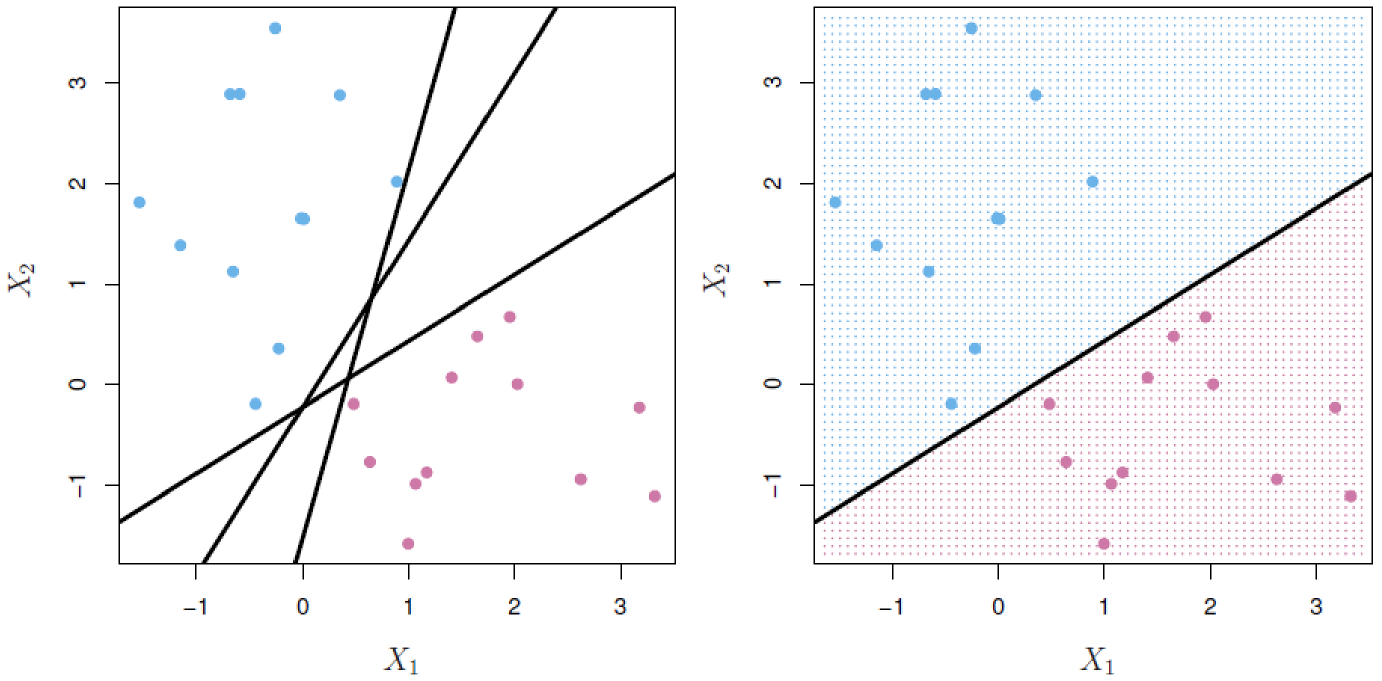


Hyperplane in 2 Dimensions



Perpendicular distance from the hyperplane

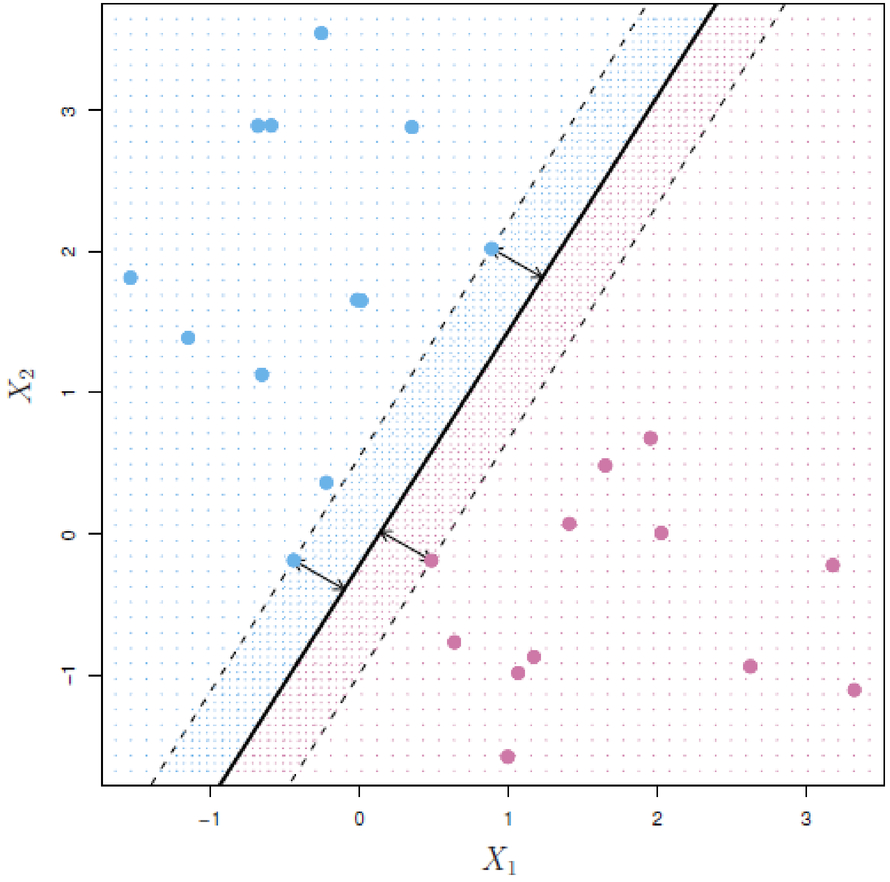
Separating Hyperplanes



- 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

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

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

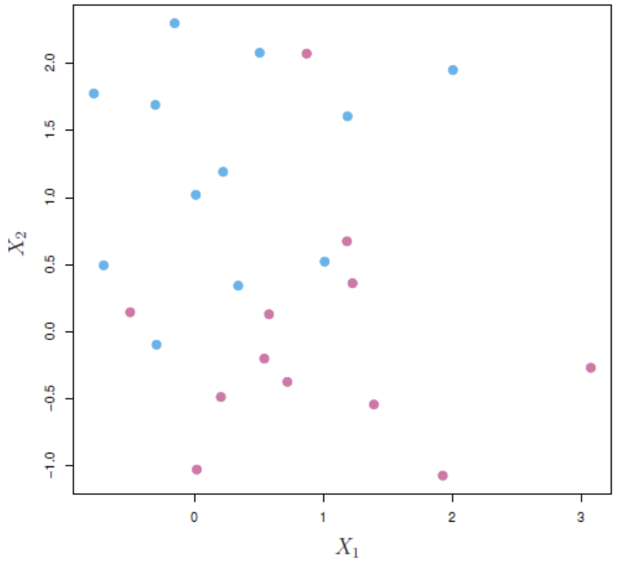
$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M$$

$$\text{for all } \; i = 1, \ldots, N.$$

Common problems

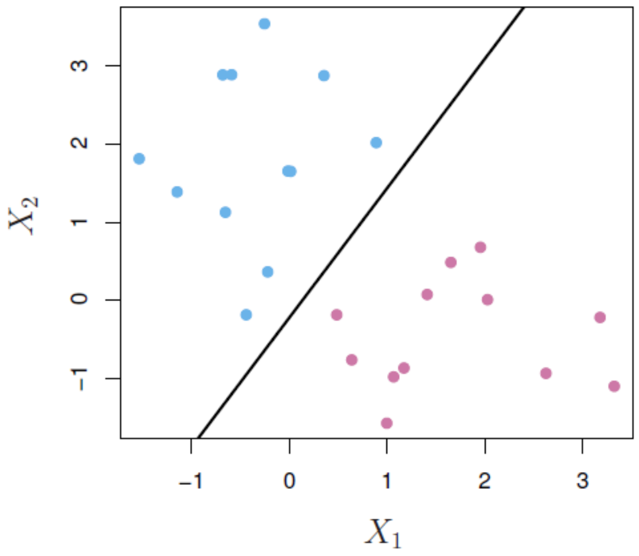
- Non-Separable data

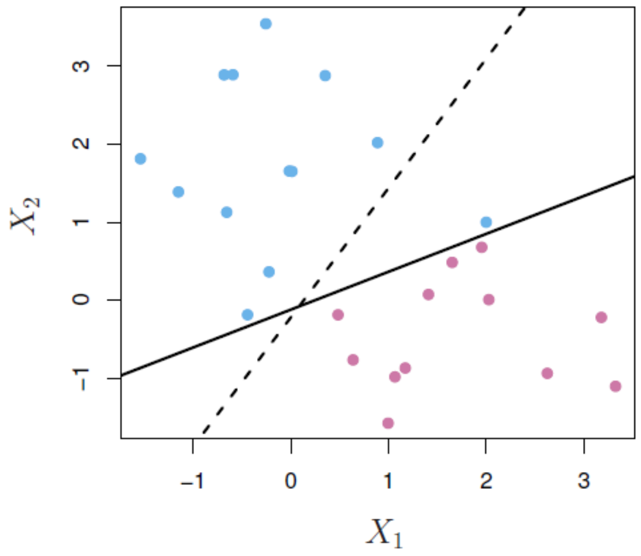
  Maximal margin classifier cannot be found



- Noisy data

  Maximal margin classifier is very sensitive to outliers





Non-Separable Data

- The data in fig 9.4 are not separable by a linear boundary.
- This is often the case since $N > p$
- we can extend the concept of a separating hyperplane in order to develop a hyperplane that *almost* separates the classes, using a so-called *soft margin*.
- The generalization of the maximal margin classifier to the non-separable case is known as the *support vector classifier*.

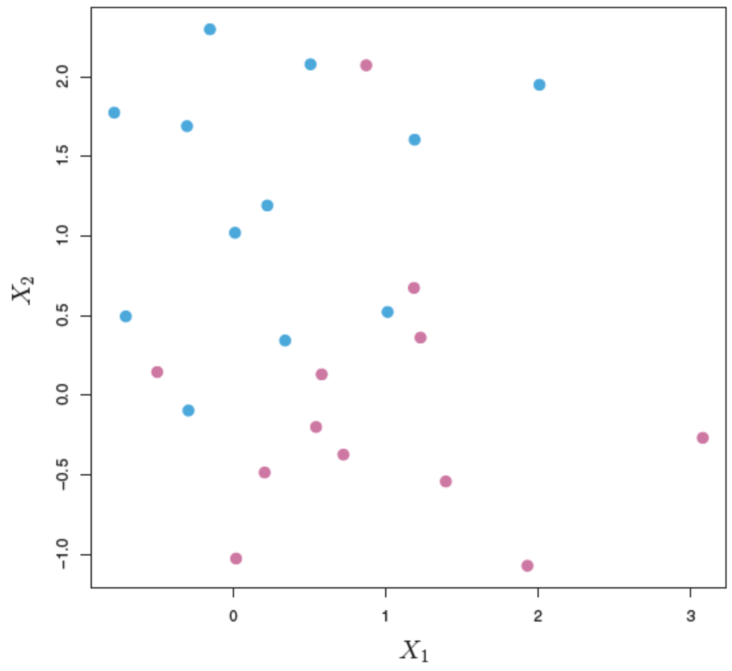


FIGURE 9.4. *There are two classes of observations, shown in blue and in purple. In this case, the two classes are not separable by a hyperplane, and so the maximal margin classifier cannot be used.*

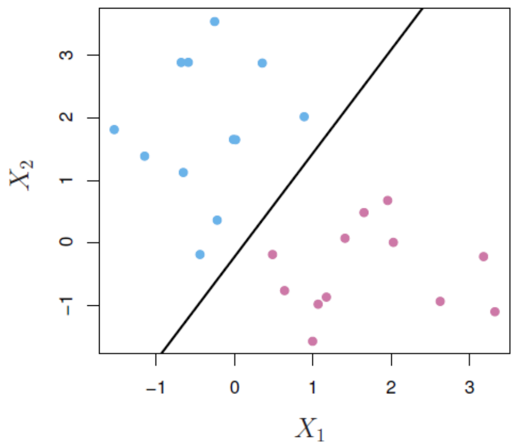
Noisy Data

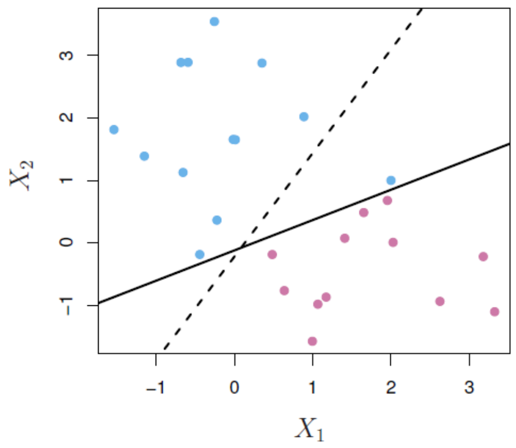
In the case of noisy data, we might be willing to consider a classifier based on a hyperplane that does *not* perfectly separate the two classes, in the interest of:

1. Greater robustness to individual observations, and
2. Better classification of *most* of the training observations

That is, it could be worthwhile to misclassify a few training observations in order to do a better job in classifying the remaining observations.

The *support vector classifier*, sometimes called a *soft margin classifier* does exactly this.





Support Vector Classifier

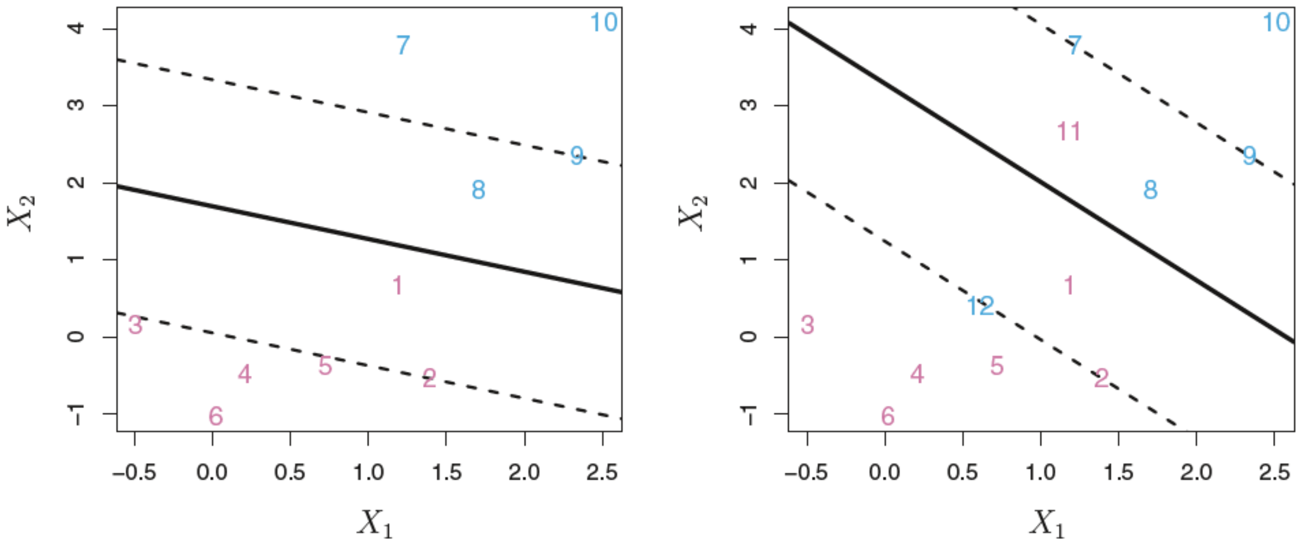


FIGURE 9.6. Left: *A support vector classifier was fit to a small data set. The hyperplane is shown as a solid line and the margins are shown as dashed lines.* Purple observations: *Observations* 3, 4, 5, *and* 6 *are on the correct side of the margin, observation* 2 *is on the margin, and observation 1 is on the wrong side of the margin.* Blue observations: *Observations* 7 *and* 10 *are on the correct side of the margin, observation* 9 *is on the margin, and observation* 8 *is on the wrong side of the margin. No observations are on the wrong side of the hyperplane.* Right: *Same as left panel with two additional points,* 11 *and* 12*. These two observations are on the wrong side of the hyperplane and the wrong side of the margin.*

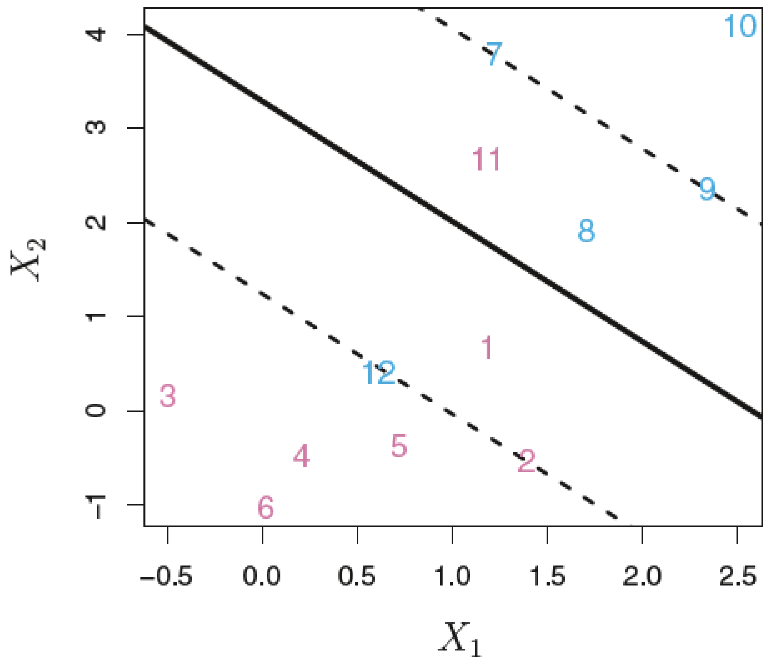
Support Vector Classifier

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,M}{\text{maximize}} \quad M$$

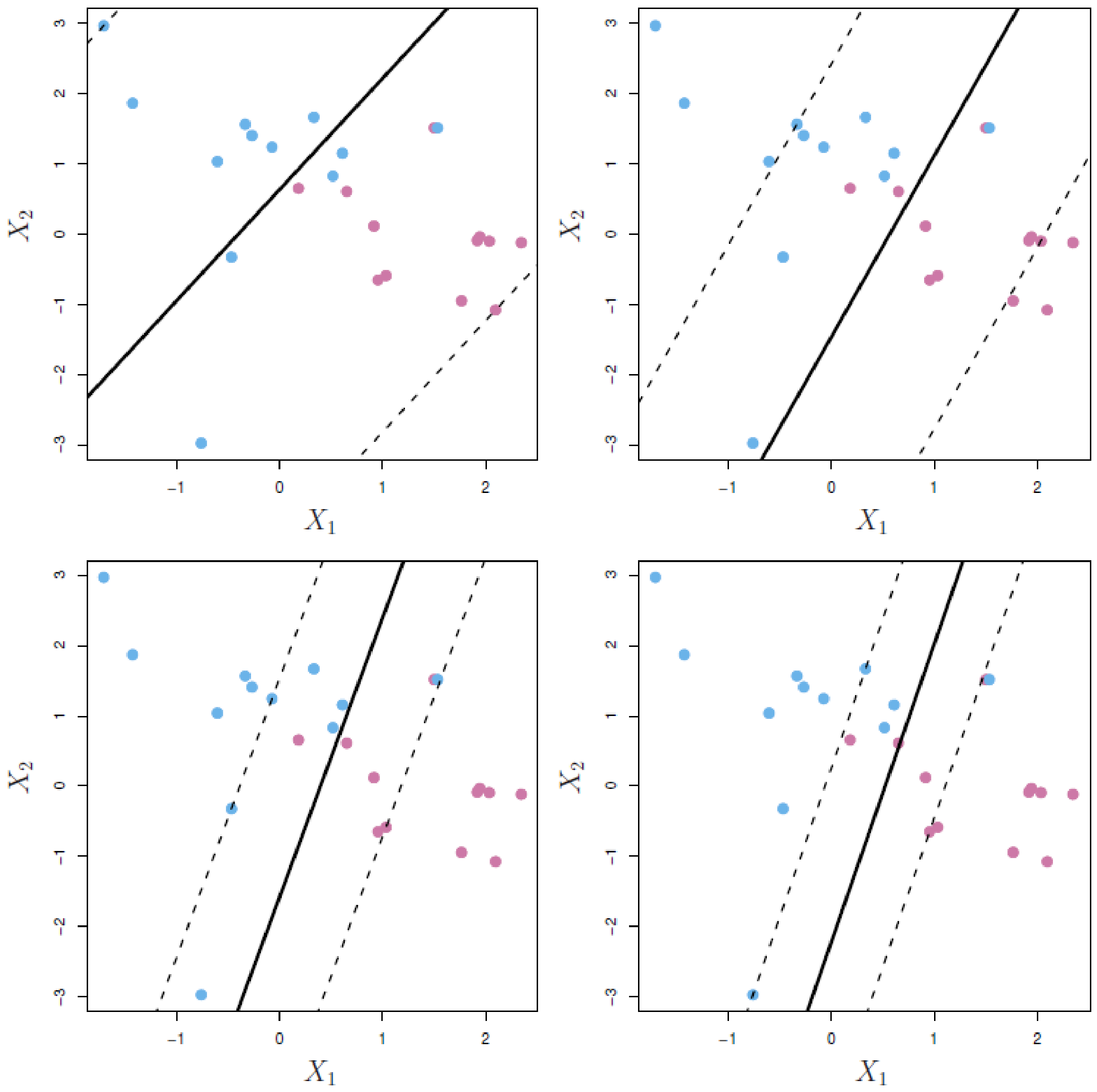
$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

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

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

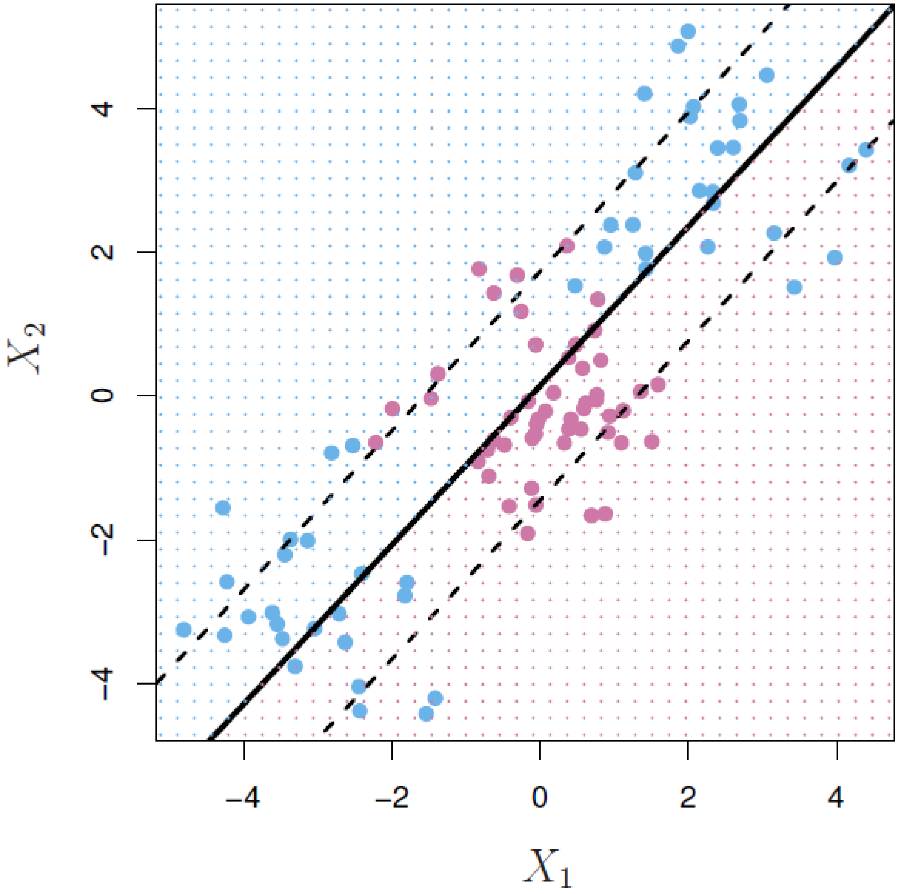


C is a tuning / regularization parameter



Failure of linear boundary

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



- What to do?

  Feature expansion (Bend the margin!)

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

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 = 0$$

This leads to nonlinear decision boundaries in the original space

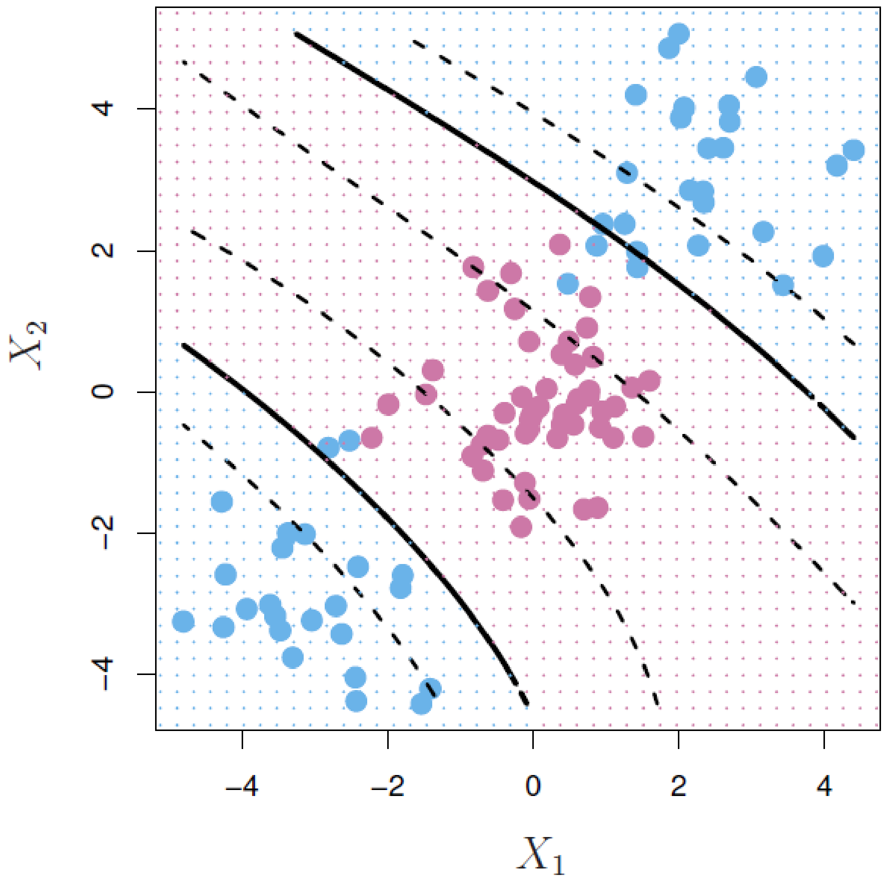
Heading

- 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

- In a 9 dimension space, the decision boundary is **a single linear boundary**.

- The projections in the 2 dimensional space are **multiple non-linear boundaries**



$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 + \beta_6 X_1^3 + \beta_7 X_2^3 + \beta_8 X_1 X_2^2 + \beta_9 X_1^2 X_2 = 0$$

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*.
- Before we discuss these, we must understand the role of *inner products* in support-vector classifiers.

Inner products and support vectors

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \quad \text{— \textit{inner product between vectors}}$$

- The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle \quad \text{— } n \text{ \textit{parameters}}$$

- To estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0, all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$ between all pairs of training observations.

It turns out that most of the $\hat{\alpha}_i$ can be zero:

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i \langle x, x_i \rangle$$

$\mathcal{S}$ is the *support set* of indices i such that $\hat{\alpha}_i > 0$. See slide 14

Kernels and Support Vector Machines

- If we can compute inner-products between observations, we can fit a SV classifier. Can be quite abstract!
- Some special *kernel functions* can do this for us. E.g.

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$$

computes the inner-products needed for d dimensional polynomials — $\binom{p+d}{d}$ basis functions!

Try it for $p = 2$ and $d = 2$.

- The solution has the form

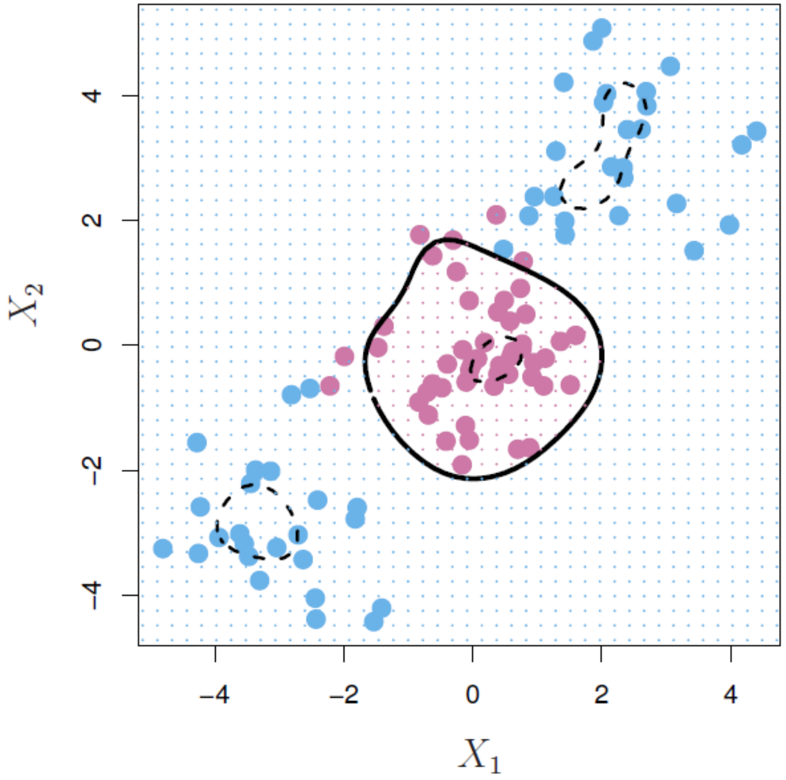
$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(x, x_i).$$

Radial Kernel

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(x, x_i)$$

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$$

- Radial Kernel, controls variance by squashing down most dimensions severely



SVM for more than 2 classes!

The SVM as defined works for $K = 2$ classes. What do we do if we have $K > 2$ classes?

OVA One versus All. Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \ldots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.

OVO One versus One. Fit all $\binom{K}{2}$ pairwise classifiers $\hat{f}_{k\ell}(x)$. Classify x^* to the class that wins the most pairwise competitions.

Which to choose? If K is not too large, use OVO.

Which to use: SVM or Logistic Regression

- When classes are (nearly) separable, SVM does better than LR.
- When not, Logistic Regression (LR) and SVM very similar.
- If you wish to estimate probabilities, LR is the choice.
- For nonlinear boundaries, kernel SVMs are popular. Can use kernels with Logistic Regression too, but computations are more expensive.

SVM in Python

- Find the SVM Sklearn documentation here
- Blackbox version of SVM in python:

```
from sklearn import svm
X = [[0, 0], [1, 1]]
y = [0, 1]
clf = svm.SVC(gamma='scale')
clf.fit(X, y)
```

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC(gamma='scale')
>>> clf.fit(X, y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```