MONASH University

ETC3555

Statistical Machine Learning

Support Vector Machines

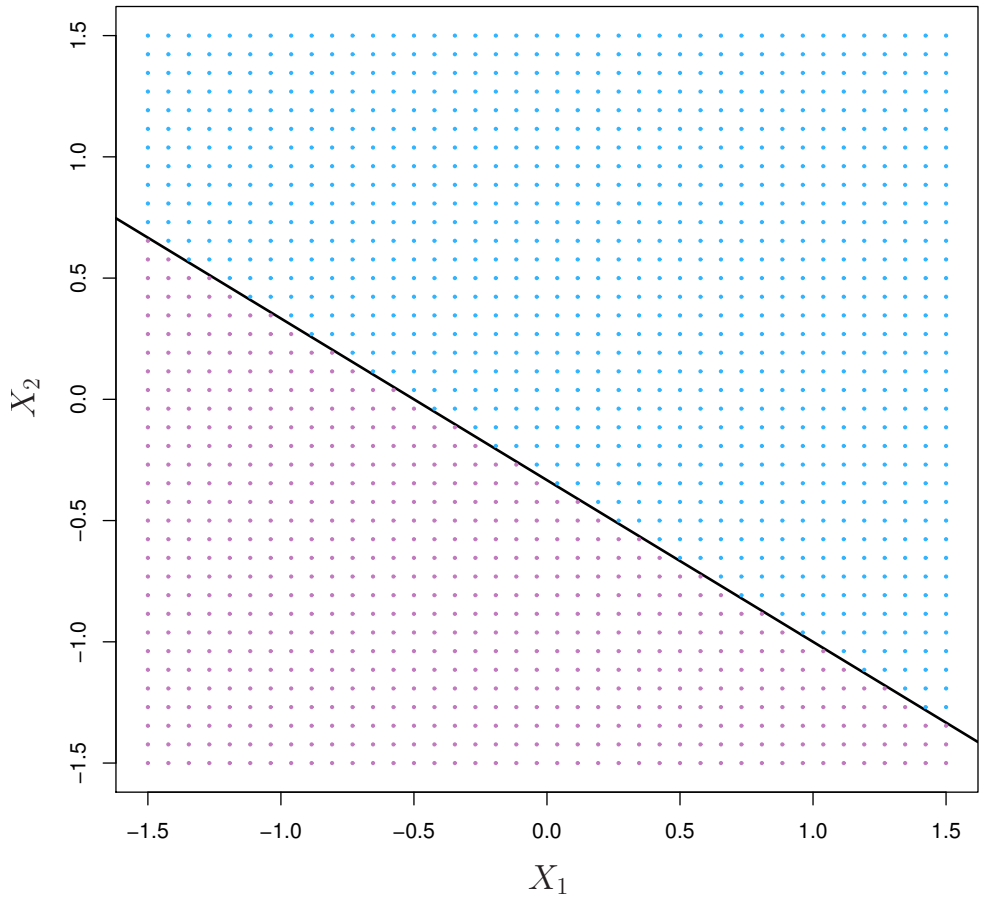
11 September 2018

Outline (subject to change)

Week	Topic
1	The learning problem
2	The learning problem
3	Linear models
4	Gradient descent
5	Neural Networks
6	Neural Networks
7	Deep Neural Networks
8	Support Vector Machines
9	Recommender systems and matrix completion
	Semester break
10	Text mining
11	Social networks
12	Project presentation

Separating Hyperplane

In a p-dimensional space, a **hyperplane** is a flat affine subspace of dimension $p - 1$.



Classification Using a Separating Hyperplane

The equation of p-dimensional hyperplane is given by

$$\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p = 0.$$

If $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$ for $i = 1, \ldots, n$, then

$$\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} > 0 \text{ if } y_i = 1,$$
$$\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} < 0 \text{ if } y_i = -1.$$

Equivalently,

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) > 0.$$

Classification Using a Separating Hyperplane

- A new observation is assigned a class depending on **which side** of the hyperplane it is located
- we classify the test observation x^* based on the **sign** of

$$s(x^*) = \beta_0 + \beta_1 x_1^* + \cdots + \beta_p x_p^*$$

- If $s(x^*) > 0$, class 1, and if $s(x^*) < 0$, class -1, i.e. $h(x^*) = \textbf{sign}(s(x^*))$.

What about the **magnitude** of $s(x^*)$?

Classification Using a Separating Hyperplane

- A new observation is assigned a class depending on **which side** of the hyperplane it is located
- we classify the test observation x^* based on the **sign** of

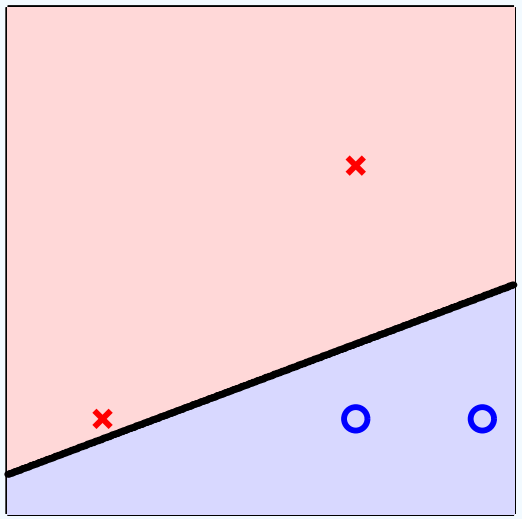
$$s(x^*) = \beta_0 + \beta_1 x_1^* + \cdots + \beta_p x_p^*$$

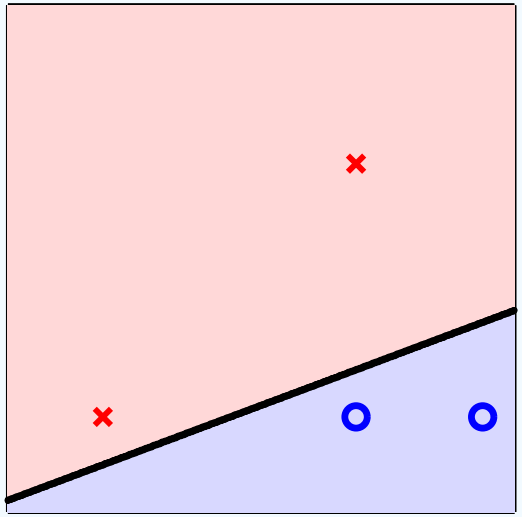
- If $s(x^*) > 0$, class 1, and if $s(x^*) < 0$, class -1, i.e. $h(x^*) = \textbf{sign}(s(x^*))$.

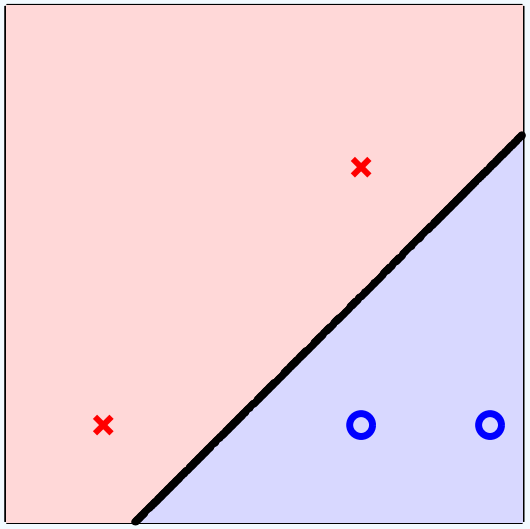
What about the **magnitude** of $s(x^*)$?

- $s(x^*)$ far from zero $\rightarrow x^*$ lies far from the hyperplane + **more confident** about our classification
- $s(x^*)$ close to zero $\rightarrow x^*$ near the hyperplane + **less confident** about our classification

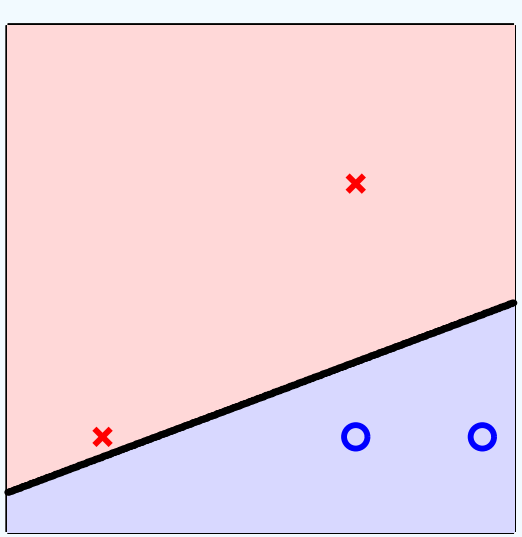
Separating Hyperplane

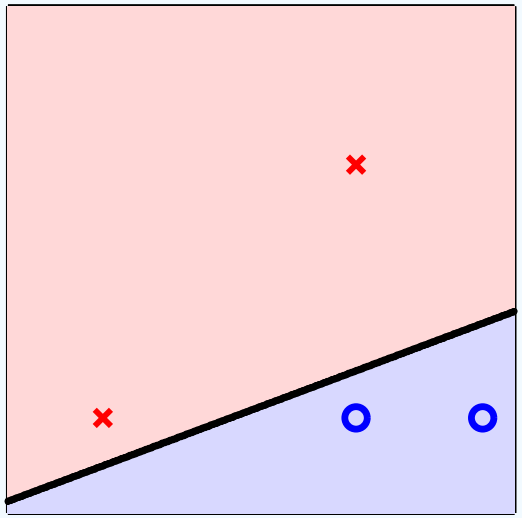


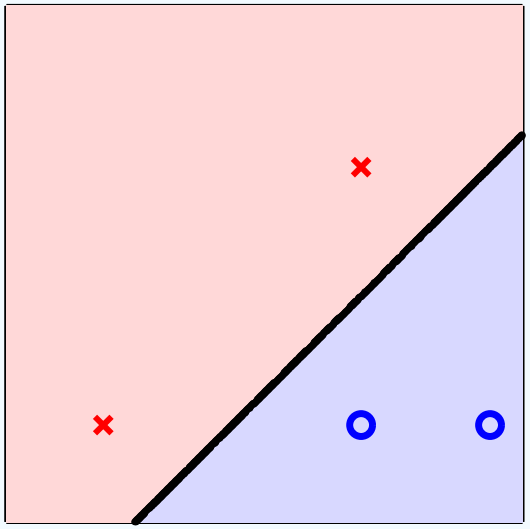


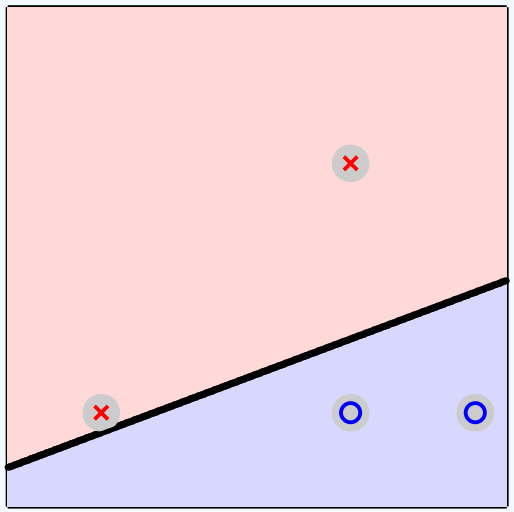


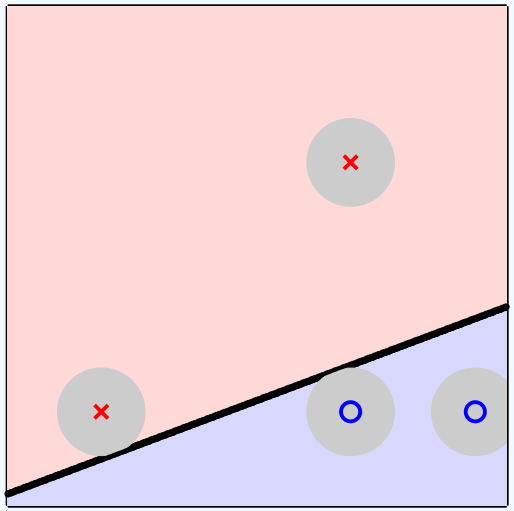
Separating Hyperplane

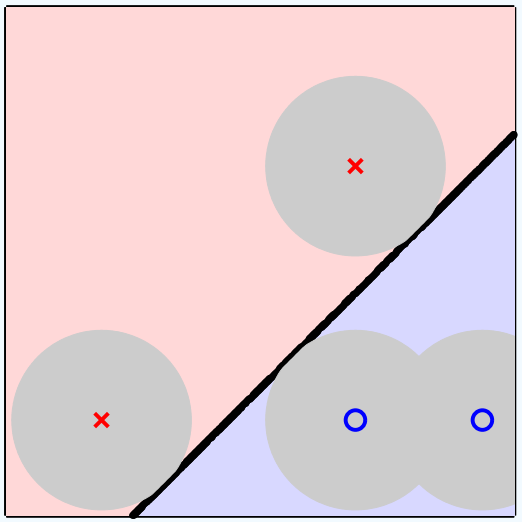




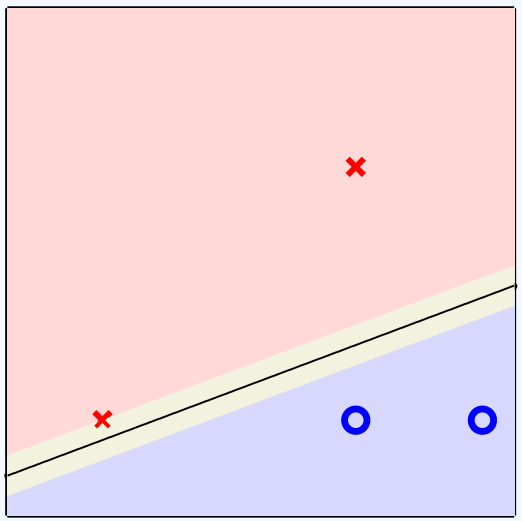


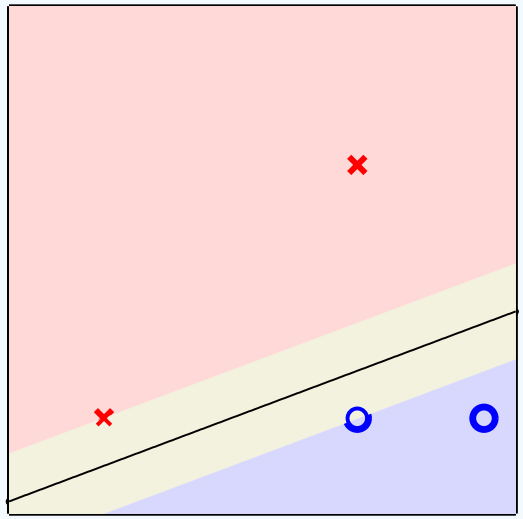


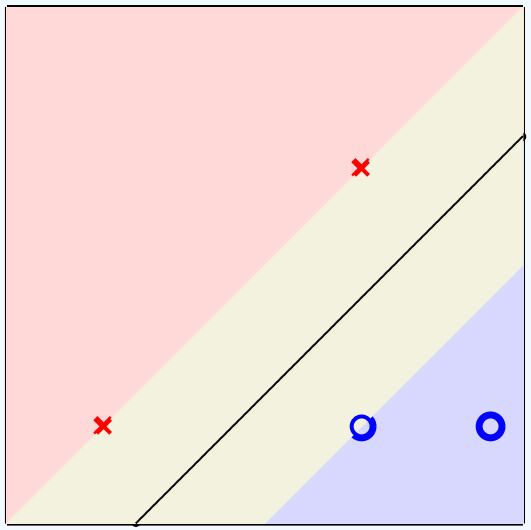




The Maximal Margin Classifier



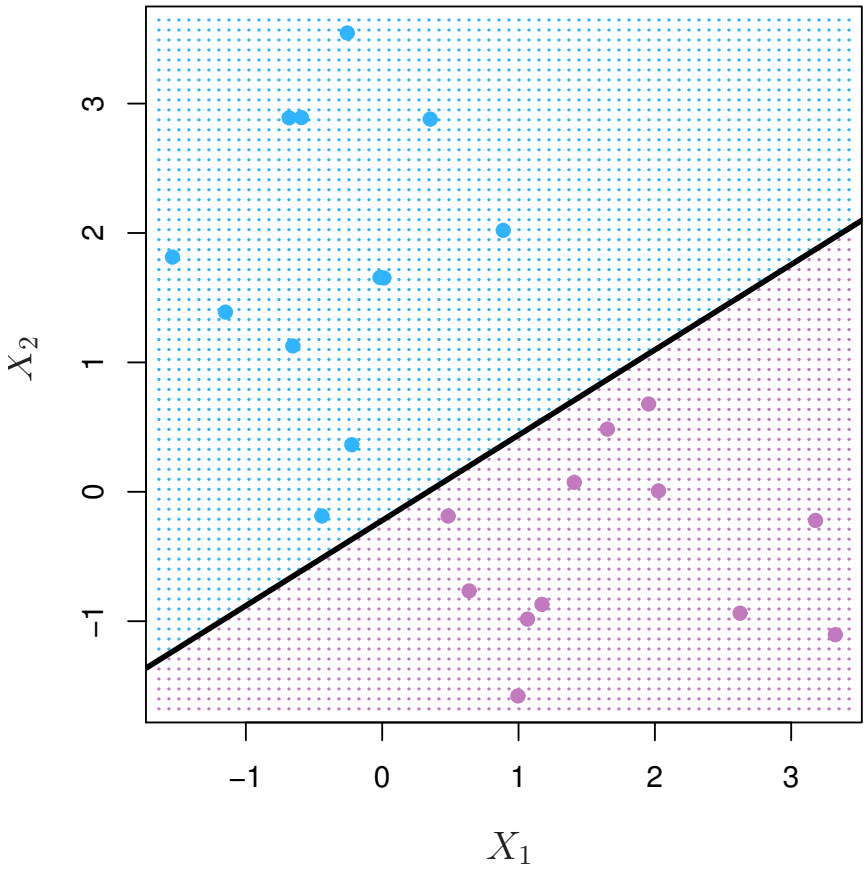


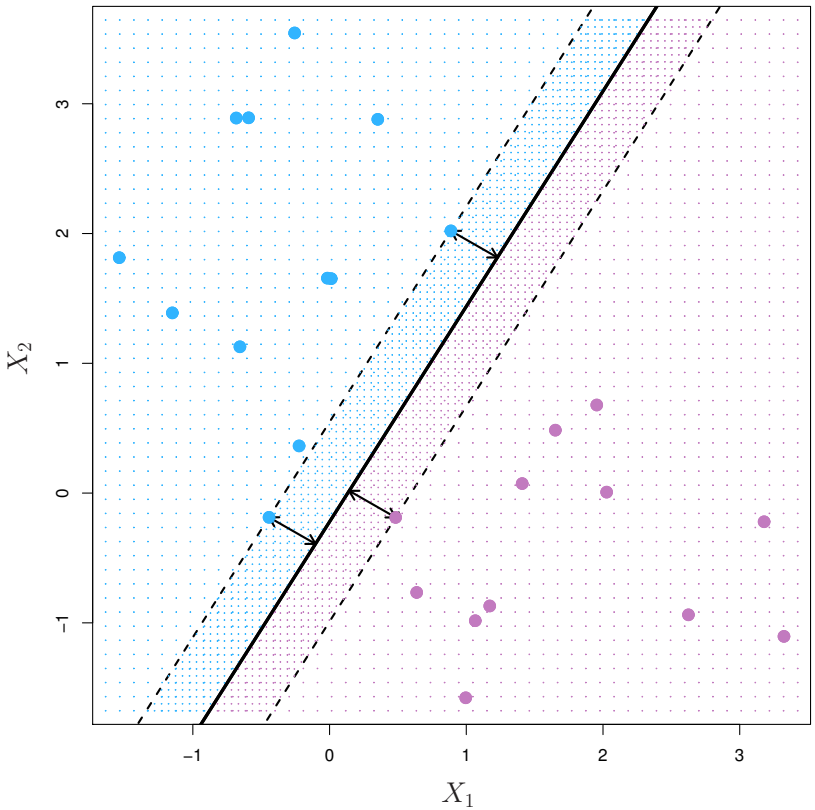


The Maximal Margin Classifier

- If our data can be perfectly separated using a hyperplane, then there will in fact exist an **infinite number of such hyperplanes**.
- We compute the the (perpendicular) distance from each training observation to a given separating hyperplane. The *smallest* such distance is know as the **margin**.
- The **optimal separating hyperplane** (or **maximal margin hyperplane**) is the separating hyperplane for which the margin is *largest*
- We can then classify a test observation based on which side of the maximal margin hyperplane it lies. This is known as the **maximal margin classifier**.

The Maximal Margin Classifier





Support vectors

- The **support vectors** are equidistant from the maximal margin hyperplane and lie along the dashed lines indicating the width of the margin.
- They **support** the maximal margin hyperplane in the sense that if these points were moved slightly then the maximal margin hyperplane would move as well
- The maximal margin hyperplane depends directly on the support vectors, but **not on the other observations**

The Maximal Margin Classifier

If $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$ for $i = 1, \ldots, n$, then the maximal margin hyperplane is the solution to the problem ($M > 0$):

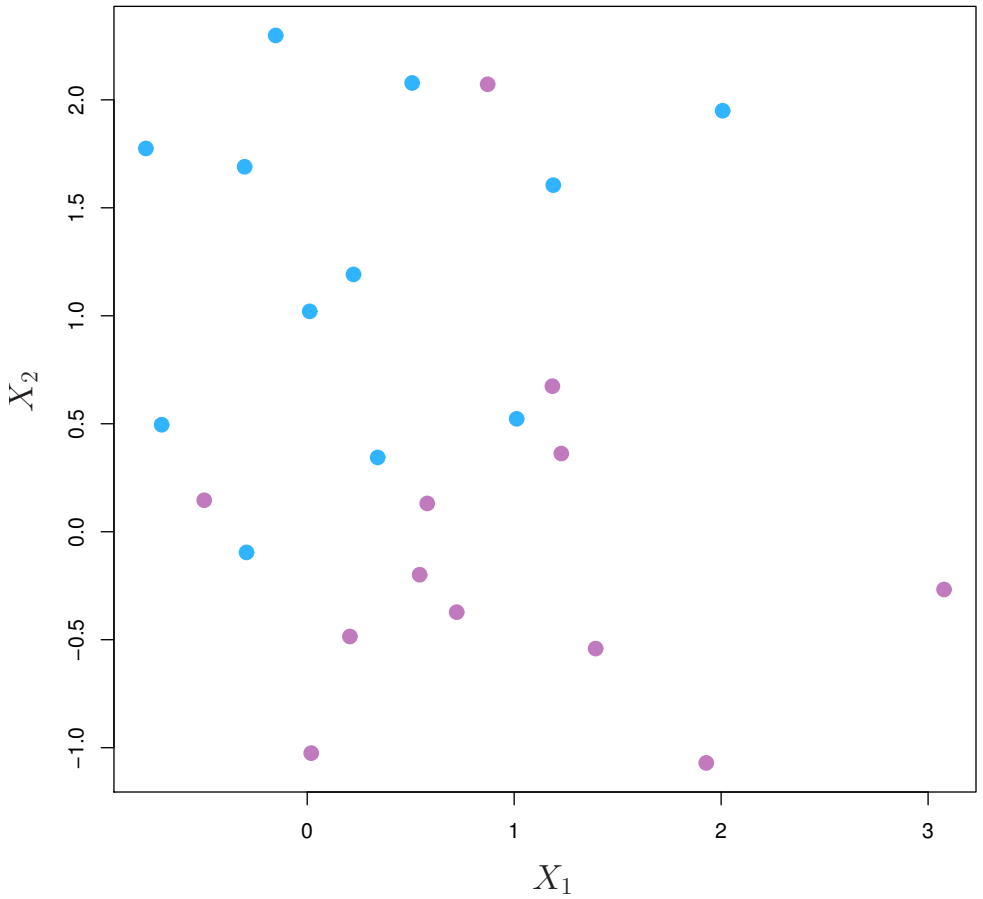
$$\underset{\beta_0, \beta_1, \ldots, \beta_p, M}{\text{maximize}} \; M$$

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

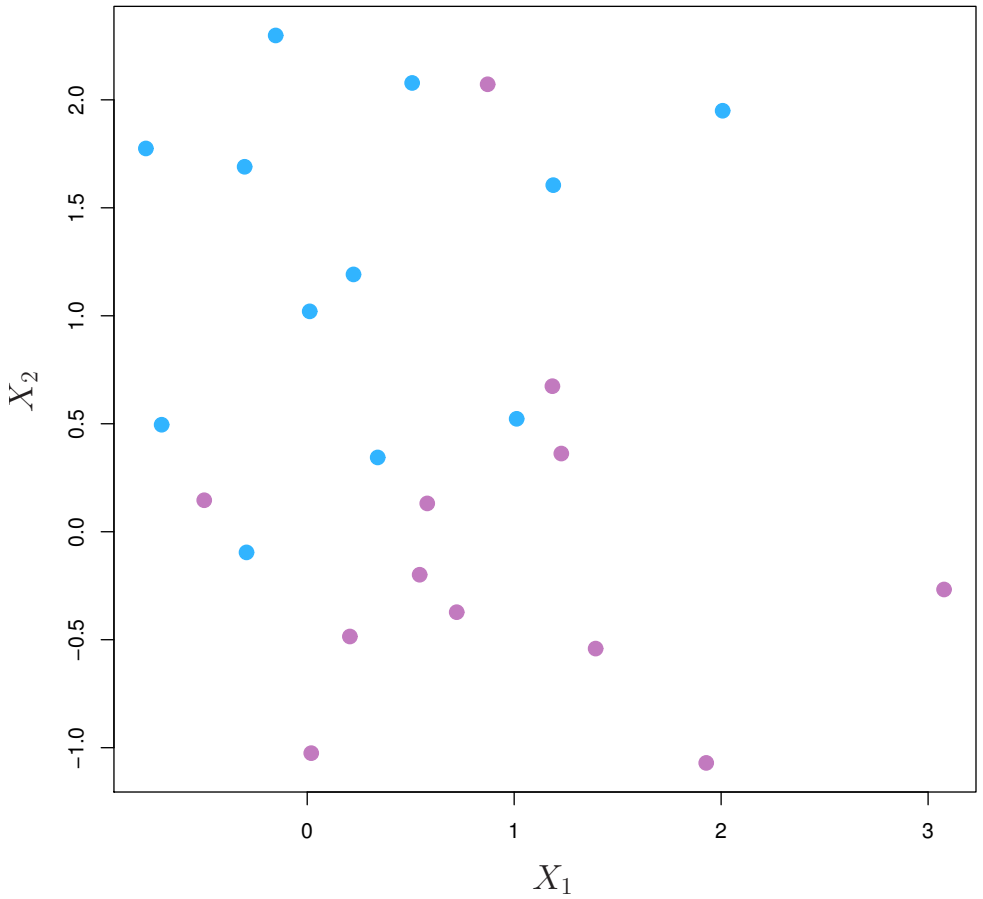
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M \;\; \forall \, i = 1, \ldots, n.$$

- For correct classification, we **only** need $y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) > 0$.
- With the equality constraint, the inequality constraint ensures that each observation is on the **correct side** of the hyperplane and **at least a (margin) distance** M from the hyperplane.

Non-separable case

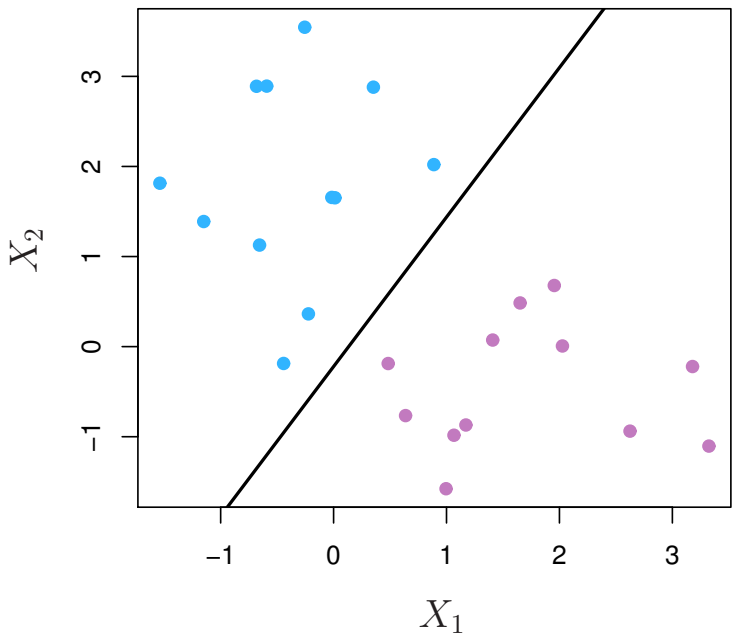


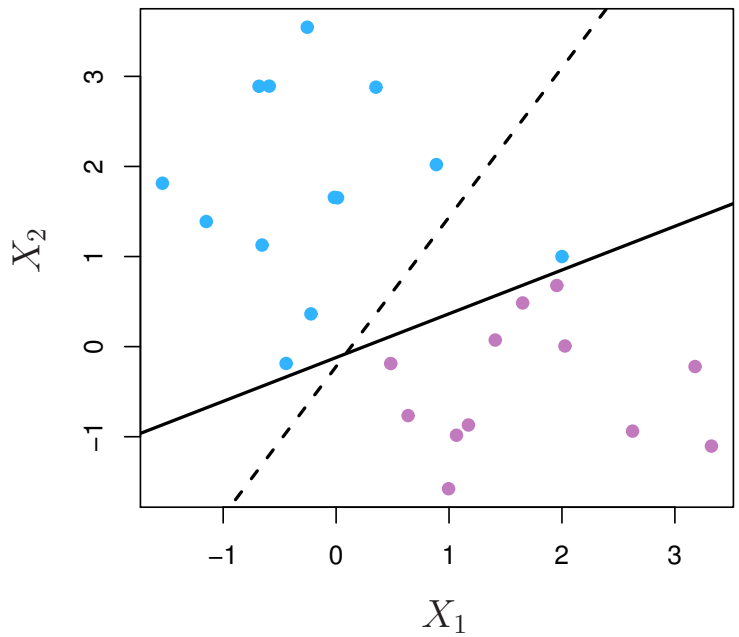
Non-separable case



The optimization problem has no solution with $M > 0$.

Lack of robustness





- The addition of a single observation leads to a large change in the maximal margin hyperplane $\rightarrow$ might overfit the training data.
- The resulting hyperplane has a tiny margin $\rightarrow$ small confidence

Support Vector Classifier

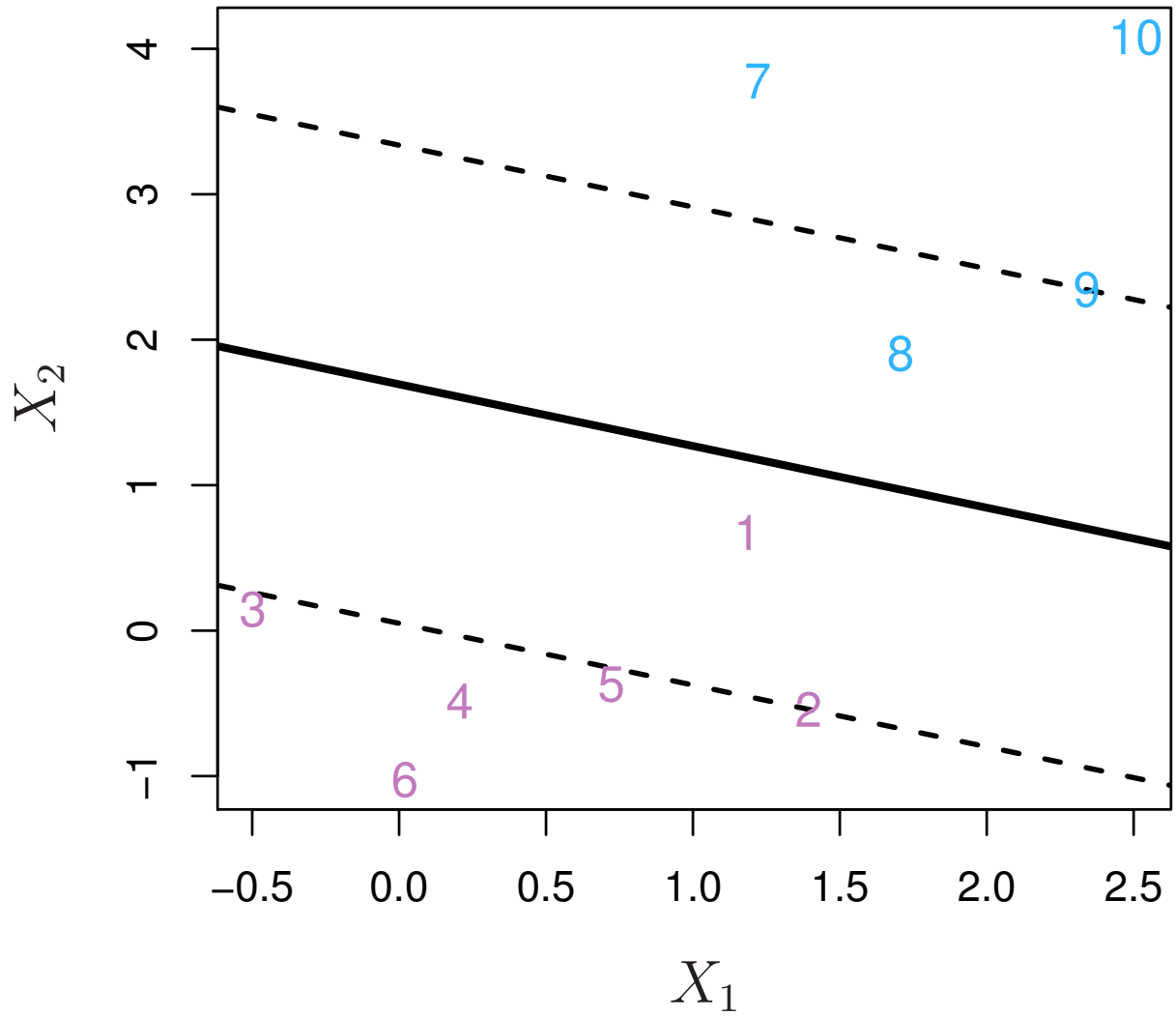
The *support vector classifier* (SVC) is based on a hyperplane that does not **perfectly separate** the two classes, in the interest of

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

The SVC is also called a *soft margin classifier* because the margin can be violated by some of the training observations.

- Most of the observations are on the **correct side of the margin**.
- Few observations are on the **wrong side of the margin**.
- Few observations are on **wrong side of the margin** and **the hyperplane**.

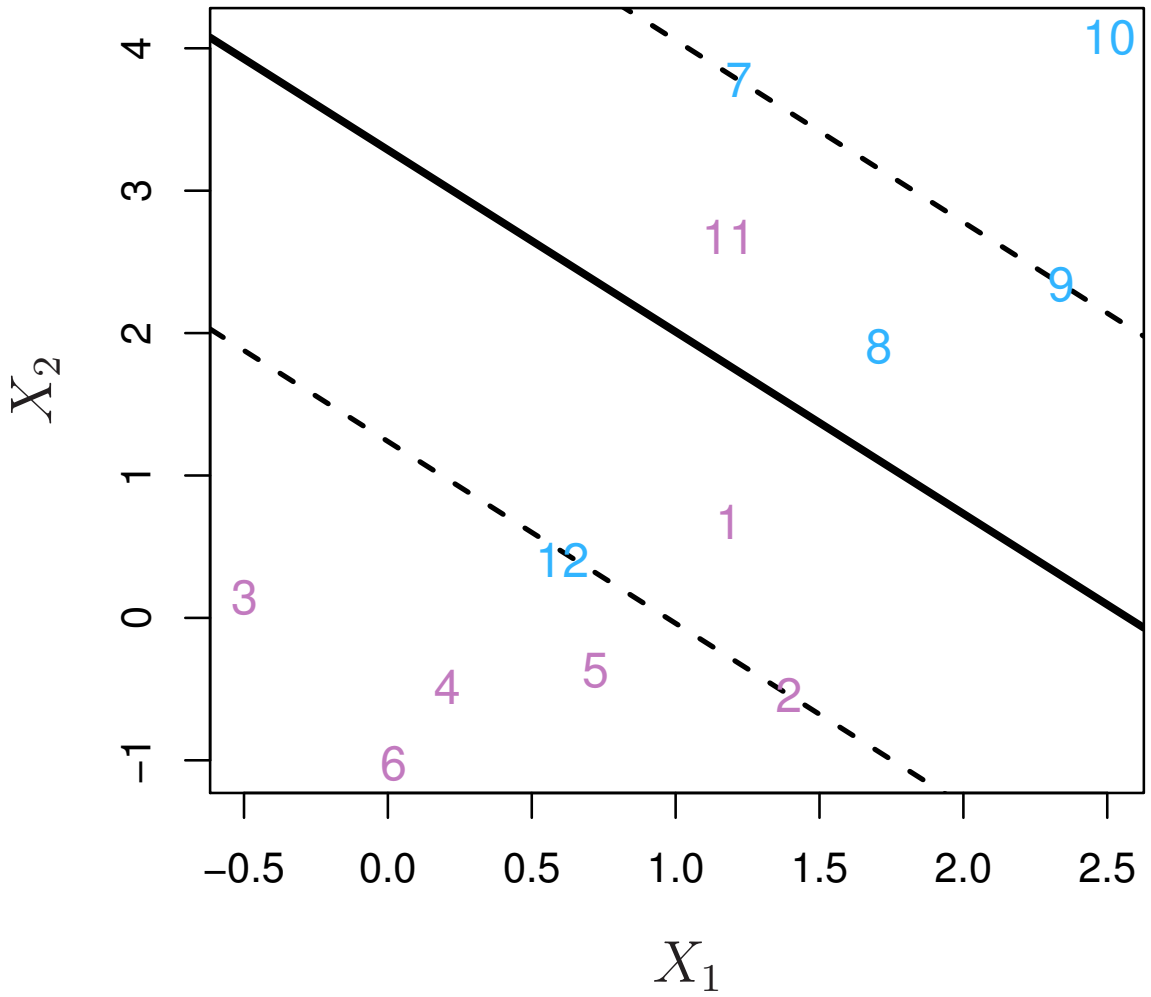
Example I



Example I

- Purple observations
  - 3, 4, 5 and 6: correct side of the margin (and correct side of the hyperplane)
  - 2 is on the margin
  - 1 is on the wrong side of the margin (and correct side of the hyperplane)
- Blue observations
  - 7, 10: correct side of the margin (and correct side of the hyperplane)
  - 9 is on the margin
  - 8 is on the wrong side of the margin (and correct side of the hyperplane)

Example II



- 11 and 12: wrong side of the hyperplane (and wrong side of the margin)

Support Vector Classifier

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

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

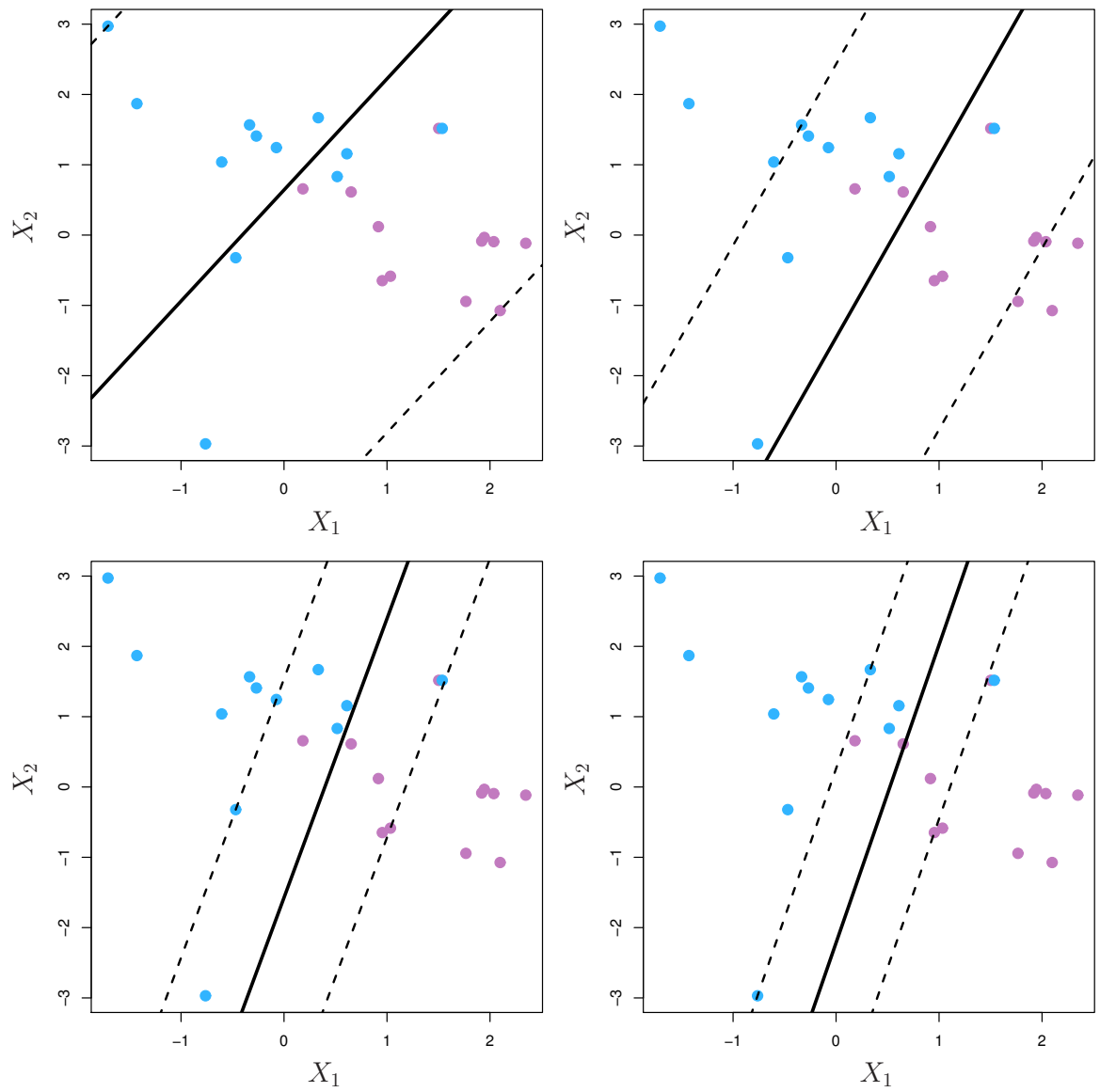
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

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

where ε_i tells us where the ith observation is located and C is a nonnegative **tuning parameter**.

- $\varepsilon_i = 0$: *correct side of the margin*,
- $\varepsilon_i > 0$: *wrong side of the margin* (violation of the margin),
- $\varepsilon_i > 1$: *wrong side of the hyperplane*.

Tuning parameter



Decreasing values of the tuning parameter C.

Tuning parameter

- C determines the **number** and **severity** of the violations to the margin (and to the hyperplane) that we will **tolerate**
  - $C = 0$: no budget for violations to the margin, and $\varepsilon_1 = \cdots = \varepsilon_n = 0$
  - $C > 0$: no more than C observations can be on the *wrong side of the hyperplane*
- C increases/decreases $\rightarrow$, margin wider/more narrow
- An observation that lies strictly on the correct side of the margin **does not affect** the classifier
- Only observations that either lie on the margin or that violate the margin (known as support vectors) **will affect** the classifier

Bias and variance tradeoff

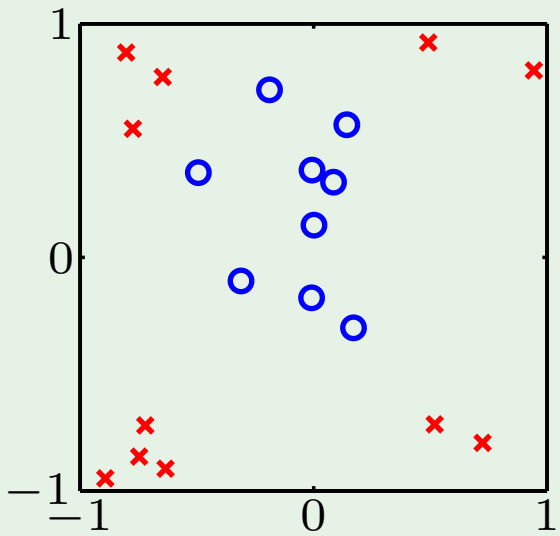
- C controls the bias-variance trade-off of the classifier
  - C large: high bias but low variance
  - C small: low bias but high variance
- When C is large, many observations violate the margin, and so there are many support vectors. Many observations are involved in determining the hyperplane.
- When C is small, then there will be fewer support vectors and hence the resulting classifier will have low bias but high variance
- In practice, C is generally chosen via cross-validation

The dependence on a few support vectors means that it is quite *robust to observations that are far away from the hyperplane*.

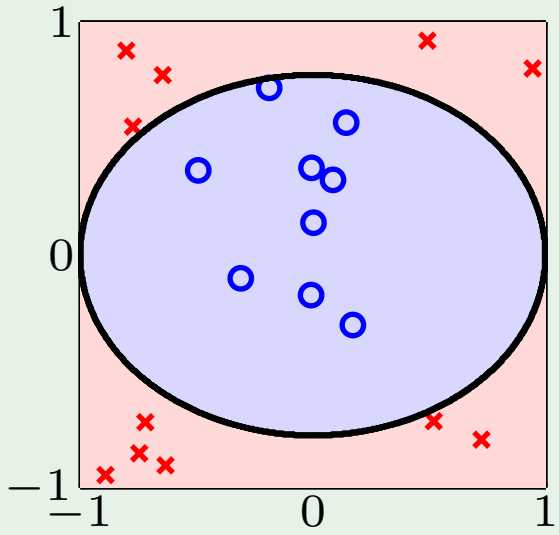
Non-linear Decision Boundaries

Linear is limited

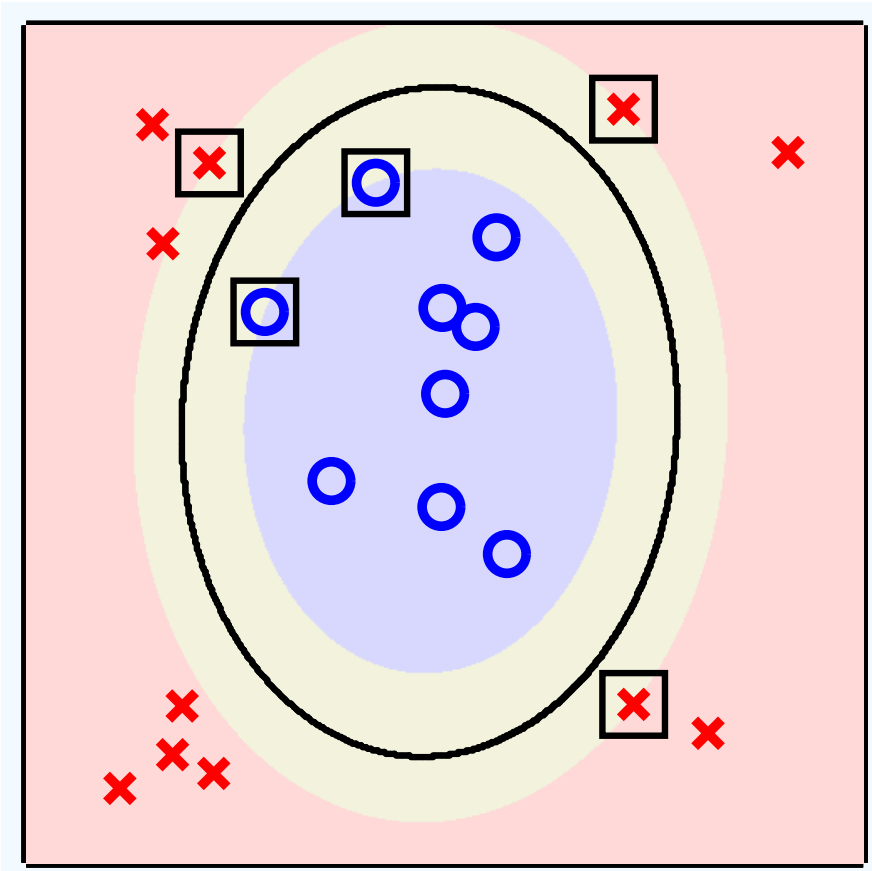
Data:

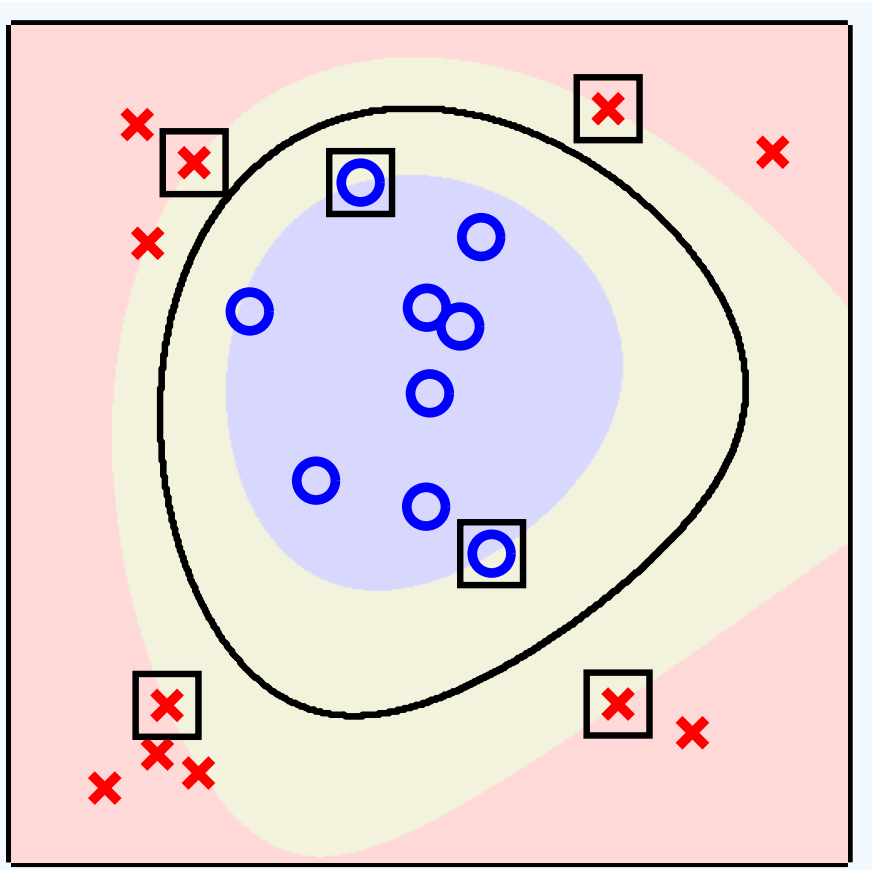


Hypothesis:



Non-linear Decision Boundaries





Enlarged feature space

We can consider **enlarging the feature space** using functions of the predictors such as *quadratic* and *cubic terms*, in order to address this non-linearity.

1. Obtain new features by computing **nonlinear transformations** of original features:

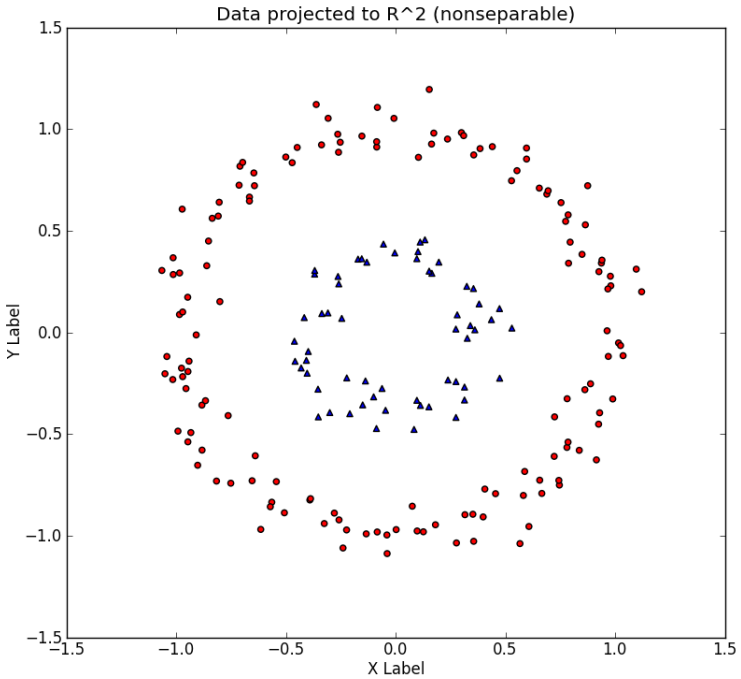
$$X = (X_1, X_2, \ldots, X_p) \xrightarrow{\Phi} Z = (Z_1, Z_2, \ldots, Z_{\tilde{p}})$$

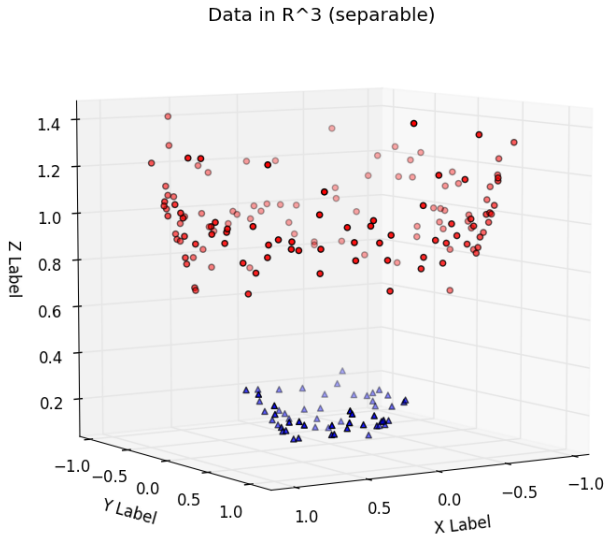
where $Z_j = \phi_j(X)$.

2. Run the support vector classifier using the **new features**:

$$h(Z) = \beta_0^{'} + \beta_1^{'} Z_1 + \beta_2^{'} Z_2 + \ldots \beta_{\tilde{p}}^{'} Z_{\tilde{p}}$$

Enlarged feature space





source: `http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html`

Original feature space: X_1, X_2

Enlarged feature space: $Z_1 = X_1, Z_2 = X_2$ and $Z_3 = X_1^2 + X_2^2$

The decision boundary of the support vector classifier will be **linear** in the enlarged feature space, but (generally) **nonlinear** in the original feature space.

Support vector machines

- There are many possible ways to enlarge the feature space, and that unless we are careful, we could end up with a **huge number of features**.
- The support vector machine (SVM) is an extension of the support vector support classifier that results from enlarging the feature space in a way that leads to **efficient computations**
- The support vector machines (SVM) uses the **kernel trick**

Support vector machines

The **inner product** of $a, b \in \mathbb{R}^p$ is $\langle a, b \rangle = \sum_{i=1}^{p} a_i b_i$.
It can be shown that

- The **linear support vector** classifier can be represented as

$$h(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

  which depends on n parameters $\alpha_i, i = 1, \ldots, n$.

- To estimate the parameters $\alpha_1, \alpha_2, \ldots, \alpha_n$ and β_0, all we need are the $\binom{N}{2}$ **inner products** between all pairs of training observations.

In representing the linear classifier $h(x)$, and in computing its coefficients, all we need are **inner products**.

Support vector machines

It can be shown that the **support vector machine (SVM)** classifier can be represented as

$$h(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \, \langle \Phi(x), \Phi(x_i) \rangle.$$

where $\Phi : \mathbb{R}^p \to \mathbb{R}^{\tilde{p}}$ is the non-linear transformation.

- The previous expression involves $\Phi(x)$ only through inner products.
- If the transform *and* the inner product can be jointly computed independently of $\tilde{p}$, the nonlinear SVM will be an efficient procedure that does not require computing/storing $\Phi(x)$ for large or even infinite $\tilde{p}$

Good news! This can be done and is called the **kernel trick**.

The kernel trick

- The kernel function, or just kernel, define a function that combines both the transform and the inner product

$$K(x, x') = \langle \Phi(x), \Phi(x') \rangle = \Phi(x)^T \Phi(x').$$

- It appears that the kernel transforms the inputs x and x' to the $\mathcal{Z}$ space and then computes the inner product. The computational complexity will depend on $\tilde{p}$!
- We will see that the kernel can be efficiently computed with a cost proportional to p instead of $\tilde{p}$

The kernel trick

Consider the second-order polynomial transform:

$$\Phi(x) = (1, x_1, x_2, \ldots, x_p, x_1x_1, x_1x_2, \ldots, x_px_p)$$

where $\tilde{p} = 1 + p + p^2$. A direct computation of

$$\Phi(x)^T\Phi(x') = 1 + \sum_{i=1}^{p} x_i x_i' + \sum_{i=1}^{p}\sum_{j=1}^{p} x_i x_j x_i' x_j'$$

takes time $O(\tilde{p}) = O(p^2)$.
However, we can write

$$\sum_{i=1}^{p}\sum_{j=1}^{p} x_i x_j x_i' x_j' = \sum_{i=1}^{p} x_i x_i' \times \sum_{j=1}^{p} x_j x_j' = (x^T x')^2.$$

and

$$K(x, x') = \Phi(x)^T\Phi(x') = 1 + (x^T x') + (x^T x')^2.$$

which takes time $O(p)$ to compute.

The kernel trick

- We can see that the previous kernel computes an **inner product** in an enlarged space
- we **do not need** to specify the transformation $\Phi(x)$ at all, but require only knowledge of the **kernel function**
- Any learning technique that uses inner products can benefit from the kernel trick

Examples of kernels

Linear kernel:

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

Polynomial kernel of degree *d*:

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

where *d* is a positive polynomial integer.

Radial kernel:

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

where γ is a positive constant.

Examples of kernels

There are a lot of other kernel choices. One can even design new kernels that better suit the learning task at hand.

$K(x_i, x_{i'})$ is a valid kernel function if and only if the kernel matrix

$$K = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \dots & K(x_1, x_n) \\ K(x_2, x_1) & K(x_2, x_2) & \dots & K(x_2, x_n) \\ \dots & \dots & \dots & \dots \\ K(x_n, x_1) & K(x_n, x_2) & \dots & K(x_n, x_n) \end{bmatrix}$$

is always symmetric positive semi-definite for any given $\{x_1, \dots, x_n\}$.

SVM and logistic regression