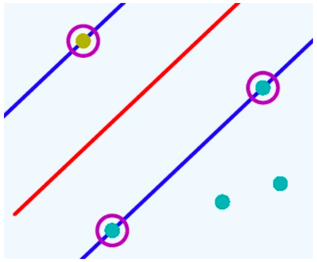
MACHINE LEARNING

COMP8220

04 – Support Vector Machine



Xuyun Zhang (Department of Computing)

Acknowledgement

- Some slides are from the S1 version designed by Rolf Schwitter (Rolf.Schwitter@mq.edu.au)

Some slides are also based on book of Christopher Bishop "Pattern Recognition and Machine Learning" Springer-Verlag New York (2006), and the slides made by Torsten Möller from this book.

Lecture Outline

- Linear Support Vector Machine
  - How SVM comes?
  - SVM learning problem formulation
  - Solving SVM optimization problem

- SVM for Non-linearity
  - SVM with soft margin
  - Kernel trick

Lecture Outline

MACQUARIE University

- Linear Support Vector Machine
  - How SVM comes?
  - SVM learning problem formulation
  - Solving SVM optimization problem

Review Classification Models

- Two methods for the discriminant function

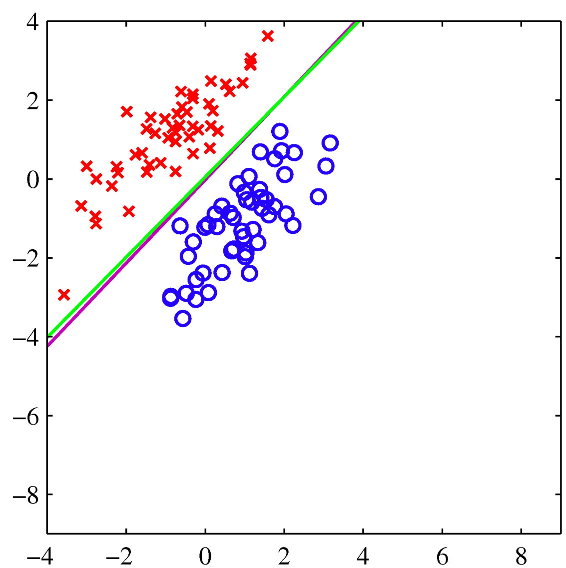
$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

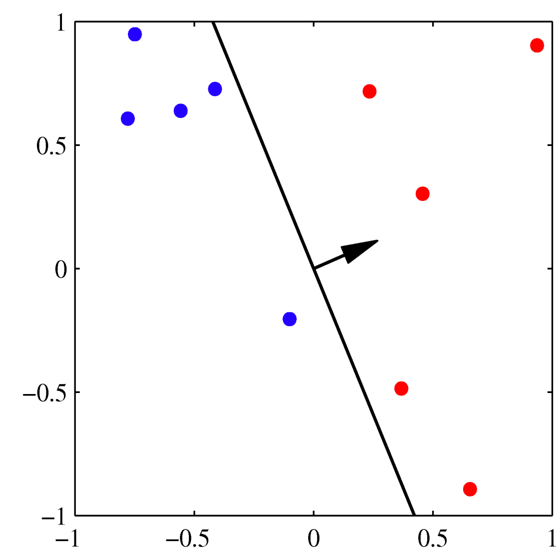
Or more generally with feature map function,

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \emptyset(\boldsymbol{x}) + w_0$$

  - Least squares for classification
  - Perceptron

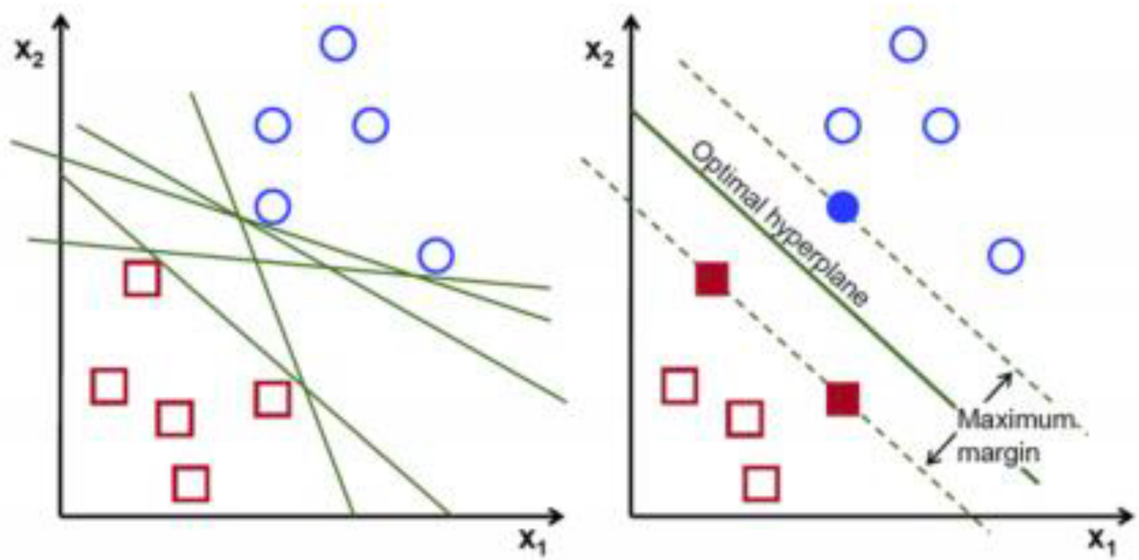
- There are many possible optimal decision boundaries for perceptron
- Which is the best?





Max Margin Criterion

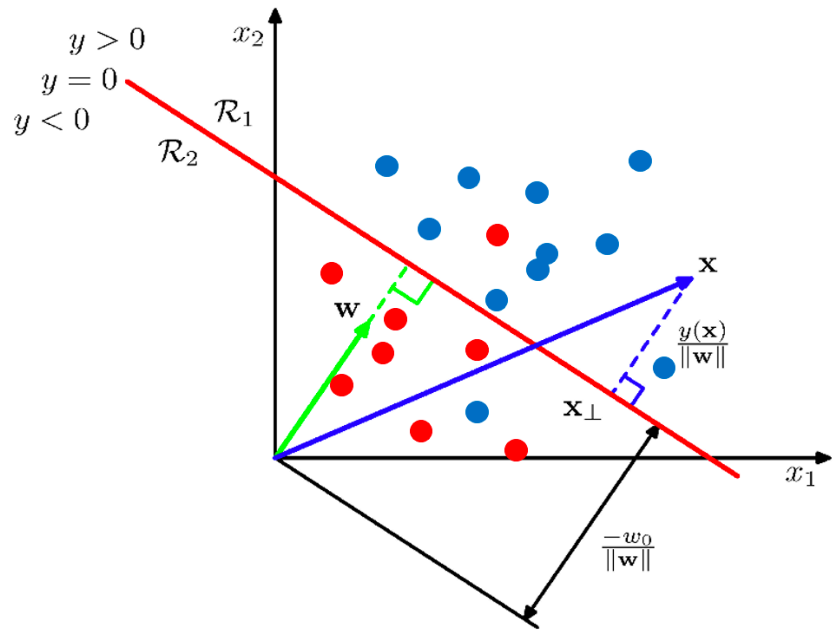
- Margin of a classification boundary (hyperplane): the minimum distance to any example



- Idea of Support Vector Machines (SVM)
  - Choose the decision boundary which maximizes the margin

Margin Calculation

- How to calculate the margin? (two-class case)
  - How to calculate distance between a point and a hyperplane?



  - Signed distance of $\boldsymbol{x}$ to the decision surface (denoted as r)

$$r = \frac{y(\boldsymbol{x})}{\|\boldsymbol{w}\|}$$

Margin Calculation

- Then, we can get (unsigned) distance

$$t\frac{y(\boldsymbol{x})}{||\boldsymbol{w}||}$$

  - t takes values in $\{1,-1\}$

- The margin can be calculated by

$$\min_n\{t_n\frac{y(\boldsymbol{x}_n)}{||\boldsymbol{w}||}\}$$

$$=\frac{1}{||\boldsymbol{w}||}\min_n\{t_n(\boldsymbol{w}^T\boldsymbol{x}_n+w_0)\}$$

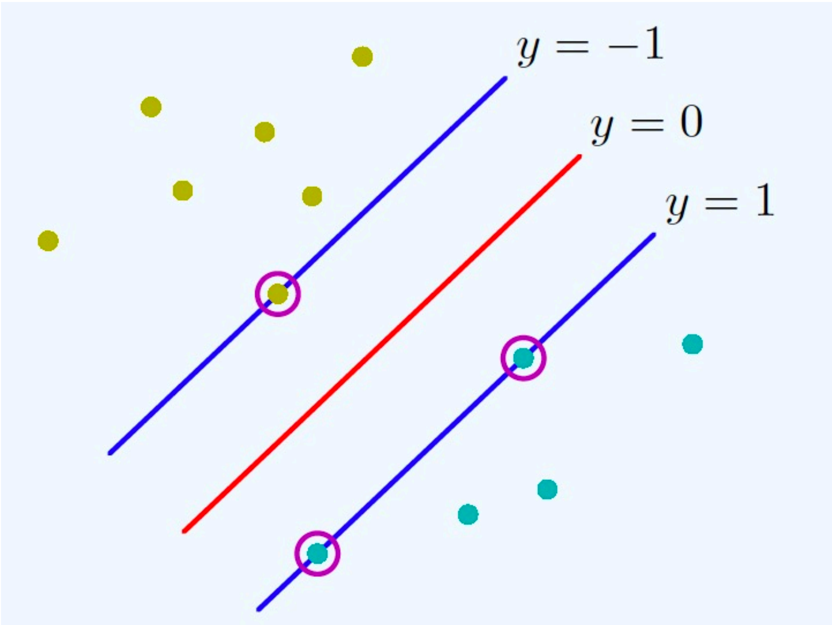
  - $||\boldsymbol{w}||$ is independent of n
  - The minimum distance to any example

SVM Learning Problem

- The learning problem is to choose $\boldsymbol{w}$ and w_0 to maximise the margin

$$\arg\max_{\boldsymbol{w},w_0}\{\frac{1}{||\boldsymbol{w}||}\min_n\{t_n(\boldsymbol{w}^T\boldsymbol{x}_n + w_0)\}\}$$

  - Points with the min value are known as support vectors



Solving Learning Problem

- The unconstrained optimization problem is complex

$$\arg\max_{\boldsymbol{w},w_0}\{\frac{1}{||\boldsymbol{w}||}\min_n\{t_n(\boldsymbol{w}^T\boldsymbol{x}_n + w_0)\}\}$$

- Let $C = \min_n\{t_n(\boldsymbol{w}^T\boldsymbol{x}_n + w_0)\}$, we can have

$$t_n(\boldsymbol{w}^T\boldsymbol{x}_n + w_0) \geq C$$

- So, the optimization problem can be re-formulated as

$$\arg\max_{\boldsymbol{w},w_0}\{C\frac{1}{||\boldsymbol{w}||}\}$$

s.t.

$$t_n(\boldsymbol{w}^T\boldsymbol{x}_n + w_0) \geq C$$

Solving Learning Problem

MACQUARIE University

- An use property of a hyperplane
  - If $\boldsymbol{w} = \kappa \boldsymbol{v}$ and $w_0 = \kappa v_0$, the two hyperplanes are the same, where κ is a non-zero constant
$$\boldsymbol{w}^T \boldsymbol{x} + w_0 = 0 \text{ and } \boldsymbol{v}^T \boldsymbol{x} + v_0 = 0$$
  - So, we can always find a κ to make
$$\min_n \{t_n(\boldsymbol{v}^T \boldsymbol{x}_n + v_0)\} = 1$$
- Then, we can have another optimization problem
$$\arg\max_{\boldsymbol{v}, v_0} \{\frac{1}{||\boldsymbol{v}||}\}$$

s.t.
$$t_n(\boldsymbol{v}^T \boldsymbol{x}_n + v_0) \geq 1$$

Solving Learning Problem

- Observation
  - The two optimization problem will produce the same hyperplane (classification decision boundary)
  - Just denotation difference: $\boldsymbol{w}$ vs $\boldsymbol{v}$, and w_0 vs v_0
  - But the second from is easier to solve!
- Then, we just need to solve the following constrained optimization problem

$$\arg\max_{\boldsymbol{w},w_0}\{\frac{1}{||\boldsymbol{w}||}\}$$

s.t.

$$t_n(\boldsymbol{w}^T\boldsymbol{x}_n + w_0) \geq 1$$

Canonical Representation

- Further observations
  - Maximising $\frac{1}{||\boldsymbol{w}||}$ is equivalent to minimizing $||\boldsymbol{w}||$
  - Minimizing $||\boldsymbol{w}||$ is equivalent to minimizing $\frac{1}{2}||\boldsymbol{w}||^2$
- So, we can have the canonical representation of the SVM learning problem

$$\boldsymbol{w}^*, w_0^* = \arg\min_{\boldsymbol{w}, w_0} \frac{1}{2}||\boldsymbol{w}||^2$$

s.t.

$$t_n(\boldsymbol{w}^T\boldsymbol{x}_n + w_0) \geq 1$$

Lagrangian Multipliers

- To solve the constrained optimization probelm, we need to use the Lagrangian Multipliers method to concert it to an unconstrained one

$$L(\boldsymbol{w}, w_0, \boldsymbol{a}) = \frac{||\boldsymbol{w}||^2}{2} - \sum_{n=1}^{N} a_n[t_n(\boldsymbol{w}^T \boldsymbol{x}_n + w_0) - 1]$$

- Set the derivatives of L w.r.t. $\boldsymbol{w}$ and w_0 to 0, we get

$$\boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{x}_n$$

$$0 = \sum_{n=1}^{N} a_n t_n$$

Dual Representation

MACQUARIE University

- Plugging those equations into L removes $\boldsymbol{w}$ and w_0, and results in a version of L where $\nabla_{\boldsymbol{w},w_0} L = 0$:

$$\tilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m \boldsymbol{x}_n^T \boldsymbol{x}_m$$

- This is the dual representation of the problem

$$\arg\max_{\boldsymbol{a}} \tilde{L}(\boldsymbol{a})$$

s.t.

$$\sum_{n=1}^{N} a_n\, t_n = 0$$

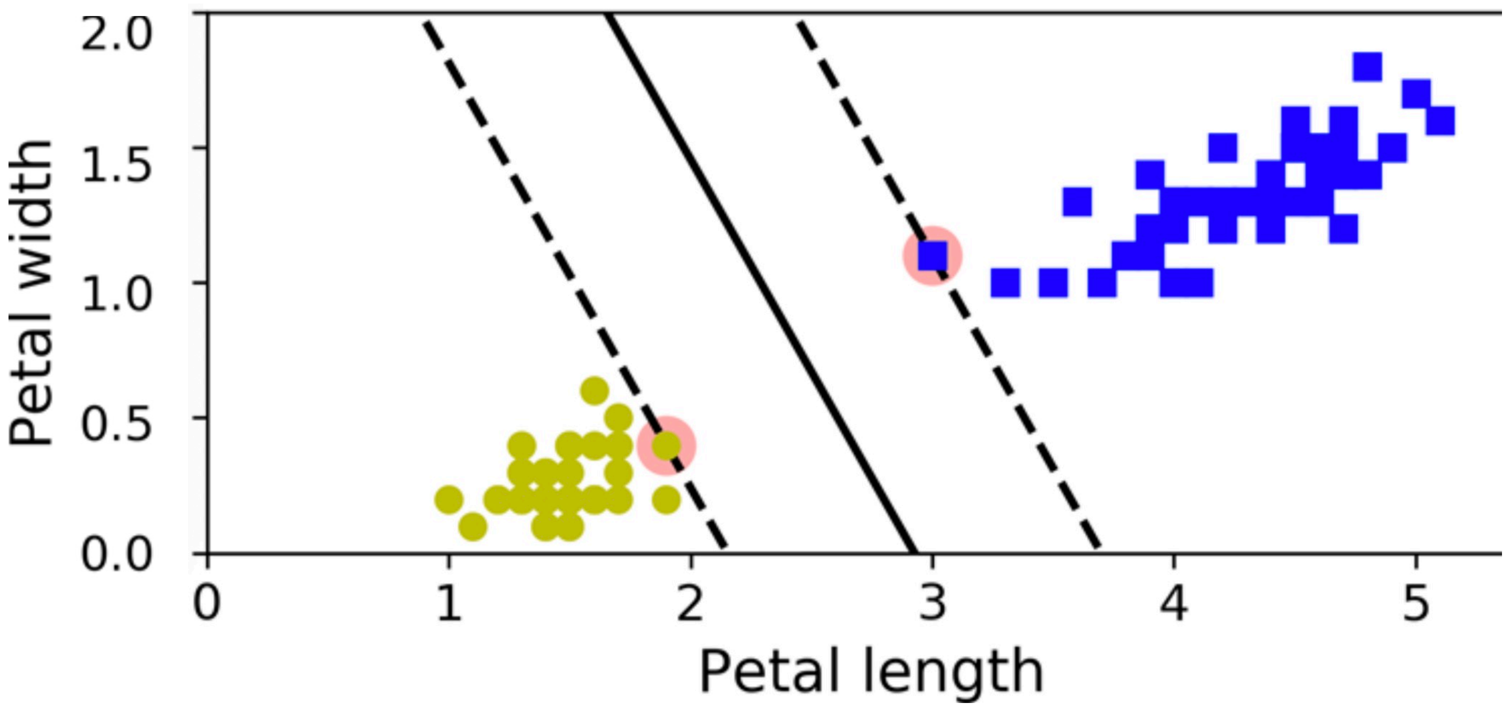
Solution

- The dual representation is easier to solve
  - Quadratic and convex in $\boldsymbol{a}$
  - Kernelized with K positive semi-definite
- Results

$$\boldsymbol{w}^* = \sum_{n=1}^{N} a_n t_n \boldsymbol{x}_n$$

  - Recall $a_n(t_n y(\boldsymbol{x}_n) - 1) = 0$ condition Lagrange. This makes either $a_n = 0$ or $\boldsymbol{x}_n$ is a support vector
  - a_n will be sparse – many zeros
  - Another formula for computing w_0^*
  - Generally, $\boldsymbol{x}_n$ can be replaced by $\emptyset(\boldsymbol{x}_n)$

Example



Lecture Outline

MACQUARIE University

- Linear Support Vector Machine
  - How SVM comes?
  - SVM learning problem formulation
  - Solving SVM optimization problem
- SVM for Non-linearity
  - SVM with soft margin
  - Kernel trick

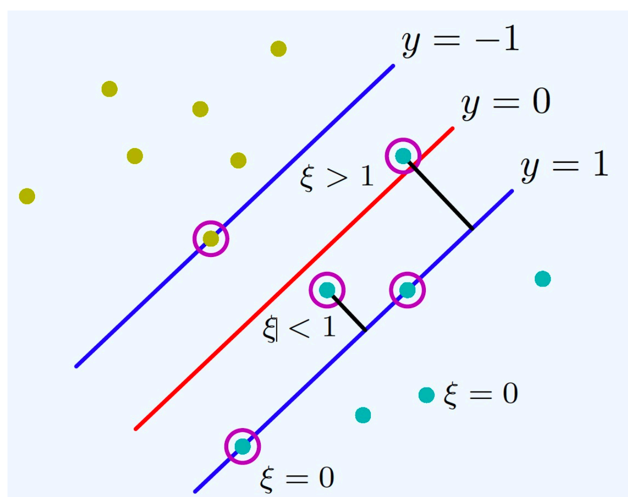
Soft Margin

❖ For most problems, data will not be linearly separable (even in feature space $\emptyset(\boldsymbol{x})$)

❖ We can relax the constraints from $t_n y(\boldsymbol{x}_n) \geq 1$ to

$$t_n y(\boldsymbol{x}_n) \geq 1 - \xi_n$$

- $\xi_n \geq 0$ are called slack variables
- $\xi_n = 0$ satisfy original problem



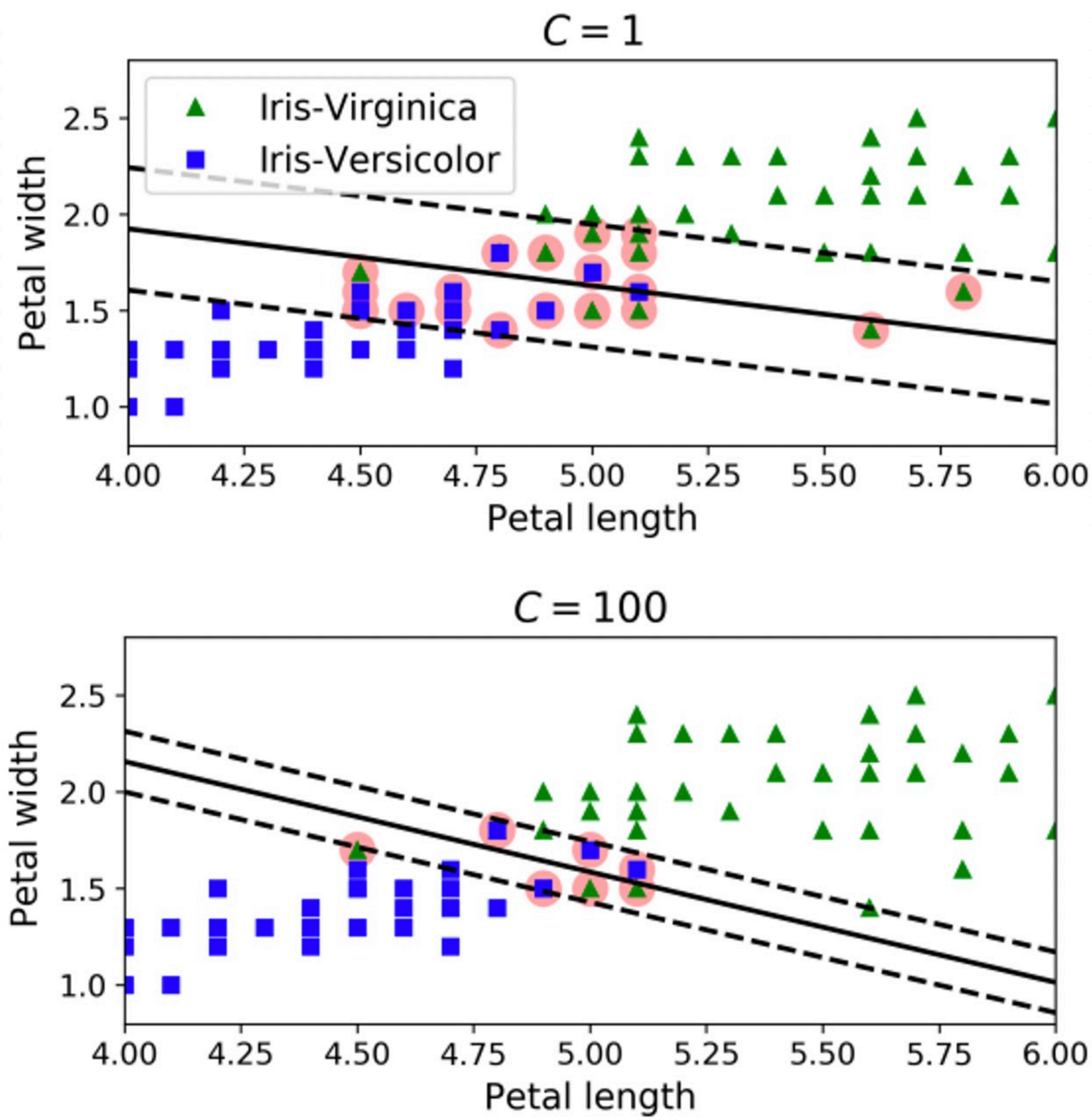
Optimization Problem

- Basic idea: non-zero slack variables are bad, penalize while maximizing the margin

$$\arg\min\{\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n=1}^{N}\xi_n\}$$

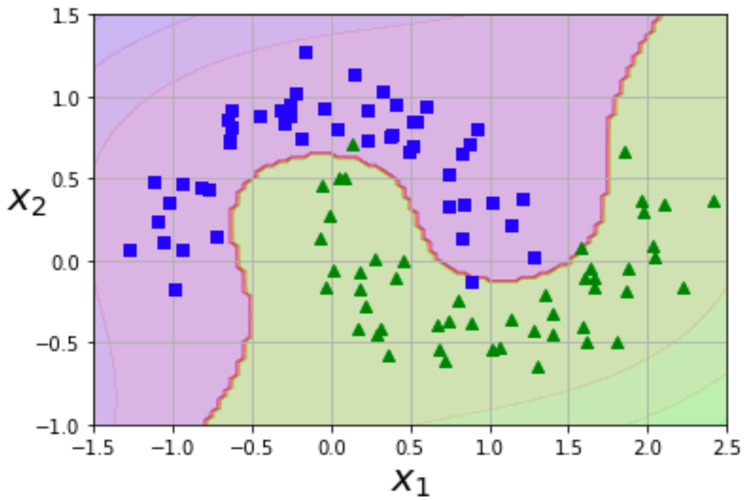
  - Optimization is same quadratic, different constraints, convex
  - Constant $C > 0$ controls the importance of large margin versus violation (non-zero slack)
    - Set using cross-validation (hyperparameter)
  - Also, C controls the complexity of the model
    - If the SVM model is overfitting, then reduce C

Soft Margin Example



Nonlinear SVM classification

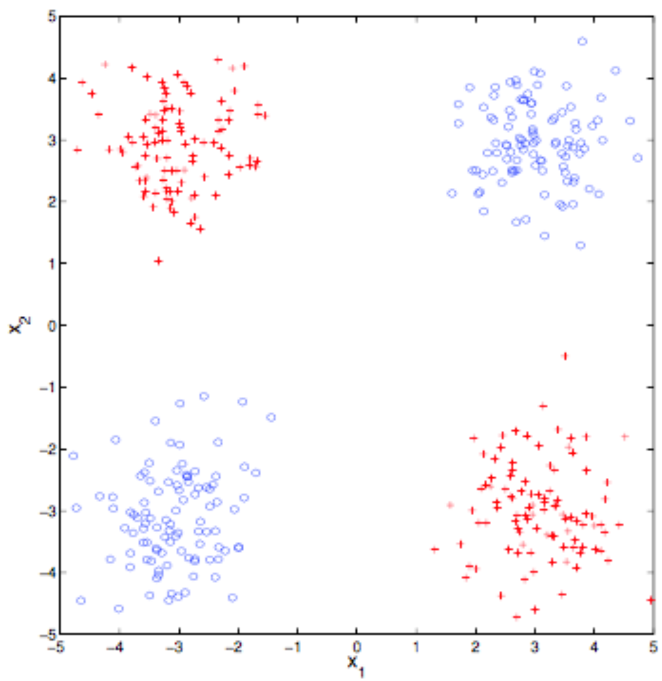
- Many datasets are not linearly separable

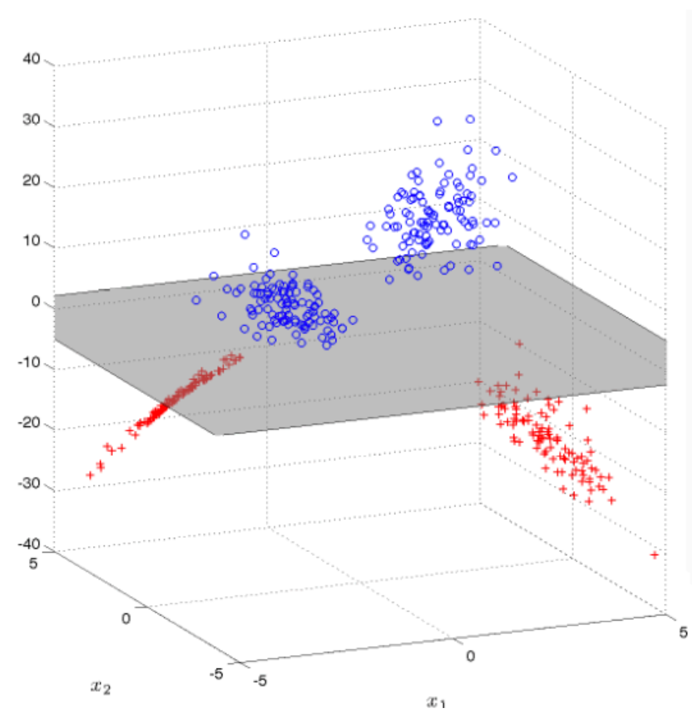


- One solution is to add polynomial features
  - May work with all sorts of ML algorithms, not only SVMs
  - This can result in linearly separable datasets
- We can then use a linear SVM classifier (SVC) together

Separation in High Dimension

- Separation may be easier in a higher dimension
  - As we investigate more features
  - XOR data example: Mapping the data from a 2-D space to a 3-D space allows us to find a separation hyperplane





Kernel Trick

- Mapping to high-dimensional feature space is promising
  - But the computation is usually expensive
- This is where the kernel trick comes in
  - The kernel trick allows us to operate in the original feature space without computing the coordinates of the data in a higher dimensional space
- A kernel is a function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ that can compute the dot (inner) product $\emptyset^T(\boldsymbol{x}_i)\emptyset(\boldsymbol{x}_j)$ based only on the original vector $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$
  - No need to know the transformation function $\emptyset(\cdot)$
  - E.g., $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T \boldsymbol{x}_j)^2$ is a 2nd-degree polynomial kernel

Kernel Trick Example

Given two points: $x = (2, 3, 4)$ and $y = (3, 4, 5)$.

We have $K(x, y) = <f(x), f(y)>$.

f is a map from n-dimensional to m-dimensional space.

$<x, y>$ denotes the dot product.

Let us first calculate $<f(x), f(y)>$.

$f(x) = (x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3)$

$f(y) = (y_1y_1, y_1y_2, y_1y_3, y_2y_1, y_2y_2, y_2y_3, y_3y_1, y_3y_2, y_3y_3)$

In our case:

$f(2, 3, 4) = (4, 6, 8, 6, 9, 12, 8, 12, 16)$ and

$f(3, 4, 5) = (9, 12, 15, 12, 16, 20, 15, 20, 25)$

Kernel Trick Example

So the dot product is:

$f(x) \cdot f(y) = f(2, 3, 4) \cdot f(3, 4, 5) =$

$(36 + 72 + 120 + 72 + 144 + 240 + 120 + 240 + 400) = 1444$

Now let us use the kernel instead:

$K(x, y) = (2*3 + 3*4 + 4*5)^2 = (6 + 12 + 20)^2 =$

$38 * 38 = 1444$

- The first method requires a lot of calculations because of projecting 3 dimensions into 9 dimensions, while using the kernel is much easier

SVM with Kernel

- As we already have

$$\boldsymbol{w}^* = \sum_{n=1}^{N} a_n t_n \emptyset(\boldsymbol{x}_n)$$

- Then, the discriminant function from SVM

$$y(\boldsymbol{x}) = \boldsymbol{w}^{*T} \emptyset(\boldsymbol{x}) + w_0$$

$$= \sum_{n=1}^{N} a_n t_n \underbrace{\emptyset^T(\boldsymbol{x}_n)\emptyset(\boldsymbol{x})}_{K(\boldsymbol{x}, \boldsymbol{x}_n)} + w_0$$

Some Kernels

Linear kernel $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{x}_1^T \boldsymbol{x}_2$

- $\phi(\boldsymbol{x}) = \boldsymbol{x}$

Polynomial kernel $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = (1 + \boldsymbol{x}_1^T \boldsymbol{x}_2)^d$

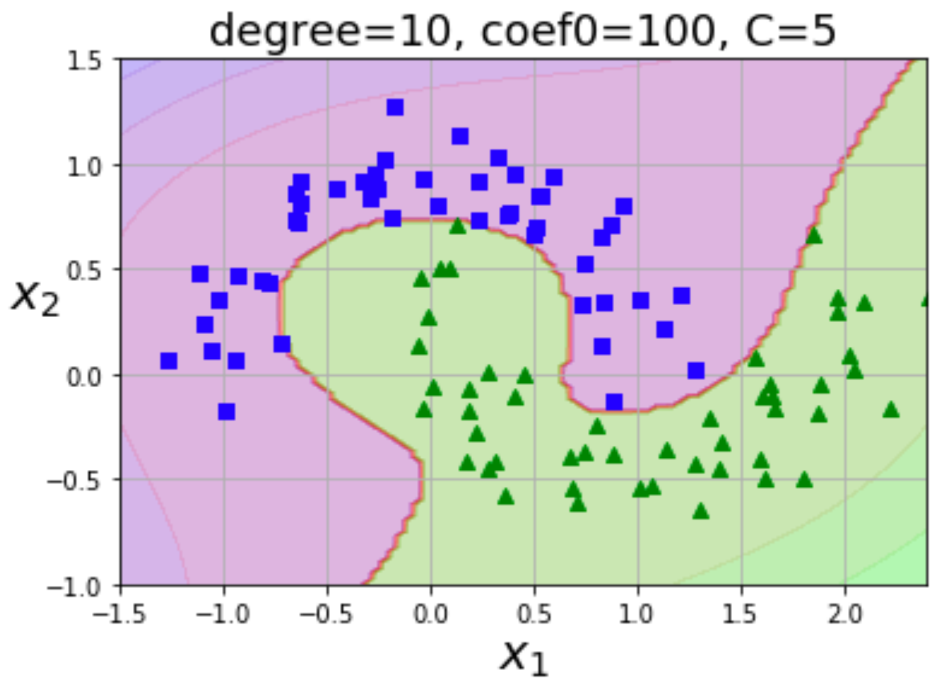
- Contains all polynomial terms up to degree d

Gaussian kernel $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(-||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2 / 2\sigma^2)$

- Infinite dimension feature space

Example

❖ Output: SVC with Polynomial Kernel



Example

❖ Output: Output: SVC with RBF Kernel