

Machine Learning (機器學習)

王麒璋 著

指導教授：丁建均老師 編修，徐嘉駿同學再編修

March, 2014

(October, 2014 由丁老師修改)

(Nov. 2015 由徐嘉駿同學修改)

第二版

Abstract

你是否想要了解 Machine Learning 的基礎概念，並且想要會用 Machine Learning 的程式，卻又沒有時間來修相關的課程或是閱讀相關的書籍？這份 Tutorial 將以淺顯易懂的方式，來介紹 Machine Learning(機器學習) 的重要概念，希望這份文件，對於想要了解 Machine Learning，但是又沒有太多時間的讀者能有所幫助。

我將從 Machine Learning 的有效性出發，來闡明為何可以透過 Machine Learning。接著，我將介紹各個較常見的 Machine Learning 演算法背後的核心概念，最後以 Machine Learning 系統設計流程及模型的驗證做收尾。

目錄

1 Introduction	3
2 When Can Machine Learn?	4
2.1 A takes H and D to get g	4
2.2 Type of Learning	4
2.3 Feasibility of Learning – No Free Lunch Theorem	5
3 Linear Regression	6
3.1 Gradient Descent Algorithm	7
3.2 Feature Scaling	9
3.3 Polynomial Regression	10
4 Logistic Regression	11

4.1 Hypothesis Representation – Logistic Function	11
4.2 Cost Function	12
4.3 Gradient Descent	12
4.4 Multi-Class	13
5 Support Vector Machine (SVM)	13
5.1 改寫 Logistic Regression 的 Cost Function	13
5.2 Nonlinear SVM	15
5.3 Using SVM	17
6 Neural Network	18
6.1 Model Representation	18
6.2 Intuition	19
6.3 Training the Weight of Neural Network	20
7 Learning Work Flow and Validation	22
7.1 Split Data and Design of the Machine Learning System	22
7.2 Feature Reduction - PCA	23
7.3 Diagnosing the Model	25
8 Conclusion	25
參考資料	25

1 Introduction

Machine Learning (機器學習)，顧名思義，就是讓機器（主要是電腦）能夠從 Data 中學習的演算法。

舉例而言，我們要如何辨識什麼東西是一顆樹？一個只有一兩歲的幼兒，可能不知道什麼東西是一棵樹。但是，如果我們經常帶他在公園裡散步，只要看到樹，就告訴他這是一顆樹…久而久之，他就知道什麼東西是一棵樹了。這其實就是人類的「學習」，學習什麼東西是一棵樹的過程。

那麼…我們該如何讓電腦能夠辨認一棵樹？同樣的，我們也可以讓電腦來「學習」。對於輸入的影像，我們先擷取這影像的特徵 (features)。接著，我們再對電腦做訓練 (training)，告訴電腦那張影像是樹，那張影像是不是樹。當訓練的資料量足夠之後，有新的影像進來時，電腦就可以判斷這張是不是樹的影像了。

這裡先介紹何謂 Features、Training、Label。

- **Features**：資料的特徵 (Features)，就是 Machine Learning 演算法的 input。就像人眼在辨認什麼東西是樹、什麼是花、什麼是草、什麼是路燈，所依據的是東西的「特徵」，電腦也是先對影像擷取出特徵之後，再來做辨識。

至於如何取出 data 的 Features，該選取哪些 Features，則完全依賴於 data 的種類和應用，以辨認「樹」而言，可以選取物體的顏色、高度、長寬比，有沒有類似樹枝或樹根的東西存在，來當成是辨識樹的「features」。若 data 是心電圖訊號，那麼 features 就可以是心跳的間隔、心電圖信號的高頻成份、低頻成份…等等。由於這份 Tutorial 只在介紹 Machine Learning 的基本概念，因此不討論每個應用領域的 Features 取法；而實際上應用 Machine Learning 的領域非常之廣，也無法一一討論，還請讀者依自己的應用領域尋找相關資料及演算法。

- **Training**：每個 Machine Learning Algorithm 都需要一些已知、現有的 data 來進行 training (訓練)。Training 會改變 Machine Learning 演算法中一些參數，使 Machine Learning 演算法能從現有的 data 特性去推測未來、未知的 data 結果。例如，我們可以持續的告訴電腦，擁有某些 features 的 data 是一棵樹或不是一棵樹。當訓練的 data 夠多，且訓練的演算法夠好，有新的 data 進來時，電腦就可以自動判斷這個 data 是否為一棵樹。

- **Label**：就是 training data 所對應的 output。就好像我們要讓幼兒了解什麼一棵樹時，就要告訴幼兒公園裡哪些東西是樹，哪些不是樹，否則幼兒將難以了解。同樣的，在對電腦做訓練時，必需要告訴電腦這些 data 所對應的 output (即 label)，否則電腦將無從學習。在訓練電腦自動地判斷樹的圖片時，我們可以將 training data 當中，不是樹的圖片的 label 設為 0，樹的圖片的 label 設為 1。

第二章將從何時可以使用 Machine Learning，Machine Learning 的有效性介紹起，第三至第六章則是敘述各個較常被使用的 Machine Learning 的 Algorithms。

2 When Can Machine Learn?

這一小節將介紹 Machine Learning 的基本概念和流程，以及 Machine Learning 在甚麼條件下才是有效的。

2.1 A Takes H and D to Get g

影像處理、音訊信號處理、生醫信號處理、甚至社會科學所處理的許多問題，是難以用簡單的公式或規則表示的。例如，光是「什麼是樹」這個問題，就難以用簡單的數學公式來表示，這時候，就可以嘗試使用 Machine Learning。

Machine Learning 的運作原理大致如圖 1，較工程化的示意圖則如圖 2 所示。透過 data 的 training，即可獲得辨識的規則（即圖 1 的 skill 或是圖二的 target function）。

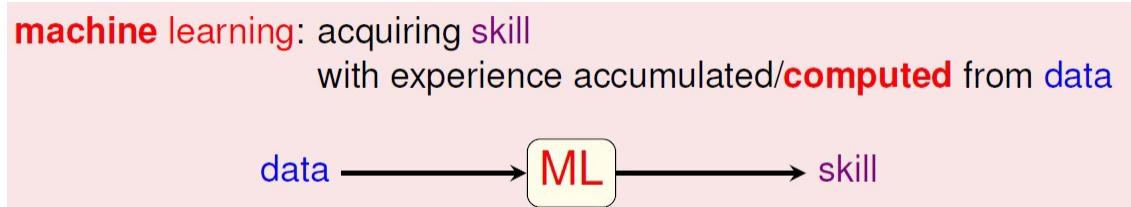


圖 1: Machine Learning (ML) 的示意圖 [2]

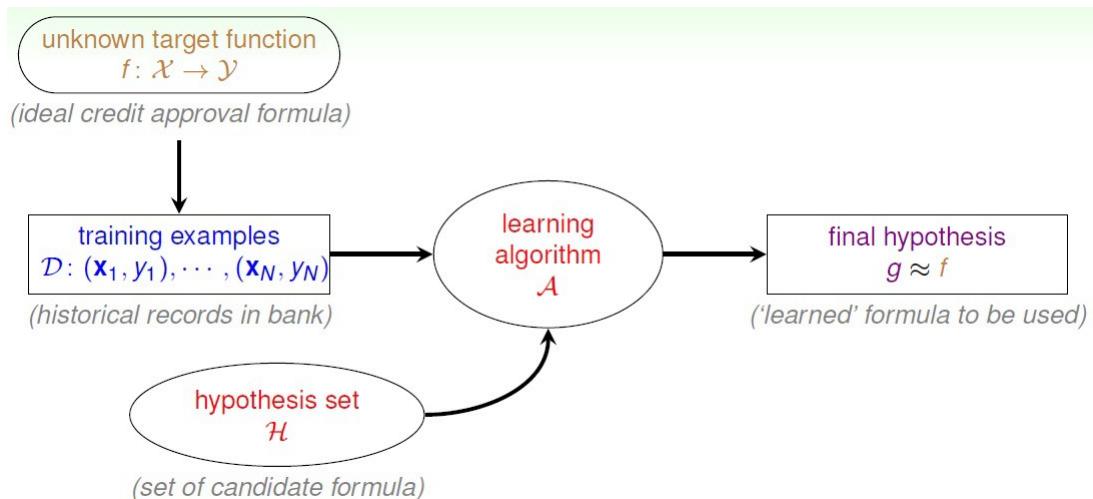


圖 2: Machine Learning (ML) 工程化的示意圖 [2]，其中 H 是 hypothesis set (所有可能的規則的集合)， D 是 training data 的集合 (其中 X_n 是第 n 個 Data， Y_n 是第 n 個 Data 所對應的 label)， A 是 machine learning algorithm， g 是根據 A 求得的規則。若 machine learning algorithm 的效果良好， g 將與真正的規則 (target function) f 相近。

2.2 Type of Learning

在 Machine Learning 的領域裡，一般有以下幾種 Learning 的種類。這份 Tutorial

介紹的工具都是屬於 Supervised Learning，也是發展較多樣且成熟的一種。

- **Supervised Learning Classification:** Training Data D 會告訴 machine learning algorithm A ，每一個 input data 對應到的 output 值（即 label）。例如，我們告訴電腦哪些圖片是棵樹，哪些不是
- **Unsupervised Learning:** Training Data D 不會告訴 A 甚麼樣 input 對應到什麼樣的 output（也就是沒有 label）。例如：我們並未告訴電腦哪些 input data 是棵樹，哪些不是。Unsupervised Learning 經常用在 Clustering Problem，分群問題。

除此之外，還有另外幾種 Learning Types，例如：

- **Semi-supervised Classification**
- **Regression:** 和 classification 不同的是，classification 的輸出是分類（例如將圖片分成是「樹」和「不是樹」兩大類），而 regression 的輸出則是一個值（例如藉由人臉的圖片判斷這個人有多少歲）

有興趣的讀者請參考 [2][8]，這份 Tutorial 將專注於一些 Supervised Learning Classification 和 Regression 演算法的介紹。

2.3 Feasibility of Learning – No Free Lunch Theorem

這一節探討 Machine Learning 的有效性。透過一些假設，比如說「若 training data 的表現是這樣，那麼未來輸入 data 表現應該也差不多」；有了這些前提，Machine Learning 的結果才有意義，否則若 training data 和未來輸入 data 的表現之間毫無關聯，那麼即使是最強的 Machine Learning Algorithm，也只能學出一個沒有使用價值的 Model。No Free Lunch Theorem 定理，就是在說明這件事。

如圖 3，前五個標成灰色的 data 為 training data， $f_1 \sim f_8$ 是這些 data 的特徵， y 則是這些 data 的所對應的 output（即 label）。根據這五個 training data，我們所得出的規則可能是：「特徵中 O 比較多的，output 應該是 O；特徵中 X 比較多的，output 應該是 X」。但這樣的規則，將難以用來判斷圖 3 的後面三個 input 所對應的 output 是什麼，因為這三筆資料有一半的特徵是 O，一半的特徵是 X。

所以，Training Data 和未來輸入的 data（即 Test Data）的分佈必需相似，是得出精確的 machine learning 的結果的必要條件。

No Free Lunch								
x	y	g	f ₁	f ₂	f ₃	f ₄	f ₅	f ₆
000	o	o	o	o	o	o	o	o
001	x	x	x	x	x	x	x	x
010	x	x	x	x	x	x	x	x
011	o	o	o	o	o	o	o	o
100	x	x	x	x	x	x	x	x
101		?	o	o	o	o	x	x
110	?	?	o	o	x	x	o	x
111	?	?	o	x	o	x	o	x

- $g \approx f$ inside \mathcal{D} : sure!
- $g \approx f$ outside \mathcal{D} : **No!** (but that's really what we want!)

learning from \mathcal{D} (to infer something outside \mathcal{D})
is doomed if any ‘unknown’ f can happen. :-(

圖 3: Example of No Free Lunch Theorem [2]

不只 training data 不能亂取，在實作上，features 也不能隨便選取，必須和 output 有邏輯上或分佈上的關係。

但有時我們無法依自己的經驗判斷 Features 是否和 output 有正向關係，此時能做的，就是將所有能想到的 Features 輸進 Machine Learning Algorithm，再透過 Feature Selection 的相關演算法挑出其中有用的 Features。關於 Features Selection 的演算法，有興趣的讀者可以參考 [3][4][6]

3 Linear Regression

線性迴歸 (linear regression)，即取一條和資料點誤差最小的直線，依照這條直線預測新的資料點應該落在何處，如圖 4。

誤差的計算一般是選擇 Least Square，也就是最小平方誤差。假設圖 4 的 x 軸是房子的大小 (input)， y 軸是房子的價格 (output)。我們現在已有一堆房子的大小和價格資料，即圖上的黑點。我們再依據這些黑點，利用 Linear Regression 畫出最能夠代表這些點的直線，也就是和這些點誤差最小的線，如圖 4 中的藍線。

接下來，若我們有一個新的房子資料，但只有大小，沒有價格。那麼我們可以依據這條直線，來判斷這樣一個大小的房子，它的價格應該是多少。

另外，值得一提的是，線性回歸是一種 Regression，和我們接下來要討論的 Supervised Learning Classification 不同。這兩者的差別在於 Classification 輸出的結果是 “label”，比如「1, 2, 3, ...」等 discrete value，代表 data 所屬分類；而 Regression 的輸出的結果則是一個 continuous value，如房屋的價格。

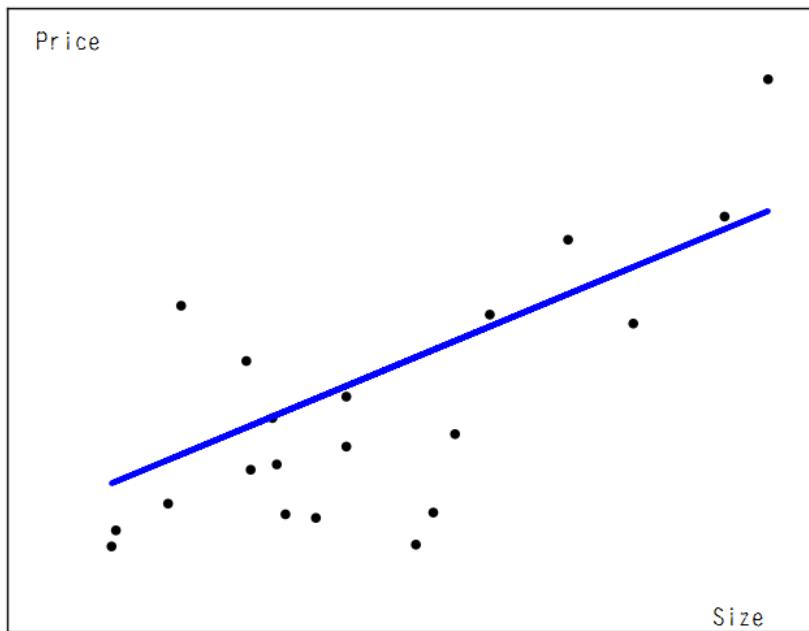


圖 4: Linear Regression 示意圖，將房子大小和房價之間的關係用直線近似

3.1 Gradient Descent Algorithm

線性迴歸 (linear regression) 在台灣的高中和大學的線性代數一般都已學過，當時是使用二次式配方法及偏微分算出迴歸線

$$y \equiv h_\theta(x) = \theta_0 + \theta_1 x$$

其中 x 是 input， y 是 output， θ_0, θ_1 是要算的係數。

當我們所處理的資料是一維的，線性迴歸的問題就如同上式那麼簡單。但是在 Machine Learning 或 Data Mining 的領域，動輒要處理好幾個 Features 時，線性迴歸的問題將變得較複雜：

$$y \equiv h_\theta(x) = \theta_0 + \sum_{j=1}^N \theta_j x_j$$

其中 x_j 是 input 的第 j 個特徵， $\theta_0, \theta_1, \dots, \theta_N$ 是 linear combination 的係數。此時可使用一些最優化演算法，比如 Gradient Descent，來將 θ_j 算出來。

在解釋什麼是 Gradient Descent 演算法之前，必須先定義 Linear Regression 的 Cost Function。在這裡 Cost Function 可以選用最小平方誤差 (如下列的式子)， m 為現有資料點數。

$$Cost(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2 \quad (1)$$

其中 $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, \dots, x_N^{(i)})$ 是第 i 個 input data 的 N 個 features， $y^{(i)}$ 是 $\mathbf{x}^{(i)}$ 對應的 output。我們要透過 Gradient Descent 演算法最小化這個 $Cost(\theta)$ 。而此時 h_θ 也

不限於二維的狀況，而是一條 $N+1$ 維線性回歸線（如果特徵數量為 N ），即

$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \dots + \theta_N x_N^{(i)}. \quad (2)$$

$\theta = (\theta_0, \theta_1, \dots, \theta_N)$ 為迴歸線的係數向量。

用 Gradient Descent 來更新 θ 值的方法的架構如下：

1. 隨機選擇起始的 $(\theta_0, \theta_1, \dots, \theta_N)$ 值
2. 以下列方式同時更新每個 θ_j 值，其中 α 稱作 learning rate，介於 0 和 1 之間

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} Cost(\theta). \quad (3)$$

3. 重複上述步驟，直到 θ_j 值收斂。

依據 Eq.(1) 和 Eq.(2)，求出 $Cost(\theta)$ 對每個 θ_j 偏微分後代入 Eq.(3) 可得 θ_0 及其它 θ_j 的更新公式

$$\begin{aligned}\theta_0 &= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)}) \\ \theta_j &= \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}.\end{aligned}$$

為什麼這樣能夠求得最佳解呢？以圖 5 為例， y 軸是 Cost Function $Cost(\theta)$ 值， x 軸則是 θ 向量當中的某個 θ_j 。圖 5 中，藍線的斜率為負，紅線的斜率為正，由 Eq.(3) 觀察，可以發現若現在的 θ_j 在藍線部分，則由於 $\frac{\partial}{\partial \theta_j} Cost(\theta)$ 為負， θ_j 的值會往右調整；反之，若 θ_j 在紅線部分，則 θ_j 將往左調整。由圖 5 可以發現重複 Gradient Descent 演算法，可以找到一個 optimal solution。

另外，可以注意的是，若 $\frac{\partial}{\partial \theta_j} Cost(\theta)$ 的絕對值大，代表 θ_j 距離 $Cost(\theta)$ 最小值發生的地方越遠， θ_j 調整的幅度也就會越大，如 Eq.(3)。

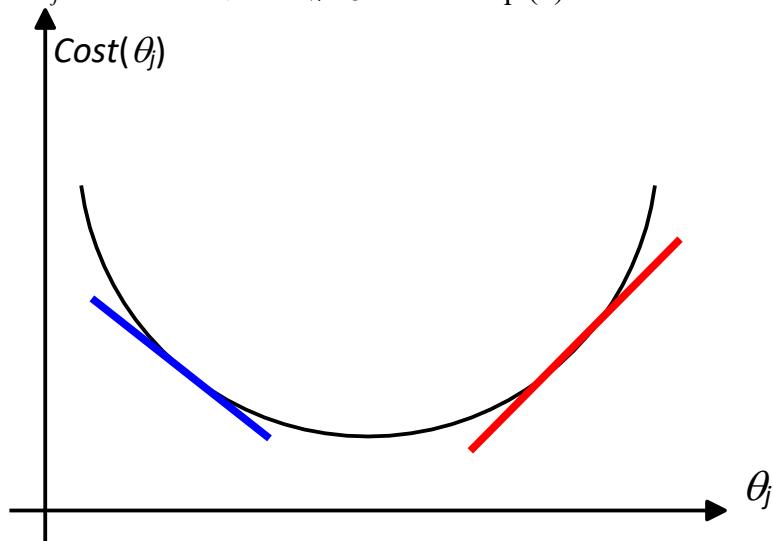


圖 5: Gradient Descent 示意圖

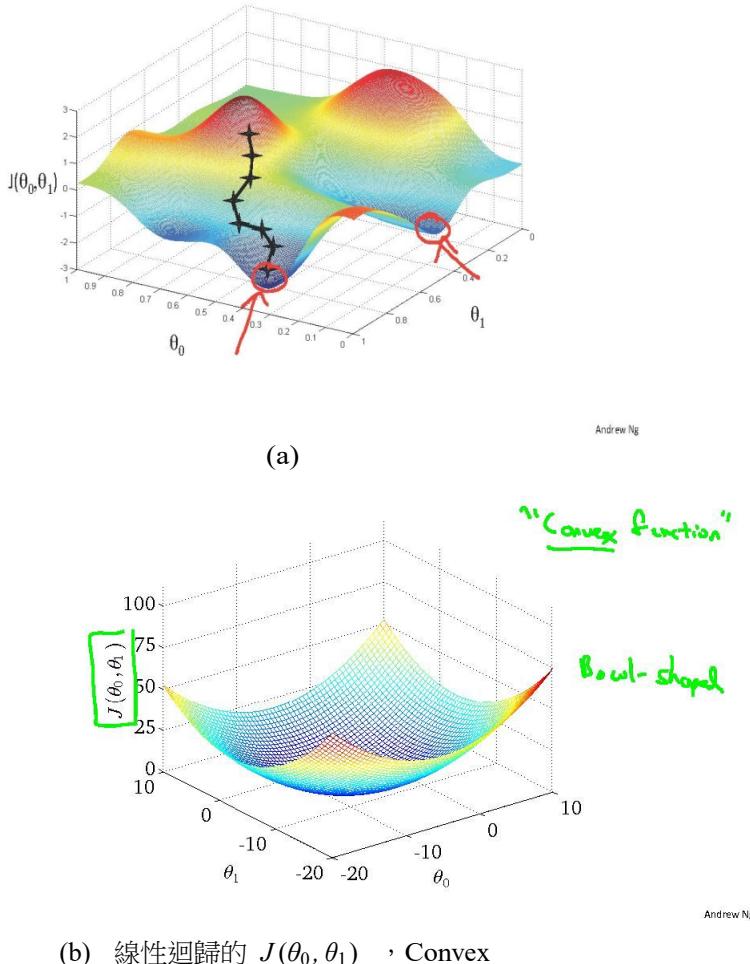


圖 6: Gradient Descent Algorithm 運作圖 [1]

但值得注意的是， θ_j 的更新值雖然會使 $Cost(\theta)$ 往低處移動。但只會停在 Local Minimum 的點，不一定是 Global Minimum (若 θ_m 是 local minimum，則 $Cost(\theta_m) < Cost(\theta_m + \Delta\theta_m)$ ，但是如果 $Cost(\theta_m) \neq \min(Cost(\theta))$ 那麼 θ_m 就不能算是 global minimum)。至於停在哪個 Local Minimum 則取決於起始值。如圖 6(a)，在一般的情況下，Gradient Descent 只能找到 Local Minimum，但如果 $Cost(\theta)$ 的形狀是如圖 6(b)的 Convex (凸形) 則能保證可以找到 Global minimum。

3.2 Feature Scaling

在 Machine Learning 問題中，Features 常常有很多很多個。如果 Features 間的 scale 差異過大，會導致某些 θ_j 的更新「步伐」，相對於其他 Features 而言會過小或過大。以[14]內的其中一個例子為例，若今天我們需要建立一個 Classifier 來分類成年人

與未成年人；利用的 Features 為身高(cm)與體重(kg)。以一個身高 175cm 體重 75kg 為例，兩者在數值上就差了 100；根據 Section 3.1 的 θ_j 的更新公式：

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

不難發現 Input 的第 j 個 Feature $x_j^{(i)}$ 的大小會影響更新的步伐，所以，像上述的體重與身高之間的數值差就會造成兩者更新速度的不同。

因此，我們必須做 Feature Scaling，使每個 Features 的值落在差不多的範圍內。以下是二個常見的 Scaling 方式。 F_{old} 代表原本的資料， F 代表經過 scaling 之後的資料：

$$F = \frac{F_{old} - \text{mean}(F_{old})}{\max(F_{old}) - \min(F_{old})}$$

如此一來， $\text{mean}(F) = 0$ 且 $\max(F) - \min(F) = 1$ 將滿足。

另一種常見的 scaling 方式為 (較建議用這方式)

$$F = \frac{F_{old} - \text{mean}(F_{old})}{s_{old}} \quad \text{其中 } s_{old} = \sqrt{\text{mean}\{(F_{old} - \text{mean}(F_{old}))^2\}} \quad (4)$$

如此一來， $\text{mean}(F) = 0$ ，且 F 的 variance 和 standard deviation 皆為 1 (註：在 Matlab 當中，正確的 standard deviation 的指令寫法為 $s_{old} = \text{std}(F_{old}, 1)$ ，要加上 “,1”)。

3.3 Polynomial Regression

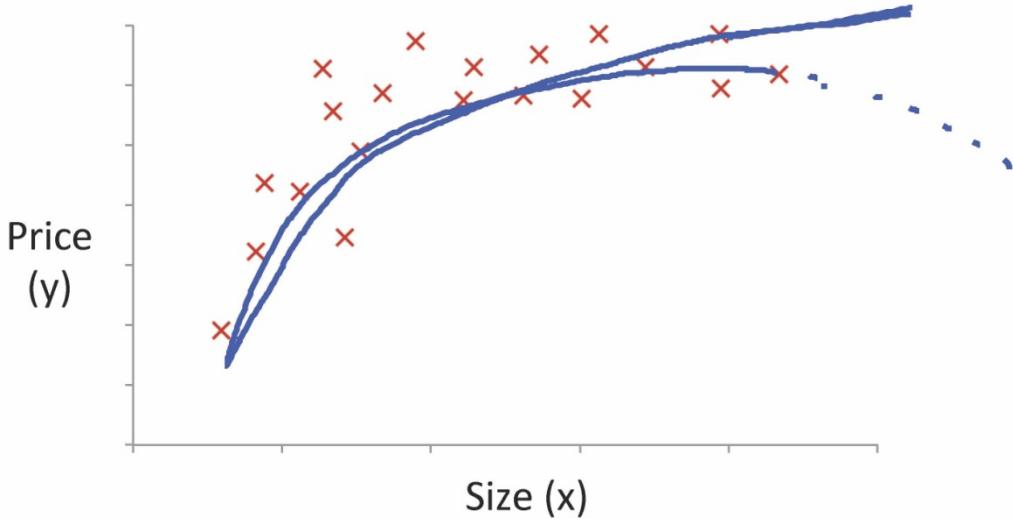


圖 7: Polynomial Regression 示意圖 [1]

線性迴歸未必對所有的問題都適用。例如，若想畫出如圖.7，則必需利用 Polynomial Regression 得出非直線的回歸線。

Polynomial Regression 即增加新的 Features。例如，若原本只有 x_1, x_2 二個 features，我們可以設定新的 features 為 $x_3 = x_1^2, x_4 = x_1^3, x_5 = x_1 x_2, x_6 = \sqrt{x_1}, x_7 = \sqrt[3]{x_2}, \dots$ ，之後再對這些 features 做 linear regression。(這樣的作法相當於把原本的 Features Space 映射到不同維度做線性回歸，因此在原本的 Features Space 看起來，回歸線就不再是直

線了。)

需特別注意的是，在做 polynomial regression 時更是需要做 Features Scaling (參考 Section 3-2)，讓 Features 落在差不多的範圍

至於該增加哪些 Polynomial 項，並沒有定論，只能透過 try and error 的方式。但是一般來說，次方越高，代表會使 Learning Model 的複雜度上升，會需要更多的 Training Data 來訓練出一個較好的 Model。

綜合而言，和線性迴歸相比，Polynomial Regression 的優缺點為：

- 優點：可以學到比 Linear Regression 更複雜的迴歸線。
- 缺點：需要更多的 Training Data，有時甚至會超出可實作範圍。

若讀者想要對 regression 做更深入的了解，或是對其他種類的 Regression 有興趣，可以參考 [1]，並建議閱讀 [2][5] 以及教科書 [3]，regression 的程式碼可以在 [8] 找到。

4 Logistic Regression

Logistic Regression 的名稱是來自它的 Cost Function，實際上這個方法並不是 Regression。它主要是解決 Supervised 的 Classification Problems，也就是分類問題。和 Regression 不同，Classification 的輸出通常是 0 或 1 等等，代表所屬分類的 Discrete Value，通常稱作 “Label”。

0 通常表示這個 data 不屬於此類，1 則表示這個 data 屬於此類。例如，要讓電腦辨認樹的圖片時，我們可將沒有樹的圖片的 label 設為 0，有樹的圖片的 label 設為 1。

4.1 Hypothesis Representation – Logistic Function

Logistic Regression 是使用 Logistic Function 來做為分類依據的 supervised classification 的方法。Logistic Function 的定義如下：

$$h_{\theta}(\mathbf{x}) = g(\boldsymbol{\theta}^T \mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}}} \quad (5)$$

其中 $\mathbf{x} = (1, x_1, x_2, \dots, x_N)^T$ 是 input feature set， $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \dots, \theta_N)^T$ 則是係數， $h_{\theta}(\mathbf{x})$ 則為 output。

我們可以將 logistic function 和 step function 的分類方式做比較。若用 step function 做分類的方式為

$$\text{Output} = u(\boldsymbol{\theta}^T \mathbf{x})$$

其中 $u(t)$ 是 step function，即

$$\text{Output} = 1 \quad \text{if } \boldsymbol{\theta}^T \mathbf{x} > 0 \quad \text{Output} = 0 \quad \text{if } \boldsymbol{\theta}^T \mathbf{x} < 0$$

也就是 output 非 1 即 0(另外，值得注意的是， $\boldsymbol{\theta}^T \mathbf{x} = 0$ 相當於一個在 N 維空間上的 $N-1$ 維的分界面或稱超平面(Hyperplane)，當 $N=2$ 時， $\boldsymbol{\theta}^T \mathbf{x} = 0$ 是一條直線，在 Pattern

Classification 中，此分界面稱為 Decision Boundary，也就是兩個不同 decision 的交界)。雖然說， $u(t)$ 的 output 是 0 或 1，看似符合 classification 的需求，但是在做最佳化時會發生問題。在做最佳化時，經常需要做微分 (例如 Lagrange multiplier 或是 Section 4-3 的 gradient descent 都需要微分)，偏偏 step function 並不是連續的，在做微分時值可能會變成無窮大，會有計算上的困難。

相對而言，logistic function 的 output $h_\theta(\mathbf{x})$ 並不是非 1 即 0，雖然不完全符合 classification 的要求，但是從 Eq.(5) 可以看出，當 $\theta^T \mathbf{x}$ 遠大於 0 時， $h_\theta(\mathbf{x})$ 的值將非常接近 1；當 $\theta^T \mathbf{x}$ 遠小於 0 時， $h_\theta(\mathbf{x})$ 的值將非常接近 0。所以，logistic function 的 output 其實和 step function 相當接近，只有在 $\theta^T \mathbf{x}$ 近於 0 的時候會有較明顯不同。而 Eq.(5) 的 logistic function 是一個連續的函式，在最佳化、做微分時將不會產生不連續的問題。

4.2 Cost Function

為了決定最佳的 θ ，我們必須先定義 Logistic Regression 的 Cost Function。首先，我們定義每個 training data 各自的 Cost：

$$\Phi(h_\theta(\mathbf{x}), y) = -y \log(h_\theta(\mathbf{x})) - (1-y) \log(1-h_\theta(\mathbf{x})) \quad (6)$$

其中 $y = 0$ or 1 ，是現有資料點 \mathbf{x} 所對應的 label， $h_\theta(\mathbf{x})$ 則是 Logistic Regression 的 output。觀察這樣的定義，可以發現當 $y = 1$ 但是 Logistic Regression 却輸出 0 時，Cost 將趨近於無限大，反之亦然。因此，這樣的 Cost 定義能夠表示 Eq.(5) 這個 Model 預測的準確性。

將上式擴展到 m 個已知資料點，可以得出 Cost Function 為

$$Cost(\theta) = \frac{1}{m} \sum_{i=1}^m \Phi(h_\theta(\mathbf{x}^{(i)}), y^{(i)}) \quad (7)$$

另外，在實作 Logistic Regression 時， $Cost(\theta)$ 常常會加上另一項 $\frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$ ，這個項稱作 regularization，通常是用來避免所算出來的 θ 值過大，以防止 Machine Learning Algorithm 過度地 fit training data 而產生 Over-fitting 的狀況。

而為何透過這樣的限制就能夠避免 Overfitting 呢？我們可以用一個簡單的例子來做理解，詳細的證明可參考[1]。首先，Overfitting 通常代表利用過多的參數來貼近 training data，也就是 $\theta^T \mathbf{x} = 0$ 的階數過高。因此，我們可以限制 $\theta^T \mathbf{x} = 0$ 但是 $\theta_j = 0$ for $j > M$ (i.e. 限制 $\theta^T \mathbf{x} = 0$ 的階數不會大於 $M+1$)。但由於這個 Constraint 的最佳化問題是 NP-Hard Problem(i.e. 至今仍未找到一個多項式 $O(n^k)$ 複雜度的決定性運算法)，因此，我們可以改為 $\sum_{i=1}^m \theta_j^2 < C$ ，避免係數和過大，間接降低 $\theta^T \mathbf{x} = 0$ 的階數。

其中 λ 是自行決定的參數，越大表示越不會 Over-fitting，但太大則會造成 Under-fitting。另外，分母的 2 是為了方便未來在微分以後的係數調整，並不影響整體的最佳化運算。

所以，加上 Regularization 之後，Logistic Regression 的 Cost Function 變為

$$Cost(\theta) = \frac{1}{m} \sum_{i=1}^m \Phi(h_\theta(\mathbf{x}^{(i)}), y^{(i)}) + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2. \quad (8)$$

4.3 Gradient Descent

有了 Cost Function 之後，我們便可以藉由許多最佳化的演算法來訓練出適合我們資料的最佳係數 θ 。以 Gradient Descent 為例（這是最常用的最佳化演算法之一），我們先求出 Cost Function，再對每個 θ_j 的偏微分得出：

$$\begin{aligned} \frac{\partial}{\partial \theta_j} Cost(\theta) &= \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial \theta_j} \Phi(h_\theta(\mathbf{x}^{(i)}), y^{(i)}) + \frac{\lambda}{m} \theta_j, \\ \frac{\partial}{\partial \theta_j} \Phi(h_\theta(\mathbf{x}^{(i)}), y^{(i)}) &= \left[-\frac{y^{(i)}}{h_\theta(\mathbf{x}^{(i)})} + \frac{1-y^{(i)}}{1-h_\theta(\mathbf{x}^{(i)})} \right] \frac{\partial}{\partial \theta_j} h_\theta(\mathbf{x}^{(i)}), \quad (\text{由 Eq. (6)}) \\ \frac{\partial}{\partial \theta_j} \Phi(h_\theta(\mathbf{x}^{(i)}), y^{(i)}) &= \frac{h_\theta(\mathbf{x}^{(i)}) - y^{(i)}}{h_\theta(\mathbf{x}^{(i)}) (1 - h_\theta(\mathbf{x}^{(i)}))} \frac{e^{-\theta^T \mathbf{x}}}{(1 + e^{-\theta^T \mathbf{x}})^2} x_j^{(i)} \end{aligned}$$

其中 $x_j^{(i)}$ 是第 i 個 input 的第 j 個特徵。又

$$h_\theta(\mathbf{x}^{(i)}) (1 - h_\theta(\mathbf{x}^{(i)})) = \frac{e^{-\theta^T \mathbf{x}}}{(1 + e^{-\theta^T \mathbf{x}})^2}$$

所以

$$\begin{aligned} \frac{\partial}{\partial \theta_j} \Phi(h_\theta(\mathbf{x}^{(i)}), y^{(i)}) &= (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)} \\ \frac{\partial}{\partial \theta_j} Cost(\theta) &= \frac{1}{m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j. \end{aligned} \quad (9)$$

將上式代入 Gradient Descent 的更新演算法之中

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} Cost(\theta), \quad (10)$$

我們就可以依循 Gradient Descent 的 Framework，得到適合我們現有資料點的最佳 θ 值。將其應用到 Section.4.1 的 $h_\theta(\mathbf{x})$ ，就可以得到訓練完成的 Logistic Regression 模型。

4.4 Multi-Class

利用 One versus All 的概念，一條 $h_\theta(\mathbf{x})$ 可分出兩類，二條 $h_\theta(\mathbf{x})$ 可分出三類， N 條 $h_\theta(\mathbf{x})$ 可分出 $N + 1$ 類。比如說現在有三個資料點 x_1, x_2, x_3 ，分別屬於 a, b, c 三類，則利用二個 Classification c_1 和 c_2 ，其中 c_1 可以分出 a 類和非 a 類，而 c_2 可以再將非 a 類分成 b 類和 c 類，如此即可做到 Multi-Class 的分類。

總體而言，logistic regression 的優點在於複雜度較低，因此可以用較少的 Training Data 就能得出有效的 Model。但缺點便是不能夠表示出太複雜的分類線。讀者若要對

logistic regression 做更深入的了解，可以參考 [1] 和 [5]。相關的程式碼也可以在 [8] 找得到。

5 Support Vector Machine (SVM)

Support vector machine (SVM，支持向量機) 是目前最被廣泛應用的 Learning Algorithm。透過一群已經分類好的資料(Labeled Data)，SVM 可以利用訓練(Training)的方式建立出一個模型；訓練結束以後，SVM 便能透過先前的資料對於未來即將輸入的尚未分類的資料進行分類。也因此，SVM 是一種 Supervised Learning Classifier(監督式學習)，其所建立的 model 只能針對具有與該分類有一定相關程度的 Data 進行分類，也正是本文在 Section 2.3 中所提及的 No Free Lunch Theorem。

由於 SVM 其背後的數學理論十分嚴謹並且複雜，本文在此僅介紹其幾何上的數學理論；這是由於，從幾何的方式出發可以讓第一次接觸 SVM 的讀者更清楚的瞭解其核心理念，而且，比起純代數的證明更容易讓人理解。此外，對於 SVM 的運作流程，這邊將從 Logistic Regression 出發。從 Logistic Regression 出發的好處在於易於解釋 SVM 運作流程。但是若想全盤了解 SVM 的最佳化方法，仍必須參考 Lagrange Multiplier [13]、KKT Conditions (一種數學最佳化的工具)，以及純粹由最佳化觀點出發的 SVM [9][10] 推導過程。

5.1 Introduction to Support Vector Machine(Geometric Approach)[16]

首先，SVM 實際上是一種 Linear Classifier，也就是 SVM 希望在特徵空間(Feature space)中尋找一個最適合的超平面(Note:在高維度的向量空間中超平面即為一組線性方程式；回到 2 級的情形時，超平面即為一條直線)，清楚的將特徵空間中不同類別的資料分開。

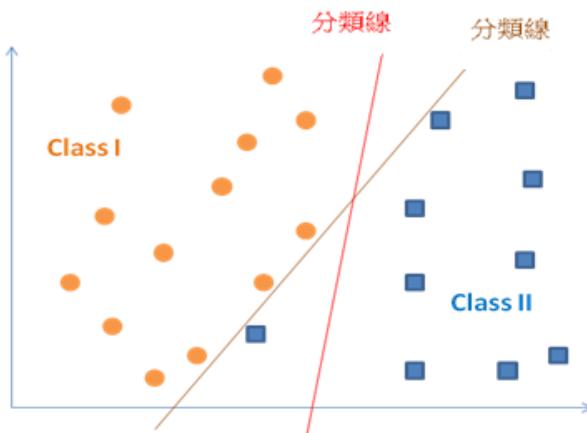


圖 8:分類線的可能情型

圖 8 為一個 2 維特徵空間的例子，可以發現到紅色及棕色分類線都能夠成功將空間中的資料分成 2 類；事實上，這空間中還存在更多條能夠區分出 Class 1 和 Class 2 的分類線；但是，哪一條分類線會比較適合呢？選出最適合的分類線便是 SVM 主要在討論的問題。

為了方便說明，以下我們以 x_i 代表第 i 個 Sample 的 Feature 且 $x_i \in R^d$ ，此外， $y^{(i)}$ 為第 i 個 sample 的標籤(Label)且 $y^{(i)} \in \{-1, 1\}$ (e.g. $y^{(i)}=-1$, 非樹； $y^{(i)}=1$, 樹)。另外，以下我們考慮一個簡單的 Case，稱為 Linear separable(可線性分割)，也就是我們的 data 可以用一個超平面使得所有同一類的樣本都落在同一邊。同樣的為了方便說明，以下我們利用 2 維說明 SVM 的幾何意義。

在幾何上， $\theta^T x = 0$ 代表著一個通過原點並且與向量 θ 正交的超平面。接著，我們將此超平面由原點沿著向量 a 的方向平移，則此超平面變為 $\theta^T(x - a) = 0$ ，其中，若我們令 $b = \theta^T a$ ，則可改寫為 $\theta^T x = b$ ，其中位移量 b 為向量 a 在 θ 上的投影。現在，我們假設 $\theta^T x = b$ 是一個可以將同一類資料若在同一邊的超平面，我們暫且稱為分隔超平面。接著，我們定義 2 個另外的超平面，其中這 2 個超平面平行於 $\theta^T x = b$ 且分別均為最靠近 Training data 其中一種類別的超平面。(i.e. 一個靠近 $y^{(i)}=-1$ ；另一個靠近 $y^{(i)}=1$)，而數學上我們稱這 2 個平面為支持超平面(Support hyperplane)，如圖 9 所示。

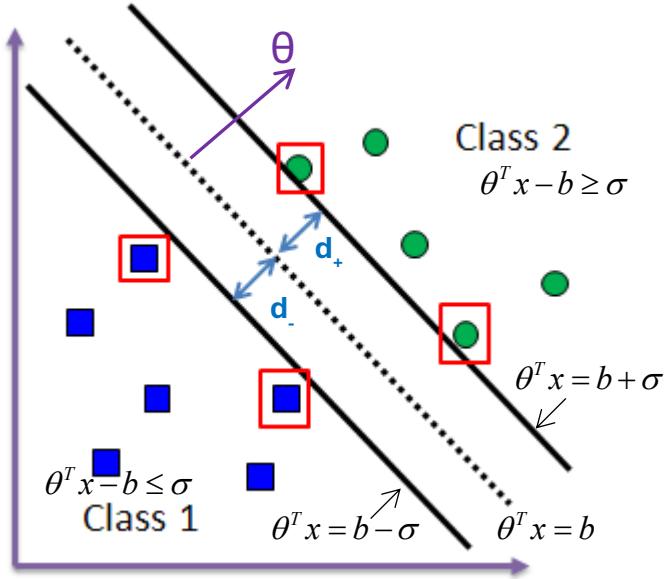


圖 9:支持超平面與分隔超平面示意圖[15]

其中， d_+ 與 d_- 為兩個超平面到分隔超平面的距離。而一個良好的分類器，我們會希望兩個類別的之間分類線與最靠近分類線的 data 的距離越大越好，這樣才比較不容易有 Mismatching 的情形。因此，若我們定義一個分隔超平面的 Margin $\nu = d_+ + d_-$ ，現在的問題就是如何找一個分隔超平面使得 ν 最大，且 $d_+ = d_-$ 。

根據上述的超平面平移的關係式，我們可以寫下兩個支持超平面的方程式：

$$\begin{aligned}\theta^T x &= b + \sigma \\ \theta^T x &= b - \sigma\end{aligned}$$

不失一般性，可以假設 σ 為 1，這是由於對等號兩邊乘上一個相同的常數，其方程式仍然是滿足。因此，可改寫為：

$$\begin{aligned}\theta^T x &= b + 1 \\ \theta^T x &= b - 1\end{aligned}$$

現在，我們可以計算 $d_+ = \frac{\theta^T(x' - x)}{\|\theta\|} = \frac{b + 1 - b}{\|\theta\|} = \frac{1}{\|\theta\|}$ ， d_- 亦然，因此， $\nu = \frac{2}{\|\theta\|}$ 。

此外，對於上面的支持超平面的定義，我們可以寫下對於分隔超平面的限制 (Constraint)：

$$\begin{aligned}\theta^T x_i - b &\geq 1 & \forall y^{(i)} = 1 \\ \theta^T x_i - b &\leq -1 & \forall y^{(i)} = -1\end{aligned}$$

這 2 個不等式透過圖 9 可以很容易的得到。假設 $y^{(i)}=1$ 代表 Class 2 的標籤，而若我們以二維直線方程式來看，要落在 Class 2 這個位置，其 x 必須滿足 $\theta^T x_i \geq b + \sigma$ 。現在 σ 為 1，所以可以得到 $\theta^T x_i - b \geq 1 \quad \forall y_i = 1$ ，而 $y^{(i)}=-1$ 的情形同理。

整合 2 個不等式以後，可以得到：

$$y^{(i)}(\theta^T x_i - b) - 1 \geq 0$$

接著，可以整理出 SVM 的原始問題：也就是最大化 ν 並且在 $y^{(i)}(\theta^T x_i - b) - 1 \geq 0$ 的限制下。也就是在訓練階段時，SVM 會選擇一個分隔超平面，並且此超平面是在支持超平面的某些限制下最大化 margin 的大小。而剛好落在支持超平面上的訓練資料點，就稱為支持向量(Support vector)，因為它們支持著支持超平面且決定了上述問題的解，故此 Classifier 我們稱為 Support Vector Machine(SVM)。

最後，我們公式化 SVM 的主要問題：

$$(1) \text{ 最大化 } \nu = \frac{2}{\|\theta\|} \text{，等同於最小化 } \frac{1}{2} \|\theta\|^2 \text{。}$$

(轉成平方的理由是希望最佳化問題變為 Convex)

$$(2) \text{ 需符合 } y^{(i)}(\theta^T x_i - b) - 1 \geq 0$$

統整以後的結果，即為 SVM 的 Cost function:

$$Cost(\theta) = \frac{\lambda}{2m} \|\theta\|^2 + \frac{1}{m} \sum_{i=1}^n [1 - y^{(i)}(\theta^T x_i - b)]$$

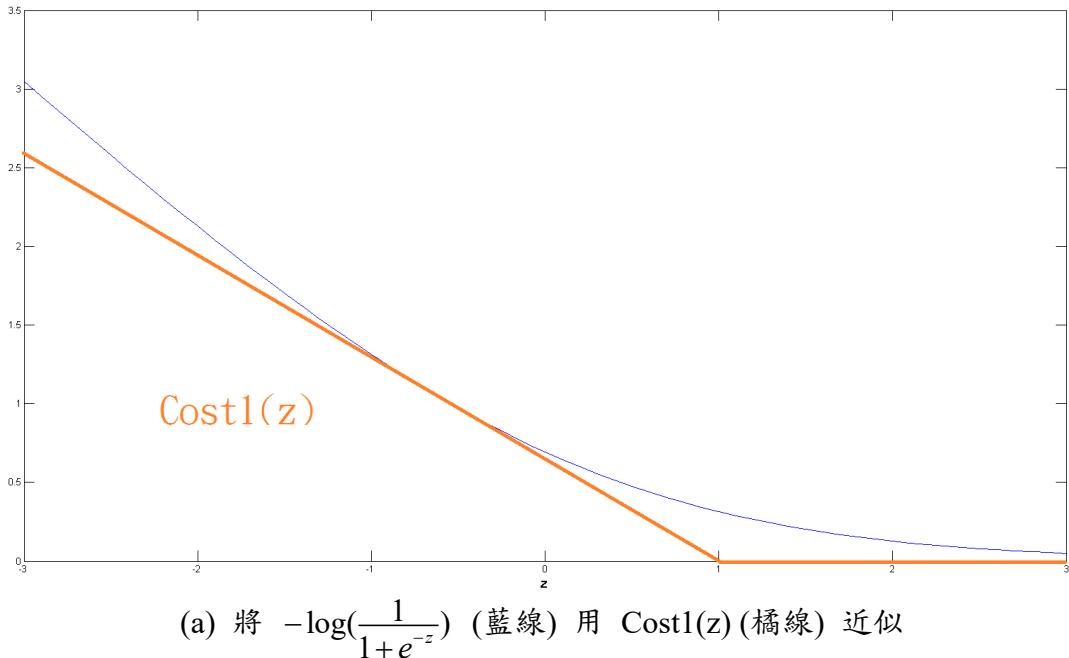
找出使得 $Cost(\theta)$ 為最小值的 θ 即可得到 SVM 的分隔平面，而當中的 λ 與 m 都是可供調整的參數。

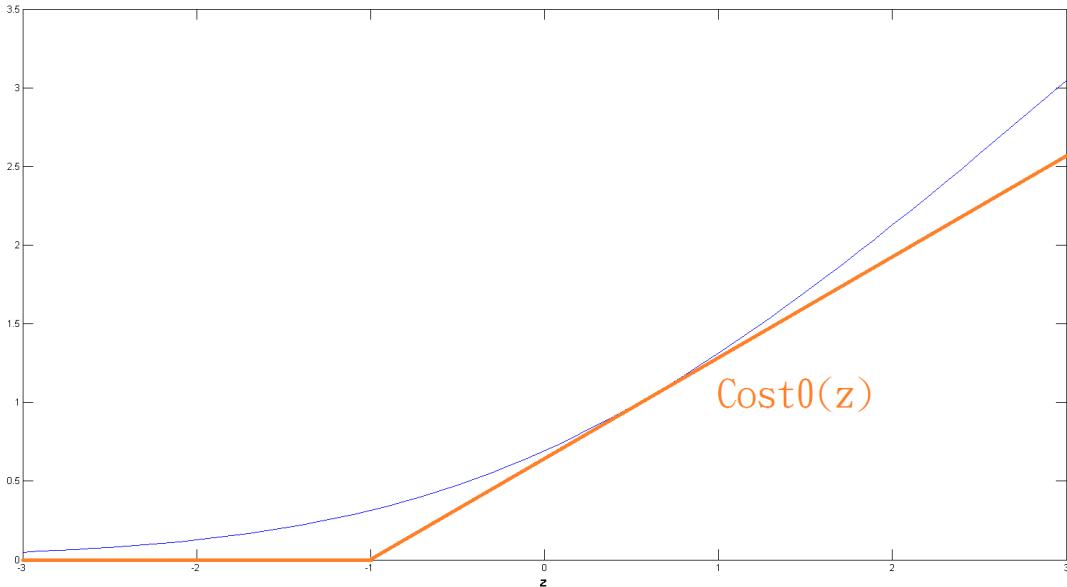
仔細觀察事實上可以發現，此 Cost function 的第二項事實上就是 Training error 的 Cost function。(因為如果 $y^{(i)}=1$ 時， $\theta^T x_i - b \leq -1$ 或 $y^{(i)}=-1$ 時， $\theta^T x_i - b \geq 1$ 都會讓這一項變為正值)因此，當我們把判斷準則換成 Logistic Regression 時，就可以得到以下的式子：

$$\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) + \frac{\lambda}{2m} \sum_{i=1}^m \theta_i^2$$

也就是將原本後面的 Cost function 換成了 Logistic function 的 Cost function。下一個 Section 我們將以這個式子，再簡化到另一個 Cost function 以利於解釋 SVM 的運作流程，至於更完整的內容則可參考[9][10]。

5.2 改寫自 Logistic Regression 的 Cost Function





(b) 將 $-\log(1 - \frac{1}{1 + e^{-z}})$ (藍線) 用 $\text{Cost}_0(z)$ (橘線) 近似

圖 9: 改寫 Logistic Regression 的 Cost Function[1]

如圖 9，將 Logistic Regression Cost Function 中的 $-\log(\frac{1}{1 + e^{-z}})$ 和 $-\log(1 - \frac{1}{1 + e^{-z}})$ 改寫，各以一條中間有曲折直線近似。可以注意的是，這兩條近似線分別在 1 之後以及 -1 之前的情況。

如此一來，原本 Logistic Regression 要求最小化的 Cost Function 為：

$$\min_{\theta} \left[\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\theta}(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(\mathbf{x}^{(i)})) + \frac{\lambda}{2m} \sum_{i=1}^m \theta_i^2 \right] \quad (11)$$

(可能少加了負號)

將變為

$$\min_{\theta} \left\{ C \left[\sum_{i=1}^m y^{(i)} \text{cost}_1(\boldsymbol{\theta}^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\boldsymbol{\theta}^T \mathbf{x}^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^m \theta_i^2 \right\}, \quad (12)$$

其中 $\text{cost}_1, \text{cost}_0$ 的定義如圖 9。而 C 是一個參數，在這裡可以視為 $C = 1/\lambda$ ，Eq. (12) 當中的 $1/m$ 僅僅是一個常數，不影響最佳化的解，因此可以直接消掉。

Eq. (12) 當中參數 C 的主要功能，是在於決定 training data 的時候分類錯誤的例子該有多大的 cost。當 C 越大，分類錯誤的 Cost 越重，SVM 會傾向於盡量 Fit Training Data，有時會產生 Over-fitting 的問題。如圖 10，當 C 很大時，SVM 會選擇一條 Margin 太小但完全 Fit Training Data 的分類線 (margin 指的是 training data 和分類線之間的距離)，如圖 10 的棕色線。一般而言，這樣的分類線有 Over-fitting 的嫌疑。而當 C 不太大時，SVM 會選擇忽略左邊的藍色方塊，選擇一條 Margin 較大的分類線，如圖 10 的紅色線。

在 machine learning 當中，over-fitting 指的是分類的結果太過於牽就 training data。相對的，under-fitting 則是太過於忽視 training data (可以看成是 Eq. (12) 的 C 值選得太小的情形)。無論是 over-fitting 和 under-fitting 皆會影響 machine learning 的準確度，

皆應該避免。

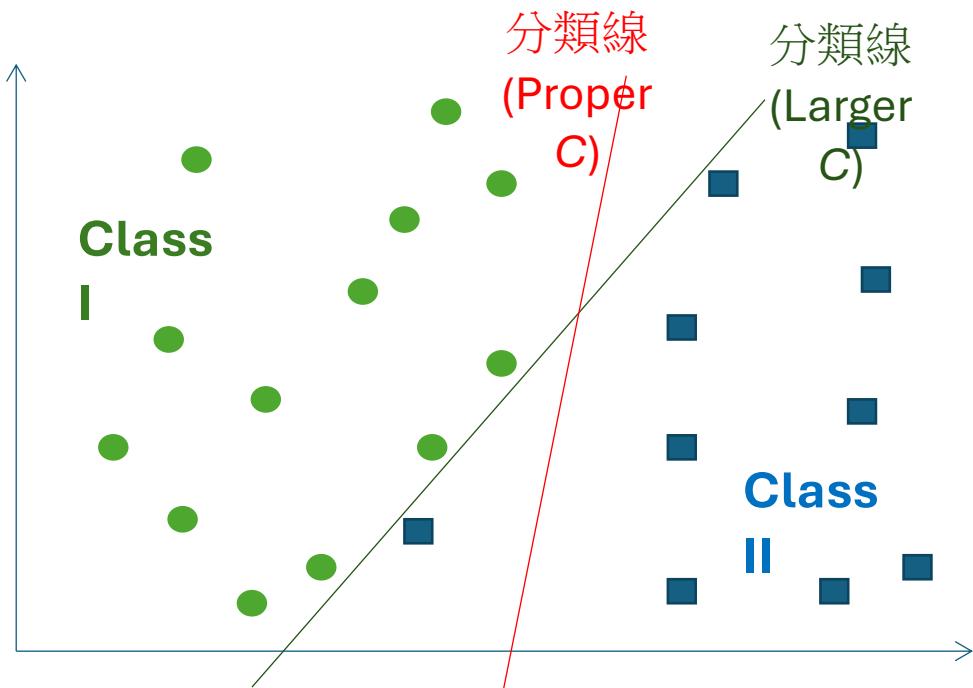


圖 10: SVM 參數 C 的功能

由於相關的數學較為複雜，這裡沒有敘述的是如何最佳化 Eq. (12)，有興趣的讀者請查閱 Lagrange Multiplier 及 KKT Conditions 的相關資料，課本 [5] 當中有對 Eq. (12) 的最佳化做說明。

5.3 Nonlinear SVM

以上敘述的是 Linear SVM，在二維空間中得到的分類線都是一條直線，這是 linear SVM 的限制。如果兩大類的資料需要藉由非線性的分割線才可以分開（如圖 11 的例子），可以考慮使用非線性的 SVM。

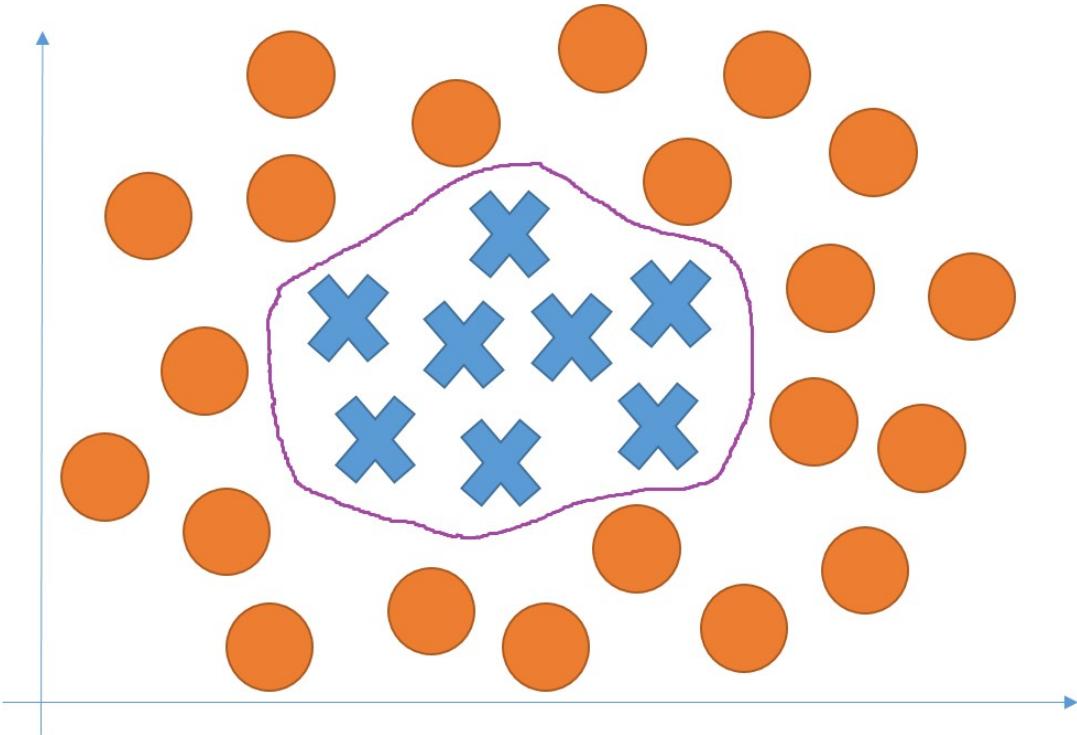


圖 11: Nonlinear SVM

如圖 11，我們需要的是一個非直線的分類線。SVM 的做法是，將 Training Data 當中的某些 Positive (Negative) cases 的 features 作為標竿 (Landmarks)，表示成 $\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3, \dots, \mathbf{L}_m$ (若 features 的數量為 N , \mathbf{L}_i 將是一個 N 維的向量， $\mathbf{L}_i = \{x_1^{(i)}, x_2^{(i)}, \dots, x_N^{(i)}\}$ ，其中 $x_n^{(i)}$ 是第 i 個被當成是 landmark 的資料的第 n 個特徵)。當有新輸入的資料進來時，若這筆資料的特徵值為 $\mathbf{x} = (x_1, x_2, x_3, \dots, x_N)$ ，我們就計算 \mathbf{x} 和 $\mathbf{L}_i (i = 1, 2, \dots, m)$ 之間的相似度：

$$\begin{aligned}
 f_1 &= \text{similarity}(\mathbf{x}, \mathbf{L}_1) \\
 f_2 &= \text{similarity}(\mathbf{x}, \mathbf{L}_2) \\
 f_3 &= \text{similarity}(\mathbf{x}, \mathbf{L}_3) \\
 &\dots \\
 f_m &= \text{similarity}(\mathbf{x}, \mathbf{L}_m)
 \end{aligned} \tag{13}$$

我們用 $\mathbf{f} = (f_1, f_2, f_3, \dots, f_m)$ 來取代 $\mathbf{x} = (x_1, x_2, x_3, \dots, x_N)$ 當成是 SVM 的輸入。關於 landmark 的選擇，最簡單的方式，是從 training data 當中任選一些資料當成是 landmarks，只要這些資料的 features 和 labels 沒有高度的一致性即可。若要對 landmarks 的選擇做最佳化，則需要用到 KKT conditions 這樣的數學工具，這不在本 tutorial 敘述的範圍內。在 Eq. (13) 當中，相似度 (similarity) 有許多種不同的定義方式。以 Gaussian Kernel 為例，

$$f_n = \text{similarity}(\mathbf{x}, \mathbf{L}_n) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{L}_n\|}{2\sigma^2}\right) \tag{14}$$

其中 σ 為 Kernel 參數，影響 Gaussian function 的胖瘦，而 $\|\cdot\|$ 指的是 2nd order norm

(平方相加再開根號)。由上式可以看出當 \mathbf{x} 和 Landmarks 的距離越遠時， $\|\mathbf{x} - \mathbf{L}_n\|$ 將越大， f_n 的值將越小。

透過 Eq.(12)，原本的 Features Vector \mathbf{x} 被 mapping 到另一個空間 $\mathbf{f} = (f_1, f_2, f_3, \dots, f_m)$ 。在這個空間中 train 出的 SVM model 具有 Nonlinear 的分類線，而此時 SVM 的 Cost Function 將由 Eq. (12) 更改為

$$\min_{\theta} \left\{ C \left[\sum_{i=1}^m y^{(i)} \text{cost}_1(\boldsymbol{\theta}^T \mathbf{f}^{(i)}) + (1-y^{(i)}) \text{cost}_0(\boldsymbol{\theta}^T \mathbf{f}^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^m \theta_i^2 \right\}. \quad (15)$$

其中 $\mathbf{f} = (f_1, f_2, f_3, \dots, f_m)$ 。

5.3 Using SVM

SVM 的參數 C 以及 Eq. (14) Gaussian Kernel 的參數 σ 對 SVM 有以下影響：

- C 越大，越傾向於 Fit Training Data。如果 SVM 的驗證結果是 under-fitting 的話，可以考慮調高 C 。反之，若出現 Over-Fitting 的狀況，可考慮降低 C 。
- Eq. (14) 的 σ 越大，Gaussian Function 越胖，使得 \mathbf{f} 的值隨著輸入資料和 Landmarks 的相似度降低而變小的速度緩慢，易同時容易受到幾個相近 Landmarks 的影響而變成 under-fitting。反之， σ 越小，instances 越不容易同時受許多 Landmarks 的影響，因此較能自由地 fit training data，較易形成 Over-fitting。

而 SVM 和 Logistic Regression 演算法的使用準則為：

- 當 training data 的數量太小（相對於 Features 的數量），使用 Model 複雜度較低的 Logistic Regression 或 Linear SVM（沒有 kernel 的 SVM）。
- 當 training data 的數量中等，而 Features 的數量很少時，使用 Model 複雜度較高的 Gaussian Kernel SVM（即 Section 5-2 的 nonlinear SVM）。
- 當 training data 的數量很大，而 Features 的數量很少時，必須再創造或增加其它 Features。

總而言之，nonlinear SVM 的優點在於可以利用 Eq.(13) 提升 Model 複雜度，以得到較為複雜的分類線，但是需要更多的 Training Data。相對的，Section 4 介紹的 Logistic Regression 則是不必用太多的 Training Data，但不能得出太複雜的分類線。

要特別注意的是，在使用 SVM 之前，需要先用 Eq.(4) 來對每個 features 做一般化，否則會造成某些特徵的影響被過度的放大的問題。這一點也是一般 SVM 使用者經常沒有注意到的地方，以致於 SVM 的效果不如預期。使用 SVM 時切勿忘了這一步。

關於 SVM 更詳細的介紹可以參考 [1][2][3]，並建議的閱讀 [5]。SVM 的程式碼可

以在 [8][11] 找到，其中 [11] 的 SVM 程式碼有很高的下載率。連結至 [11] 所列的網頁之後，將網頁網下拉，就可以看到“Download LIBSVM”，接下來就可以下載 zip file 或是 tar.gz file，解壓縮並安裝之後，即可使用。

[14] 則是運用 [11] 所下載的 SVM 的 Matlab code 來執行 SVM 的二個範例，分別是運用身高和體重來判斷一個人是成人還是未成年人的範例，以及運用經緯度來判斷一個地方是台北市或是新北市的範例。

6 Neural Network

Neural network (類神經網路) 是一種試圖模擬大腦的演算法，在 1980 及 1990 早期被廣泛使用，但由於被證明在雙螺旋 (two spirals) 的問題中 training time 是呈 exponential 的速度成長，從 1990 年代後期開始，它在 machine learning 上的地位逐漸被 SVM 取代。(雙螺旋問題是指一種對於任何直線，或任何複雜度過低的分類線而言，錯誤率和正確率皆在 50% 左右的資料分佈。如圖 11，可以看出任一條直線分類線都大致將黑、白兩種資料點平分)

然而，在 2010 年代初期，由於 deep learning 的興起 (deep learning 其實就是多個階層的 neural network)，使得 neural network 的地位又重新被提升了。

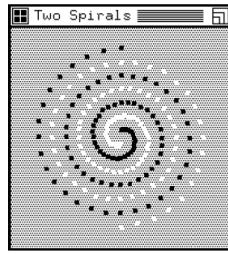


圖 12: 雙螺旋 (Two Spirals) 分佈 [7]

6.1 Model Representation

6.1.1 Neural Unit

Neural network unit 是模仿人類的神經元，它基本上是一個 Logistic Model，如圖 13。左方 a_i 是 input， g 是 logistic function， a_j 是 output，而 w_{ij} 是權重 (weight)。

$$in_j = w_{0,j} + \sum_i w_{i,j} a_i, \quad a_j = g(in_j) = g\left(w_{0,j} + \sum_i w_{i,j} a_i\right), \quad (16)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

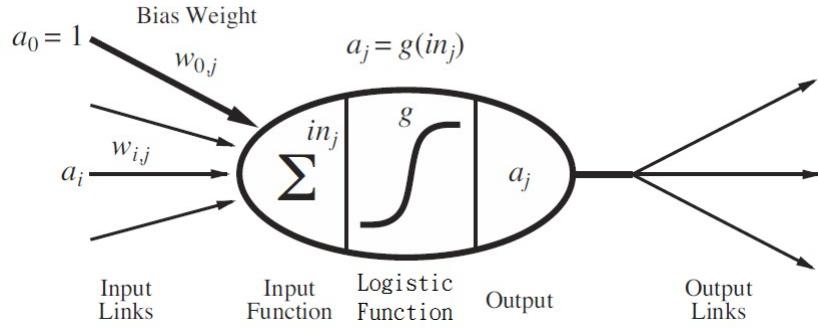


圖 13: Neural Network Unit [7]

6.1.2 Neural Network

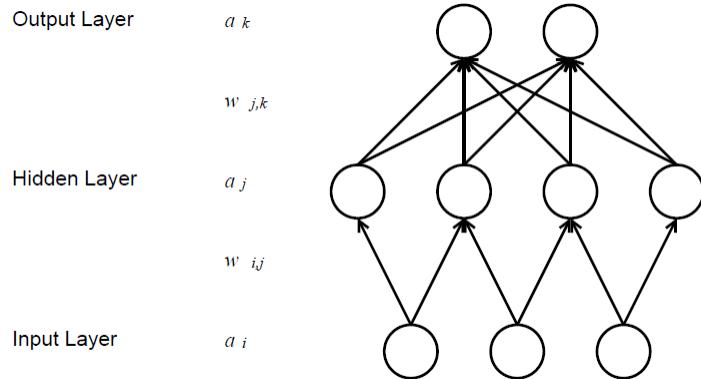


圖 14: 3-Layer Neural Network [7]。每一個 node 皆為如同圖 12 的 neural network unit

將 Section.6.1.1 圖 13 敘述的 neural network unit 排列、組合如圖 14，即成為 Neural Network。每個 Unit 之間的 Edge 有各自權重。圖 14 是普遍而言較受歡迎的 3-Layer Neural Network，但每一層的 unit 數及整體架構的層數並不受任何限制。唯一要注意的是雖然越多層、越多 unit，Neural Network 就越能學到複雜的分類線，但同時也需要更多的 training data 和 training time (甚至會超出可實作的範圍)。

以下將給出幾個簡單的例子，說明為何透過調整 neural network unit 之間的權重可以達到 Machine Learning 效果。

6.2 Intuition

根據 Eq.(16) 的 logistic function，當 z 越大時， $g(z)$ 的值越接近於 1，反之則接近於 0。如圖 15，透過給予 Edge 適當權重，我們可以得到和 logical “AND” 一樣的真值表。圖 16 則是 logical “OR”的例子。

可以發現。透過給予 input 不同的 weights (權重)，每個 Neural Network Unit 可以做出不同的 Function。如果，將這些 Functions 放在一起，Neural Network 幾乎可做出

任何形狀的分類線。缺點是需要的 training data 數量和所花費的 training time 都相當的龐大。

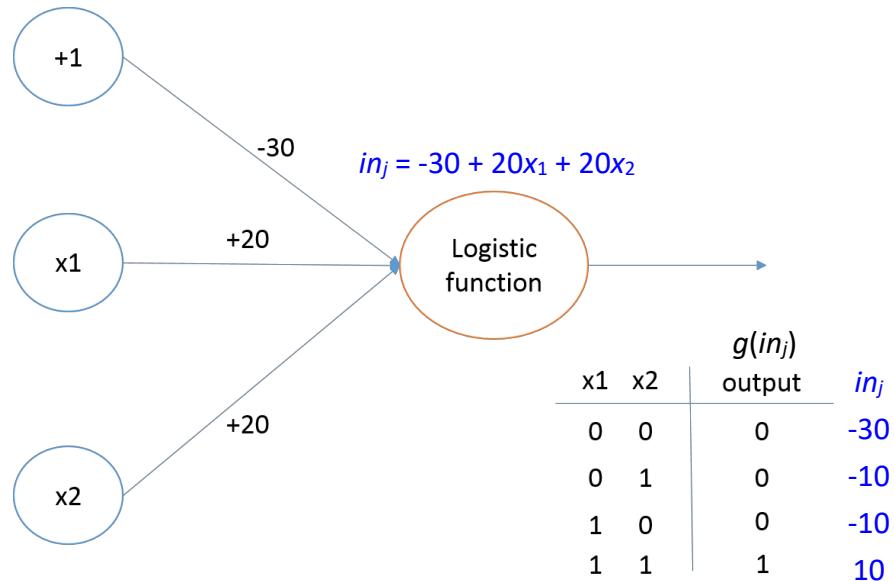


圖 15: Neural Network – AND function [1]

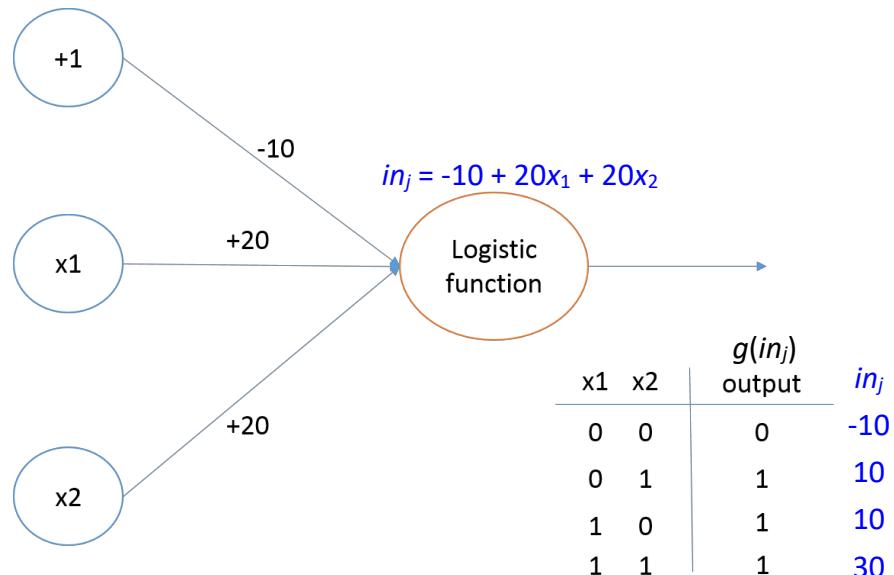


圖 16: Neural Network – OR function[1]

6.3 Training the Weight of Neural Network

這裡的 Training 方式，是使用 Back-Propagation 得出 Gradient 後，配合 Gradient Descent Algorithm 訓練出 Weight。

首先定義 Neural Network Unit 的 Cost function E 為

$$E \equiv \frac{1}{2}(y - o)^2, \quad (17)$$

其中 $o = g(in_j) = \frac{1}{1 + \exp(-in_j)}$ (in_j 和 $g(in_j)$ 定義於 Eq. (16) 和 圖 13) 為 Neural Network Unit 的 output (相當於圖 12 的 a_j)， $y = 0$ 或 1 為 training data 對應的真正的 output (經常稱作 label)。則依據 Gradient Descent Algorithm，每個 Edge 的 Weights 更新方式為

$$w_{i,j} \leftarrow w_{i,j} - \alpha \frac{\partial E}{\partial w_{i,j}} \quad (18)$$

其中

$$\frac{\partial E}{\partial w_{i,j}} = (y - o) \frac{\partial}{\partial w_{i,j}} (y - g(in_j)) = -(y - o) g'(in_j) a_i$$

由 Eq. (16)，我們可以證明 Logistic Function $g(z)$ 滿足 $g'(z) = g(z)(1 - g(z))$ ，所以 Weights 的更新變為

$$w_{i,j} \leftarrow w_{i,j} + \alpha (y - o) \cdot o \cdot (1 - o) \cdot a_i. \quad (19)$$

Eq. (19) 是 neural network 只有單一的 layer 的情形下 weights 的更新方式 (其實，單一 layer 的 neural network 和第四章的 logistic regression 頗為相似，只是 cost function 不同)。接下來，我們討論在 multi-layer Neural Network (即圖 14) 當中，該如何使用 Gradient Descent 的方法來更新 Weights $w_{i,j}$ 和 $w_{j,k}$ 。

此時採用 Back-Propagation 演算法。“Back-Propagation”顧名思義，就是將 Output Layer 的 Error 往回傳到 Hidden Layer，然後依據每個 Edge 的權重，「回推」出每個 Unit 需要對 Output 的錯誤負多少責任，從而更改 Edges 的權重 $w_{i,j}$ 。

根據這樣的觀念，我們可以假設 Hidden Layer 的 Error function Δ_j 和 Output Layer k 的 Error function Δ_k 會有如下的關係

$$\Delta_j = g'(in_j) \sum_k w_{j,k} \Delta_k \quad (20)$$

其中 in_j 為神經元 j 的 input 且 $\Delta_k = (y_k - o_k)o_k(1 - o_k)$ ，其中 o_k 為 Output Layer 的 neuron k 的輸出， y_k 為理想的第 k 個 output， $(y_k - o_k)$ 即為 Output layer 的 Error。由此可以推測 Output Layer 及 Hidden Layer 的 Weights 更新方式為

$$\begin{aligned} w_{j,k} &\leftarrow w_{j,k} + \alpha \cdot (y_k - o_k)o_k(1 - o_k)a_j, \\ w_{j,k} &\leftarrow w_{j,k} + output_j \cdot \Delta_k, \end{aligned} \quad (21)$$

$$\begin{aligned} w_{i,j} &\leftarrow w_{i,j} - \alpha \cdot \frac{\partial E}{\partial w_{i,j}} = w_{i,j} - \alpha \frac{\partial}{\partial w_{i,j}} \sum_k \frac{1}{2} (y_k - o_k)^2 \\ &= w_{i,j} + \alpha \sum_k \frac{\partial o_k}{\partial w_{i,j}} (y_k - o_k) = w_{i,j} + \alpha \frac{\partial in_j}{\partial w_{i,j}} \frac{\partial o_j}{\partial in_j} \sum_k \frac{\partial in_k}{\partial o_j} \frac{\partial o_k}{\partial in_k} (y_k - o_k) \\ &= w_{i,j} + \alpha \cdot a_i g'(in_j) \sum_k w_{j,k} o_k (1 - o_k) (y_k - o_k), \end{aligned}$$

$$w_{i,j} \leftarrow w_{i,j} + \alpha \cdot output_i \cdot \Delta_j \quad (22)$$

.....

其中 $output_i$ 和 $output_j$ 為神經元 i 和 j 的 outputs ($output_i = g(in_i)$, $output_j = g(in_j)$)。藉由 Back – Propagation，不斷地讓 Neural Network 調整各個 Edges 的權重，最後達到我們需要的 Model。

然而，要注意的是，Eqs. (20) 和 (22) 都缺乏嚴謹的理論基礎，使得不少的學者並不相信 neural network 的結果，這也是近年來 neural network 並未像 SVM 那麼普及的原因之一。

Neural Network 的優點是能夠得出非常複雜的分類規則，但是缺點是需要的 training data 數量和 training time 非常的可觀。

如果讀者必須使用 Neural Network 的話，建議採用 Cascade Correlation Neural Networks，或者至少使用 Quick Back – Propagation，這是一種 Back – Propagation 的變形，能夠較快 Train 出 Neural Network Model。

讀者若要對 neural network 有更深的了解，可以參考 [1][2][3][5] 等資料。

7 Learning Work Flow and Validation

這個章節說明如何檢驗 Learning Algorithm 的效果。首先必須弄清楚 Training Error，Test Error，和 Validation Error 之間的不同。在某些文章中，Test Error 和 Validation Error 兩詞會被混用。在這裡兩者並不相同，前者指 Test data 的 Error，後者指 Validation data 的 Error (test data 和 validation data 的差別將在 Section 7-1 說明)；而 Training Error 則僅僅只是 model 符合 Training data 的程度，有時 Training Error 過小反而表示整個系統對於 Training data 太過於 Over-fitting 了，在新的、未知的 data 上未必有好表現。

7.1 Split Data and Design of the Machine Learning System

假設現在手上有 m 個 instances (m 筆資料)。在做 machine learning 時，我們不能將 m 個 instances 都拿去訓練 Model，因為這樣無法驗證訓練出來的 Model 是否對未知的、新的 data 也同樣有效。

因此，一般做法有兩種，要驗證的都是「訓練出來的 model 是否能夠廣泛應用到未知的、新的 instances 上」：

- 將資料分做兩群，比如取 60 – 70% 的 instances 去 train model，當作 training set，剩下的 instances 當作 test set。training 完 Model 之後，將其應用到 test set 上，以得到的 test error 檢驗 model 是否有效，若效果不佳則回頭調整 model 參數或增減 features。
- 將資料分做三群，比如取 60% 當作 training set，20% 當作 validation set，剩下的

20% 當作 test set。

先以 training set 訓練出 model 以後，將其應用到 validation set 上，以 validation error 檢驗 model 是否有效，若效果不佳則回頭調整 model 參數或增減 features。最後，當達到滿意的 validation error 後，以 test set 檢驗這個 model 的有效性，若對 test set 的效果不佳，這時通常是 features 的 information 不足或是有太多不相干的 features 了，必須從頭思考如何取 features。

和資料分作兩群的作法相比，再多分一群的好處是能夠確保我們保有一些 instances (即 test set) 是在調整 model 時，完全沒有被 model 看到的，這樣能夠保證我們對 model 的調整不會只是符合現有 instances，而是能夠確實一般化到完全未知的新 instances 上。

Machine Learning 系統的設計流程，大致上如下列的步驟所述。

(Step 1) 拿到 Data 並將 Data 隨機分成 Training Data、Validation Data、以及 Test Data 三組。

(Step 2) 使用 training data 來訓練 Machine Learning Model

(Step 3) 用 Validation Data 來檢驗 Model 是否有效。

(Step 4) 以 Test Data 做最後檢驗。

在最後的結果不佳，或者需要加速 Learning Algorithm 的速度時，需要在 Training 步驟前應用 PCA 做 feature reduction。

7.2 Features Reduction – PCA

這裡介紹一個常常用來做 Feature Reduction (或稱作 dimensionality reduction，降維) 的工具，Principal Component Analysis，(PCA)。假設原本的 features 是個 n 維向量，PCA 主要的概念是留下最重要的前面 k 個向量，將原本 n 維的 features，投影到較小的 k 維空間中。這個方法主要是為了解決當我們擁有很多 Features 時，很有可能各個 Features 之間存在著相關性，而這些相關性較高的 Features 就並不需要每一個都採用，因為相關性高就代表著採用其中幾個較重要的 Features 就可以得到大部分的資訊，不需要把每一個 Features 都拿進來，而所謂的 Principal Component 就是那些較重要的 Features。

PCA 的概念和線性代數中求 eigenvectors 和 eigenvalues 相似，通常 eigenvalue 的絕對值越大，就代表這個 eigenvector 越重要，在降維時越應該被保留。

7.2.1 PCA Algorithm

藉由 Matlab 或 Octave 的 svd 指令，可以用以下流程做 PCA：

- 先對 features 做 normalization 確保每個 feature 有 zero-mean (必要時也需做 feature scaling)。

- 假設有 n 個 features，input $\mathbf{x}^{(i)}$ 將是有 n 個 entries 的 column vector ($^{(i)}$ 代表第 i 筆資料)。找出現有 n 維 features 的 Covariance Matrix： $\Sigma = \frac{1}{m} \sum_{i=1}^m \mathbf{x}^{(i)} (\mathbf{x}^{(i)})^T$, Σ 是一個 $n \times n$ 的矩陣
- 使用 Matlab 或 Octave 的 *svd* 指令： $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \text{svd}(\Sigma)$ ，其中我們需要的是 $n \times n$ 的 \mathbf{U} matrix。
- 將原本的 features \mathbf{x} 投影到 \mathbf{U} 的前 k 個 columns 上，即得到 PCA reduction 後的 k 維 feature vector \mathbf{z}

$$\mathbf{z}^{(i)} = [\mathbf{U}(:, 1:k)]^T \mathbf{x}^{(i)}$$

關於 k 值的選擇，有約定俗成的描述方式： \mathbf{z} 向量保留了 \mathbf{x} 向量的 99%, 95%, 90%, 85%, ... variance。它的意思大略是 \mathbf{z} 向量保留了 \mathbf{x} 向量的多少 information。這個 Variance 的算法可藉由 *svd* 指令的 output matrix, \mathbf{S} , 得到。選取保留 99% 的 k 值公式為：

$$\frac{\sum_{i=1}^k S[i,i]}{\sum_{i=1}^n S[i,i]} \geq 0.99 \quad (23)$$

其中 0.99 可隨意換成想保留的 Variance 多寡。

以下附上 Matlab 的 PCA 範例程式碼（使用 Matlab 內建的 *svd* 指令）：

```
%X is a 2-D matrix
% featureNormalize
[n, m]=size(X); % n: num. of features, m: num. of instances
mu = mean(X);
X_minus_mean = X - mu'*ones(1,m);
sigma = std(X_minus_mean);
X_norm = X_minus_mean./ ( sigma'*ones(1,m));

% PCA
Covariace_Matrix = 1/m*(X_norm*X_norm');
[U, S, V] = svd(Covariace_Matrix); % by the built-in svd function

% project to top-k dimensions of U
% K-value selected by user and smaller than col
U_reduce = U(:, 1:K);
X_PCA_result = U_reduce' *X_norm;
```

7.2.2 Applying PCA

PCA 的用途，一般而言有三個：

- Compression – 降低用來儲存資料的記憶體或硬碟需求
- 純粹只是想要在低維度（一維、二維）的空間上觀看資料分佈
- 加速 Machine Learning Algorithm

PCA 有時會被拿來處理 Over-fitting 的問題。的確，藉由 PCA 減少 Features 數目可以一定程度 上改善 over-fitting 的狀況，但除非是有非常高度的運算效率需求，降維不宜降維得過多。原因是 PCA 畢竟會減少 features 所含的 information 量，它不一定能精準判斷哪些 features 是有用的、哪些是沒用的。

7.3 Diagnosing the Model

最後，這裡提供一些調整 Machine Learning Model 的原則，主要由觀察 training error 和 validation error，來判斷 model 是否是 over-fitting 或 under-fitting。

1. Training Error 大，Validation Error 也大：此時為“under-fitting”，需增加 features 數量或增加 model 的複雜度。以 SVM 為例，可改用 Section 5-2 非線性的 SVM。
2. Training Error 很小，Validation Error 也很小：通常這樣是不錯的結果，可以應用到 test set 上觀察最後的結果了。
3. Training Error 很小，但 Validation Error 很大：此時為“over-fitting”，可以透過取得新的 instances，增加 training set 的大小改善。也可以減少 Features，或使用複雜度較低的 learning model，如 logistic model。

8 Conclusion

關於 machine learning 的簡單介紹，在此告一個段落。這個 tutorial 主要是提供想要會用 machine learning，想要對 machine learning 有基礎概念的讀者來參考。若你不以基礎概念的了解為滿足，想要對 machine learning 的原理做更全面的了解，甚至想針對 machine learning 的理論做研發，可以參考這個 tutorial 最後附的參考資料。其中 [1] 是全世界最知名的 machine learning 線上學習資料，[5] 則是一本詳細且清楚的介紹 machine learning 的書本，[11] 是由台大資工林智仁教授所架設的網頁，每天都有來自世界各地的學者來此下載 SVM 的程式碼，[12] 也是和 SVM 相關值得參考的資料。另外，[8] 也是一個頗負盛名的 machine learning 程式碼下載網站。

參考資料

- [1] Andrew Ng, *Machine Learning*, Stanford Cousera, available from <https://www.coursera.org/course/ml>
- [2] Hsuan-Tien Lin, “Machine Learning Foundations”, NTU Cousera, available from <https://www.coursera.org/course/ntumalone>
- [3] Y. S. Lin, Abu-Mostafa, M. Magdon-Ismail, and H. T. Lin, *Learning from Data: A Short Course*, AMLBook.com, 2012.
- [4] F. Rosenblatt. ”The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, 65(6):386-408, 1958.
- [5] Tom Mitchell, *Machine Learning*, McGraw-Hill, 1997.
- [6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The Elements of Statistical Learning*, Springer, 2009
- [7] Stuart J. Russel and Peter Norvig, *Artificial Intelligence: A Modern Approach*, second edition
- [8] Scikit , Python (實作的 Learning Algorithm Library) 。<http://scikit-learn.org/stable> 。
- [9] Suykens, Johan AK, and Joos Vandewalle. ” Least squares support vector machine classifiers.” *Neural processing letters* 9.3 (1999): 293-300.
- [10] Tong, Simon, and Daphne Koller. ” Support vector machine active learning with applications to text classification.” *The Journal of Machine Learning Research* 2 (2002): 45-66.
- [11] C. C. Chang and C. J. Lin, “LIBSVM -- A library for SVM,” available from <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- [12] C. W. Hsu, C. C. Chang, and C. J. Lin, “A practical guide to support vector classification,” available from <http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>
- [13] D. Klein, “Lagrange multipliers without permanent scarring.” University of California at Berkeley, Computer Science Division (2004).
- [14] “Examples for SVM using Matlab codes,” available from <http://djj.ee.ntu.edu.tw/SVMExamples.zip>
- [15] “Happyman’s Studio -- Support Vector Machine,” available from <https://cg2010studio.wordpress.com/2012/05/20/%E6%94%AF%E6%8C%81%E5%90%91%E9%87%8F%E6%A9%9F%E5%99%A8-support-vector-machine/>
- [16] 李根逸，支持向量機教學文件，台灣大學通訊與多媒體實驗室。網址：<http://www.cmlab.csie.ntu.edu.tw/~cyy/learning/tutorials/SVM1.pdf>