機器學習演算法

Machine Learning Algorithms

葛竑志 著

丁建均 教授 編修

Feb, 2020

CONTENT

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF TABLES

Chapter 1 Instance Based Learning

Instance 具有「物件」的意思，這一類機器學習演算法將每個 training data 視為一種物件，在物件與物件之間的比較之中來獲得對於 label 的預測，又稱作 Memory Based Learning。相對於許多機器學習模型旨在建立一套通則化（generalized）的模型來對新的資料進行預測，這種方法強調的是將各個已標記（labeled）的 training features 儲存形成一個巨大的資料庫，當要對新的資料預測時去比較各個 feature 的相似度，從而找出最適合的歸類，也因此這類學習的時間複雜度與 training data 的數量成正比。

1.1 K-Nearest Neighbor (k-NN)

k-NN 沒有實質上的 training 過程，可以想像一組具有 n 個 features 的 training data 分佈於 $\mathbb{R}^n$ 空間中，當預測資料的 feature 則會在這個空間中與這 n 個 training data 計算彼此的距離。其中，k 屬於一種重要的 hyper-parameter，必須視使用情況決定 k 的大小。

如圖 1-1，當$K = 1$時 *k*-NN 會直接將距離最近的 instance 的 label 作為輸出，等價於 Nearest Neighbor method。當$K = 3$時則有可採 label voting 的方式決定，得票最高者來做為 classification 結果。同樣地，k-NN 也適合用於 non-discrete 的預測，僅需要將 neighbor label 的決定方式改為平均值 (mean) 或中位數 (median) 處理即可。

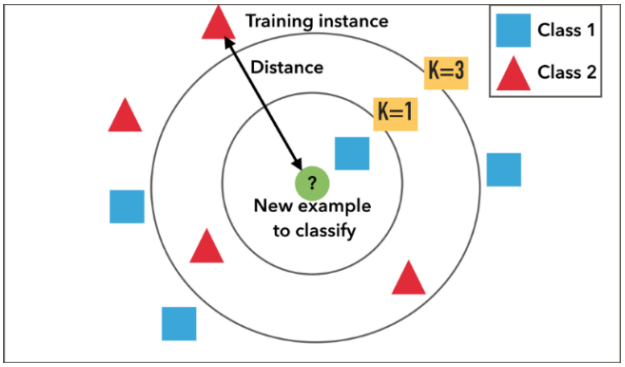


圖 1-1　k-NN 對於 new example 的預測示意圖。[1]

另外，某些版本的 k-NN 會較為用距離來決定權重 (weight) [2]，依照距離反比的值給予該 label 的權重，如式 1-1，其中x代表待預測的資料的 feature，x_i代表x的 neighbor，與其對應的 label y_i，w_i為權重，$dist(.,.)$為距離公式

$$y = f(x) = \frac{\sum_i w_i y_i}{\sum_i w_i}，w_i = \frac{1}{dist(x, x_i)}$$

式 1-1　Distance-weighted k-NN 公式 (The case where there are only two labels)

K-NN 同時適合用於線性與非線性的資料預測上，但其不可避免的缺點為過於龐大的時間與空間複雜度，為了建立這樣的資料庫，不得不將所有 training data 如實儲存起來，進行預測的階段上也勢必要花掉很多時間在距離的量測上。此外，維度詛咒 (curse of dimensionality) 也在 k-NN 上有所展現，在真實世界的多維資料通常只有少數的 attribute 是有用的，倘若某些 attribute 彼此具有高度正相關性，將會使得 k-NN 容易受到誤導。

1.2　Locally Weighted Learning (LWL)

LWL 與 k-NN 相同，沒有實質上的 training 過程，亦屬於 lazy learning 的代表，主要應用於 regression 上。LWL 和一般的 regression 不同。一般的 regression

其實屬於一種 global fitting 的方法，嘗試在某種假設下（如次方項的大小）以最小平方法找到適合的參數（weight）。然而，倘若給定的預測 features 超出 training data 的範圍過多，往往不會有很好的預測精準度。相對而言，LWL 則是強調 local fitting 的概念，希望在預測某個 feature 時只參考鄰近的點作為參考，而不是會受到距離遙遠資料的影響。[5][6]

如圖 1-2，令橫軸為x，縱軸為y。圖左為 Linear regression 的模式，以最小平方法找到一條無論x在何處都以這條線為預測的 regression line。圖右則是 LWL 模式下，當我們想要知道（Query）x對應y為多少時，則會先依據鄰近的 Query 的鄰居產生一條 regression line 後，再計算出y的值，也因此 LWL 的漸近線，實際上是不受到次方項假設的一條曲線。

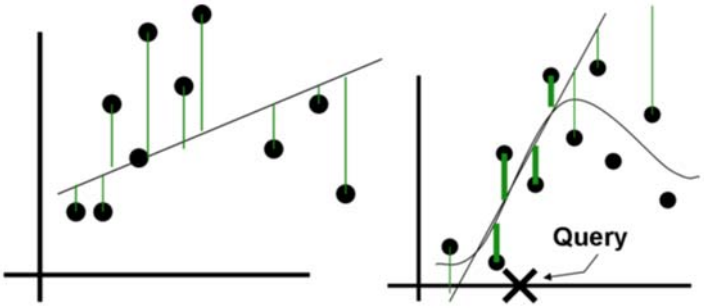


圖 1-2　(左圖) Linear regression 與 (右圖) LWL 的比較[3]

以數學表示計算針對x的 regression linear 的 optimal weight 的方式如式 1-2，x為要 query 的點，x_i為 training feature，N代表以x為中心於x軸上的 neighbor set（可採 k-NN 模式或 distance threshold 的方式定義之），$K(.,.)$代表計算 neighbor 距離

的 kernel function，通常為 Gaussian function。物理意義上來看，其實 (式 1-2) 正好是傳統 linear regression 所用到的最小平方法的 weighted 版本。

$$w^* = argmin_w \sum_{x_i \in N} K(x, x_i)(w^T x_i - y_i)^2$$

$$= \arg\min_w \sum_{x_i \in N} \frac{e^{-\frac{(x-x_i)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} \left(w^T x_i - y_i\right)^2$$

式 1-2　LWL 之 optimal weight 定義

計算出 weight w^* 之後，則可以求出對應該 query x 的 regression line，$y^* = w^{*T} x$，從而得出符合 local 結構的預測。為了比較漸進的效果，以下列出數種不同 regression 方法的 regression line：自圖 1-3 至圖 1-5 中給定了三個 training data（圖左，training data 以黑點表示）與五個 training data（圖右）時，將會產生形狀不同的 regression line，其中可見 LWL 的表現往往較符合預期。

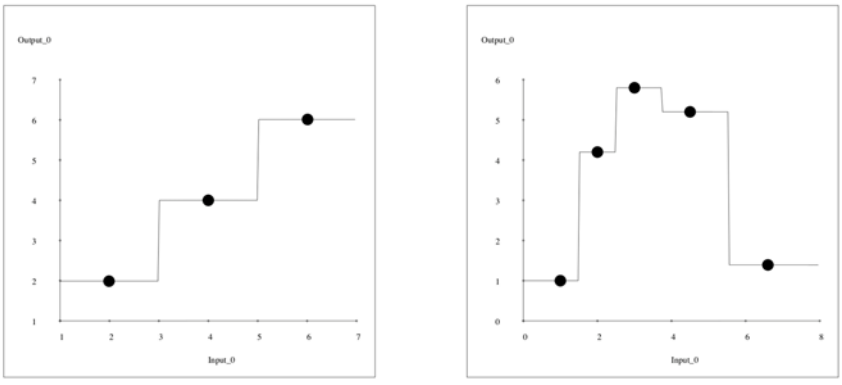


圖 1-3　Nearest Neighbor Regression[4]

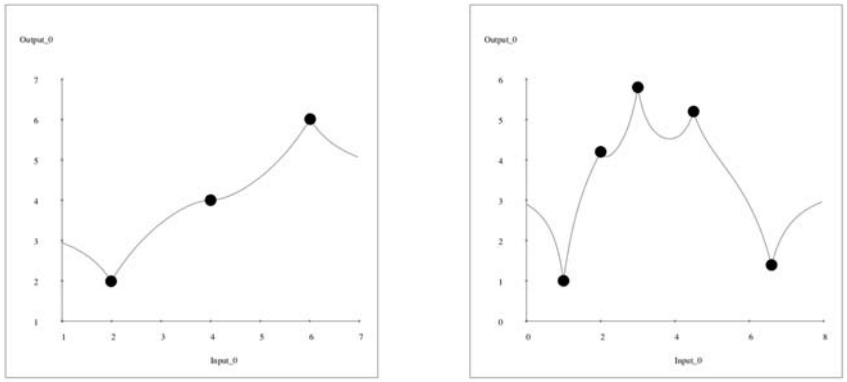


圖 1-4　Weighted Average Regression[4]

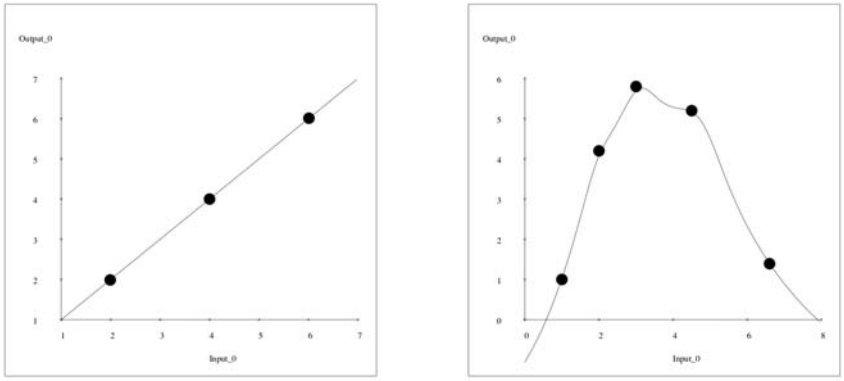


圖 1-5　Locally Weighted Learning (LWL) Regression[4]

1.3　Learning Vector Quantization (LVQ)

LVQ 和 Vector Quantization(VQ)有很密切的關係，在介紹 LVQ 之前，不妨理解一下 VQ 是什麼。VQ 是一種傳統信號處理上用來壓縮資料的方法，目的是將原本需要多個位元編碼的信號壓縮至只需要較少位元進行編碼的 codeword 上，為此，在壓縮時必須犧牲掉一些細微資訊，以「最能夠代表原始資訊」的 codeword (可以想成特徵平面上每個區域的中心點) 來代表原始的信號進行傳輸。[9]

Codeword 的集合稱之為 Codebook，建立 codebook 的方法有三種：(i) K-means、(ii) LBG algorithm 及 (iii) Centroid splitting algorithm [7]。 此處不會特別說明如何計算 codeword，但以概念性的例子來說明就如圖 1-6 所示的 Voronoi diagram (為經典的演算法課題，假使其中一個 codeword 稱為y_i，則「與y_i同屬一區的任一點

到y_i」的距離必定小於等於「該點到其他 codeword」的距離）。當然，在某些無法理解原理的資訊上，我們無從事先得知怎麼樣的 codeword 最能夠代表一群資訊，所以以上三種方法皆屬於 unsupervised clustering 的方法。

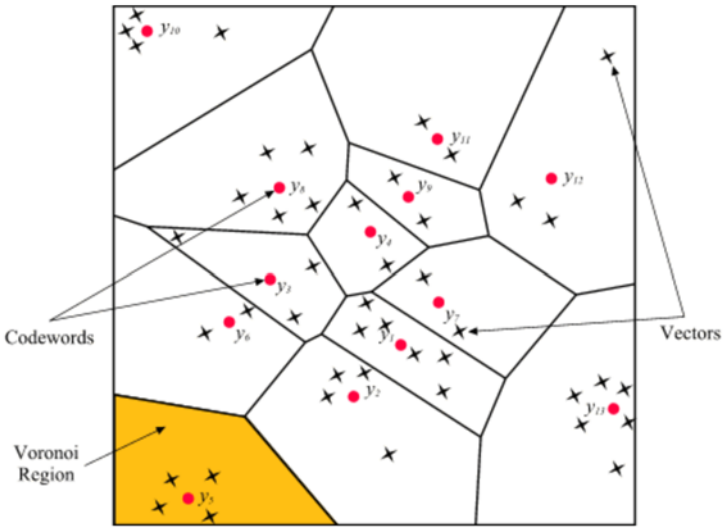


圖 1-6　對於二維 vector 在平面上所建立的 Codewords 與其 Voronoi Region [8]

LVQ 與 VQ 的最大不同在於前者是能夠事先知道 label 的 (所以 LVQ 屬於一種 supervised clustering 方法)，並且藉此來訓練出最能代表一群 vector 的代表 codeword (即 clustering)。因此，以數學表示 LVQ 的目標函數如式 1-3 所示，其中x代表任一個 training data，x_i代表 codeword，K為 codeword 的數量，$u_i(.)$用以判斷x是否屬於x_i所代表的類別，期望值則是對所有的x取期望。

$$J_{LVQ} = E\left[\sum_{i=1}^{K} u_i(x)\|x - x_i\|^2\right], \qquad u_i(x) = \begin{cases} 1 & \text{if } x \in i^{th} \text{ region} \\ 0 & \text{otherwise} \end{cases}$$

式 1-3　LVQ 之目標函數

然而，LVQ 中的 quadratic form 具有對於 outlier 特別敏感的問題，因此有些人提出以 1-norm 取代$\|x-\bar{x_i}\|^2$項，或是在 2-norm 項外圍加上$\Phi(x)=||x||^{\alpha}$ 的 kernel 如 (式 2-4)所示，其中 α 可以小於 2。

$$J_{R-LVQ}=E\left[\sum_{i=1}^{K}u_i(x)\|x-x_i\|^{\alpha}\right],\qquad u_i(x)=\begin{cases}1 \text{ if } x\in i^{th} \text{ region}\\ 0 \qquad \text{otherwise}\end{cases}$$

式 1-4　Robust LVQ 之目標函數

1.4 Self-Organizing Map (SOM)

SOM 與 K-means 同屬於一種 unsupervised clustering 方法，相較於 K-means 與 LVQ，SOM 的 clustering 方法更具有可解釋性。SOM 是在 1980 年代類神經網路誕生之初的產物之一，起初許多學者都爭相模仿神經生理學上的特徵來構思類神經網路的架構。

LVQ 與 SOM 屬於「競爭式學習」框架下的方法，試圖讓 neuron 彼此競爭來「贏取」代表某個類別的權利，且在獲得該權力的同時為了增加 neuron 的專一性，必須去降低其他神經元獲得該權力的機會，此正是所謂的贏者全拿（winner-takes-all）的「側向抑制」機制。然而，根據一些神經生理學的研究，學者發現有些神經網路則是有「側向連結」的特徵，即當某個神經元活化的時候，周遭的神經元也有一起被活化而學習的現象，因此 Kohonen 在 1982 年提出 SOM 的時候使用了如圖 1-7 的回饋函數。

SOM 主要由兩個層組成，如圖 1-8 分為輸入層與輸出層。輸出層通常為一維或二維空間中排列的神經元，經過 training 後，輸出層的神經元會以原始資料的拓樸結構展現在一維或二維空間上，屬於某種程度上具有拓樸意義的降維視覺化方

法。SOM 的輸出神經元具有與 training feature 同等的維度，命名為w且$w \in \mathbb{R}^d$，在 training 過程中必須使得 winner 神經元能夠越接近「它所負責的輸入」越好。

第一步為對 w 進行 initialization。令第 j 個 neuron 的 feature 為 $w_j, j = 1, \ldots, N$。

第二步則輸入其中一個 training data $x_i \in \mathbb{R}^d$，找出 winner neuron，如式 1-5

$$j^* = arg\ \min_j \left\| x_i - w_j \right\|^2, for\ j = 1, \ldots, N$$

式 1-5　SOM 之 winner neuron 決定公式

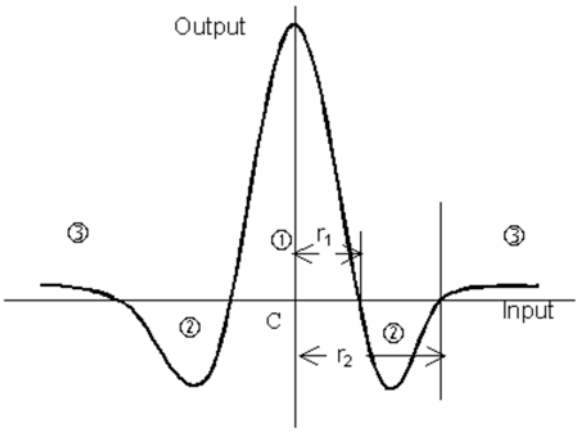


圖 1-7　墨西哥帽 (Mexican hat) 函數[10]可做為回饋函數

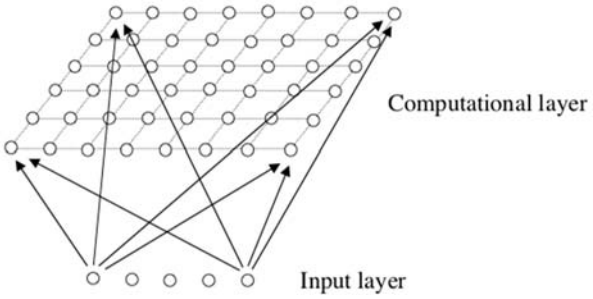


圖 1-8　SOM 基本結構 [11]

第三步則是對w_j進行 update，如式 1-6 所示，其中，$N_{j^*}(.)$代表以 winner 神經元 j^* 為中心所圈出的 neighbor set，$\phi(., k, j^*)$為以「第k個 neuron 與 winner 神經元 j^* 的距離」作為考量所賦予的學習權重函數，定義如式 1-7 所示，其中的h_k代

表第k個 neuron 在 output layer 上的幾何座標，係用以衡量 neuron 彼此距離所設置的虛擬座標。其中 t 代表第 t 次迭代數，可觀察到 w、$\phi(.,.,.)$ 與 $N_{j^*}(.)$皆屬於迭代變數。

$$w_k(t+1) = \begin{cases} w_k(t) + \phi(t,k,j^*)[x_i - w_k(t)], & k \in N_{j^*}(t) \\ w_k(t), & k \notin N_{j^*}(t) \end{cases}$$

式 1-6　SOM 之 neuron feature update 公式

$$\phi(t,k,j^*) = exp\left(-\frac{\|h_k - h_{j^*}\|}{2\sigma^2(t)}\right)$$

式 1-7　SOM 之學習權重函數

第四步則是重新進行二、三步驟直到迭代終點。

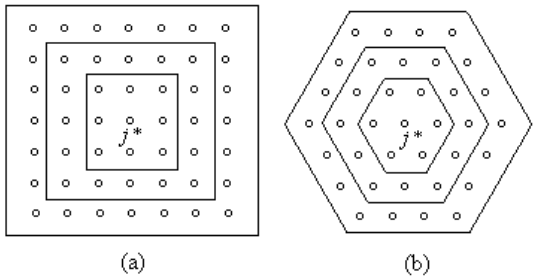


圖 1-9　SOM 之 Neighbor set 定義範圍 [49]

不妨注意到式 1-7 的$\sigma^2(t)$也屬於迭代變數，通常會希望 SOM 在訓練的過程中從「排列」朝向「收斂」方向發展，因此通常會定義$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{T}\right)$，其中以$\sigma_0$表示初始變數，$T$表示總迭代數。另一方面，$N_{j^*}(.)$基本上可使用如圖 1-7 的定義，往往也可以使用不同 pattern 如圖 1-9，training 過程中範圍由大至小限縮。

圖 1-10 表示了 SOM 的訓練過程：符號 o 代表 SOM 的 output neuron，並以連線表示 neuron 間的關係（已較多連線相鄰的 neuron，代表彼此的距離越長），符號 x 則代表從目標空間（黃底處）所抽樣出來的 training data。在第一個步驟時，

選出了某一個 training data 作為 input，而此時標記為實心黑的 neuron 正是 winner，將會代表周邊的 neuron 一起 update 成類似 input data 的 feature，在下一個步驟則是換成另一個 training data 作為 input，如此往復便能產生一種近似原始空間分佈的拓撲結構。圖 1-11 的例子亦顯示了 SOM 展開 output neuron 的過程。

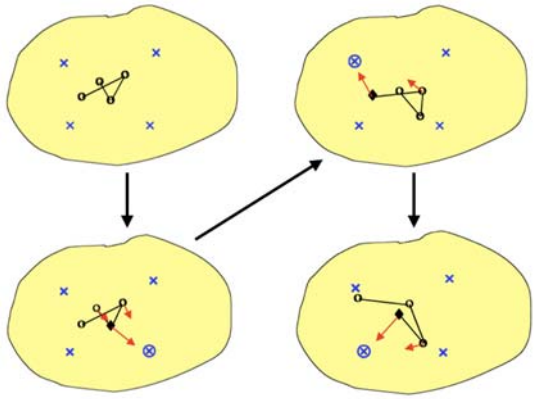


圖 1-10 SOM 訓練過程之視覺化表示法之一[11]

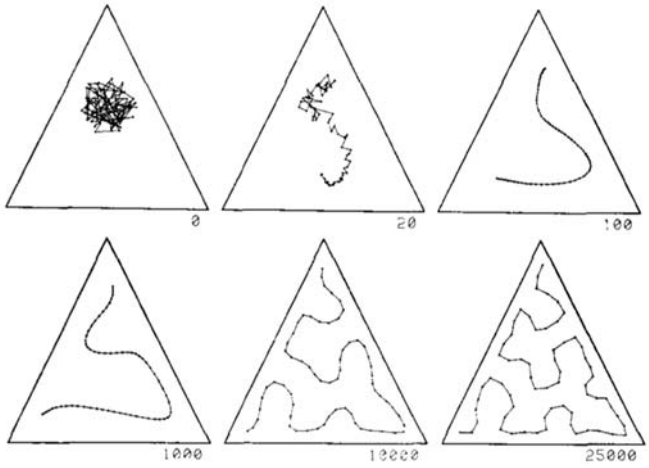


圖 1-11 SOM 訓練過程之視覺化表示法之一[12]。右下角的數字代表 iteration 的次數。

Chapter 2 Dimensionality Reduction

Dimensionality Reduction 是一種以更少資訊用來描述原始高維資料特性的學問，降低維度的好處除了能夠節省記憶體空間，更能夠減少高維資料常有的 noise，並加速機器學習模型的收斂速度。

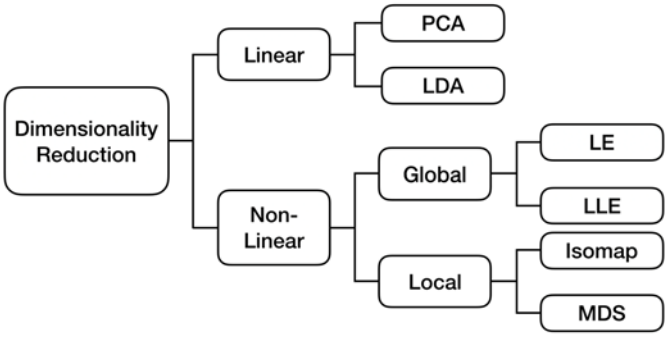


圖 2-1 Dimensionality Reduction 方法架構

2.1 Linear Methods

2.1.1 Principle Component Analysis (PCA)

PCA 主要目的在於找出互相正交的 principle axis，使得原始高維資料投影在這些 axis 上時能夠去突顯其 variances。如此一來，若能將 principle axis 儲存作為 dictionary，即可以高維資料的投影值作為降維後的輸出。舉例來說，原始資料具有 d 維，則可藉 PCA 產生 k 個 d 維的 principle axis（其中$k < d$），則投影後的資料僅有 k 維。

一般定義 PCA 的 objective function 如式 2-1，其中v_i代表 principle axis，$\langle v_i, x \rangle$代表v_i與x的投影內積，而需要去找出一組v_i能夠 minimize 這種 reconstruction error。

$$J_{\text{PCA}} = E\left[\left\| x - \sum_{i=1}^{K} \langle v_i, x \rangle v_i \right\|^2\right]$$

式 2-1 PCA 之目標函式（一）

另一種等價的 objective function 定義則從直觀的理解去 maximize 投影後的 variance 如式 2-2，其中σ^2代表投影後變異數，v^T為 principle axis 的 matrix form，μ_x為x_i的平均數，N為 feature 的數目。在 feature 已預先做過 normalization 的前提下，可知投影後變異數可改寫為 covariance 的關係。如此一來，即可寫作如式 2-3 所示，其中C代表 covariance matrix，需要在$\|v\| = 1$的前提下做 maximization。

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(v^T x_i - \mu_x)^2$$

式 2-2　PCA 之目標函數（二），其中 v, x_i, μ_x 皆為 column vectors

$$v = \underset{v \in R^d, \|v\|=1}{argmax}\ v^T C v$$

式 2-3　PCA 之目標函式（二）最佳化形式

$$\|v\| = 1$$

式 2-4　對投影向量的限制

解這類問題時，亦可以用 singular value decomposition (SVD) 的方式，詳細的解法可以參考 [44] 的附錄。PCA 是個普遍且常用的工具，但必須考慮到 PCA 的一大假設在於 eigenvalue 的大小決定了資訊的重要性，也就是本身 eigenvalue 較小的 component 往往只能夠代表 noise 的成分，因此我們可以忽略掉過小的 component 進行降維。另一方面，以 eigen decomposition 的方式求解也使得對於 principle axis 的解釋性降低。

2.1.2　Linear Discriminant Analysis (LDA)

LDA 與 PCA 同為線性的降維方法，但是更加入了對於 label 的考量（supervised），因此 LDA 的 cost function 是以「最小化組內變異」及「最大化組間變異」為考量而設計的。

為了說明 LDA 的公式，此處先進行名詞與符號的定義：

v 為投影矩陣，v 總共有 d 個 columns (也就是說，$v^T x$ 是將輸入 x 投影成 d 維的輸出)，$\|v_n\| = 1$ for all n 其中 v_n 是 v 的第 n 個 column。C_j代表第j組之集合。m_j為第j組之原始平均，$\mu_j = v^T m_j$為第j組之投影後平均。C_j代表第j組之集合，S_j^W代表第j組之原始組內變異數，S_j^{within}代表第j組之投影後組內變異數。S_j^B代表第j組之原始與他組平均數的變異數，$S_j^{between}$代表第j組之投影後與他組平均數的變異數。由此，我們可定義 LDA 之組內及組間總變異數如式 2-5、式 2-6，則可以導引出能夠符合 LDA 最小化投影後組內變異與最大化投影後組間變異的目標函數式 2-7。

$$S^{within} = \sum_{j=1}^{L} S_j^{within} = \sum_{j=1}^{L} \sum_{i \in C_j} \left(v^T x_i - v^T m_j\right)\left(v^T x_i - v^T m_j\right)^T$$

$$= \sum_{j=1}^{L} v^T \left(\sum_{i \in C_j} \left(x_i - m_j\right)\left(x_i - m_j\right)^T \right) v = \sum_{j=1}^{L} v^T S_j^W v = v^T \sum_{j=1}^{L} S_j^W \, v = v^T S_W v$$

式 2-5　LDA 之投影後組內總變異數

$$S^{between} = \sum_{j=1}^{L} S_j^{between} = \sum_{j=1}^{L} \sum_{i=1, i \neq j}^{L} \left(v^T m_i - v^T m_j\right)\left(v^T m_i - v^T m_j\right)^T$$

$$= \sum_{j=1}^{L} v^T \left(\sum_{i=1, i \neq j}^{L} \left(m_i - m_j\right)\left(m_i - m_j\right)^T \right) v = \sum_{j=1}^{L} v^T S_j^B v = v^T \sum_{j=1}^{L} S_j^B \, v = v^T S_B v$$

式 2-6　LDA 之投影後組間總變異數

$$J_{LDA} = \frac{S^{between}}{S^{within}} = v^T S_B v (v^T S_W v)^{-1} = v^T (S_W S_B^{-1}) v$$

式 2-7　LDA 之目標函數

式 2-8 為以限制投影後組內差異為 1 的前提下所進行的最大化解，我們可以觀察到這種形式與 PCA 求解的脈絡相當相似，因此得到偏微分解如式 2-9，因此接著以 eigen decomposition 的脈絡找出適合的 principle axis 即為 LDA 的解。

$$v = \underset{v^T S_W v = 1}{argmax}\, v^T S_B v$$

式 2-8　LDA 之目標函數最佳化形式

$$\frac{\partial f(v,\lambda)}{\partial v} = 0 \rightarrow 2S_B v - 2\lambda S_W v = 0 \rightarrow S_B v = \lambda S_W v \rightarrow S_W^{-1} S_B v = \lambda v$$

$$\text{其中} \quad f(v,\lambda) = v^T S_B v - \lambda v^T S_w v$$

式 2-9　LDA 目標函數之 Lagrange Multiplier 解

也就是說，v 的每個 columns 其實是 $S_W^{-1} S_B$ 的 eigenvectors，λ 則為 eigenvalues。

2.2 Non-Linear Methods

2.2.1 Multi-Dimensional Scaling (MDS)

MDS 旨在降維過程中保留原始 feature 之間的 pairwise distance 特性。舉例而言，若今有一個圖表表示各個城市之間的距離，在不考慮原始資料的維度的情況下，我們能夠透過 MDS 在低維空間中將它們的彼此的關係展現出來。

	London	Berlin	Oslo	Moscow	Paris	Rome	Beijing	Istanbul	Gibraltar	Reykjavik
London	–									
Berlin	570	–								
Oslo	710	520	–							
Moscow	1550	1000	1020	–						
Paris	210	540	830	1540	–					
Rome	890	730	1240	1470	680	–				
Beijing	5050	4570	4360	3600	5100	5050	–			
Istanbul	1550	1080	1520	1090	1040	850	4380	–		
Gibraltar	1090	1450	1790	2410	960	1030	6010	1870	–	
Reykjavik	1170	1480	1080	2060	1380	2040	4900	2560	2050	–

圖 2-2　MDS 中的 Distance Matrix[14]

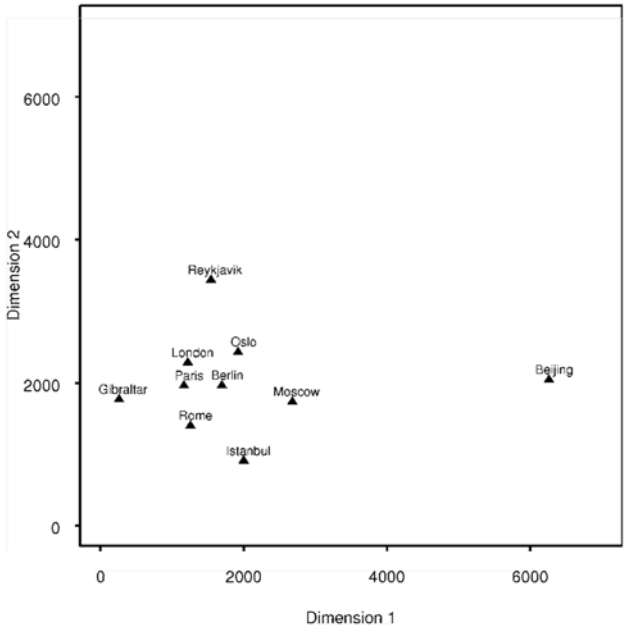


圖 2-3　將圖 2-2 之結果以 MDS 方式降至二維平面表示[14]

一般情況下，我們常以 Euclidean distance 作為高維空間中兩點的距離，即$d_{ij} = \|x_i - x_j\|, x \in R^d$。但退一步思考，Euclidean distance 所描述的是幾何空間的距離，要是高維度的資料並非落在這種空間時，是否就不適用這種方法來描述所謂的「距離」呢？因此，與其使用絕對的距離作為表示，許多學者更傾向以 Dissimilarity 來表示 feature 彼此之間的距離，代表兩個越相類似的點其 Dissimilarity 越低。

Dissimilarity 越高代表兩者之間的距離越遠，越低則越短。相反地，Similarity 越高代表兩者越近，越低則越遠。一般以式 2-10 的方式描述兩者的關係，其中d_{ij}為i與j兩者的 dissimilarity，s_{ij}為i與j兩者的 similarity。

$$d_{ij} = \sqrt{s_{ii} + s_{jj} - s_{ij}}$$

式 2-10　Dissimilarity 與 Similarity 關係

首先，建立一個類似於圖 2-2 的 Dissimilarity matrix D，其 entry 定義為$d_{ij} = \|x_i - x_j\|, x \in R^d$。則接下來需要將$D$轉換為 Similarity 的形式，如

$$A = \left[a_{ij} = \frac{1}{2} d_{ij}^2\right]$$

$$B = [b_{ij} = a_{ij} - a_{i.} - a_{.j} + a_{..}]$$

式 2-11 將 Dissimilarity matrix 轉換為 Similarity matrix

一般來說，MDS 以此分為 Metric 與 Non-metric 兩種方法，在 Metric MDS 的範疇下又有一種 classical MDS，是將 dissimilarity 定義為 Euclidean distance 的方法，在之後的推導我們可發現其降維的方法與 PCA 是等價的。

2.2.2 Isomap

相較於 MDS 在高維空間中進行距離的量測，Isomap 則是採測地線距離（Geodesic distance）來描述流形（manifold）的距離。舉例來說如圖 2-4，Swiss Roll Dataset 為一種分布在三維空間中的資料集，以其宛如一張二維的紙所捲成的形狀為名，顏色越相近的點代表彼此是越相近的。當我們要量測圖 2-5A 圖中兩點的距離時，若使用傳統的幾何距離直接做量測（如藍色虛線）便會誤以為這兩點是十分相近的，實際上卻不然，我們通常會沿著該平面在空間中的走向畫出一條線（如藍色實線）來合理地代表這兩點的距離，這正是所謂的 Geodesic distance。

Isomap 首要遇到的難題是，當人們對高維資料的分布一無所知時，該如何去合理地畫出 geodesic line？它的想法也相當簡單，如果無從得知整體的樣貌，不如先從局部結構去建立起 feature 之間的關係，也就是 Neighbor graph，最後還是可以在遙遠的兩點之間拉起一條 geodesic line 的。如圖 2-5(C) 為建立出來的 neighbor graph，自兩點之間畫起的 shortest path distance 正好與藍色實線拉直的效果相去不遠。

首先，可先以 k-NN 模式或定義一定幾何距離以下的點作為 neighbor 的條件，接著在對每個 vertex 做 neighbor 判斷時，將兩者互為 neighbor 的情況下以 edge 相連彼此，如此一來，neighbor graph 就完成了。接著可採 Dijkstra's algorithm [45]或 Floyd-Warshall algorithm [46][47]，計算各個 vertex pair 的 shortest path distance 後就能建立 Dissimilarity Matrix，而這種 Dissimilarity Matrix 可再透過 MDS 演算法的方式找進一步出低維度的表示法。一言以敝之，Isomap 相較於 MDS 只是改變了計算 distance 的方式。

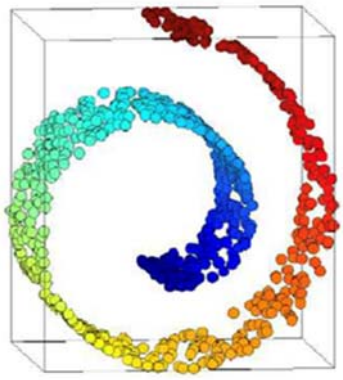


圖 2-4　Swiss Roll dataset[16]

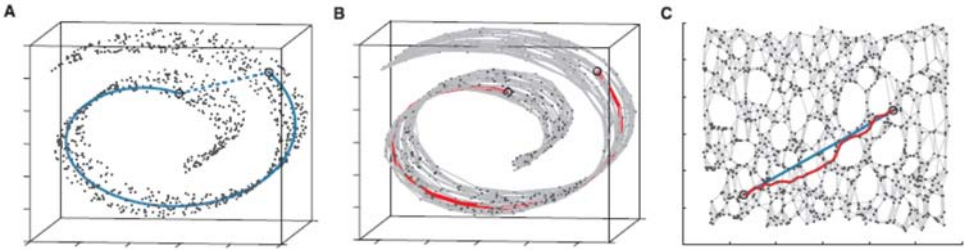


圖 2-5　Geodesic distance 在 Swiss Roll 上之實現[15]

Isomap 可以應用於諸如圖片類型的高維資料型態，舉圖 2-6 的例子，將一段鳥類飛行的圖片以 Isomap 進行降維後，可在低微空間中逐 frame 劃出一條

Trajectory，觀察這種 trajectory 可發現影片中存在著鳥類展翼（紅線，週期性的畫圓模式與展翼的動作相符）與滑行（黃線）的兩種不同型態。

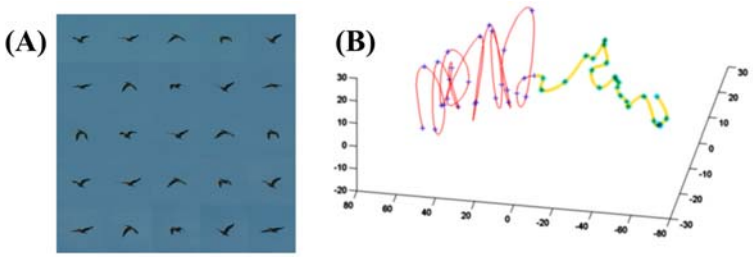


圖 2-6　Isomap 應用於鳥類飛行影片與其在低微空間中的 Trajectory[17]

2.2.3　Linear Locally Embedding (LLE)

任選流形上的某一點放大觀察，我們能發現該處通常會是一個相當平坦的區域。LLE 則是建立在這樣的觀察下，而假設局部區域的點都是其鄰近點的「線性」組合，如果這種線性關係在降維的過程中得以被保留，理應能夠為流形的結構做充足的解釋。同樣舉 Swiss Roll Dataset 的例子如圖 2-8，能夠發現 LLE 的降維結果相當不錯。[18][19]

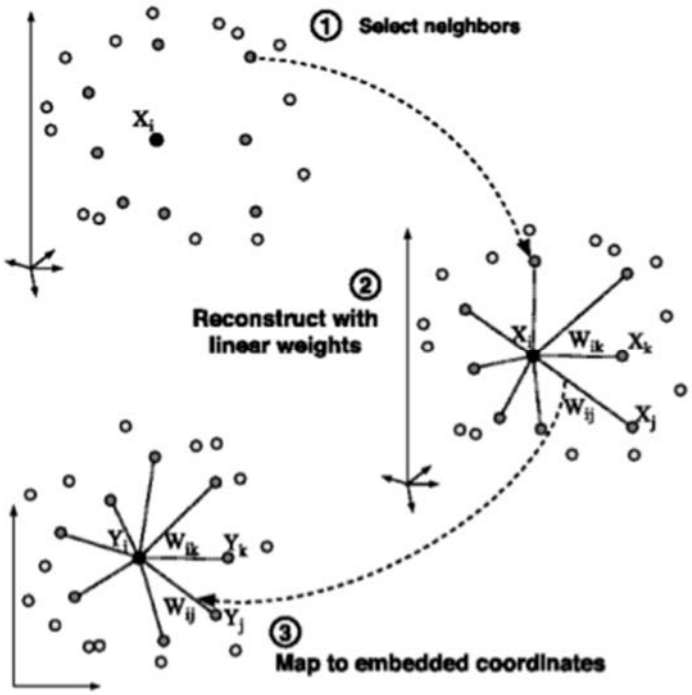


圖 2-7　LLE 中某一點為其鄰近點之線性組合示意圖[19]

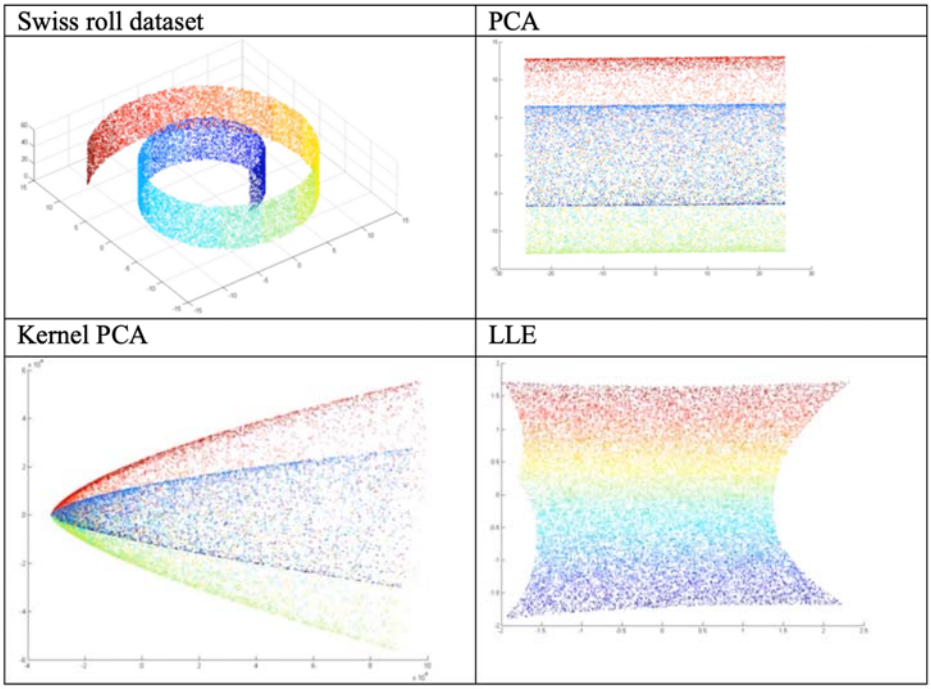


圖 2-8　三種降維方法於 Swiss Roll Dataset 上的表現[20]

令W_{ij}為一個代表第i個 feature 與第j個 feature 是否為 neighbor 的 sparse matrix，

我們可在高維空間中，就可以運用 features 之間的關聯性將W_{ij}計算出來，以此來

定義如式 2-12 的 objective function，其中y_i代表第i個 feature 的低維度空間座標，希望去找到一組y能夠 minimize 這種 reconstruction error。(也就是希望降維之後 features 之間的關係還是能夠維持)

$$\underset{y}{argmin}\sum_{i}\left(y_i-\sum_{j}W_{ij}y_j\right)^2$$

式 2-12 LLE 之 objective function

將上式改寫為 Matrix form 後可以得到式 2-13，其中$M=(I-W)^T(I-W)$，並再加上對於Y的限制來避免 degeneration（全為 0）。再以 Lagrange multiplier 方式改寫後，發現這類問題與 PCA、LDA 等相同，屬於 eigen decomposition 求解的脈絡如式 2-14。因此，對M做 eigen-decomposition，並加 eigenvalue 由大而小做排序，其中前數個非零 eigenvalue 對應的 eigenvector 即為投影向量。

$$(Y-WY)^2=Y^T(I-W)^T(I-W)Y=Y^T\mathrm{MY}$$
$$Y=\underset{Y,Y^TY=1}{\operatorname{argmin}}\,Y^T\mathrm{MY}$$

式 2-13 LLE 之 optimal solution

$$f(Y,\lambda)=Y^TMY-\lambda(Y^TY-\mathbf{1})$$
$$\frac{\partial f(Y,\lambda)}{\partial Y}=2MY-2\lambda Y=0\ \rightarrow MY=\lambda Y$$

式 2-14 LLE 之以 Lagrange multiplier 求解過程

2.2.4 Laplacian Eigenmap (LE)

前面所提到的 LLE 假設每一個點都能從其 neighbor 的線性關係所組成，並在降維的過程中保持這種關係維持不變。LE 則是假設每一個 feature 都只與其 k 個 neighbor features 相似，並在降維的過程中保持這種相似關係。

LE 會在高維空間中建立 feature 之間的 neighbor graph，建立方法則是一般的 ε-neighborhood（距離小於ε時就是 neighbor）或 k-nearest neighborhood。另一方面對於這些 neighbor 則需要一個 weighted matrix 去定義彼此的相似度，此時令W_{ij}為一個代表第 i 與第 j 個 features 相似程度的 sparse matrix，兩者的幾何距離越近則W_{ij}的值會越高（也代表越相似），通常是以 Heat kernel：$W_{ij} = e^{-\frac{\|x_i - x_j\|^2}{t}}$作為輸出，其中$t$為一種 hyper-parameter。當$t \to 0$時，則是一種只要是 neighbor 時 weight 就是 1，否則為 0 的狀況。在高維資料中建立W_{ij}後，則可以此來定義如式 2-15 的 objective function，其中y_i代表第i個 feature 的低維度空間座標，希望去找到一組y能夠 minimize 這種 reconstruction error。（不妨注意到 LE 的 objective function 只與 LLE 差在W_{ij}的權重位置）

$$\sum_{i,j}(y_i - y_j)^2 W_{ij} = \sum_{i,j}(y_i^2 - 2y_i y_j + y_j^2)W_{ij}$$

式 2-15 LE 之 objective function

則此時再令$D_{ii} = \sum_j W_{ij}$，$D_{jj} = \sum_i W_{ij}$，則可將 objective function 整理如式 2-16，這時$L = D - W$係為 Laplacian matrix，LE 則是因此而得名。由此，我們可推導出如式 2-17 的 optimal solution，同樣地，為了避免落入 trivial 解（y全為 0），必須對y做限制。

$$\sum_i y_i^2 D_{ii} + \sum_j y_j^2 D_{jj} - 2\sum_{i,j} y_i y_j W_{ij} = 2\left(\sum_i y_i^2 D_{ii} - \sum_{i,j} y_i y_j W_{ij}\right) = 2y^T L y$$

式 2-16 LE 之 objective function

$$\underset{y, y^T D y = 1}{argmin}\ y^T L y$$

式 2-17 LE 之 optimal solution

補充：在圖論問題上，D為一種 diagonal matrix 稱作 degree matrix，W為一種 symmetric matrix 稱作 adjacency matrix。Laplacian matrix 常用於解決 Graph 問題 [21]，具有如 positive semidefinite 等良好特性（eigenvalue 皆大於等於 0），因此在此被用來對 objective function 求解。

接著就按照類似於 LLE 的處理方式，如式 2-18 採 Lagrange multiplier 與 eigen decomposition 的方式解出y。則將 eigenvalue 由大而小做排序，其中前數個非零 eigenvalue 對應的 eigenvector 即為投影向量。

$$f(y,\lambda) = y^T L y - \lambda(y^T D y - 1)$$

$$\frac{\partial f(y,\lambda)}{\partial y} = Ly - \lambda D y = 0$$

$$\frac{\partial f(y,\lambda)}{\partial \lambda} = -(y^T D y - 1) = 0$$

式 2-18 LE 之以 Lagrange multiplier 求解過程

2.2.5 SNE

SNE (Stochastic Neighbor Embedding) 由 Hinton, Roweis 所提出 [23]，亦屬於一種保留空間結構的方法(Embedding)。這種方法將高維空間的歐式距離轉換為相似度的條件機率分佈：在衡量兩點相關性時，並非直接測量x_i與x_j的距離，而是將x_i設為高斯分布的中心後，x_j有多少機率屬於x_i的 neighbor。令降維過程$x_i \to y_i$，其在高維的定義如式 2-19，其中σ_i的選擇是相當重要的，由於資料分布的稠密性不同，必須要有一套方法動態調整σ_i的值（稠密處使用較小的σ_i，反之則大）。[23]

$$p(j|i) = \frac{exp(\frac{-\|x_i - x_j\|^2}{2\sigma_i^2})}{\sum_{k \neq i} exp(\frac{-\|x_i - x_k\|^2}{2\sigma_i^2})}$$

式 2-19 SNE 之高維 neighborhood PDF

在此，SNE 使用 perplexity (定義如下) 測量各個 conditional PDF P_i的優劣，perplexity 越小代表該分佈在預測時的效果較佳，使用者可預先指定 perplexity 在 5~50 範圍內，接著以 binary search 找出各個P_i所對應的σ_i。

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$$

$$Perp(P_i) = 2^{H(P_i)}$$

式 2-20 Perplexity 之定義

另一方面，在低維的定義則是直接令 $\sigma_i = \frac{1}{\sqrt{2}}$ 如式 2-21，以試圖讓降維後的分佈 scaling 能夠盡可能一致。

$$q(j|i) = \frac{exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} exp(-\|y_i - y_k\|^2)}$$

式 2-21 SNE 之低維 neighborhood PDF

SNE 以 KL divergence (KLD) 作為衡量 $p(j|i)$ 和 $q(j|i)$ 兩個條件機率分佈相似的標準，KLD 越小代表兩個分佈越相似。則將 KL divergence 對y_i 做偏微，便可找到降低 KLD 的 gradient 如式 2-22

$$C = \sum_i KL(P_i \| Q_i) = \sum_i \sum_j p(j|i) log \frac{p(j|i)}{q(j|i)}$$

$$\frac{\delta C}{\delta y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j) \quad \text{(???)}$$

式 2-22 KL divergence 之梯度

2.2.6 *t*-SNE

t-SNE 改進之處在於(1)將 SNE 的 cost function 改為 symmetric 的版本 (2)在低維表示中以 student-*t* distribution 取代 Gaussian。以下將分項描述：

(1) 觀察 SNE 中採用的 KLD metric，其實$KL(P_i||Q_i) \neq KL(Q_i||P_i)$。此時會有個問題: 使用分散的低維分佈來表示鄰近的高維分佈時(即以小$q(j|i)$表示大$p(j|i)$)，KLD 較大。但是相反過來，使用鄰近的低維表示分散的高維時，KLD 會較小，這是為何 KLD 不對稱的原因。也因此，SNE 將焦點放在保留 local structure 上。t-SNE 為了使 cost function 變得較為對稱，則是將 conditional PDF 改為 joint probability 如式 2-23

$$p_{ij} = \frac{\exp\left(-\|x_i - x_j\|^2/2\sigma^2\right)}{\sum_{k\neq l}\exp\left(-\|x_k - x_l\|^2/2\sigma^2\right)}$$

$$q_{ij} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k\neq l}\exp\left(-\|y_k - y_l\|^2\right)}$$

式 2-23 t-SNE 對 neighborhood PDF 的改進（一）

但此時會有一些問題：當x_i為 outlier 時，p_{ij}將會非常小，連帶使得y_i在 cost 的影響上顯得比較微弱，因此就以直觀的方式令式 2-23 的P_{ij}改為式 2-24。做完以上改進後，則梯度的表示法如式 2-25 所示。

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

式 2-24 t-SNE 對 neighborhood PDF 的改進（二）

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

式 2-25 t-SNE 改為 joint probability 的梯度

(2) 這種技巧主要是去解決 Crowding problem。舉例而言，一個在 m 維空間中半徑為 r 的球，其體積為 r^m，假設在高維空間中在此體積中均勻分佈各種 data point，在降到低維空間的過程中就會發生 crowding 的現象，因為體積是隨著指數倍下降的。另一個例子如圖 2-9 則是在 2d 空間中均勻分佈的點，拓展至高維空間中後，會發現 bar 圖會顯得較為集中在某個幾個距離點上，顯示越高維的資料降維後 crowding problem 越嚴重。為了減輕 crowding 的影響，t-SNE 在低維表示中採用 t-student distribution (DOF=1)這種重尾的分佈，如式 2-26 及圖 2-10。

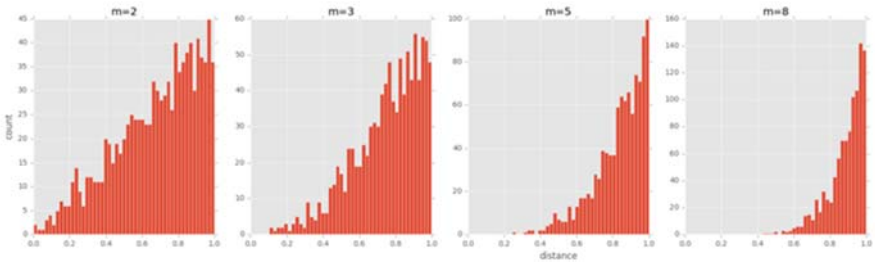


圖 2-9 降維的 crowding problem[22]

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

式 2-26 t-SNE 將低維的 neighborhood PDF 改為 *t*-student distribution

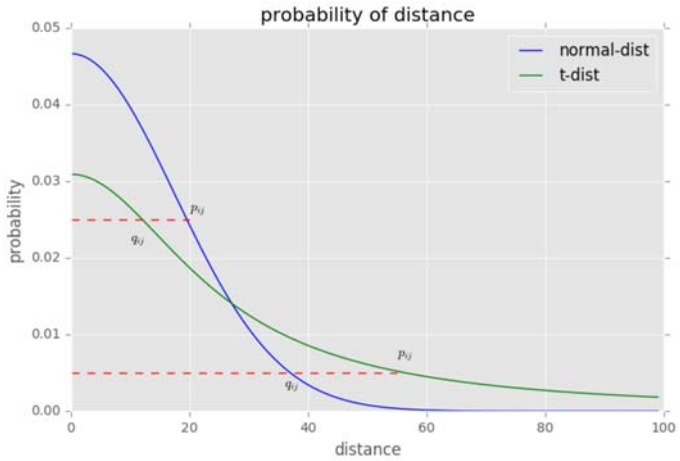


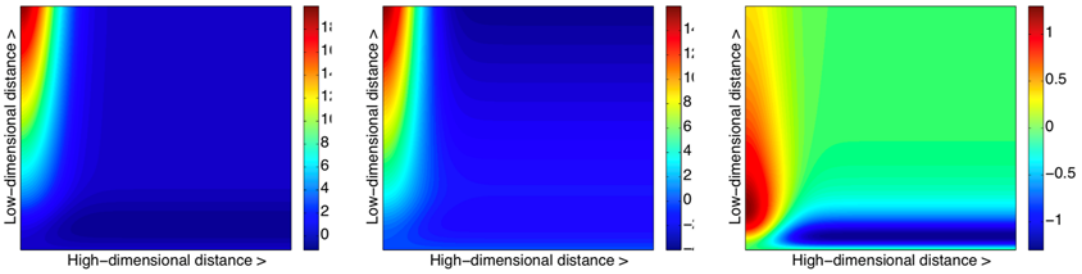
圖 2-10 normal-distribution 與 *t*-distribution 的比較[22]

此時有個相當好的特性：當兩點相距很遠時，q_{ij}就可以符合 inverse square law。這種特性讓本來相距甚遠的兩點的 joint probability 幾乎是 scale-invarient 的。此外，採用 t-student distribution 時，能讓近者更近，遠者更遠。因此將式 2-26 代回到式 2-25，最後進行優化時所使用的 gradient 將如式 2-27 所表示，且使得 gradient map 變得較為合理如圖 2-11，其中正的 gradient 代表吸引力，負的代表相斥力，

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1 + \|y_i - y_j\|^2\right)^{-1}$$

式 2-27 t-SNE 之梯度

和式 2-25 相比，式 2-28 算出的 gradient 將會隨著 $\|y_i - y_j\|^2$ 而減小。



(a) Gradient of SNE. (b) Gradient of UNI-SNE. (c) Gradient of t-SNE.

圖 2-11 SNE 至 t-SNE 之梯度演進[43]

Chapter 3 Decision Tree

Decision Tree (DT) 是對 feature 進行一系列分類問題：如 feature 中的哪些 attribute 需要符合哪些條件等，能夠將該 feature 分類到其所對應的 label 的樹狀圖。DT 所學習的 parameter 則是各個 node 上用來分類的 criteria，屬於一種 supervised learning 的方法。[24]

此處必須先瞭解 Information gain（IG）的概念，其定義如式 3-1，其中 D 代表 data 的集合，D_p代表 node p以下的 data 集合。N 代表 data 的數量，N_p代表 node p以下的 data 數量。m 取決於 node p底下有多少個 edge，若為二元分類問題，則 $m = 2$。$I(.)$代表 Impurity，高代表資料中有許多類別，低則代表資料中單一類別的成員較多。DT 的目標是使得末端的$I(.)$值越低越好，因此必須要在 node p上尋找最能夠提高$IG(.)$的 criteria f。

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j)$$

式 3-1 Information Gain 定義 (將 D_p 分成多個 D_j 的 information gain)

對於$I(.)$的定義有許多種，如表 3-1 所列，當 label 數量只有 2 時，不同定義的$I(.)$會有不同樣貌如圖 3-1。其中$p(i|t)$的定義如式 3-2 所示，t代表 node，D_t代表 node t以下的 data 集合，N_t代表 node t以下的 data 數量，i代表類別的 index，$1(.)$代表條件符合時輸出為 1，否則為 0。因此式 3-2 的物理意義上即是去表示對於 node t底下的 dataset，有多少比例的 data 被歸類為 class i。

$$p(i|t) = \frac{1}{N_t} \sum_{x \in D_t} 1(x = i)$$

式 3-2 $p(i|t)$定義

表 3-1　Impurity 定義

Gini Impurity	$I_G(t) = \sum_{i=1}^{c} p(i\|t)(1 - p(i\|t)) = 1 - \sum_{i=1}^{c} p(i\|t)^2$
Entropy	$I_H(t) = -\sum_{i=1}^{c} p(i\|t) \log_2 p(i\|t)$
Misclassification	$I_M(t) = 1 - \max(p(i\|t))$

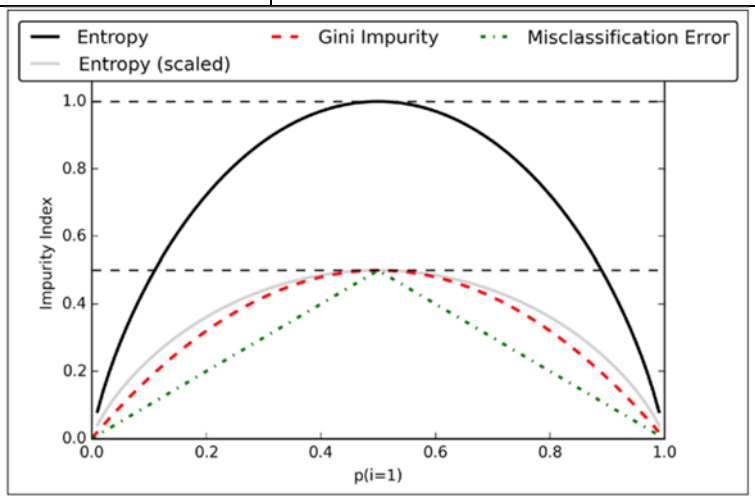


圖 3-1　二元分類時三種 Impurity index 所測量的值[26]

舉例來說，今有兩種分類法 A 與 B，分別將 40 個「1」類與 40 個「2」類的資料進行分類的結果如圖 3-2，再分別以 Gini impurity 與 Entropy 的方式計算 IG 得到如表 3-2 的結果。可觀察到兩種定義所算出來的 information gain (IG) 都以 B 分類法較大，因此 B 分類法屬於較好的分類法。

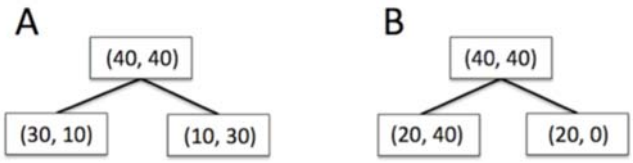


圖 3-2　舉例：二元分類結果[26]

Gini Impurity	Entropy
$I_G(D_p)=1-(0.5^2+0.5^2)=0.5$	$I_H(D_p)=-(0.5\log_2(0.5)+0.5\log_2(0.5))=1$
$A: I_G(D_{left})=1-\left(\left(\frac{3}{4}\right)^2+\left(\frac{1}{4}\right)^2\right)=\frac{3}{8}=0.375$	$A: I_H(D_{left})=-\left(\frac{3}{4}\log_2\left(\frac{3}{4}\right)+\frac{1}{4}\log_2\left(\frac{1}{4}\right)\right)=0.81$
$A: I_G(D_{right})=1-\left(\left(\frac{1}{4}\right)^2+\left(\frac{3}{4}\right)^2\right)=\frac{3}{8}=0.375$	$A: I_H(D_{right})=-\left(\frac{1}{4}\log_2\left(\frac{1}{4}\right)+\frac{3}{4}\log_2\left(\frac{3}{4}\right)\right)=0.81$
$A: I_G=0.5-\frac{4}{8}0.375-\frac{4}{8}0.375=0.125$	$A: IG_H=1-\frac{4}{8}0.81-\frac{4}{8}0.81=0.19$
$B: I_G(D_{left})=1-\left(\left(\frac{2}{6}\right)^2+\left(\frac{4}{6}\right)^2\right)=\frac{4}{9}=0.\overline{4}$	$B: I_H(D_{left})=-\left(\frac{2}{6}\log_2+\left(\frac{2}{6}\right)+\frac{4}{6}\log_2+\left(\frac{4}{6}\right)\right)=0.92$
$B: I_G(D_{right})=1-(1^2+0^2)=0$	$B: I_H(D_{right})=0$
$B: IG_G=0.5-\frac{6}{8}0.\overline{4}-0=0.1\overline{6}$	$B: IG_H=1-\frac{6}{8}0.92-0=0.31$

表 3-2　舉例：將二元分類結果以不同 impurity 定義計算[26]

另一方面，若為 regression 的情況，則 impurity 可直接定義為該集合對於集合平均的 mean square error (MSE) 或 mean absolute error (MAE) 如 (式 3-3)。

$$\boldsymbol{I_C = \frac{1}{N_t}\sum_{x\in D_t}(x-\bar{x})^2}$$

$$I_C = \frac{1}{N_t}\sum_{x\in D_t}|x-\bar{x}|$$

式 3-3　Impurity 於 continuous 情況下的定義

DT 的建立演算法有許多種，但大致上都是如下所述的脈絡。首先（1）演算法會先對 training data 切割出 training set 與 validation set。（2）使用 training set 去找出最大的樹狀結構：以 recursive 的方式持續進行分類，當無法再產生新的分類（最大 IG 為 0 時），或是都被分在同一類的時候即停止。（3）以 validation set 進行樹修剪（Tree pruning）的過程來避免 overfitting，這時會使每個 node 去紀錄分類時的 IG，若一些 node 的 IG 太小代表其分類的意義不大，則會被修剪掉。（4）

Cross-validation：依此概念將原始資料集隨機切成數種不同 training set 與 validation set 的組合，對各個組合重複 1 至 3 步驟後，可建立出數個不同的 DT，從中選出這些 DT 在各自 validation set 上擁有最小 misclassification rate 的 tree，便完成了 DT 的過程。[27]

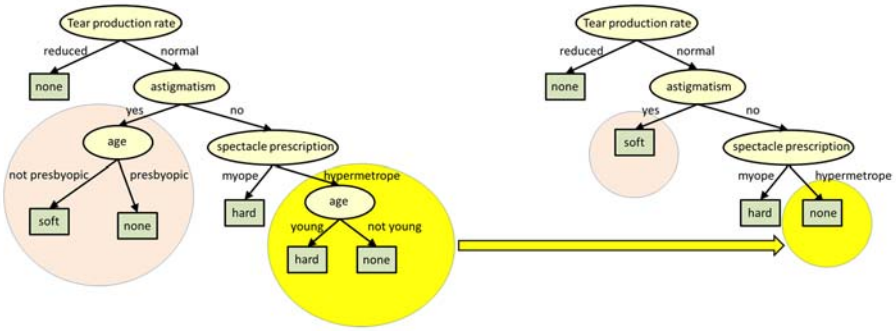


圖 3-3　Tree pruning 過程[27]

Chapter 4 Ensemble Learning

傳統機器學習的方法為搜尋（Search），即透過搜尋所有可能函數構成的空間集合，來找出最能夠逼近未知函數$f(.)$的近似函數$g(.)$，即我們常稱的假說（hypothesis），因此常需要使用特定的 metrics 作為衡量預測精準度的標準，並以與問題相關的知識對兩函數做一致性（consistence）檢定。

Ensemble learning 則取代傳統機器學習產生單一 hypothesis 的模式，改採多個假說組合而成的整體假說，也就是綜合多個 models 的結果。舉例而言，在 discrete output case 下採多個預測模型投票 (voting) 的方式作為最後輸出；在 continuous output case 下採多個 output 線性組合的方式整合。

Ensemble 之所以有效的原因有數種：多半認為 model 往往在過於龐大的 hypothesis space 進行 searching，過多的 training data 可能導致有多個表現相當良好的 hypothesis 產生，但最後仍然只能冒著資料稍有偏頗的風險在這之中選擇其中一個 hypothesis。也有人認為，當 hypothesis space 不包含$f(.)$時，則永遠都無法挑選出足夠良好的$g(.)$，此時透過 Ensemble 的 linear combination 效應則可有效擴大$g(.)$所在函數空間，進而達到更好的表現。[28]

4.1 Bootstrap

指的是從 data set 隨機抽樣一些數量相同的子集的過程。此處抽樣後的 sample 是可以放回原本的 data population 以進行下一輪抽樣的（random sampling with replacement）。

Bootstrapping 後的 data subset 可作為衡量 model overfitting 程度的指標：由於 subsets 提供了更多樣化的 probability distribution，影響著整體資料的 mean 與

variance 等，在 overfitted model（如 decision tree 就是很容易 overfit 的模型）上的表現上很容易就能看出問題，例如預測精準度大幅下降、Loss 上升等等。Bootstrapping 也能應用於各個 model 的訓練階段，如 Bagging 與 Boosting。

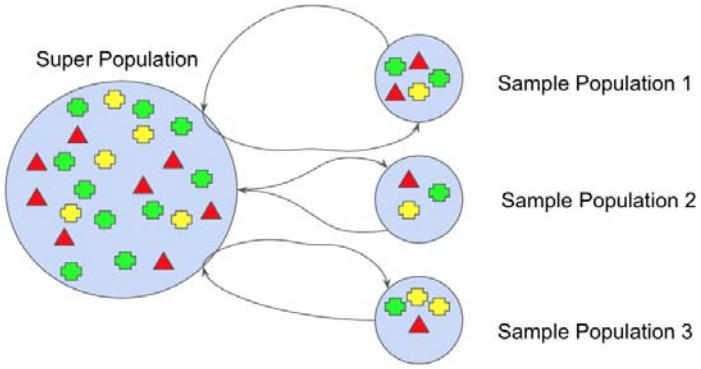


圖 4-1 Bootstrap 示意圖 [29]

4.2 Bagging and Random Forest

Bagging 為 Bootstrap aggregating 的代稱，首次出現於 Leo Breima [30] 著作中，強調的是以 Bootstrapping 處理 training dataset，並以此 data subsets 送到各個 sub-model 進行訓練、整合結果的過程，如圖 4-2。

直覺上觀察，Bootstrapped data 具有不同的 probability distribution，因此在將各個 model 的結果整合起來時，能夠將一些來自組內 mean、median 或 variance 等統計 noise 的成分消弭掉。透過數學證明 [31]，可觀察到 output variance 可透過 Bagging 的方式大幅下降，與此同時，並不會增加額外的 bias（因為通常 bias 與 variance 之間有 trade-off）。由於 error 幾乎從 variance 貢獻而來，Breiman 也提到 Bagging 特別適合用在數值較為 unstable 的 model 上（即 input 偏差會導致 output 的巨大變化）。

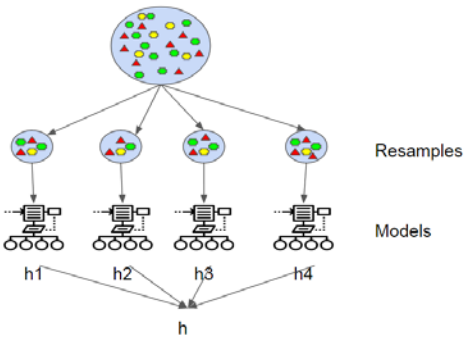


圖 4-2　Bagging 架構示意圖[29]

$$\hat{g}_{\mathcal{L}}^{A}(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{M} g_{\mathcal{L}_i}(\mathbf{x}).$$

式 4-1　將各個 regressor 整合起來的表示法

$$\hat{g}_{\mathcal{L}}^{A}(\mathbf{x}) = \arg\max_{j} \sum_{i=1}^{M} I[g_{\mathcal{L}_i}(\mathbf{x}) = j]$$

式 4-2　將各個 classifier 整合起來的表示法

Random Forest 的概念從 Bagging 而來，一般相信 Decision Tree 容易 Overfit，因此通常會限制 Tree 的層數來避免之，在這種情況下，Random forest 就是建立許多 subtree model，各個 subtree 具有一定的隨機性，如以抽樣方式選擇訓練資料或是特定選用幾個 feature 的組合來做訓練，將結果以式 4-1 或式 4-2 的方式結合再一起後就是 random forest model 的輸出，一般來說效果會比單棵 tree 的表現來的佳。

4.3 Boosting

Boosting 與 Bagging 主要不同處在於抽樣階段上為 random sampling without replacement（代表抽樣後的樣本不能放回母體），但在訓練的 evaluation 階段時預測錯誤的 sample 將會抽樣一部分加入訓練下一個 sub-model 的 training dataset 中。

也因此，不同於 Bagging 能夠平行化處理，Boosting 必須在訓練資料的準備階段上參考 sub-model 的 outcome 而有所變化。

在 ensemble k 個 sub-model 的情況下，其 training pipeline 如表 4-1

* Initialization: set up a null set for misclassified data, $Mis_0 = \{\}$ * For $i = 1, 2, \ldots, k$, do: (1) Draw a sub-dataset D_i without replacement from original dataset D (2) Add 50% of elements in Mis_{i-1} to D_i (3) Use D_i to train a sub-model c_i (4) Evaluation by sending D to c_i, set this misclassified sample set as Mis_i.

表 4-1　Boosting 演算法

若訓練對象為 classifier，Infer 階段時便是從 k 個 sub-model 的預測中取 max voting。一般可觀察到初期的 sub-model 表現不會太好，但 sub-model 會隨著 misclassified data 的加入而逐漸變強。Breiman 在 1996 的論文提到這種方法也可在降低 variance 的同時降低 bias。

4.4 Adaboost

Adaboost 在 1997 由 Freund 與 Schapire 等人所提出。與 Bagging 不同之在於這裡不對 dataset 做 Bootstraping，每個 sub-model 都必須接受所有 dataset 的訓練。其 adaptive 之處在於：各個 iteration 中將動態地改變賦予 data 的 weight。

對普通的 classification case 而言，objective function 可表示如式 4-3，其中1(.)唯有在括號內條件成立時輸出為 1，否則為 0。但 Adaboost 卻會表示如式 4-4，目的是為了強調 misclassified sample 的影響，即上一個 iteration 中被 misclassified 的

sample 將會在下個 iteration 中得到較大的 weight。如此一來，在進行 gradient descent 時就會有較多來自該 sample error 的梯度組成。

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N} 1(g(x_i) \neq y_i)$$

式 4-3　objective function for a classifier

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N} w(x_i)1(g(x_i) \neq y_i)$$

式 4-4　weighted objective function for a classifier

Adaboost 的精神與 Boosting 相同，其目的脈絡如下：在前一次 iteration 中訓練好一個 sub-model 後，希望產生一組會使得該 sub-model 的ε提高至 0.5 的 weight 組合，如此一來，在使得前一個 sub-model fail 掉的情況下，去訓練下一個 sub-model，應可以有更好的表現（代表降低 correctly classified sample 的 weight，用以提升 misclassified sample 的 weight）。透過數學證明[31]，weight 的更新模式表示如式 4-5，其中t代表 iteration，ε_t表示第t次迭代時的整體誤差，此時若模型設計妥當可使ε必小於 0.5 時，$\sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}$ 必定能夠大於 1，也因此能夠針對錯誤的 sample 加強其 loss weight。

$$w_{t+1}(x_i) \leftarrow \begin{cases} w_t(x_i)\sqrt{\dfrac{1-\varepsilon_t}{\varepsilon_t}}, when\ g(x_i) \neq y_i \\ w_t(x_i)\sqrt{\dfrac{\varepsilon_t}{1-\varepsilon_t}}, when\ g(x_i) = y_i \end{cases}$$

式 4-5　Adaboost 之 weight 更新關係

考慮二元分類問題，$x \epsilon \mathcal{X}$，$y\epsilon \mathcal{Y} = \{-1, +1\}$，$g: \mathcal{X} \to \mathcal{Y}$。Ensemble k 個 sub-model 的情況下，其 training pipeline 如表 4-2，其中a_t為一種根據ε_t對 weight 進行調整的函數，正確率較高的 sub-model 將會有較高的a_t，反之正確率低的其a_t則低。最後在 inference 時的預測則如式 4-6 所示。

* Initialization (1) For all data x_n: $w_1(x_n) = 1$ * For $t = 1, \dots, T$, do: (2) Select a hypothesis $g_t(.)$ (i.e., some classification model) that minimize the objective function $$\varepsilon_t = \frac{1}{N}\sum_{i=1}^{N} w_t(x_i)1(g_t(x_i) \neq y_i)$$ (3) Let $a_t = \frac{1}{2} ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$, for each data x_n, update the weight $$w_{t+1}(x_n) \leftarrow w_t(x_n) exp\left(-a_t y_n g_t(x_n)\right)$$ (Note that when the prediction is error, then $y_n g_t(x_n)$ = -1).

表 4-2　Adaboost 演算法

$$h(x) = sign(\sum_{t=1}^{T} a_t g_t(x))$$

式 4-6　Adaboost 之輸出結果

a_t 對ε_t的函數上其實就是 sigmoid 函數($y = \frac{1}{1+e^{-t}}$)順時針轉 90 度的版本如圖 4-3：當$\varepsilon < 0.5$時(在 first iteration 中，可理解為超過半數的 sample 被正確地 classified)，a_t會相當快速地朝正值成長。在$\varepsilon = 0.5$的 special case 中則會認為這次 iteration 產生的 sub-model 無法對預測做出貢獻，因此$a_t = 0$，將不會對 weight 做任何調整，也不會在 infer 階段做出貢獻。

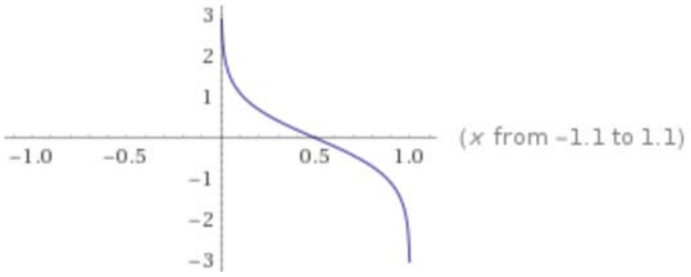


圖 4-3　ε_t和a_t之間的關係[33]

為說明 Adaboost 的原理，此處介紹了ε的 upper bound 與 convergence 的推導，詳細說明可參考 [48]。首先可定義Z_t以代表著第t個 iteration 的 weight summation，其具有如式 4-7 之關係，其中N為 data 數量。此時回顧表 4-2 中對於 weight 的迭代關係，將之拆解為分類正確與分類錯誤的項之後，與式 4-7 對應，可得到Z_t的迭代關係如式 4-8，其中$Z_t\varepsilon_t$代表分類錯誤項的 weight summation，$Z_t(1-\varepsilon_t)$則是分類正確項的 weight summation，又將a_t本身的定義帶入，則Z_{t+1}可簡化為$2Z_t\sqrt{\varepsilon_t(1-\varepsilon_t)}$，且因$\varepsilon_t$必小於 0.5，由此可證明$Z_{t+1}$必會隨著迭代數的增加而減少。

$$Z_1 = N, \qquad Z_{t+1} = \sum_{i=1}^{N} w_t(x_n) exp\big(-a_t y_n g_t(x_n)\big)$$

式 4-7　Z_t之定義

$$Z_{t+1} = Z_t\varepsilon_t exp(a_t) + Z_t(1-\varepsilon_t)exp(-a_t) = 2Z_t\sqrt{\varepsilon_t(1-\varepsilon_t)}$$

式 4-8　Z_t之迭代關係

另一方面，為了推導 upper bound，首先可關注在ε本身的定義上，也就是$\varepsilon = \frac{1}{N}\sum_{n=1}^{N} 1(y_n g(x_n) < 0)$，其實是受到 $\frac{1}{N}\sum_{n=1}^{N} exp\big(-y_n g(x_n)\big)$ 作為 upper bound 所限制的，即

$$\frac{1}{N}\sum_{n=1}^{N} 1(y_n g(x_n) < 0) \quad < \quad \frac{1}{N}\sum_{n=1}^{N} exp\big(-y_n g(x_n)\big)$$

再者，可關注在Z_T也就是收斂時所達到的 weight summation 的值。此時可從式 4-7 當中關於Z_t的定義中的$w_t(x_n)$發現，$w_t(x_n)$其實具有如表 4-2 的迭代關係，因此可將Z_{T+1}表示為一種連乘的形式如式 4-9，進而簡化記號為$\sum_{n=1}^{N} exp\left(-y_n g(x_n)\right)$。此時可發現$\frac{1}{N} Z_{T+1}$恰巧就是本段開頭所述$\varepsilon$的 upper bound，那麼藉由式 4-8 的迭代關係便可將Z_{T+1}改寫為與ε有關的形式，便可得式 4-10 之關係。又因ε_t必小於 0.5，由此可發現ε可隨 upper bound 隨迭代逐漸變小而逐漸變小，由此證明 Adaboost 的收斂。

$$Z_{T+1} = \sum_{n=1}^{N} \prod_{t=1}^{T} exp\left(-a_t y_n g_t(x_n)\right) = \sum_{n=1}^{N} exp\left(-y_n \sum_{t=1}^{T} a_t g_t(x_n)\right) = \sum_{n=1}^{N} exp\left(-y_n g(x_n)\right)$$

式 4-9　Z_{T+1}的推導

$$\varepsilon = \frac{1}{N} \sum_{n=1}^{N} 1(y_n g(x_n) < 0) < \frac{1}{N} Z_{T+1} = \prod_{t=1}^{T} 2 \frac{1}{N} N \sqrt{\varepsilon_t (1 - \varepsilon_t)}$$

式 4-10　ε之 upper bound

4.5 Gradient Boosting

先前介紹的 boosting 都有一項特點，便是在進行輸出時以線性組合或者 max voting 的方式將各個 sub-model 的結果整合出來，例如 $g(x) = \sum_{t=1}^{T} g_t(x_n)$（為求簡化，此處設 weight 皆為 1）。Gradient boosting 的概念則不然，它要求的是讓第 *t* 個 sub-model 的輸出值為「前 *t*-1 個 sub-model 集合而成的 output」與「true value」的 residual。這件事其實也很有道理，因為許多細緻的資料分布並不是一蹴而就的，往往是有 coarse 的特徵加上 fine 的特徵做修飾後才會得到最後的結果 [35][36]。

令有一個 base model 的預測值為 $g_1(x)$，則 residual 為 $y - g_1(x)$。此時 loss 與 gradient 如式 4-12 所示，則可以 gradient descent 的方式確保式 4-12 的事項。

$$L(y, g_1(x)) = \frac{1}{2}(y - g_1(x))^2$$

$$\frac{\partial L(y, g_1(x))}{\partial g_1(x)} = -(y - g_1(x))$$

式 4-11 GB 目標函數之目標函數與梯度

$$L(y, g_1(x)) \geq L\left(y, g_1(x) - \alpha \frac{\partial L(y, g_1(x))}{\partial g_1(x)}\right) = L\left(y, g_1(x) + \alpha(y - g_1(x))\right)$$

式 4-12 GB 目標函數之確保

當以 boosting 模式組合第二個 sub-model 時，output 更新為$g_1(x) + g_2(x)$，其中$g_2(x)$為第二個 sub-model 的 output，必須盡可能地近似前一個 iteration 的 residual：$g_2(x) \approx y - g_1(x) = -\alpha \frac{\partial L(y, g_1(x))}{\partial g_1(x)}$。不難注意到第二個 sub-model 的近似目標正好就是前一個 iteration 中 loss 對 overall model 的 negative gradient。

傳統機器學習是採 gradient descent 方式將「loss」對「sub-model 內部參數」偏微來進行優化，其實 GB 的每一個 sub-model 的 fitting 過程皆屬之。除此之外，GB 在各個 iteration 中將「loss」對「current overall model」偏微，採這種 negative gradient 作為 sub-model 的 fitting 目標，其實就是在引導 overall model 在更廣層級上朝 local minimum 前進。其 pipeline 如下

* Initialization
Set up a base learner $g_0(x) = 0$, set an overall model $F(x) = g_0(x)$
* For $t = 1, \ldots, T$, do:
Calculate former iteration's negative gradient: $-\frac{\partial L(y, F(x))}{\partial F(x)}$
Fit a model to negative gradient: $g_t(x) = -\frac{\partial L(y, F(x))}{\partial F(x)}$
Update overall model: $F(x) \coloneqq F(x) + \rho g_t(x)$
* Infer stage
$F(x)$ is the final overall model.

表 4-3 Gradient boosting 演算法

GB 所使用的 objective function 並非只適用於 square loss。舉例來說，在 absolute loss 的情況下，其 objective function 與 gradient 如式 4-13，Huber loss 則如式 4-14。這兩種 objective function 相較於 square loss 較不受 outlier 的影響，依情況可以在 GB 時採用此類 objective function。因此，negative gradient 並非往往都等於 residual。

$$L(y, F(x)) = |y - F(x)|$$
$$\frac{\partial L(y, F(x))}{\partial (F(x))} = sign(y - F(x))$$

式 4-13 Absolute loss 之 objective function 與 gradient

$$L(y, F(x)) = \begin{cases} \frac{1}{2}(y - F(x))^2, |y - F(x)| \le \gamma \\ \gamma\left(|y - F(x)| - \frac{1}{2}\gamma\right), |y - F(x)| > \gamma \end{cases}$$
$$\frac{\partial L(y, F(x))}{\partial (F(x))} = \begin{cases} y - F(x), |y - F(x)| \le \gamma \\ \gamma * sign(y - F(x)), \quad |y - F(x)| > \gamma \end{cases}$$

式 4-14 Huber loss 之 objective function 與 gradient

4.6 Xgboost (Gradient Boosting Decision Tree)

Xgboost 名稱由 Extreme(X) Gradient Boosting 而來，屬於基於 Gradient Boost Decision Tree 的進化版本（GBDT 即是以 GB 方法組合的 DT，為了避免 overfit，各個 sub-tree 的深度往往不深）。

Xgboost 同樣延續 boosting 精神，以 recursive 方式決定該 iteration 的 objective function。由於 Xgboost 在 objective function 中加入了限制 tree 複雜程度的項次，此時必須先提及關於 tree 的參數：令一個有 T 個 leaf nodes 的 tree，如式 4-15，其中$q(.)$為一種可將 feature domain 投射到 leaf domain 的函數，每個 leaf 則對應著 w_k代表第k個 leaf 所代表的 output，$f_t(x_i)$代表 prediction 修正量。

$$f_t(x) = w_{q(x)}$$
$$where\ q(x): X \to T$$
$$where\ w_k, k = 1, \dots, T$$

式 4-15 tree 的表示法

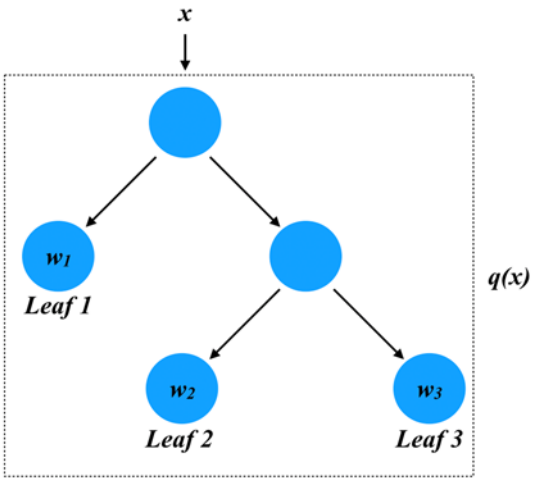


圖 4-4 Tree 結構說明

由於 tree 屬於容易 overfitting 的結構，因此需要對 leaf nodes 的數量T以及各個 leaf 的 output w_k做限制如式 4-16 所示。因此，可想像 Xgboost 是一種以 gradient boosting 與對 tree 的結構做限制的架構下去進行預測。在「希望第 t 棵樹能夠接近前一次 iteration 預測值與真實值的 residual 的目的」下，可以將 objective function 寫作如式 4-17 所示，其中 $l(.,.)$代表任一種常見的 loss function 如 square loss 或 absolute loss，$\hat{y_i}$代表 ground truth，$y_i^{(t-1)}$代表前一個 iteration 的 prediction，$f_t(x_i)$代表第t個 iteration 的 prediction 修正量。

$$\Omega(f_t) = \gamma T + \lambda \frac{1}{2} \sum_{k=1}^{T} w_k^2$$

式 4-16 Xgboost 之對於 tree 結構的限制

$$L_t = \left[\sum_{i=1}^{N} l\left(\hat{y_i}, y_i^{(t-1)} + f_t(x_i)\right) \right] + \Omega(f_t)$$

式 4-17 Xgboost 之目標函數

為了求出第 t 個 iteration 的 optimal solution w^*，必須先以 Taylor expansion 拆解$l(.,.)$，並令$g_i = \frac{\partial l\left(\hat{y_i}, y_i^{(t-1)}\right)}{\partial y_i^{(t-1)}}$，$h_i = \frac{\partial^2 l\left(\hat{y_i}, y_i^{(t-1)}\right)}{\partial \left(y_i^{(t-1)}\right)^2}$時可將 objective function 寫如式 4-18。並且因為$l\left(\hat{y_i}, y_i^{(t-1)}\right)$在第$t-1$個 iteration 時已被決定而可捨去，可整理如式 4-19。

$$L_t \cong \left[\sum_{i=1}^{N} l\left(\hat{y_i}, y_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t)$$

式 4-18 Xgboost 之 optimal solution 推導過程（一）

$$\widehat{L_t} \cong \left[\sum_{i=1}^{N} g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t)$$

式 4-19 Xgboost 之 optimal solution 推導過程（二）

接著拆解f_t，並定義$I_j = \{i|q(x_i) = j\}$代表落在第j個 leaf 的x集合(當j互異時，各I_j彼此互斥），則得到式 4-20。此時再令$G_j = \left(\sum_{i \in I_j} g_i\right)$，$H_j = \left(\sum_{i \in I_j} h_i\right)$，則可得到較為簡潔的式子如式 4-21。

$$\widehat{L_t} \cong \left[\sum_{i=1}^{N} g_i w_q(x_i) + \frac{1}{2} h_i w_q^2(x_i)\right] + \gamma T + \lambda \frac{1}{2} \sum_k w_k^2$$

$$= \left[\sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i\right) w_j + \frac{1}{2}\left(\sum_{i \in I_j} h_i + \lambda\right) w_j^2\right]\right] + \gamma T$$

where $w_j = w_q\left(x_i\right)\Big|_{i \in I_j}$.

式 4-20 Xgboost 之 optimal solution 推導過程（三）

$$\widehat{L_t} \cong \left[\sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}\left(H_j + \lambda\right) w_j^2\right]\right] + \gamma T$$

式 4-21 Xgboost 之 optimal solution 推導過程（四）

使$\hat{L}_t$對w_j求偏導數後，便能得到 optimal solution 與 optimal loss 如式 4-22。這個 objective value 可以代表一個 tree 的 structure score，與其他衡量方法相同，structure score 越小代表此 tree 的結構越佳。

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$\hat{L}_t^* = -\frac{1}{2}\left[\sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda}\right] + \gamma T$$

式 4-22 Xgboost 之 optimal solution

回顧到 Information gain 的概念，通常我們希望在每個 leaf notes 的 iteration 中，選擇一個能讓 IG 最大的分割方式。先定義I為目標 leaf 以下的 instance set，I_L為分出 node 之後左枝的 instance set，I_R則為右枝。此時的 IG 與$G_.$及$H_.$的關係則如式 4-23 表示，由於窮舉樹的結構是不可能的，所以許多演算法都採 greedy 的做法，即在一群 instance 之中排列組合，對於每個組合計算一次IG，最後採IG最大的組合作為 split 的方式。

$$IG = L(I) - (L(I_L) + L(I_R))$$

$$= \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G_I^2}{H_I + \lambda}\right] - \gamma$$

式 4-23 IG 與樹結構之關係

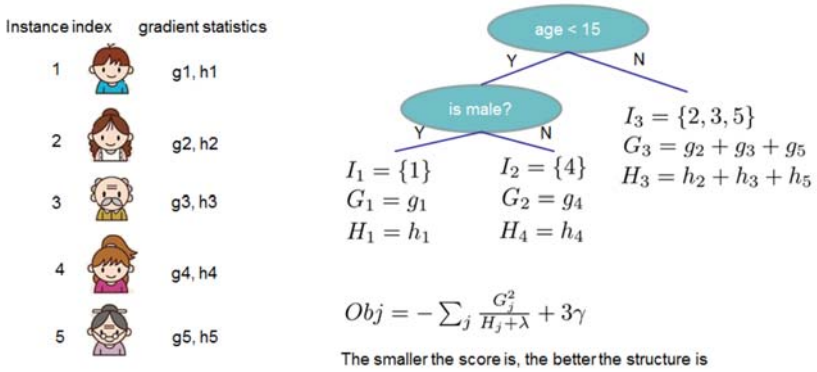


圖 4-5 Xgboost 之 objective function 計算方式[37]

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

表 4-4　Xgboost 演算法[37]

Chapter 5 Alternative Models

5.1 Radial Basis Function Network (RBFN)

Radial Basis Function Network 是以 Radial basis function 作為神經元基底的神經網路，於 1988 年由 D.S. Broomhead 與 David Lowe 所提出。其名稱具有兩種用意：Radial function 表示一種以 Euclidean distance 作為變數的函數，Basis 則代表將這種 Function 作為 Network 的 Kernel。[38]

Radial Function 其基本定義為計算某點$\vec{x}$與原點的 Euclidean distance，考慮到作為比較的基準點並非原點，而可以是在 Euclidean space 上任一點$\vec{c}$作為圓心時，則寫作式 5-1。

$$\varphi(\vec{x}) = \|\vec{x} - \vec{c}\|_2^2$$

式 5-1 Radial function

Gaussian function	$\phi(r) = e^{-(\varepsilon r)^2}$
Multi-quadratic	$\phi(r) = \sqrt{1 + (\varepsilon r)^2}$
Inverse multi-quadratic	$\phi(r) = \frac{1}{\sqrt{1 + (\varepsilon r)^2}}$
Inverse quadratic	$\phi(r) = \frac{1}{1 + (\varepsilon r)^2}$
Poly-harmonic spline	$\phi(r) = \begin{cases} r^k, & k = 1,3,5 \dots \\ r^k \ln(r), & k = 2,4,6 \dots \end{cases}$
Thin plate spline	$\phi(r) = r^2 \ln(r)$

表 5-1 Radial function 類型的活化函數

將 RBF network 與 Neural network 做比較。可以觀察到最主要的不同在於 Hidden layer 上的處理方式。NN 主要先計算 Weight 與 Input 的 Inner product，再通過一種 Sigmoid 形的 Activation function 得到 Output。RBFN 則是計算 Input 與 Center 的距離，再通過 Gaussian 形的輸出函數得到 Output。

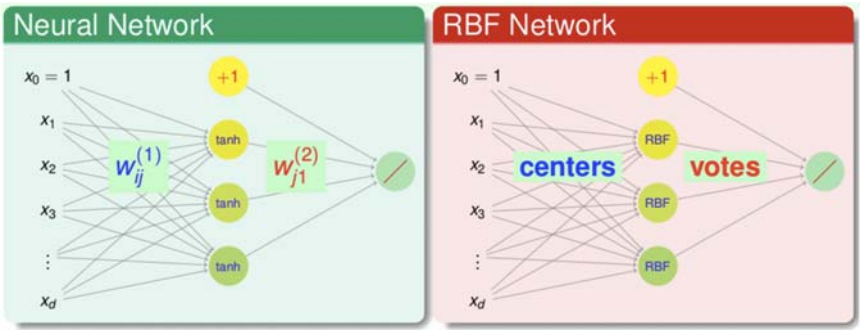


圖 5-1　NN 與 RBFN 的架構比較 [50]

通常為三層結構：Input 層、Hidden 層及 Output 層，假設 input $\vec{x} \in \mathbb{R}^N$，output $y \in \mathbb{R}$，則 RBFN 可視為$\mathbb{R}^N \to \mathbb{R}$的函數，表示為式 5-2，其中$M$為 hidden layer 中 basis 的數目，$\overrightarrow{C_{\mathrm{m}}}$與$w_m$代表第$m$個 basis 的中心與權重。

$$y = f(\vec{x}) = \sum_{m=1}^{M} w_m \, \phi\left(\left\|\vec{x} - \overrightarrow{C_m}\right\|_2^2\right)$$

式 5-2　RBFN 之 output

今假設使用了最常用的 Gaussian function 作為活化函數，則 kernel 可以寫作式 5-3 的形式，比較真實資料集$(\vec{x}, \hat{y})$與預測資料集$(\vec{x}, y)$時，其 objective function 可以定義為式 5-4，其中 k 為 iteration，j為 training data index。

$$\phi\left(\left\|\vec{x} - \overrightarrow{C_m}\right\|_2^2\right) = \exp\left(-\frac{\left\|\vec{x} - \overrightarrow{C_m}\right\|_2^2}{2\sigma_m^2}\right)$$

式 5-3　RBFN 之 radial function

$$\mathrm{e(k)} = \frac{1}{2}(\hat{y}_j - y_j)^2$$

式 5-4　RBFN 之目標函數

則為了求解直接對式 5-4 進行偏微即可，如式 5-5。進而可以得到 RBFN 在下個 iteration 中更新後的參數如式 5-6，其中，k 為迭代數，$\eta_i, i = 1,2,3$為各個參

數的學習率。這種架構與類神經網路相似，但類神經網路只有調整 w_m, 這裡則是也對 $\vec{C}_m$ 與 σ_m 做調整，於是訓練 RBFN 的架構整理如。

$$\frac{\partial E(\mathrm{k})}{\partial w_m}=0,\frac{\partial E(\mathrm{k})}{\partial \overrightarrow{C_m}}=0,\frac{\partial E(\mathrm{k})}{\partial \sigma_m}=0$$

式 5-5　RBFN 目標函數之偏導數

$$w_m(k+1)=w_m(k)+\eta_1 e(k)\phi\left(\left\|\vec{x}-\overrightarrow{C_m}\right\|_2^2\right)$$

$$\overrightarrow{C_m}(k+1)=\overrightarrow{C_m}(k)+\eta_2\frac{w_m e(k)}{\sigma_m^2}\phi\left(\left\|\vec{x}-\overrightarrow{C_m}\right\|_2^2\right)\left(\vec{x}-\overrightarrow{C_m}\right)$$

$$\sigma_m(\mathrm{k}+1)=\sigma_m(k)+\eta_3\frac{w_m e(k)}{\sigma_m^3}\phi\left(\left\|\vec{x}-\overrightarrow{C_m}\right\|_2^2\right)\left\|\vec{x}-\overrightarrow{C_m}\right\|_2^2$$

式 5-6　RBFN 參數之迭代關係

* Initialization Set hyperparameters: the number of basis M. Initialization of adjustable parameters: $w_m, \overrightarrow{C_m}$, m = 1, …, M * K-means: to find initial cluster centers $\overrightarrow{C_m}$ in an unsupervised way. * Update parameters $w_m, \overrightarrow{C_m}, \sigma_m$ according to the gradient.

表 5-2　RBFN 訓練過程

此處補充關於 *K*-means 的說明：*K*-means 屬於 unsupervised clustering 方法，常被用以決定適合的分類中心，其 objective function 被定義為去最小化 within cluster sum of square 如式 5-7，其中 k 為迭代數，M 為中心數目，N 為資料量，J_i表示第 i 個中心對所有資料點的 cost，$\overrightarrow{x_j}$為第j個資料，$\overrightarrow{C_i}$為第i個中心。w_{ij}代表第j個資料對於第i個中心所產生的權重（物理意義上代表在所有 M 個中心當中，$\overrightarrow{C_i}$屬於距離 $\overrightarrow{x_j}$最近的中心時就將其分類進去），其定義與限制如式 5-8。由此，K-means 將會

在本次 iteration 中決定新的 cluster center 如式 5-9，代表以被分類在此中心的資料的質心作為新的重心。

$$\mathrm{J(k)} = \sum_{i=1}^{M} J_i = \sum_{i=1}^{M}\sum_{j=1}^{N} w_{ij}\left\|\overrightarrow{x_j} - \overrightarrow{C_i}\right\|_2^2$$

式 5-7　K-means 之目標函數

$$w_{ij} = \begin{cases} 1, if\left\|\overrightarrow{x_j} - \overrightarrow{C_i}\right\|_2^2 < \left\|\overrightarrow{x_j} - \overrightarrow{C_m}\right\|_2^2, \forall m \neq i \\ 0, otherwise \end{cases}$$

$$\sum_{i=1}^{M} w_{ij} = 1 \quad \forall j = 1, \ldots, N; \sum_{i=1}^{M}\sum_{j=1}^{N} w_{ij} = N$$

式 5-8　K-mean 之權重定義

$$\overrightarrow{C_i} = \frac{\sum_{j=1}^{n} w_{ij}\,\overrightarrow{x_j}}{\sum_{j=1}^{n} w_{ij}}$$

式 5-9　K-means 之 cluster center

5.2　Context Relevant Self Organizing Map

由 Hartono(2015)提出，建立在 Self Organizing Map (SOM) 概念上，並結合 RBF 作為輸出的架構。能夠進行具有 Supervised Learning 特性的 Dimensionality Reduction。

機器學習通常具有兩個步驟：DR (Dimensionality Reduction) 及 Classification。常見的神經網路 (NN) 模型則不會特別做區分。DR 屬於一種特徵抽取的概念 (Feature Extraction)，主要用途為(1) 排除 independent noise (2) 去除過多的自由度，以幫助 classification 能夠更容易，通常可從是否有 Supervised 的性質做區分。Unsupervised 的範疇包含了 PCA、LDA、LLE (Locally Linear Embedding)、NCA (Neighboring Component Analysis)，以及基於統計特性的方法如 SNE、t-SNE。

supervised 則有 MRE(Multiple Relational Embedding)、SNeRV(Supervised Neighbor Retrieval Visualizer)、S-Isomap 等。

對非 NN 的模型架構來說，在訓練時通常會面臨如何做 DR（Feature selection）及 Classification 的問題，通常可分為 Filter、Wrapper 及 Embedded 法。

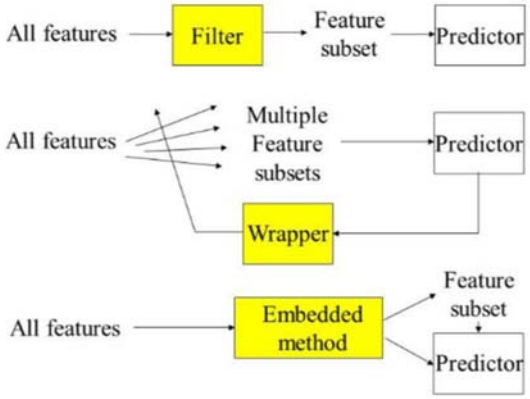


圖 5-2　Feature selection 之三種方法[42]

Filter 法僅採 Feature 本身的特性作為 DR 的選擇，待適合的 mapping 找到後，就將其固定，再進行後續的 classification，屬於最常使用的方法。Wrapper 法則是會以 model 最後的輸出作為 DR mapping 的評斷準則，通常表現會較 filter 法好，但問題將會變得 non-convex，且多餘的參數會使得計算量大增(有人提出一種加入輔助變數的演算法，能夠降低複雜度[40])。Embedded 法則是在 objective function 中同步進行 feature selection 與 classification，常見的方式便是 Lasso（L_1 norm）與 Ridge regression（L_2 norm）如圖 5-3，其中$\Omega(.)$雖然是對 weight 做限制，但卻可以達到 feature selection 的效果。

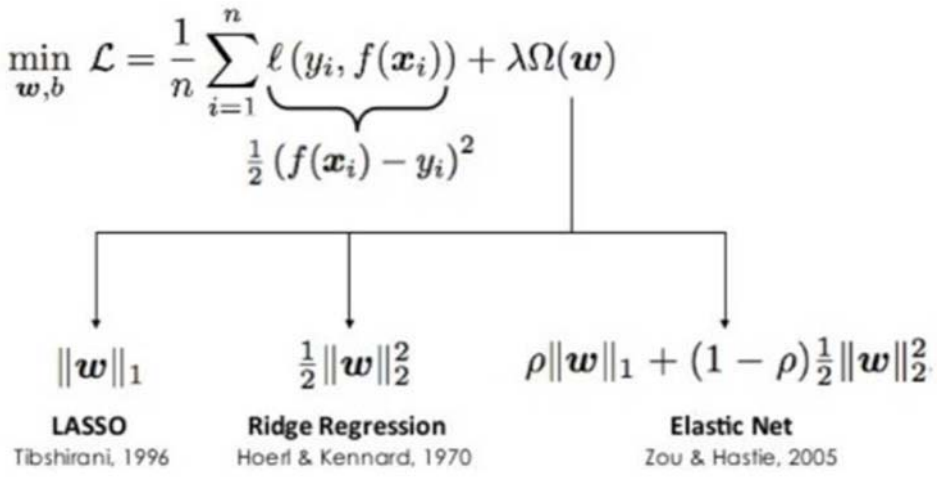


圖 5-3 Embedded 的方法[42]

為了說明 Lasso 與 Ridge regression 的效果，可以簡單的圖形如圖 5-4 說明之，考慮由兩種 weight β_1與β_2為軸建立的 loss function 平面與 contour，其中$\hat{\beta} = \{\widehat{\beta_1}, \widehat{\beta_2}\}$代表抵達 minima 的 optimal weight，而藍色區域則分別為 Lasso(左)與 Ridge(右)對 weight 的限制(即β必須要落在區域內)。假設模型只是簡單的$y = \beta_1 x_1 + \beta_2 x_2$，則這種對 weight 的限制在一定程度上限制了 feature 對 output 的影響效果，這就是 embedded 法的精神所在，即利用 objective function 的設計同步進行 feature selection 與 classification。另外值得一提的是，一般認為 Lasso 的 feature selection 效果會較 Ridge regression 來得強，是因為在 weight 在順著 gradient 朝向 contour minima 前進的過程中，會比 Ridge regression 來得容易抵達邊角的位置，此時意味著某個軸（某個 feature）的貢獻度為 1，而另一個軸（某個 feature）的貢獻度為 0。

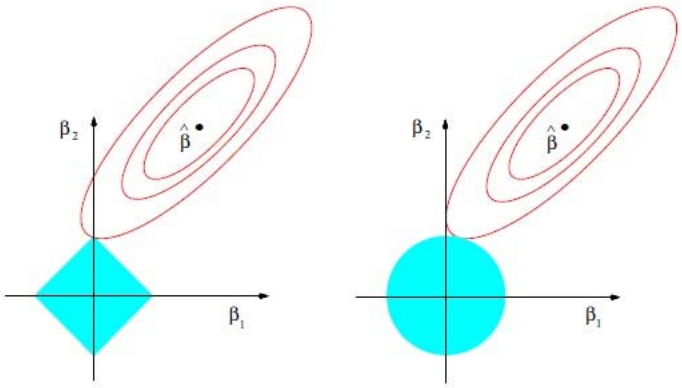


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \le t$ *and* $\beta_1^2 + \beta_2^2 \le t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

圖 5-4　Lasso 與 Ridge regression 比較 [51]

為了解模型之運作（增加模型透明度），通常會希望 DR 後的低維分佈能夠對資料的類別關係有一定程度的代表性，如此一來，就能夠透過資料的可視化幫助了解複雜資料的關係。因此，這種架構經過訓練後，能夠在 hidden layer 中獲得資料的輔助表示。

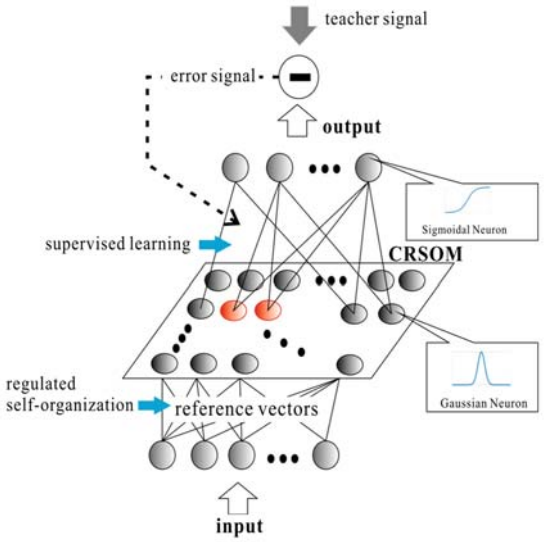


圖 5-5　CRSOM（Context Relevant Self Organizing Map）之架構[39]

按照 SOM 的典型架構，將 hidden layer 二維平面上的每個 neuron 都與 input 比較，得到差異值I_j作為 hidden layer 的 input 如式 5-10，其中$\mathrm{X}(t) \in \mathbb{R}^D$代表輸入資料，$W_j(t) \in \mathbb{R}^D$代表第$j$個 neuron 的 reference vector，$\alpha > 0$代表 scaling factor，t代表迭代數。則此時定義能使$I_j^{hid}(t)$最小的 neuron 為 winner neuron 如式 5-11

$$I_j^{hid}(t) = \frac{\left\| X(t) - W_j(t) \right\|_2}{\alpha}$$

式 5-10 hidden layer 之 input

$$j^*(t) = \arg\min_j I_j^{hid}(t)$$

式 5-11 決定 hidden layer 上的 winner neuron

則可以依照 winner neuron 在 hidden layer 上與其他 neuron 的相對位置得到高斯距離函數如式 5-12，其中h_j表示第j個 neuron 在 hidden layer 的座標，並以此函數作為調控該 hidden layer neuron 的 output 之權重如式 5-13。(距離 winner neuron 越遠的 neuron 權重越小，且 reference vector 與$X(t)$差越遠的 neuron 輸出值也越小)

$$\phi(j^*(t), j) = e^{-\frac{\left\| h_{j^*(t)} - h_j \right\|_2^2}{S(t)}}$$

式 5-12 hidden layer 之高斯距離函數, h_j 和 $h_{j*(t)}$ 越近則調越多

$$O_j^{hid}(t) = \phi(j^*(t), j) e^{-I_j^{hid}(t)}$$

式 5-13 Hidden layer 之 output

接著，output layer 將會對 hidden layer output 進行線性組合來作為 Output layer input 如式 5-14，其中$v_{jk}(t)$為 output layer 的第 k 個 neuron 與 hidden layer 的第 j 個 neuron 間的權重，$\theta_k(t)$為 output layer 的第 k 個 neuron 的 bias。則這些值也須通過一個 Sigmoid 函數活化方為最後的 output，如

式 5-15。Output layer 外具有真實值 (ground truth value) $T(t) \in \mathbb{R}^P$，則可將 output layer 的 output 集合成向量表示$O^{out}(t) = [O_1^{out}(t), O_2^{out}(t), \ldots, O_P^{out}(t)]$，計算兩者之間損失值$E(t) = \frac{1}{2}\|T(t) - O^{out}(t)\|_2^2$。如此一來，CRSOM 的 forward 步驟就完成了，架構整理如圖 5-6。

$$I_k^{out}(t) = \sum_j v_{jk}(t) O_j^{hid}(t) - \theta_k(t)$$

式 5-14 Output layer 之 input

$$O_k^{out}(t) = \varphi\left(I_k^{out}(t)\right), \qquad \varphi(x) = \frac{1}{1+e^{-x}}$$

式 5-15 Output layer 之 output

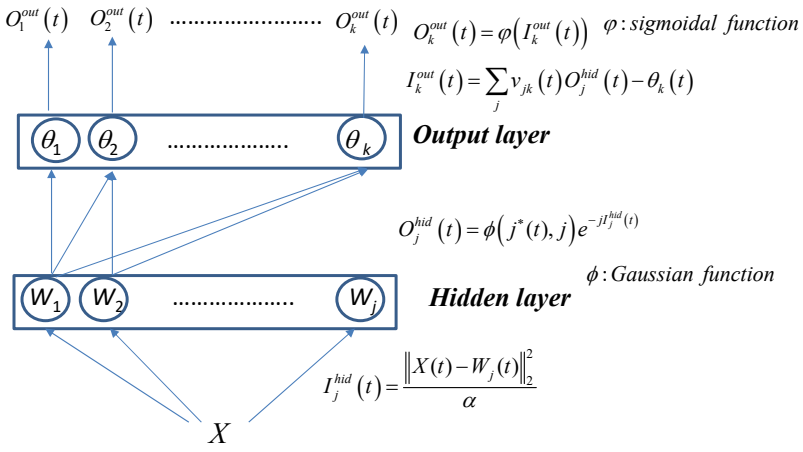


圖 5-6 CRSOM 之架構與函數

接下來是 backpropagation 的部分，以 Gradient descent 法針對 output layer 中第 k 個 neuron 可作如式 5-16 之更新，其中η_1代表學習速率。進一步拆解偏微分項後得式 5-17 之結果，為求方便表示此處令$\delta_k^{out}(t) = (-T(t) + O_k^{out}(t))(O_k^{out}(t)(1 - O_k^{out}(t)))$，則可得到兩個偏微分項的結果式 5-19。

$$v_{jk}(t+1) = v_{jk}(t) + \eta_1 \frac{\partial E(t)}{\partial v_{jk}(t)}$$

$$\theta_k(t+1) = \theta_k(t) + \eta_1 \frac{\partial E(t)}{\partial \theta_k(t)}$$

式 5-16 output layer 參數之更新函數

$$\frac{\partial E(t)}{\partial v_{jk}(t)} = \frac{\partial E(t)}{\partial O_k^{out}(t)} \frac{\partial O_k^{out}(t)}{\partial I_k^{out}(t)} \frac{\partial I_k^{out}(t)}{\partial v_{jk}(t)}$$

$$= \left(-T(t) + O_k^{out}(t)\right)\left(O_k^{out}(t)\left(1 - O_k^{out}(t)\right)\right) O_j^{hid}(t)$$

式 5-17 output layer 參數之更新函數推導過程（一），這裡用到$E(t) = \frac{1}{2}\|T(t) - O^{out}(t)\|_2^2$ 以及 式 5-18

$$\frac{\partial E(t)}{\partial v_{jk}(t)} = \delta_k^{out}(t) O_j^{hid}(t)$$

$$\frac{\partial E(t)}{\partial \theta_k(t)} = -\delta_k^{out}$$

式 5-19 output layer 參數之更新函數推導過程（二）

更深入一層，若要對 hidden layer 的參數做更新，則要做更多偏微分的步驟，例如處理 hidden layer 第 j 個 neuron 時的梯度如式 5-20 所示。同樣地，為求簡化，令$\delta_j^{ref}(t) = -2\left(\sum_k \delta_k^{out}(\mathrm{t})\, v_{jk}(t)\right) e^{-I_j(t)}$，則可對 reference vector 進行如式 5-21 之更新，其中η_2代表學習速率。

$$\frac{\partial E(t)}{\partial W_j(t)} = \sum_k \frac{\partial E(t)}{\partial O_k^{out}(t)} \frac{\partial O_k^{out}(t)}{\partial I_k^{out}(t)} \frac{\partial I_k^{out}(t)}{\partial W_j(t)}$$

$$= \left(\left(\sum_k \delta_k^{out}(t)\, v_{jk}(t) \right) O_j^{hid}(t) \right) \frac{-\partial I_j^{hid}(t)}{\partial W_j(t)}$$

$$= 2 \left(\sum_k \delta_k^{out}(\mathrm{t})\, v_{jk}(t) \right) O_j^{hid}(t)(X(t) - W_j(t))$$

式 5-20 hidden layer 參數之更新函數推導過程

$$\mathrm{W}_j(t+1) = \mathrm{W}_j(t) + \eta_2 \delta_j^{ref}(t) \phi(j^*(t), j)(X(t) - W_j(t))$$

式 5-21 hidden layer 參數之更新函數

綜上所述，W_j因為受到δ_j^{ref}的影響，將會受到來自 true label 的影響而重建出 organizing map，因此相較於原本 SOM 組成的 map 會更具有其 label 的意義。

5.3 Joint Optimization of Mapping and Classifier

本節以數學原理角度解釋 mapping 與 classification 之訓練途徑，以及最佳化的方法 (輔以 auxiliary coordinate 的作用可幫助加速訓練)。將 Dimension Reduction 視為一種 mapping 稱作F，後續的 Classification 稱為g，則整個分類器可以描述為兩個函數的連鎖過程為g∘F。[40]

在 DR 的處理上，主流的分類器皆採 “filter” 模式，分開進行 DR 及 Classification 步驟。第一步，這種模式會將原始資料(x_n, y_n)預先處理為$(F(x_n), y_n)$，其中F可透過不同 supervised 或 unsupervised 方法來學習。第二步則會固定F，以$F(x_n)$作為 feature 送到g去對應到y_n後，從而學習 g，屬於大多數機器學習模型所注重的階段。這種模式被視為是在F之上建立起一種代理目標函數(Proxy objective function) ，從而去最佳化總體分類器g∘F的表現。

另一種則稱為 “wrapper” 模式，則將重點放在直覺性的思考：透過代理目標函數學習到的g和F理論上並非最佳解，取而代之，應該要對整個g∘F去做最佳化

才能夠得到較好的g和F才是。然而 wrapper 卻會面臨 non-convex 問題，且由於 DR 的 mapping 可為非線性對應，此舉將會大幅增加變數間的 dependence，而加提高優化參數的難度。在維持g∘F關係的前提下進行優化，可藉由輔助座標系的幫助，有機會能夠將問題降為 convex problem 如式 5-27。

接下來，我們討論最佳化的方法。此處模型使用非線性的 mapping function F，以及 linear SVM 作為g，則目標函數可描述如式 5-22（recall：SVM 的 objective function 為式 5-23），其中$R(F)$為對 F所進行的 regularization term，舉例來說便是常見的對於 weight 的 L_1 或 L_2 norm。w，b，ξ_n皆屬g的參數，分別為 weight，bias 及 slack variable。N 表示 sample 數。下方的限制則是 linear SVM w/ slackness 的基本形，即在不可能以線性方法完全分開兩類的情況下，允許一定的誤差。

$$\lambda R(F) + \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N}\xi_n$$
$$s.t\begin{cases} y_n(w^T F(x_n) + b) \geq 1 - \xi_n \\ \xi_n \geq 0 \end{cases}, \forall n$$

式 5-22 Mapping 與 Classifier 之總和目標函數（一）

$$\frac{1}{2}\|w\|^2 \; s.t. \; y_n(w^T x_n + b) \geq 1$$
$$\frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N}\xi_n \;\; s.t. y_n(w^T x_n + b) \geq 1 - \xi_n$$

式 5-23 Linear SVM 與 soft-margin linear SVM 之目標函數[41]

物理意義上，λ及C分別為控制 regularization 及 slackness 的 hyper-parameter，$\frac{1}{2}\|w\|^2$為 linear SVM 為了極大化 gap 所做的 term。目標函數的目的是去優化F，ξ_n及g隱含的參數w，b 。然而此處所觀察到的 bottleneck 為來自F的 non-convexity。

Method of auxiliary coordinates (MAC)，此法的目的在於解除 $g \circ F$ 的巢狀結構，從而產生較淺的 mapping 如 $g(z)$ 與 $F(.)$。因此，藉由設定一個 latent space 中新的變數 z 來代替$F(x)$，將有助於這種結構的產生，為此，我們將條件中的$F(x)$改寫為 z，並新增一個條件為$z_n = F(x_n), \forall n$ ，將此條件以 quadratic penalty 方式處理後，將可以移動到目標函數中為式 5-24，新增加的 $\frac{\mu}{2}\sum_{n=1}^{N}\|z_n - F(x_n)\|^2$ 項代表 Z 與$F(x)$的誤差，並加上μ的權重。此處可觀察到 $\mu \to \infty$ 時，將會收斂回原本的關係式。改寫成這種關係後，目標函數需要去優化的參數除了原本的F，g，ξ_n，更增加了z_n。然而這也導致變數之間的依賴關係消失，可分步驟進行優化過程。

$$\lambda R(F) + \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N}\xi_n + \frac{\mu}{2}\sum_{n=1}^{N}\|z_n - F(x_n)\|^2$$

$$s.t \begin{cases} y_n(w^T z_n + b) \geq 1 - \xi_n \\ \xi_n \geq 0 \end{cases}, \forall n$$

式 5-24 Mapping 與 Classifier 之總和目標函數（二）

第一部分為 g-step，固定 F 及 Z 後，優化w及b 的問題純粹只是典型的 SVM 問題$z_n \mapsto y_n$。如果是 K-class 的問題，則可以以 one-vs-all 的模式去解決。

第二部分為 F-step，固定 g (即 w與b)及 Z 後，優化 F 也是典型的 regularized regression 問題$x_n \mapsto z_n$。

第三部分為 Z-step，固定 F 及 g (即 w與b) 後，則欲優化的參數為ξ_n及z_n，因此總體目標函數可改寫式 5-25，又因寫成這種形式後不必從總和計算目標函數，可將各個 sample 視為單一個體而去進行優化，寫作式 5-26。這種 convex quadratic programming 的函數具有一種 closed form 如式 5-27，依據γ的選擇，optimal solution

有可能是正好落在目標函數的 minima 或是符合限制式條件但盡量貼近 minima 的邊緣位置。

$$\frac{2C}{\mu}\sum_{n=1}^{N}\xi_n+\sum_{n=1}^{N}\|z_n-F(x_n)\|^2$$

$$s.t\begin{cases}y_n(w^Tz_n+b)\geq 1-\xi_n\\ \xi_n\geq 0\end{cases},\forall n$$

式 5-25 Z-step 之目標函數（一）

$$\frac{2C}{\mu}\xi+\|z-F(x)\|^2$$

$$s.t\begin{cases}y(w^Tz+b)\geq 1-\xi\\ \xi\geq 0\end{cases}$$

式 5-26 Z-step 之目標函數（二）

Lagrange multiplier: $\frac{2C}{\mu}\xi+\|z-F(x)\|^2+\gamma(y(w^Tz+b)-1+\xi)$

$$z=F(x)+\gamma yw$$

式 5-27 z 之 closed form 解

整體而言，像這種將 optimization 過程解離的方法，將有助於收斂速度的提升與計算複雜度的降低。

REFERENCE

[1]. https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7

[2]. https://www.slideserve.com/marcie/instance-based-learning

[3]. https://alliance.seas.upenn.edu/~cis520/dynamic/2018/wiki/index.php?n=Lectures.LocallyWeightedRegression

[4]. Atkeson, C. G., Moore, A. W., Schaal, S.: Locally weighted learning for control. In *Lazy learning.* 75-113 (1997)

[5]. https://slideplayer.com/slide/5261502/

[6]. https://www.slideserve.com/marcie/instance-based-learning

[7]. https://www.slideserve.com/opa/vector-quantization-vq

[8]. http://www.mqasem.net/vectorquantization/vq.html

[9]. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=2ahUKEwj6rpi3v9ThAhWRwJQKHdIFAzQQFjABegQIAhAC&url=http%3A%2F%2Fyuhmoon.myweb.hinet.net%2Fann%2Fchapter08.ppt&usg=AOvVaw25sA3xGpEL99HWGis_Ll5P

[10]. http://ccy.dd.ncu.edu.tw/~chen/course/Neural/ch3/SOM.pdf

[11]. http://www.cs.bham.ac.uk/~jxb/NN/l16.pdf

[12]. Kohonen, T.: The self-organizing map. *Proceedings of the IEEE* **78**(9). 1464-1480 (1990)

[13]. https://medium.com/@chih.sheng.huang821/%E6%A9%9F%E5%99%A8-%E7%B5%B1%E8%A8%88%E5%AD%B8%E7%BF%92-%E4%B8%BB%E6%88%90%E5%88%86%E5%88%86%E6%9E%90-principle-component-analysis-pca-58229cd26e71

[14]. https://www.bristol.ac.uk/media-library/sites/cmm/migrated/documents/chapter3.pdf

[15]. http://www.cmlab.csie.ntu.edu.tw/~cyy/learning/tutorials/NDR.pdf

[16]. Kingravi, H. A., Celebi, M. E., Rajauria, P. P.: Unsupervised learning of manifolds via linear approximations. In *IEEE International Workshop on Database and Expert Systems Applications.* 54-58 (2007)

[17]. Pless, R.: Images spaces and video trajectories: using isomap to explore video sequences. In *International Conference on Computer Vision* **3**(2). 1433-1440 (2003)

[18]. Saul, L. K., Roweis, S. T.: An introduction to locally linear embedding. Available at: http://www.cs.toronto.edu/~roweis/lle/. (2000)

[19]. Roweis, S. T., Saul, L. K.: Nonlinear dimensionality reduction by locally linear embedding. *Science* ***290***(5500). 2323-2326 (2000).

[20]. https://www.youtube.com/watch?v=scMntW3s-Wk

[21]. https://en.wikipedia.org/wiki/Laplacian_matrix

[22]. http://www.datakit.cn/blog/2017/02/05/t_sne_full.html

[23]. Maaten, L. v. d., Hinton, G.: Visualizing data using t-SNE. *Journal of Machine Learning Research* **9**. 2579-2605 (2008)

[24]. http://biostatdept.cmu.edu.tw/doc/epaper_b/38.pdf

[25]. https://medium.com/jameslearningnote/%E8%B3%87%E6%96%99%E5%88%86%E6%9E%90-%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E7%AC%AC3-5%E8%AC%9B-%E6%B1%BA%E7%AD%96%E6%A8%B9-decision-tree-%E4%BB%A5%E5%8F%8A%E9%9A%A8%E6%A9%9F%E6%A3%AE%E6%9E%97-random-forest-%E4%BB%8B%E7%B4%B9-7079b0ddfbda

[26]. https://sebastianraschka.com/faq/docs/decision-tree-binary.html

[27]. http://www.cs.cmu.edu/~bhiksha/courses/10-601/decisiontrees/

[28]. https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

[29]. https://hackernoon.com/how-to-develop-a-robust-algorithm-c38e08f32201

[30]. Breiman, L.: Bagging predictors. *Machine learning* **24**(2). 123-140 (1996)

[31]. Mehta, P., Bukov, M., Wang, C. H., Day, A. G., Richardson, C., Fisher, C. K., Schwab, D. J.: A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*. 36-39 (2019)

[32]. https://medium.com/@chih.sheng.huang821/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-ensemble-learning%E4%B9%8Bbagging-boosting%E5%92%8Cadaboost-af031229ebc3

[33]. Wolframe Alpha

[34]. https://medium.com/@cwchang/gradient-boosting-%E7%B0%A1%E4%BB%8B-f3a578ae7205

[35]. http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/

[36]. http://www.ccs.neu.edu/home/vip/teach/MLcourse/4_boosting/slides/gradient_boosting.pdf

[37]. Chen, T., Guestrin, C.: Xgboost: a scalable tree boosting system. In *ACM Conference on Knowledge Discovery and Data Mining.* 785-794 (2016)

[38]. Chang, G. W., Chen, C. I., Teng, Y. F.: Radial-basis-function-based neural network for harmonic detection. *IEEE Transactions on Industrial Electronics* **57**(6). 2171-2179 (2010)

[39]. Hartono, P.: Classification and dimensional reduction using restricted radial basis function networks. *Neural Computing and Applications* **30**(3). 905-915 (2018)

[40]. Wang, W., Carreira-Perpinan, M. A.: The role of dimensionality reduction in classification. In *Conference on Artificial Intelligence*. (2014)

[41]. https://slideplayer.com/slide/1579281/

[42]. https://zhuanlan.zhihu.com/p/24635014

[43]. http://www.cs.columbia.edu/~verma/classes/uml/lec/uml_lec8_tsne_slides.pdf

[44]. http://djj.ee.ntu.edu.tw/ADSP7.pptx

[45]. Dijkstra, E. W.: A note on two problems in connexion with graphs. *Numerische mathematik* **1**(1). 269-271 (1959)

[46]. Floyd, R. W.: Algorithm 97: shortest path. *Communications of the ACM* **5**(6). 345 (1962)

[47]. Warshall, S.: A theorem on boolean matrices. *Journal of the ACM* **9**(1). 11-12. (1962)

[48]. https://www.youtube.com/watch?v=tH9FH1DH5n0

[49]. http://www.cis.hut.fi/projects/somtoolbox/documentation/somalg.shtml

[50]. https://www.itread01.com/p/1419320.html

[51]. Hastie, T., Tibshirani, R. & Friedman, J. (2009), The Elements of Statistical Learning; Data Mining, Inference and Prediction, Springer. Second edition.