淺度機器學習：分類器的原理與評比實驗

Multinomial Logistic Regression 、 Support Vector Machine 與 Neural Network

April 28, 2025

機器學習最常見的應用是「教會」機器對多變量資料進行分群，尤其面對變數多、資料量龐大及需要及時反應的場景，人類必須仰賴機器快速計算的優勢。這類的機器也稱分類器，可以單純的只是一個統計方法，也可以是複雜到必須外掛 GPU 的演算法。本文介紹三種常見的統計分類方法: 多元羅吉斯回歸（Multinomial Logistic Regression, MLR）、支援向量機（Support Vector Machine, SVM）與神經網路（Neural Network）在影像分群的應用及如何進行評比實驗。其中多元羅吉斯回歸的分類原則就是現行許多深度機器學習在輸出端使用的 Softmax 分類器。

本章將學到關於程式設計

〈本章關於 Python 的指令與語法〉
指令：
sklearn.linear_model: LogisticRegression, LogisticRegressionCV
sklearn.svm: SVC, LinearSVC
sklearn.neural_network: MLPClassifier
sklearn.decomposition: PCA
sklearn.preprocessing: StandardScaler
sklearn.model_selection: train_test_split
sklearn.metrics: classification_report, ConfusionMatrixDisplay, accuracy_score

1 背景介紹

1.1 多元羅吉斯回歸

多元羅吉斯回歸將一般性的羅吉斯回歸從二元分類擴展為多元，符合現代機器學習面對多類別辨識的需求，並將其納入作為輸出單元，也稱為 Softmax 或 Maximum Entropy classifier。多元羅吉斯回歸做為分類器的目的，是將一組多變量資料 **x** 的線性組合依機率之大小，分配到 K 個類別中機率最大的類別。數學式表達為（參考 [1]）

$$\begin{aligned}
\log \frac{Pr(G=1|X=\mathbf{x})}{Pr(G=K|X=\mathbf{x})} &= \beta_{10} + \boldsymbol{\beta}_1^T \mathbf{x} \\
\log \frac{Pr(G=2|X=\mathbf{x})}{Pr(G=K|X=\mathbf{x})} &= \beta_{20} + \boldsymbol{\beta}_2^T \mathbf{x} \\
&\vdots \\
\log \frac{Pr(G=K-1|X=\mathbf{x})}{Pr(G=K|X=\mathbf{x})} &= \beta_{(K-1)0} + \boldsymbol{\beta}_{K-1}^T \mathbf{x}
\end{aligned} \tag{1}$$

其中 $Pr(G=k|X=\mathbf{x})$ 稱為後驗機率（posterior probability），兩個後驗機率的比值取對數後，稱為對數勝算比（log-odds ratio）或邏輯轉換（logit transformation）。並以此對數勝算比做為線性回歸的應變數。式 (1) 可以改寫為

$$\begin{aligned}
Pr(G=k|X=\mathbf{x}) &= \frac{e^{\beta_{k0}+\boldsymbol{\beta}_k^T \mathbf{x}}}{1+\sum_{l=1}^{K-1} e^{\beta_{l0}+\boldsymbol{\beta}_l^T \mathbf{x}}}, \quad k=1,2,\cdots,K-1, \\
Pr(G=K|X=\mathbf{x}) &= \frac{1}{1+\sum_{l=1}^{K-1} e^{\beta_{l0}+\boldsymbol{\beta}_l^T \mathbf{x}}}
\end{aligned} \tag{2}$$

式 (2) 表示多變量資料 **x** 屬於 K 個類別的個別機率，且此 K 個後驗機率之總和為 1，代表除此之外別無選擇。於是在分類的判別上，便將 **x** 判給機率最高的那一類。式 (2) 多元羅吉斯回歸模型牽涉到需要估計的參數為

$$\Omega = \{\beta_{10}, \boldsymbol{\beta}_1, \beta_{20}, \boldsymbol{\beta}_2, \cdots, \beta_{(K-1)0}, \boldsymbol{\beta}_{K-1}\}$$

假設多變量資料 **x** 有 p 個變數，則參數 Ω 共有 $(p+1)(K-1)$ 個參數需要估計。參數的估計在機器學習領域裡，被稱為學習（learning）或訓練（training），僅就羅吉斯回歸這部分的參數估計，一般以最大概似估計配合適當的演算法進行。假設共有 N 筆多變量資料 $\mathbf{x}_i, i=1,2,\cdots,N$，則其對數概似函數寫為[1]

[1] $f(\mathbf{x_i}, y_i|\Omega)$ 稱第 i 個樣本的概似函數，其中 $y_i \in \{1,2,\cdots,K\}$。

$$l(\Omega) = \ln \Pi_{i=1}^{N} f(\mathbf{x_i}, y_i|\Omega) = \tag{3}$$

於是多元羅吉斯回歸的參數估計問題寫為：

$$\max_{\Omega} l(\Omega) \tag{4}$$

接著，我們以 `sk-learn` 提供的羅吉斯回歸套件應用在人臉的分辨。在此之前，先拿較單純的資料試驗看看。

範例 1. `sklearn` 提供了兩個多元羅吉斯回歸的套件：`LogisticRegression` 及 `LogisticRegressionCV`。[a] 先來參考 `sk-learn` 的範本程式，該範例以義大利地區三個紅酒產區的 178 瓶酒所萃取出 13 種葡萄酒成分作為自變數，以三個產區為標籤。[b]本範例計畫以下列資料做多元羅吉斯回歸的分類練習：

1. 原始資料標準化與未標準化。
2. 以 13 種葡萄酒成分作為輸入資料。
3. 以 13 種葡萄酒成分的 q 個主成分為輸入資料。

比較原始的 13 種成分資料與較低維度的 q 主成分資料的分類準確率與執行效率。

[a] `LogisticRegressionCV` 可視為高階版的 `LogisticRegression`。這兩個套件所使用的參數估計方式較之前述的最大概似函數估計，多了兩個控制項以避免過度訓練。而 `LogisticRegressionCV` 對控制項的掌握更細緻（簡單地說）。

[b]範例程式來自 `sklearn` 網站，以關鍵字「Importance of Feature Scaling」查詢。

當進行分類器學習時，通常將資料分成兩部分：訓練資料與測試資料。以訓練資料估計分類器模型的參數，待估計完成後，再以測試資料檢驗訓練結果。若重複訓練與測試的結果表現不佳，則必須調整參數估計的演算法，包括演算法的停止條件、迭代數量、... 甚至更換演算法等。假設表現依然不被接受，還可以調整資料，譬如進行標準化、改為特徵資料或改變特徵的選取... 等。一連串的實驗嘗試直到窮其一切辦法為止。下列以三段程式碼展示最陽春的做法。

第一段：準備資料（讀入資料、分離資料、標準化資料）

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.model_selection import train_test_split

# Read data
df = pd.read_excel('data/Wine.xlsx')
X = np.array(df.iloc[:, :-1]) # 排除最後一欄標籤
y = np.array(df.iloc[:, -1]) #標籤欄

# Split data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30)

# Standardize data
scaler = StandardScaler()
X_train_ = scaler.fit_transform(X_train)
X_test_ = scaler.fit_transform(X_test)
```

注意：訓練資料與測試資料必須分開標準化，而非標準化後再分成訓練與測試資料。上述程式碼將測試資料規劃為 30%。讀者不妨試著調整為 20% 或 25%。

第二段：以標準化後之原始資料的訓練資料學習，並以測試資料測試準確率

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'lbfgs'  # 'lbfgs' is the default
# solver = 'liblinear'
# solver = 'newton-cg'
clf_original = LogisticRegression(solver = solver, **opts)
clf_original.fit(X_train_, y_train)

y_pred = clf_original.predict(X_test_)
# 測試資料之準確率回報
print(f"{accuracy_score(y_test,␣y_pred):.2%}\n")
print(f"{clf_original.score(X_test_,␣y_test):.2%}\n")
print(classification_report(y_test, y_pred))
```

請注意這裡將分類器（LogisticRegression）參數估計的演算法所需的調整細節放在 opts 中一併陳列。另，回報測試資料對於訓練完成的分類器的分類準確率，以兩種不同方式呈現，其中 accuracy_score 比對了測試資料的標籤（y_test）與分類預測值（y_pred），而 clf_original.score 直接給出準確

率。兩者結果是一樣的。最後一個指令給出較完整的報告，列印出如圖 1 ，其中的幾項指標，如 precision、recall、f1-score 都是統計名詞，讀者很容易查到其定義。此外，上述程式碼故意將演算法（`solver`）的選項註解，留著給讀者參考選用。

	precision	recall	f1-score	support
1	0.86	1.00	0.92	12
2	1.00	0.92	0.96	24
3	1.00	1.00	1.00	18
accuracy			0.96	54
macro avg	0.95	0.97	0.96	54
weighted avg	0.97	0.96	0.96	54

圖 1: 測試資料的分類準去度報告

第三段：以標準化後之原始資料的主成分之訓練資料學習，並以測試資料測試準確率

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)

opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'lbfgs'  # 'lbfgs' is the default
# solver = 'liblinear'
# solver = 'newton-cg'

clf_PCA = LogisticRegression(solver = solver, **opts)
clf_PCA.fit(Z_train, y_train)
y_pred = clf_PCA.predict(Z_test)

print(f"{clf_PCA.score(Z_test,␣y_test):.2%}\n")
```

請注意：上述程式碼採用兩個主成分，以對比前一段程式採用了 13 個輸入變數。其執行效率肯定比較快，但是準確率則有待實驗觀察。讀者可以將主成分逐漸調高（直到 13），觀察準確率的變化，甚至可以畫一張折線圖來比較看看（與 scree plot 對比）。

當使用 `sklearn` 的羅吉斯回歸指令（`LogisticRegression`）時，必須了解演算法的概念，必要時必須嘗試不同的參數以取得較佳的表現，譬如演算法的選擇（如 `solver = 'lbfgs'`）、停止條件的設定（`tol = 1e-6, max_iter`

= int(1e6))，而非一味地採用預設的參數，避免因訓練不足導致錯誤的結果。

如果仔細地實驗範例中的幾個提問，也是一件不小的事。讀者最好事先訂好計畫，一步步地仔細執行，並隨時觀察各種值得一看的數據。另外，也可以試試 LogisticRegressionCV 的表現，譬如

```
Cs = np.logspace(-5, 5, 20)
clf_original = LogisticRegressionCV(solver = solver, \
                Cs=Cs, cv=5, **opts)
```

LogisticRegressionCV 的 CV 代表加入了 Cross Validation（交叉驗證）的機制，在指定的 c 值中 (Cs) 挑選最佳 c 值。[2]以避免演算法陷入過度訓練的麻煩。簡單說，是在訓練資料裡面再撥出一部分，當作在訓練過程中的測試資料，一旦測試結果不再進步，便會提早停止訓練。至於交叉驗證的方式也可以從 cv 選項自訂，詳細介紹請看 https://scikit-learn.org/stable/modules/cross_validation.html。

此外，機器學習常見一個學習方法伴隨著多個參數的選擇問題。這是來自每個存活至今的學習方法都經歷許多研究學者的改良，愈來愈能應付各種不同的資料型態與訓練需求，但也因此引入愈來愈多的參數選擇，譬如演算法 solver、約束機制 c 值, 交叉驗證的方式 cv、、、等。但愈完善的選項代表另一種麻煩，便是哪一組選項才是最佳選擇？於是出現了像 sklearn 提供的 GridSearchCV 用來搜尋最佳組合。譬如下列程式碼列出 15 種組合給 LogisticRegression 進行計算，最後選出最佳者。

```
opts = dict(tol = 1e-6, max_iter = int(1e6))
# parameters for GridSearchCV
parameters = {'solver':['lbfgs', 'liblinear', \
'newton-cg', 'sag','saga'], 'C':[0.1, 1, 10]}
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.3) # 5-fold CV
grid = GridSearchCV(estimator=LogisticRegression(**opts), \
       param_grid=parameters, cv=cv,
       scoring=['accuracy','f1_macro'], refit="accuracy")
grid.fit(X_train_, y_train)

cv_logistic = pd.DataFrame(data = grid.cv_results_)
```

[2]在 LogisticRegression 中的 c 值與 LogisticRegressionCV 中的 cs 都是指 regularization strength（約束強度），目的為避免演算法在最後階段陷入過度擬合（過度訓練）而強加入的約束機制，c 值代表約束強度。cs 值則是從多個指定的 c 值中透過 Cross Validation 的方式挑選最佳的 c 值。這也是 LogisticRegressionCV 與 LogisticRegression 最大不同之處。

```
print(grid.best_params_)
print(grid.best_score_)
print(grid.best_estimator_)
```

範例 2. 依前範例對於紅酒產區的辨識，將資料轉為人臉影像。同樣地，先採用資料量較小、類別較少的人臉影像，譬如 AT&T 人臉資料。[a]這組影像資料共有 40 人（類），每人 10 張影像，每張影像大小為 64×64。試著使用多元羅吉斯回歸作為分類器（`LogisticRegression` 或 `LogisticRegressionCV`），而提供分類器訓練的人臉影像資料可分為原型資料與主成分特徵資料。觀察各種組合的分類準確率。

[a]影像下載自：https://github.com/daradecic/Python-Eigenfaces

範例 3. 同上，但使用 Yale Face 38 人的 2410 張人臉圖像，每張大小 192×168。由於這組人臉圖像所提供的 y 值是每個人的張數，並沒有提供每張圖像所屬的組別標籤，必須自行製作。

樣本數 2410 與 $192 \times 168 = 32256$ 的變數總量，讓學習的時間變長許多，這便凸顯了主成分分析的重要。假設取前 200 個主成分進行學習，則與原來資料的 32256 個變數相差百倍以上，所節省的學習時間值得期待，而學習效果也值得觀察。

1.2 支援向量機 SVM

支援向量機的原理是在二維平面建立一條直線，或在高維空間建立一個超平面（hyperplane），[3]

$$\mathbf{w}^T\mathbf{x} + \mathbf{b} = \mathbf{0} \tag{5}$$

其中 $\mathbf{x}$ 代表變數向量 $[x_1\ x_2\ \ldots x_p]^T$，使得式 (5) 的超平面能將兩個群組分開。當面對超平面（線性）無法分開的群組資料時，可以將資料投射到更高維的空間，使得該資料在高維度空間為線性可分。這個觀念引入了非線性的核函數（kernels），式 (5) 改寫為一般形式的

[3]網頁 https://codeshellme.github.io/2021/01/ml-svm1/ 以平實易懂的方式介紹 SVM，值得參訪。

$$\mathbf{w}^T\phi(\mathbf{x}) + \mathbf{b} = \mathbf{0} \tag{6}$$

參數 **w** 與 **b** 來自這個最小值問題：[4]

$$\begin{aligned} &\min_{\mathbf{w},\mathbf{b},\zeta} \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\zeta_i \\ &\text{subject to } \; y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + \mathbf{b}) \geq 1 - \zeta_i, \;\; \zeta_i \geq 0 \end{aligned} \tag{7}$$

其中 $y_i \in \{-1, 1\}$，代表資料 $\mathbf{x}_i$ 的類別值，即非 1（正方）即 -1（負方）。如果這個最佳的超平面能完美分開群組資料（同為正方或負方），則 $y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + \mathbf{b}) \geq 1$。但實務資料往往無法完美分割，因此加上 $C\sum_{i=1}^{N}\zeta_i$ 作為處罰項（penalty），也就是允許不完美切割，以 ζ_i 作為第 i 個樣本點被錯誤分開的距離，且以常數 C 代表處罰項的強度。一般稱這個處罰項為正規化（或約束）參數（regularization parameters），是處理過度訓練的手段之一。

以上對 SVM 的描述以兩個類別為主，式 (7) 決定了兩個類別群組之間的分界線（超平面）。對於多元類別（multiple classes）的分類一般以兩種方式處理：（假設共有 K 個類別）

1. One vs one (ovo)：針對 K 個類別內的每兩個類別製作一個分類器（分界線），總共有 $\frac{K(K-1)}{2}$ 個分類器。當面對一個新資料 $\mathbf{x}_{new}$ 的分類時，讓 $\frac{K(K-1)}{2}$ 個分類器都做分類，再計算 $\mathbf{x}_{new}$ 被分到哪一個類別的次數最多，以此類別作為最終分類的結果。

2. One vs the rest (ovr)：顧名思義，就是一個類別對比其餘類別的整合。換句話說，將 K 個類別分成兩組，一組為某單一類別，歸為正方；其餘類別同歸於負方，如此建立一個分類器（分界線）。當面對新資料時，同樣計算每個類別被分到的次數。遇到同分，則以距離支援向量短者為勝。`sk-learn` 以 ovr 作為預設值。

`sk-learn` 套件提供了幾個 SVM 的學習方式，如 `SVC, LinearSVC, NuSVC`，[5]其中以 `SVC` 較常被使用。以下的練習便以 `SVC` 為主。

[4]參考 `sk-learn` 官網關於 `SVM` 的介紹：`https://scikit-learn.org/stable/modules/svm.html#svm-classification`。

[5]`SVC` 與 `NuSVC` 對於多元群組資料，在決策函數（decision function）的選擇有兩種：一、採一對其他（one-vs-rest），即為每一群組建立的分界線是以其他群組全部作為對立面，且為預設選項；二、採一對一（one-vs-one）分組方式，即為每一個群組都建立與其他個別群組的超平面界線；而 `LinearSVC` 則是採一對其他的模式。讀者可以試著改變 `SVC` 的參數 `decision_function_shape='ovo'`，比較一對一（ovo）與一對其他（ovr）兩種模式的表現。

範例 4. 利用 SVC 重作前述範例的三組資料，並與 LogitRegression 並列一起做，也就是使用相同的訓練與測試資料。觀察這兩個分類方法的優劣或差異。

以下程式碼示範 SVC 典型的用法：

```
from sklearn.svm import SVC, LinearSVC

C = 1  # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
# opts = dict(C = C, decision_function_shape = 'ovo', \
#             tol = 1e-6, max_iter = int(1e6))

clf_svm = SVC(kernel="linear", **opts)
# clf_svm = SVC(kernel="rbf", gamma=0.2, **opts)
# clf_svm = SVC(kernel="poly", degree=3, gamma="auto", **opts)
# clf_svm = LinearSVC(**opts) # one vs the rest
clf_svm.fit(X_train, y_train)
predictions = clf_svm.predict(X_test)
print(classification_report(y_test, predictions))
```

SVC 預設的 kernel 是 rbf(radial basis function)，另外還有其他選項，如多項式 ploy, sigmoid 等，讀者可以多方嘗試，找到最適合的組合。上述程式碼為了比較不同分類法的表現，因此採用變數名稱 clf_svm 代表 SVM 的分類器，此時做為對比的多元羅吉斯分類器可以取名 clf_LR 以利區別。

1.3 神經網路（Neural Network）

這裡所指的神經網路為傳統的前饋式（feedforward）多層次感知器（Multi-Layer Perception, MLP），而非目前深度學習使用的神經網路，譬如 CNN 這類更複雜、龐大的系統。圖 2 展示具備一個隱藏層的典型前饋式類神經網路，[6]其中幾個須留意的數字為左邊輸入端（Input）標示為 p 個變數個數（圖中 $p = 14$），中間隱藏層（Hidden Layer）有 q 個神經元（圖中 $q = 10$）及最右邊的輸出層（Output Layer），共有 r 個輸出變數（圖中 $r = 3$）。輸入與輸出變數的個數 p, r 依問題的結構而定，譬如用於人臉辨識，則 p 代表影像的大小（或特徵成分的大小），r 等於辨識的類別。值得一提的是中間的隱藏層與輸出層，特別是隱藏層的結構與所含的神經元數量 q。q 越大，代表輸出與輸入之間的關係越複雜，從數學關係的角度來說，便是彼此間的非線性程度越高。

[6]本圖從 MATLAB 輸出

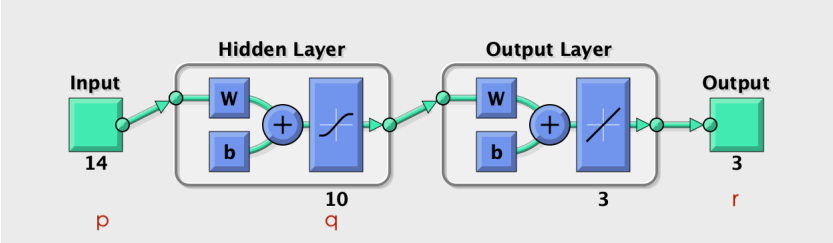

圖 2: 具備一個隱藏層的典型前饋式神經網路

假設輸入端的 p 個變數表示為 $x_1, x_2, \cdots, x_p$，輸出端的 r 個變數表示為 $\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_r$，則前饋式類神經網路的輸出與輸入間的數學關係寫成

$$\hat{y}_k = \sum_{i=1}^{q} w_{ki}^2 g\left(\sum_{j=1}^{p} w_{ij}^1 x_j + b_i^1\right) + b_k^2, \quad 1 \le k \le r \tag{8}$$

其中函數 $g(\cdot)$ 稱為激發函數（activation function），[7]可以選擇為（$-1 \le g(z) \le 1$）

$$g(z) = c_1 \frac{1 - e^{-c_2 z}}{1 + e^{-c_2 z}} \tag{9}$$

激發函數 $g(z)$ 的長相便如圖 2 的隱藏層所繪製的函數圖。函數 $g(z)$ 也可以用在輸出層，以增加複雜度，不過通常使用如圖 2 的線性函數 $y = x$，也就是圖 2 的輸出層所繪製直線方程式。式 (8) 中的 w_{ij}^1 與 b_i^1 代表第一個隱藏層的第 i 個神經元與第 j 個輸入的權重係數（Weightings）與位階係數（Biases）。同樣地，w_{ki}^2 與 b_k^2 則是第二層（在此為輸出層）的第 k 個神經元與前一隱藏層的第 i 個輸出的權重係數與位階係數。

類神經網路根據已知的輸入與輸出資料 $x_j(n)$ 與 $y_k(n)$，調整係數 $w_{ij}^1, w_{ki}^2, b_i^1$ 與 b_k^2，使得真實資料 $y_k(n)$ 與類神經網路輸出資料 $\hat{y}_k(n)$ 的誤差為最小。假設共有 N 筆真實資料，則所謂類神經網路的訓練階段，寫成多變量函數的最小值問題，即

$$\min_{\Omega} e(\Omega) \tag{10}$$

[7]激發函數的意義來自神經細胞受到其他細胞的電力刺激，當刺激的電力上升到某個門檻值時，將激發與它連結的其他神經細胞。激發函數便是模擬神經細胞的激發狀態。每個神經細胞的激發函數不同，有些是較和緩的爬升，有些則是陡峭的如一個開關，非關即開。在深度學習的神經網路中，常用的激發函數稱為 relu 函數，長相近似式 (9) 的 sigmoid 函數，差別在中間那段非線性部分變成線性，以此降低整個系統的非線性程度，達到運算速度加快，也同時具備避免神經網路過度擬合的副作用。

其中損失函數（loss function）[8]

$$
\begin{aligned}
e(\Omega) &= \sum_{n=1}^{N}\sum_{k=1}^{r}\left(y_k(n)-\hat{y}_k(n)\right)^2 \\
&= \sum_{n=1}^{N}\sum_{k=1}^{r}\left(y_k(n)-\sum_{i=1}^{q} w_{ki}^2 g\left(\sum_{j=1}^{p} w_{ij}^1 x_j + b_i^1\right) + b_k^2\right)^2
\end{aligned} \tag{11}
$$

其中參數 $\Omega = \{w_{ij}^1, w_{ki}^2, b_i^1, b_k^2\}_{i=1,2,\cdots,q;j=1,2,\cdots,p;k=1,2,\cdots,r}$，共 $pq+qr+q+r$ 個參數。若以圖 2 的類神經網路架構（$p=14, q=10, r=3$）為例，式 (10) 的損失函數 $e(\Omega)$ 的變數共 183 個。因此類神經網路可視為一個非常複雜、非線性程度很高到函數，能將輸入（X）與輸出（Y）的關係配適的很完美。

`sk-learn` 相對應的套件為 `MLPRegressor` 與 `MLPClassifier`，分別用在輸出端為連續型與類別型變數。以下影像辨識的範例便以 `MLPClassifier` 為主。

> 範例 5. 以 `MLPClassifier` 重作前述範例的三組資料，並與 `LogitRegression, SVC` 並列一起做，也就是使用相同的訓練與測試資料。觀察這三個分類方法的優劣或差異。

以下程式碼為典型的 `MLPClassifier` 使用方式：

```
from sklearn.neural_network import MLPClassifier

# hidden_layers = (512,) # one hidden layer
# activation = 'relu' # the default
hidden_layers = (30,)
activation = 'logistic'
opts = dict(hidden_layer_sizes = hidden_layers, verbose = True, \
        activation = activation, tol = 1e-6, max_iter = int(1e6))
# solver = 'sgd' # not efficient, need more tuning
# solver = 'lbfgs' # not suitable here
solver = 'adam' # default solver
clf_MLP = MLPClassifier(solver = solver, **opts)
clf_MLP.fit(X_train, y_train)
```

[8] Loss function 是機器學習的靈魂角色，一切的學習都是繞著它轉。關於 loss function 的基本觀念可以參考這個網頁的介紹：https://chih-sheng-huang821.medium.com/機器-深度學習-基礎介紹-損失函數-loss-function-2dcac5ebb6cb。簡單說，損失函數呈現真實輸出值與模型輸出值的差異，其變數為模型參數，是一個非常高維度的多變量非線性函數。其形態依輸出值為連續型與類別型，有所不同。譬如類別型輸出的分類問題，常用的損失函數稱為交叉熵（cross-entropy），用來量測資訊的不確定性。本文以類別辨識為主，採用的神經網路套件 `MLPClassifier` 的損失函數常稱為 softmax，也是交叉熵；又如前述的多元羅吉斯回歸的對數概似函數若再進一步分析，最後的型態就是負的 cross-entropy（參考 https://en.wikipedia.org/wiki/Cross_entropy）。

```
predictions = clf_MLP.predict(X_test)
print(classification_report(y_test, predictions))
```

神經網路需要「琢磨」的地方很多，譬如隱藏層的數量、每個隱藏層的神經元個數、激發函數的選擇（relu, logistic, tanh）、演算法的選擇（adam, lbfgs, sgd,...）。不同的資料可能需要不同的組合，導致機器學習的訓練過程往往非常冗長。不過這樣的實驗過程也會讓執行者能深刻了解每一項選擇的意義，譬如當選擇激發函數為預設的 relu 時，所需要的隱藏層的神經元要相對於 logistic 多些，因為 relu 接近線性的函數對應，會降低整個網路的非線性關係，雖然可以避免過度訓練，但它的近乎線性的對應關係，需要較多的神經元才能達到「學習」的目的。所以上述程式碼有一組註解的 `hidden_layer = (512,)` 與 `activation = 'relu'`，便是建議讀者試試。至於需要多少個隱藏層才夠？或是每個隱藏層需要多少神經元？必須透過不斷嘗試性的實驗才能得到結論，可以肯定得是，絕非越多越好，因此測試資料的準確度非常重要。

另外，神經網路的演算法也很關鍵，演算過程 loss function $e(\Omega)$ 的遞減速度與趨勢也值得觀察，如圖 3。

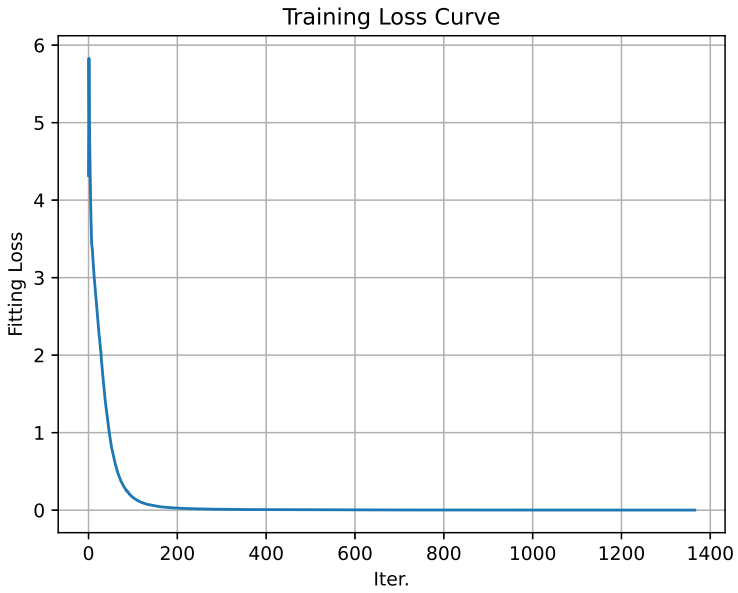

圖 3: loss function $e(\Omega)$ 隨演算的遞迴逐漸變小的趨勢

如果類別不是太多，也可以繪製 confusion matrix 來觀察每一個類別被分類的準確度，進而了解哪些類別或該類別的資料有甚麼特殊之處，以致造成分類的誤差較大。這便提供了輸入端可以做哪些事來提高分類準確度。製作圖 4 的程式碼如下：

```
from sklearn.metrics import ConfusionMatrixDisplay

fig, ax = plt.subplots(1, 1, figsize=(12,12))
score = 100*clf_MLP.score(X_test, y_test)
title = 'Testing␣score␣={:.2f}%'.format(score)
disp = ConfusionMatrixDisplay.from_estimator(
        clf_MLP,
        X_test,
        y_test,
        xticks_rotation=45, #'vertical',
        # display_labels=class_names,
        cmap=plt.cm.Blues,
        normalize='true',
        ax = ax
    )
disp.ax_.set_title(title)
plt.show()
```

References

[1] T. Hastie, R. Tibshirani, J. Friedman, "The Elements of Statistical Learning：Data Mining, Inference, and Prediction".

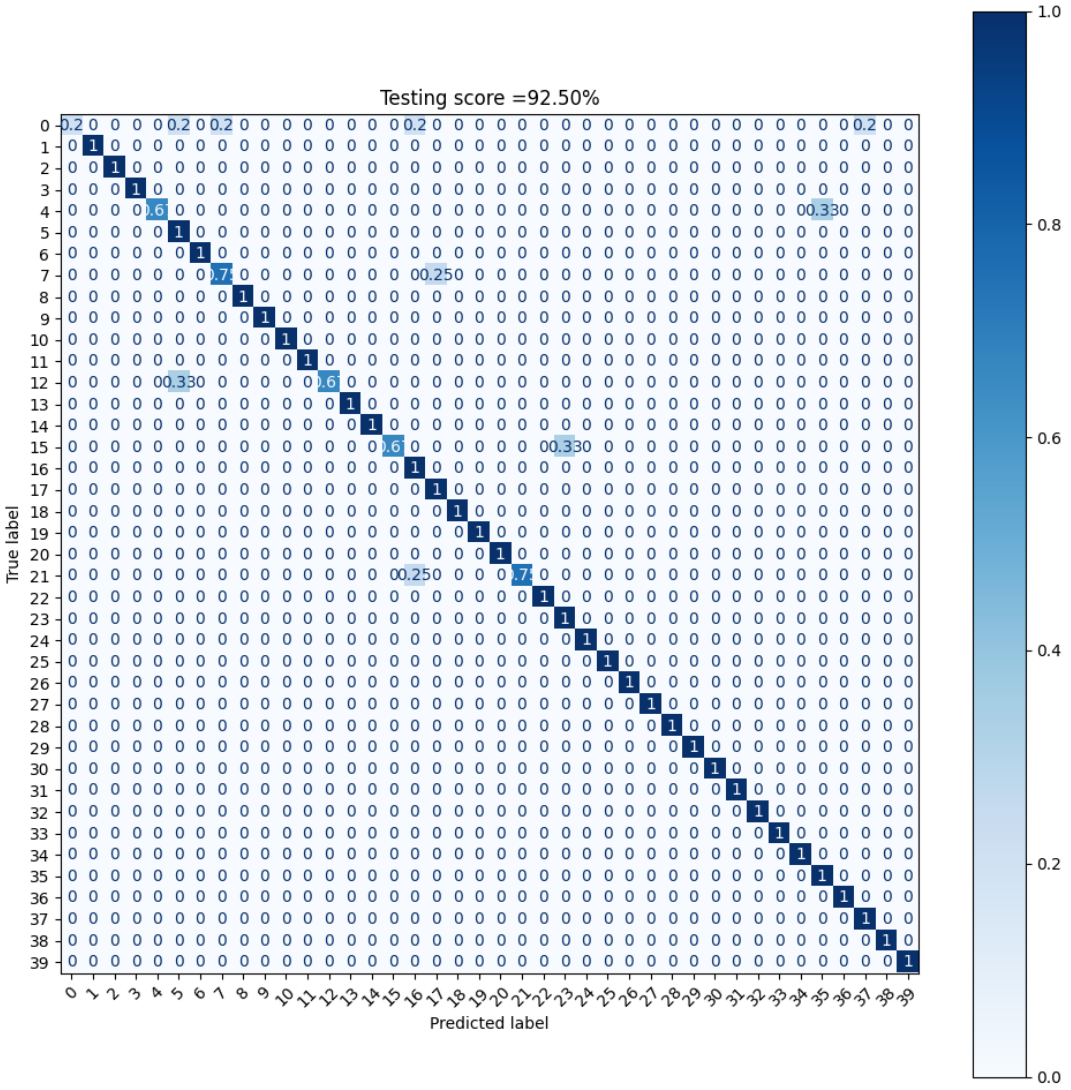

圖 4: 測試資料的 confusion matrix。