海大資工 AI 機器學習 - 01057006 作業報告

(一)實驗結果

系統代碼	執行環境	實驗資料	正確率
A	Scikit-learn - DecisionTree	訓練 vs 測試 4:1	99.85%
B	Scikit-learn - DecisionTree	10-fold cross-validation	99.53%
C	Scikit-learn - RandomForest	訓練 vs 測試 4:1	99.56%
D	Scikit-learn - RandomForest	10-fold cross-validation	99.24%
E	Scikit-learn - Naive Bayes - GaussianNB	訓練 vs 測試 4:1	7.5%
F	Scikit-learn - Naive Bayes - GaussianNB	10-fold cross-validation	8.3%
G	Scikit-learn - Naive Bayes - MultinomialNB	訓練 vs 測試 4:1	95.29%
H	Scikit-learn - Naive Bayes - MultinomialNB	10-fold cross-validation	95.17%
I	Scikit-learn - Naive Bayes - BernoulliNB	訓練 vs 測試 4:1	92.94%
J	Scikit-learn - Naive Bayes - BernoulliNB	10-fold cross-validation	91.31%
K	Scikit-learn - SVM - linear	訓練 vs 測試 4:1	97.06%
L	Scikit-learn - SVM - linear	10-fold cross-validation	96.26%
M	Scikit-learn - SVM - poly	訓練 vs 測試 4:1	94.85%
N	Scikit-learn -	10-fold cross-validation	93.08%

	SVM - poly		
O	Scikit-learn - SVM - rbf	訓練 vs 測試 4:1	95.74%
P	Scikit-learn - SVM - rbf	10-fold cross-validation	93.97%
Q	Scikit-learn - KNN(鄰居數量 5)	訓練 vs 測試 4:1	96.03%
R	Scikit-learn - KNN(鄰居數量 5)	10-fold cross-validation	94.91%
S	Scikit-learn - KNN(鄰居數量 10)	訓練 vs 測試 4:1	96.47%
T	Scikit-learn - KNN(鄰居數量 10)	10-fold cross-validation	94.76%
U	Scikit-learn - KNN(鄰居數量 20)	訓練 vs 測試 4:1	96.32%
V	Scikit-learn - KNN(鄰居數量 20)	10-fold cross-validation	94.44%
W	Scikit-learn - MLPClassifier	訓練 vs 測試 4:1	98.26%
X	Scikit-learn - MLPClassifier	10-fold cross-validation	98.15%
Y	Tensorflow - DNN	訓練 vs 測試 4:1	97.46%
Z	Tensorflow - CNN	訓練 vs 測試 4:1	99.33%
AA	Tensorflow - MLP	訓練 vs 測試 4:1	97.35%

(二)系統描述

主程式架構:

1. 載入資料、資料處理

```
from scipy.io import arff # 提供加載和解析 ARFF 文件的功能
import pandas as pd
# 載入資料
data = arff.loadarff('/content/sample_data/hypothyroid_modified_cjlin.arff')
df = pd.DataFrame(data[0]) # 建立二維的資料表格 ( 方便對資料做操作 )
```

```
# Artribute ( 訓練樣本集的列 ) 移除分類類別
data_x = df.drop(columns=['Class'])
# 目標變量 ( 模型評估和預測 )
data_y = df['Class']
```

```
# 將類別特徵進行one hot decode。
# 將原來的類別 Artribute 被拆分成多個二元 Artribute，方便模型進行訓練
data_x = pd.get_dummies(data_x)
```

```
# 實例化 LabelEncoder
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
# 將類別型變量轉換為數值型變量 將每個類別映射到一個整數值
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.fit_transform(y_test)
```

2-1. 實驗資料 - 訓練:測試 4:1 、 random_state = 42

```
# 將資料集劃分為 訓練集(80%) 和 測試集(20%)
# 隨機種子為42 確保每次分割資料時都得到相同的結果
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data_x, data_y, test_size=0.2,random_state=42)
```

2-2. 實驗資料 – 10 交叉驗證

```
from sklearn.model_selection import cross_val_score
# 使用 cross_val_score 進行交叉驗證 ( 10 交叉驗證 )
scores = cross_val_score(clf, data_x, data_y, cv=10)
```

3. 訓練、預測、評估模型

```
# 訓練模型
clf.fit(x_train, y_train_encoded)

# 進行預測
y_pred = clf.predict(x_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test_encoded, y_pred)
print("Accuracy:", accuracy)
```

- 系統 A、B：Scikit-learn – DecisionTree
  - 決策樹是一種基於樹狀結構的監督式學習算法，用於分類和回歸任務
  - 不需要對資料進行正規化或標準化，不需要假設資料服從特定的分佈
  - random_state: 確保每次運行模型時得到相同的結果

```
# 實例化 DecisionTreeClassifier 作為分類器
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=42)
```

- 系統 C、D：Scikit-learn – RandomForest
  - 隨機森林是一種集成式學習方法，用於解決分類和回歸問題

- 集成: 結合多個決策樹來提高預測的準確性和穩定性
- 穩定性、泛化能力較高
- random_state: 確保每次運行模型時得到相同的結果

```
from sklearn.ensemble import RandomForestClassifier
# 實例化 DecisionTreeClassifier 作為分類器
clf_rf = RandomForestClassifier(random_state=42)
```

- 系統 E、F：Scikit-learn – Naive Bayes GaussianNB
  - 假設每個特徵都服從高斯分佈（常態分佈），每個特徵之間是獨立的
  - 計算每個類別的事後機率，
    並選擇具有最高事後機率的類別作為預測結果
  - 適用於處理小規模資料集和高維度特徵的分類問題
  - 不需要調整超參數，對資料分佈的假設較弱

```
from sklearn.naive_bayes import GaussianNB
# 實例化 GaussianNB 作為分類器
clf = GaussianNB()
```

- 系統 G、H：Scikit-learn – Naive Bayes MultinomialNB
  - 假設特徵的分佈是多項式分佈，通常用於文字分類任務
  - 對於高維稀疏特徵的資料集效果較好
  - 對於連續特徵或非整數計數的資料較不適用

```
from sklearn.naive_bayes import MultinomialNB
# 實例化 MultinomialNB 作為分類器
clf = MultinomialNB()
```

- 系統 I、J：Scikit-learn – Naive Bayes BernoulliNB
  - 假設特徵是二元的(存在或不存在)，通常用於處理二元特徵的資料集
  - 不能處理連續特徵，並且對於特徵之間的相關性敏感

```
from sklearn.naive_bayes import BernoulliNB
# 實例化 BernoulliNB 作為分類器
clf = BernoulliNB()
```

- 系統 K、L：Scikit-learn – SVM poly
  - 使用多項式 kernel 函數進行非線性分類
  - 參數 d，表示多項式的次數，可以控制特徵空間的維度
  - C 參數(懲罰參數)和 gamma 參數(控制核函數的寬度)，用於調整模型的複雜度和泛化能力

```
from sklearn.svm import SVC  # 引入支持向量分類器
# 實例化 SVC linear 作為分類器
# linear、poly、rbf
svm_clf = SVC(kernel='poly')
```

(無使用特定參數)

- 系統 M、N：Scikit-learn – SVM linear
  - 線性 kernel 函數用於在特徵空間中找到一個超平面，將不同類別的資料分隔開來
  - 在高維度的情況下計算效率高
  - 適用於特徵空間線性可分的問題，並且具有較好的解釋性

```
svm_clf = SVC(kernel='linear')
```

- 系統 O、P：Scikit-learn – SVM rbf
  - RBF kernel 函數將特徵空間映射到高維空間中，使原本線性不可分的資料變得線性可分

```
svm_clf = SVC(kernel='rbf')
```

- 系統 Q、R、S、T、U、V：Scikit-learn – KNN
  - 計算已知樣本資料與待分類資料之間的距離
    - 通常使用歐氏距離或曼哈頓距離等距離度量方式
  - 從訓練資料集中選取 K 個最近的鄰居
  - 根據這 K 個鄰居的類別(對於分類任務)或數值(對於回歸任務)，使用多數表決法(分類)或平均值(回歸)來決定待分類資料點的類別或數值

```
from sklearn.neighbors import KNeighborsClassifier
# 創建 KNeighborsClassifier 實例
# 指定鄰居的數量，
knn_clf = KNeighborsClassifier(n_neighbors=5)
knn_clf.fit(X_train, y_train_encoded)
```

(本實驗只調整鄰居數量做分析)

[補充] 以下為可調整的參數:

在 `KNeighborsClassifier` 中，一些常用的可以調整的參數以及它們的意義如下：

1. `n_neighbors`：指定要考慮的最近鄰居的數量(預設值為5)
2. `weights`：指定在計算鄰居的權重時所使用的權重函數。 可選值包括：
   - "uniform"：所有鄰居權重相等
   - "distance"：權重與距離成反比
   - 自訂函數：可以傳遞一個自訂的函數來計算權重

3. `algorithm`：指定用於計算最近鄰居的演算法。 可選值包括：
   - "auto"：根據資料的情況自動選擇合適的演算法
   - "ball_tree"：使用BallTree演算法
   - "kd_tree"：使用KDTree演算法
   - "brute"：使用暴力搜尋方法
4. `leaf_size`：指定使用BallTree或KDTree演算法時的葉子大小(預設值為30)
5. `p`：用於Minkowski距離的冪參數 (預設值為2)
   - `p=1`：表示曼哈頓距離
   - `p=2`：表示歐氏距離
6. `metric`：指定用於計算距離的度量方式(預設值為"minkowski")
   - "manhattan" (曼哈頓距離)
   - "euclidean" (歐氏距離)

- 系統 W、X：Scikit-learn – MLPClassifier

```
from sklearn.neural_network import MLPClassifier
# 創建 MLPClassifier 實例
mlp_clf = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', solver='adam', random_state=42)
```

  - hidden_layer_sizes: 指定 MLP 模型中每個隱藏層的神經元數量
  - activation = 'relu': 指定了隱藏層和輸出層的激勵函數
    這裡使用的是 ReLU（Rectified Linear Unit）激勵函數，它在深度學習中經常使用，因為它能夠解決梯度消失的問題
  - solver = 'adam': 用於優化權重的優化器
    在這個例子中，使用的是 Adam 優化器，
    它是一種常用的隨機梯度下降優化器
  - random_state = 42:確保每次運行程式時都能得到相同的結果

- 系統 Y：Tensorflow – DNN

```
from keras.utils import to_categorical
# 將數字表示的目標變量轉換為 one-hot 編碼格式
y_train_categorical = to_categorical(y_train_encoded,4)
y_test_categorical = to_categorical(y_test_encoded,4)
```

```
# 將特徵和目標變量轉換為 NumPy 數組
x_train = np.array(x_train)
x_test = np.array(x_test)
y_train_encoded = np.array(y_train_categorical)
y_test_encoded = np.array(y_test_categorical)
```

```
model = tf.keras.Sequential()
# 向模型中添加了三個全連接層（Dense layers），分別具有 16、8 和 4 個神經元。
# 第一層需要指定輸入形狀（input_shape），即特徵的形狀
model.add(tf.keras.layers.Dense(16, input_shape=(x_train.shape[1],), activation='sigmoid'))
model.add(tf.keras.layers.Dense(8, activation='sigmoid'))
model.add(tf.keras.layers.Dense(4, activation='softmax'))
model.summary()
```

1. tf.keras.layers.Dense(16, input_shape=(x_train.shape[1],), activation='sigmoid'):

- 模型的第一個全連接層，具有 16 個神經元
- input_shape = (x_train.shape[1],) 指定輸入特徵的形狀
  x_train.shape[1]表示特徵的數量，即輸入層的大小
- activation = 'sigmoid' 指定了使用 sigmoid 激活函數

2. tf.keras.layers.Dense(8, activation='sigmoid'):
   - 模型的第二個全連接層，具有 8 個神經元
   - 不需要再指定輸入形狀，因為 Keras 會自動從前一層推斷輸入形狀
   - 仍然使用 sigmoid 激活函數
3. tf.keras.layers.Dense(4, activation='softmax'):
   - 模型的輸出層，具有 4 個神經元，對應於目標變量的類別數量
   - 使用 softmax 激活函數，將神經元的輸出轉換為每個類別的機率

```
# 創建一個Adam優化器的實例 ， 指定學習率的初始值 = 0.001
opt = tf.keras.optimizers.Adam(learning_rate=0.001)
# 將優化器、損失函數和評估指標添加到模型中
# 1. 使用 Adam 優化器（optimizer）編譯模型
# 2. 指定損失函數（loss function）為分類交叉熵（Categorical Crossentropy）
# 3. 加入評估指標，這裡使用的是分類準確率（Categorical Accuracy）
model.compile(optimizer=opt, # from_logits = True表示模型的輸出未經過softmax激活函數轉換
              loss=tf.losses.CategoricalCrossentropy(from_logits=True),
              metrics=[tf.metrics.CategoricalAccuracy(name='accuracy')])
```

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# 模型的訓練過程
history = model.fit(x_train, # 訓練資料集的特徵資料
            y_train_encoded,# 訓練資料集的目標
            epochs=40, # 進行 40 輪訓練
            batch_size=20, # 每輪訓練使用 20 個樣本
            # 在每個訓練輪次結束時，使用測試資料集來驗證模型的表現
            validation_data=(x_test, y_test_encoded),
            verbose=1) # 以詳細模式顯示訓練過程
```

[補充]

batch_size = 20

- 在每個訓練步驟中，模型將使用 20 個樣本進行前向傳播(Forward Propagation)和反向傳播(Backward Propagation)，並更新模型的參數

verbose = 1

- 顯示每個訓練輪次的進度條，以及訓練和驗證集上的損失和指標值

- 系統 Z：Tensorflow – CNN

```
# 載入 MNIST 數據集
(x_train, y_train), (x_test, y_test) = datasets.mnist.load_data()

# 對數據進行正規化和形狀調整
# 將像素值縮放到0到1之間，並將資料的形狀調整為(28, 28, 1)的格式，以符合CNN的輸入要求
x_train = x_train.reshape((60000, 28, 28, 1)).astype('float32') / 255
x_test = x_test.reshape((10000, 28, 28, 1)).astype('float32') / 255

# 將類別標籤進行 one-hot 編碼 (二進制向量)
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)
```

```
# 定義 CNN 模型
def create_cnn_model(input_shape, num_classes):
    model = models.Sequential()
    # 添加卷積層和池化層
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    # 展平特徵圖
    model.add(layers.Flatten())
    # 添加全連接層
    model.add(layers.Dense(64, activation='relu'))
    # 輸出層
    model.add(layers.Dense(num_classes, activation='softmax'))
    return model
```

定義 CNN 模型，包括多個卷積層、池化層、全連接層和輸出層。

模型的架構包括：

- 三個卷積層：每個卷積層都包含 32、64 和 64 個 3x3 的卷積核，使用 ReLU 激活函數
- 兩個池化層：每個池化層使用 2x2 的池化窗口進行最大池化
- 一個全連接層：包含 64 個神經元，使用 ReLU 激活函數
- 輸出層：使用 softmax 激活函數，輸出類別機率

```
# 指定輸入形狀和類別數量
input_shape = (28, 28, 1)  # MNIST 數據集的輸入形狀為 (28, 28, 1)
# 對於 MNIST 數據集，每個圖像都屬於 0 到 9 中的一個類別
num_classes = 10  # 類別數量為 10

# 創建模型
model = create_cnn_model(input_shape, num_classes)

# 編譯模型
model.compile(optimizer='adam', # 使用Adam優化器
              loss='categorical_crossentropy', # 損失函數為分類交叉熵
              metrics=['accuracy']) # 評估指標為準確率
```

```
# 訓練模型
history = model.fit(x_train, y_train,
                    epochs=5, # 訓練5個epochs
                    batch_size=64, # 每個batch包含64個樣本
                    validation_data=(x_test, y_test))
```

- 系統 AA：Tensorflow – MLP

```
# 定義 MLP 模型
# 創建一個序列模型，並添加三個全連接層（Dense layers）
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, input_shape=(X_train.shape[1],), activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax')
])
```

層數	神經元個數	激活函數
第一層	64	ReLU
第二層	32	ReLU
第三層	4	softax

```
# 編譯模型
model.compile(optimizer='adam', # 使用Adam優化器
              loss='categorical_crossentropy', # 損失函數為分類交叉熵
              metrics=['accuracy']) # 評估指標為準確率（accuracy）
```

```
import tensorflow as tf
# 對目標數據進行 one-hot 編碼
y_train_encoded = tf.keras.utils.to_categorical(y_train, num_classes=4)
y_test_encoded = tf.keras.utils.to_categorical(y_test, num_classes=4)
```

```
# 訓練模型
history = model.fit(X_train,
                    y_train_encoded,
                    epochs=10, # 訓練10個epochs
                    batch_size=32, # 每個batch包含32個樣本
                    validation_data=(X_test, y_test_encoded))
```

[補充]

常見的評估指標

1. 準確率（accuracy）：模型正確分類的**樣本數**與**總樣本數**之比
2. 精確率（precision）：被**正確預測**為正類別的樣本數與**所有被預測**為正類別的樣本數之比
3. 召回率（recall）：被**正確預測**為正類別的樣本數與**所有真實**為正類別的樣本數之比
4. F1 分數（F1-score）：**精確率**和**召回率**的**調和平均數**，可看作是精確率和召回率的**加權平均數**
5. ROC-AUC（ROC 曲線下的面積）：ROC 曲線下的面積，用於評估**二元分類**模型的性能
6. PR-AUC（PR 曲線下的面積）：PR 曲線下的面積，用於評估**二元分類**模型的性能

(三)結論

1. 在使用 Scikit-learn 中的不同機器學習模型上，DecisionTree 和 RandomForest 在訓練 vs 測試(4:1)的準確率都較高，而且在 10-fold cross-validation 中也保持了較高的準確率。

2. Naive Bayes 方法的表現相對較差，特別是 GaussianNB，其準確率遠低於其他模型。

3. 在使用 SVM 模型時，linear 核函數的準確率較高，而 poly 和 rbf 核函數的準確率較低。

4. 在 KNN 模型中，鄰居數量 5、10、20 對準確率並沒有太大的影響，但是可以發現在訓練 vs 測試(4:1)的準確率都較 10-fold cross-validation 高

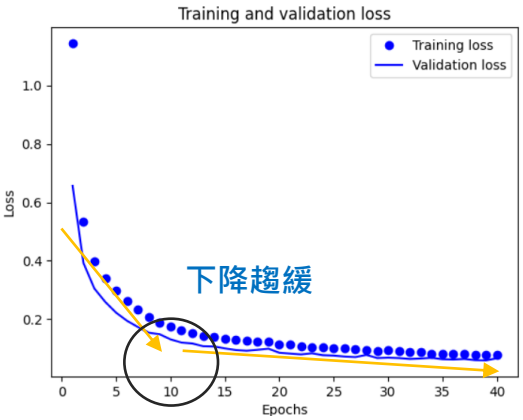
5. MLPClassifier 在訓練與測試(4:1) 以及 10-fold cross-validation 中的準確率都相對較高。

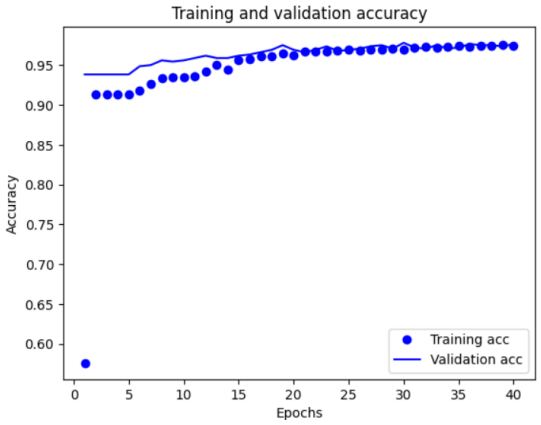
6. TensorFlow 的 CNN 模型的準確率較其他 TensorFlow 模型來得高。

7. 透過以下圖形可以發現，訓練至 10 個 epochs 時為最佳值。

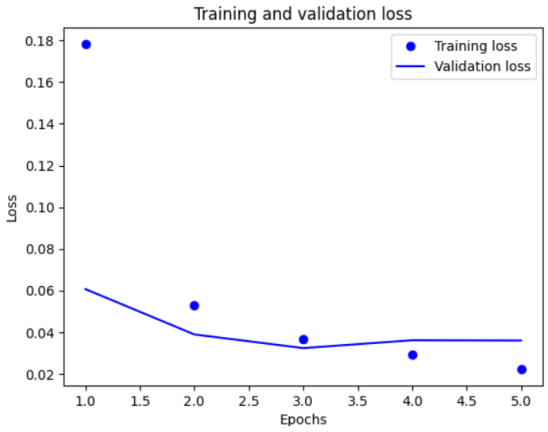
TensorFlow 模型作圖:

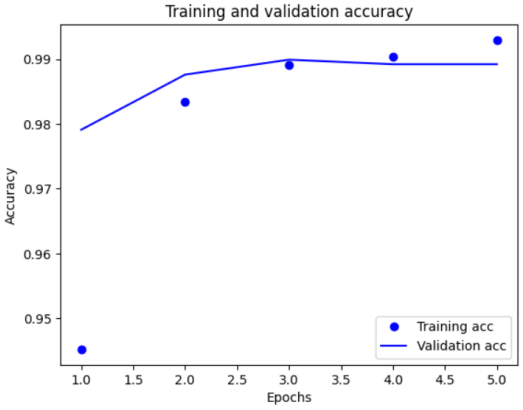
- DNN 模型 - 訓練 40 個 epochs



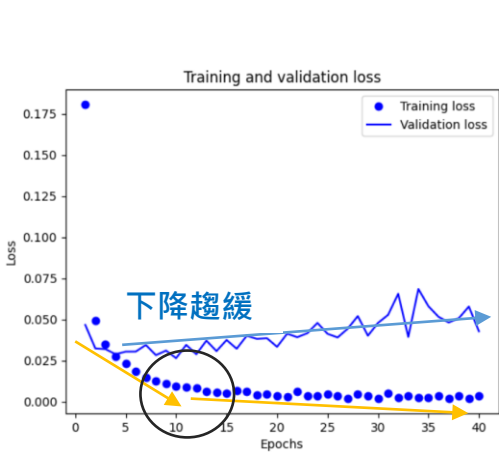


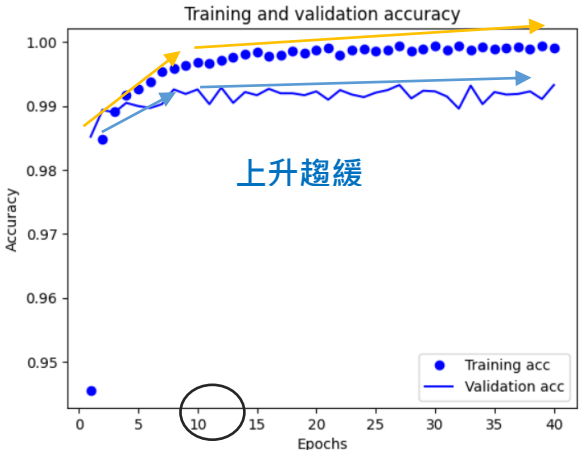
- CNN 模型 - 訓練 5 個 epochs



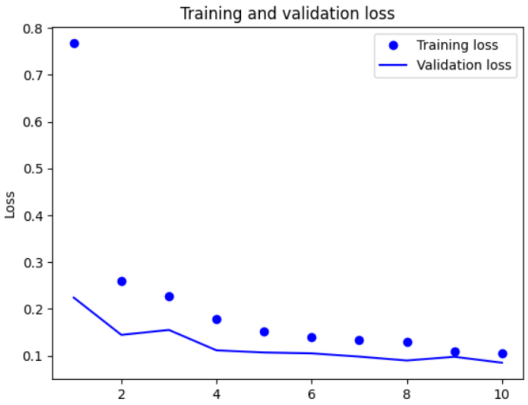


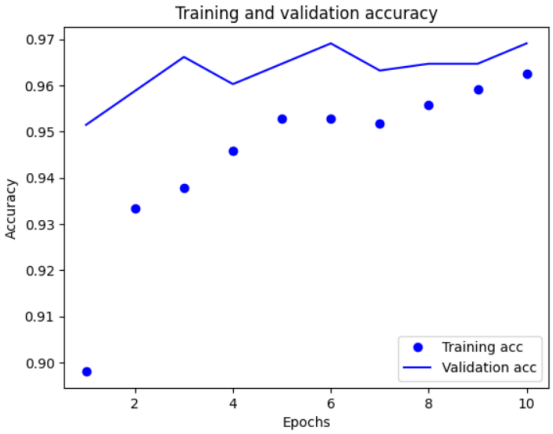
- CNN 模型 - 訓練 40 個 epochs (模型的學習已經趨於飽和)



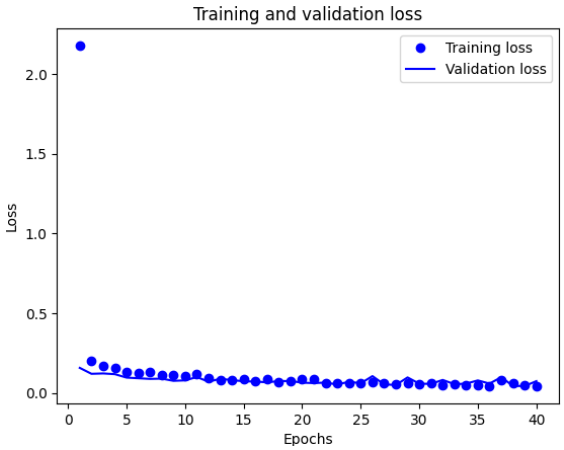


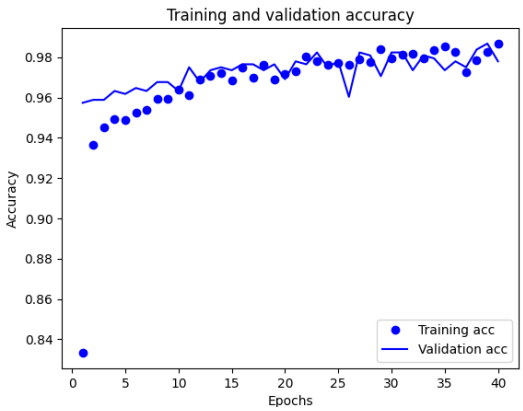
- MLP 模型 - 訓練 10 個 epochs





- MLP 模型 - 訓練 40 個 epochs





Q: 哪種系統或哪種設定對本套實驗資料較有效?

A: 根據以上分析，對於這套實驗資料，Scikit-learn 的 DecisionTree(99.85%)、RandomForest(99.56%)、MLPClassifier(98.26%) 和 TensorFlow 的 CNN(99.33%) 模型是最有效的系統。

Q: Train loss 與 Valid loss 之間的關係?

A: 當訓練損失和驗證損失都在下降並保持相對接近時，表示模型在訓練集和驗證集上都有良好的表現。如果驗證損失持續下降，而訓練損失趨於穩定(例如: CNN 模型 - 訓練 40 個 epochs)，可能表明模型的學習已經趨於飽和，或者需要調整正則化參數以改善泛化能力。

(四)參考資料

1. Supervised learning — scikit-learn 1.4.2 documentation

2. All symbols in TensorFlow 2 | TensorFlow v2.16.1

3. 深度学习当中 train loss 和 valid loss 之间的关系？ - 知乎 (zhihu.com)