國立彰化師範大學 113 學年度

資訊管理學系人工智慧課程

學期筆記

指導教授：翁政雄

學生：陳毓欣

目錄

20240909. 臉部偵測、人臉辨識

壹、 相關知識

一、 流程：讀檔 → 找圖片中的臉的部分 → 確定他是誰

二、 資料量下降（特徵擷取）是關鍵

三、 訓練模型（face.py）耗時最久

貳、 程式碼解讀

一、 載入 Haar Cascade 模型：

1. 使用 cv2.CascadeClassifier 來載入人臉檢測的 Haar 特徵模型
2. 預設使用 haarcascade_frontalface_default.xml 檔案

二、 建立 LBPH 人臉識別模型：

使用 cv2.face.LBPHFaceRecognizer_create()創建 Local Binary Patterns Histogram (LBPH)的人臉識別器

三、 載入模型與人臉檢測器：

1. 使用 cv2.face.LBPHFaceRecognizer_create()啟用人臉識別器，並讀取訓練好的模型檔案 face2.yml
2. 加載 Haar Cascade 人臉檢測模型 haarcascade_frontalface_default.xml

四、 開啟攝影機：

使用 cv2.VideoCapture(0)打開默認攝影機。如果攝影機無法開啟，程式會提示錯誤並退出

五、 讀取影像並進行人臉檢測：

1. 從攝影機讀取影像，縮放到 540x300（提高效能）
2. 轉換成灰度影像以提高檢測速度與準確性
3. 使用 face_cascade.detectMultiScale()檢測影像中的人臉，返回人臉的外框座標

六、 人臉識別：

1. 迴圈處理每個檢測到的人臉
2. 使用 recognizer.predict()進行人臉識別，返回 ID 和信心值（confidence）
3. 如果信心值小於 60，視為有效識別，並顯示對應的名字；否則顯示???

七、 讀取圖片進行人臉檢測：

1. 載入 face01/9.jpg，轉換為灰度影像以加速檢測
2. 用 detectMultiScale 檢測人臉，返回所有檢測到的人臉座標

參、 附錄 – 程式碼

一、 PicRead

```
import cv2
import numpy as np
#detector =
cv2.CascadeClassifier('xml/haarcascade_frontalface_default.xml')
#載入人臉追蹤模型
#filename = 'C:/Users/petew/anaconda3/envs/TF/Lib/site-
packages/cv2/data/haarcascade_frontalface_default.xml'
filename =    'haarcascade_frontalface_default.xml'
detector = cv2.CascadeClassifier(filename)
#載入人臉追蹤模型
recog = cv2.face.LBPHFaceRecognizer_create()
# 啟用訓練人臉模型
faces = [] #儲存人臉模型位置大小的串列
ids = [] #紀錄該人臉 id 的串列

for i in range(1,11):

    #img = cv2.imread(f'face01/{i}.jpg')
#依序開啟每一張蔡英文的照片
    #img = cv2.imread('D:/PythonCode/Opencv/face01/{i}.jpg')
#依序開啟每一張蔡英文的照片
    #filename = f'D:/PythonCode/Opencv/face01/{i}.jpg'
#讀取圖片檔案
    filename = f'face01/{i}.jpg'
#讀取圖片檔案
    print(filename)
    img = cv2.imread(filename)
    cv2.imshow('My Image', img)

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#色彩轉換成黑白
    img_np = np.array(gray,'uint8')
    face = detector.detectMultiScale(gray)
```

```
#按下任意鍵則關閉所有視窗
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    '''
    str=f'D:/PythonCode/Opencv/face01/{i}.jpg'
    print(str)
    '''
```

二、 face

```
#pip uninstall opencv-python
#pip install opencv-contrib-python

import cv2
import numpy as np
#detector =
cv2.CascadeClassifier('xml/haarcascade_frontalface_default.xml')
#載入人臉追蹤模型
#filename = 'C:/Users/petew/anaconda3/env3/TF/Lib/site-
packages/cv2/data/haarcascade_frontalface_default.xml'
filename = 'haarcascade_frontalface_default.xml'
detector = cv2.CascadeClassifier(filename)
#載入人臉追蹤模型

recog = cv2.face.LBPHFaceRecognizer_create()
#啟用訓練人臉模型方法
faces = []
ids = []

#注意圖片張數
for i in range(1,17):
        #img = cv2.imread(f'face01/{i}.jpg')
#依序開啟每一張蔡英文的照片
        #filename = f'D:/PythonCode/Opencv/face01/{i}.jpg'
        filename = f'face01/{i}.jpg'
        print(filename)
        img = cv2.imread(filename)
#依序開啟每一張蔡英文的照片
        #cv2.imshow('My Image', img)
```

```
#顯示圖片，可以省略或是註解


        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#色彩轉換成黑白（加大效率，前提是黑白跟彩色的辨識程度差不多）
        img_np = np.array(gray,'uint8')
        face = detector.detectMultiScale(gray)
        for(x,y,w,h) in face:
            faces.append(img_np[y:y+h,x:x+w])
            ids.append(1)

#注意圖片張數
for i in range(1,11):
        #img = cv2.imread(f'face02/{i}.jpg')
#依序開啟每一張蔡英文的照片
        #filename = f'D:/PythonCode/Opencv/face02/{i}.jpg'
        filename = f'face02/{i}.jpg'
        print(filename)
        img = cv2.imread(filename)
#依序開啟每一張蔡英文的照片


        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#色彩轉換成黑白
        img_np = np.array(gray,'uint8')
        face = detector.detectMultiScale(gray)
        for(x,y,w,h) in face:
            faces.append(img_np[y:y+h,x:x+w])
            ids.append(2)


print('training...')
recog.train(faces,np.array(ids))
recog.save('face1.yml')
#face.yml
print('OK!')
```

三、 detection（使用相機）

```
#pip install opencv-contrib-python
import cv2
```

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
#啟用訓練人臉模式方法
recognizer.read('face2.yml')
#讀取人臉模型檔案
#cascade_path = "xml/haarcascade_frontalface_default.xml"
#載入人臉追蹤模型
#filename = 'C:/Usres/petew/anaconda3/envs/TF/Lib/site-
package/cv2/data/haarcascade_frontalface_default.xml'
filename = 'haarcascade_frontalface_default.xml'
cascade_path=filename

face_cascade = cv2.CascadeClassifier(cascade_path)
#啟用人臉追蹤

cap = cv2.VideoCapture(0)
#開啟攝影機
if not cap.isOpened():
    print("Cannot open camera")
    exit()
while True:
    ret, img = cap.read()
    if not ret:
        print("Cannot receive frame")
        break
    img = cv2.resize(img, (540,300))
#縮小尺寸，加快辨識效率
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#轉換成黑白
    faces = face_cascade.detectMultiScale(gray)
#追蹤人臉(目的在於標記出外框)

    #建立姓名和 ID 的對照表
    name = {
        '1':'Tsai',
        '2':'Trump',
        '3':'MA',
        '4':'???'
```

```
            }

        #依序判斷每張臉屬於哪個 ID
        for(x,y,w,h) in faces:
            cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
#標記人臉外框
            idnum,confidence = recognizer.predict(gray[y:y+h,x:x+w])
#取出 id 號碼以及信心指數 confidence
            if    confidence < 60:
                text = name[str(idnum)]
#如果信心指數小於 60，取得相應的名字
            else:
                text = '???'
#不然名字就是 ???
            # 在人臉外框旁加上名字
            cv2.putText(img, text, (x,y-
5),cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2,
cv2.LINE_AA)

        cv2.imshow('oxxostudio', img)
        if cv2.waitKey(5) == ord('q'):
            break
#按下 q 鍵停止

cap.release()
cv2.destroyAllWindows()
```

四、 detection_pic（偵測圖片）

```
#人臉辨識
#讀取照片，辨識人臉身分
#designed by 翁政雄 博士/教授

#pip install opencv-contrib-python

import cv2

#(A) 載入 人臉辨識模型(face.yml)，face.py 進行訓練
recognizer = cv2.face.LBPHFaceRecognizer_create()
```

```
#啟用訓練人臉模型方法
recognizer.read('face.yml')
#讀取人臉模型檔

#(B) 載入 人臉擷取模型， 擷取人臉影像(讀取人臉圖片 或 開啟攝影機)
filename='haarcascade_frontalface_default.xml'
cascade_path=filename
face_cascade = cv2.CascadeClassifier(cascade_path)
#啟用人臉追蹤

#(C) 人臉影像擷取(讀取人臉圖片)
filename='face01/9.jpg'
#讀取人臉圖片 或 開啟攝影機 擷取圖片
print(filename)
img = cv2.imread(filename)

gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#轉換成黑白
faces = face_cascade.detectMultiScale(gray)
#追蹤人臉(目的在於標記出外框)
print(len(faces))
#僅擷取 1 張圖片，len=1

#建立姓名和 id 的對照表
name = {
    '1':'Tsai',
    '2':'Trump',
    '3':'???'
}

#(D)辨識 圖片 及 顯示 姓名
for(x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
#標記人臉外框
    idnum,confidence = recognizer.predict(gray[y:y+h,x:x+w])    #
取出 id 號碼以及信心指數 confidence
```

```
        if    confidence < 60:
                text = name[str(idnum)]
#如果信心指數小於 60，取得相應的名字
        else:
                text = '???'
#不然名字就是 ???
        # 在人臉外框旁加上名字
        cv2.putText(img, text, (x,y-
5),cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2,
cv2.LINE_AA)
        print(text)

while True:
        cv2.imshow('Face Detyection', img)
        if cv2.waitKey(5) == ord('q'):
                break
#按下 q 鍵停止
cv2.destroyAllWindows()
```

20240916. 車牌辨識

壹、 相關知識

一、 當原始資料集資料筆數不足時，解決方案如下：

1. 加入雜訊
2. 使用其他資料集補充
3. 使用移動平均填充
4. 使用機器學習/深度學習預測法先預測一次

二、 蒙地卡羅方法（Monte Carlo Method）是一種基於隨機數模擬的數值計算方法，用於解決包含隨機性或複雜概率分佈的問題。該方法以概率統計為基礎，通過隨機抽樣和重複試驗來逼近問題的解決方案。常見應用於機器學習，例：強化學習和遊戲人工智慧中（如 AlphaGo）。

貳、程式碼解讀

一、 人臉檢測與圖像處理
1. 使用 CascadeClassifier 檢測人臉區域，並對圖像進行灰度轉換
2. 採用 OpenCV 的 cv2.CascadeClassifier 和 LBPHFaceRecognizer_create()訓練面部識別模型

二、 圖像大小調整(resize)
1. 調整目標文件夾中的圖像大小以統一格式
2. 提及使用 PIL.Image.LANCZOS 方法進行高質量縮放

三、 車牌檢測與裁剪
1. 使用 Haar 特徵檢測模型檢測車牌
2. 對檢測到的車牌進行裁剪並保存

四、 車牌數字識別
1. 進一步處理裁剪後的車牌圖片，分割出各個字符
2. 將分割的字符輸出為獨立文件，方便後續訓練

五、 模型訓練與測試
1. 使用 Keras 建立卷積神經網絡(CNN)訓練模型，用於車牌字符的識別
2. 提供了一個驗證準確性的步驟

六、 車牌識別應用
1. 利用訓練好的模型進行車牌字符辨識
2. 圖像分割處理後，每個字符送入模型，輸出最終車牌號碼

參、附錄 – 程式碼

一、 picRead

```
import cv2
import numpy as np
#detector =
cv2.CascadeClassifier('xml/haarcascade_frontalface_default.xml') #載入人臉追蹤
#filename='C:/Users/petew/anaconda3/envs/TF/Lib/site-
```

```
package/cv2/data/haarcascade_frontalface_default.xml
filename = 'haarcascade_frontalface_default.xml'
detector = cv2.CascadeClassifier(filename) #載入人臉追蹤模型
recog = cv2.face.LBPHFaceRecognizer_create() #啟用訓練人臉模型方法
faces = [] #儲存人臉位置大小的串列
ids = []

for i in range(1,11):
    #img = cv2.imread(f'face01/{i}.jpg) #依序開啟每一張蔡英文的照片
    #img = cv2.imread('D:/PythonCode/Opencv/face01/{i}.jpg') #依序開啟每一張蔡英文的照片
    #filename = f'D:/PythonCode/Opencv/face01/{i}.jpg' #讀取圖片檔案
    filename = f'face01/{i}.jpg' #讀取圖片檔案
    print(filename)
    img = cv2.imread(filename)
    #cv2.imshow('My Image', img) #顯示圖片，可以省略或是註解

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #色彩轉換為黑白
    img_np = np.array(gray, 'uint8') #轉換為指定編碼的 numpy 陣列
    face = detector.detectMultiScale(gray) #擷取人臉區域

    #按下任意鍵則關閉所有視窗
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    '''
    str = f'D:/PythonCode/Opencv/face01/{i}.jpg'
    print(str)
    '''
```

二、 resize

```
def emptydir(dirname):                    #清空資料夾
    if os.path.isdir(dirname):            #資料夾存在就刪除
```

```
            shutil.rmtree(dirname)
            sleep(2)                                    #需延遲否則會
出錯
        os.mkdir(dirname)                               #建立資料夾

def dirResize(src,dst):
        myfiles = glob.glob(src + '/*.JPG')         #讀取資料夾全部 jpg
檔案
        emptydir(dst)
        print(src + '  資料夾：')
        print('開始轉換圖形尺寸！')
        for f in myfiles:
            fname = f.split("\\")[-1]
            img = Image.open(f)
            #img_new = img.resize((300, 225),
PIL.Image.ANTIALIAS) #ANTIALIAS 已經停用
            img_new = img.resize((300, 225), PIL.Image.LANCZOS) #
尺寸 300x225

            img_new.save(dst + '/' + fname)
            print('fname:',fname)
        print('轉換圖形尺寸完成！\n')

import PIL
from PIL import Image
import glob
import shutil, os
from time import sleep

dirResize('realPlate_sr', 'realPlate')
dirResize('predictPlate_sr', 'predictPlate')
```

三、 cropPlate

```
def emptydir(dirname):   #清空資料夾
        if os.path.isdir(dirname):   #資料夾存在就刪除
            shutil.rmtree(dirname)
            sleep(2) #需延遲否則會出錯
        os.mkdir(dirname) #建立資料夾
```

```
import cv2
from PIL import Image
import glob
import shutil, os
from time import sleep

print('開始擷取車牌！')
print('無法擷取車牌圖片：')
dstdir = 'cropPlate'
myfiles = glob.glob("realPlate\*.JPG")
emptydir(dstdir)
for imgname in myfiles:
    filename = (imgname.split('\\'))[-1]      #取得檔案名稱
    img = cv2.imread(imgname) #讀入圖形
    detector = cv2.CascadeClassifier('haar_carplate.xml')
    signs = detector.detectMultiScale(img, scaleFactor=1.1,
minNeighbors=4, minSize=(20, 20))     #框出車牌
    #割取車牌
    if len(signs) > 0 :
        for (x, y, w, h) in signs:
            image1 = Image.open(imgname)
            image2 = image1.crop((x, y, x+w, y+h))   #擷取車牌
圖形
            #image3 = image2.resize((140, 40),
Image.ANTIALIAS)
            image3 = image2.resize((140, 40), Image.LANCZOS)
#轉換尺寸為 140x40
            image3.save(dstdir + '/tem.jpg')      #轉換尺寸為
140x40
            image4 = cv2.imread(dstdir + '/tem.jpg')      #以
opencv 讀車牌檔
            img_gray = cv2.cvtColor(image4,
cv2.COLOR_RGB2GRAY) #灰階
            _, img_thre = cv2.threshold(img_gray, 100, 255,
cv2.THRESH_BINARY)   #黑白
            cv2.imwrite(dstdir + '/' + filename, img_thre)
```

```
                print('fname:', filename)

            else:
                print(filename)

os.remove(dstdir + '/tem.jpg')   #移除暫存檔
print('擷取車牌結束！')
```

四、 cropNum

```
def emptydir(dirname):   #清空資料夾
    if os.path.isdir(dirname):   #資料夾存在就刪除
        shutil.rmtree(dirname)
        sleep(2) #需延遲否則會出錯
    os.mkdir(dirname) #建立資料夾

import cv2
import shutil, os
from time import sleep

emptydir('cropMono')
image = cv2.imread('cropPlate/7238N2.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)   #灰階
_,thresh = cv2.threshold(gray, 127, 255,
cv2.THRESH_BINARY_INV) #轉為黑白
contours1 = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) #尋找輪
廓
contours = contours1[0] #取得輪廓
letter_image_regions = []    #文字圓形串列
for contour in contours:      #依序處理輪廓
    (x, y, w, h) = cv2.boundingRect(contour)      #單一輪廓資料
    letter_image_regions.append((x, y, w, h))    #輪廓資料加入串
列
letter_image_regions = sorted(letter_image_regions, key=lambda x:
x[0]) #按 x 座標排序
print(letter_image_regions)
#存檔
i=1
```

```
for letter_bounding_box in letter_image_regions:      #依序處理輪廓資料
    x, y, w, h = letter_bounding_box
    if w>=5 and h>32 and h<40:   #長度>6 且高度在 33-39 才是文字
        letter_image = gray[y:y+h, x:x+w]     #擷取圖型
        letter_image = cv2.resize(letter_image, (18,38))
        cv2.imwrite('cropMono/{}.jpg'.format(i), letter_image)   #存檔
        i+=1
```

五、 cropNum_all

```
def emptydir(dirname):   #清空資料夾
    if os.path.isdir(dirname):   #資料夾存在就刪除
        shutil.rmtree(dirname)
        sleep(2) #需延遲否則會出錯
    os.mkdir(dirname) #建立資料夾

import cv2
import glob
import shutil, os
from time import sleep

print('開始擷取車牌數字！')
emptydir('cropNum')
myfiles = glob.glob("cropPlate\*.jpg")
for f in myfiles:
    filename = (f.split('\\'))[-1].replace('.jpg', '')
    emptydir('cropNum/' + filename)
    image = cv2.imread(f)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _,thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV) #轉為黑白
    contours1 = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) #尋找輪廓
    contours = contours1[0] #取得輪廓
    letter_image_regions = []     #文字圓形串列
```

```
    for contour in contours:       #依序處理輪廓
        (x, y, w, h) = cv2.boundingRect(contour)       #單一輪廓資料
        letter_image_regions.append((x, y, w, h))     #輪廓資料加入串列
    letter_image_regions = sorted(letter_image_regions, key=lambda x: x[0]) #按 x 座標排序
    print(letter_image_regions)
    #存檔
    i=0
    for letter_bounding_box in letter_image_regions:       #依序處理輪廓資料
        x, y, w, h = letter_bounding_box
        if w>=5 and h>32 and h<40:   #長度>6 且高度在 33-39 才是文字
            letter_image = gray[y:y+h, x:x+w]     #擷取圖型
            letter_image = cv2.resize(letter_image, (18,38))
            cv2.imwrite('cropMono/{}.jpg'.format(i), letter_image) #存檔
            i+=1
            print('fname:',filename, '',i='', i)
print('擷取車牌數字結束！')
```

六、 makefont

```
def emptydir(dirname):   #清空資料夾
    if os.path.isdir(dirname):   #資料夾存在就刪除
        shutil.rmtree(dirname)
        sleep(2) #需延遲否則會出錯
    os.mkdir(dirname) #建立資料夾

import glob
import shutil, os
from time import sleep

print('開始建立文字庫！')
emptydir('platefont')
fontlist = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
```

```
'Z']
for i in range(len(fontlist)):  #建立文字資料夾
    emptydir('platefont/' + fontlist[i])
dirs = os.listdir('cropNum')    #讀取所有檔案及資料夾
picnum = 1  #開啟計數器，讓檔名不會重複
for d in dirs:
    if os.path.isdir('cropNum/' + d):    #只處理資料夾
        myfiles = glob.glob('cropNum/' + d + '/*.jpg')
        for i, f in enumerate(myfiles):
            shutil.copyfile(f, 'platefont/{}/{}.jpg'.format(d[i],
picnum))  #存入對應資料夾
            picnum += 1
print('建立文字庫結束！')
```

七、 makedata

```
def emptydir(dirname):  #清空資料夾
    if os.path.isdir(dirname):  #資料夾存在就刪除
        shutil.rmtree(dirname)
        sleep(2) #需延遲否則會出錯
    os.mkdir(dirname) #建立資料夾

import cv2
import random
import glob
import shutil, os
from time import sleep

fontlist = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
'Z']
print('開始建立訓練資料！')
emptydir('data')
for n in range(len(fontlist)):
    print('產生 data/' + fontlist[n] + ' 資料夾')
    emptydir('data/' + fontlist[n])
    myfiles = glob.glob('platefont/' + fontlist[n] + '/*.jpg')
    for index, f in enumerate(myfiles):
        pic_total = 500
```

```
            pic_each = int(pic_total / len(myfiles)) + 1
            for i in range(pic_each):
                img = cv2.imread(f)
                for j in range(20):
                    x = random.randint(0, 17)
                    y = random.randint(0, 37)
                    cv2.circle(img, (x, y), 1, (0, 0, 0), -1)
                cv2.imwrite('data/', fontlist[n] +
'/{:0>4d}.jpg'.format(index*pic_each+i+1, img))
print('建立訓練者資料結束！')
```

八、 train

```
#pip install opencv-python
#pip install imutils

import cv2
import os.path
import numpy as np
from imutils import paths
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Flatten, Dense

#取消警告訊息
import warnings
warnings.filterwarnings("ignore")

imagedir = "data" #訓練資練
modelname = "carplate_model.hdf5" #模型名稱
data = []    #資料串列
labels = [] #標籤串列

#讀取資料
for image_file in paths.list_images(imagedir):
    image = cv2.imread(image_file)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
        label = image_file.split(os.path.sep)[-2]
        data.append(image)
        labels.append(label)
data = np.array(data)
labels = np.array(labels)

#訓練資料佔 85%
(X_train, X_test, Y_train, Y_test) = train_test_split(data, labels,
test_size=0.15, random_state=0)
#標準化資料
X_train_normalize = X_train.reshape(X_train.shape[0], 38, 18,
1).astype("float") / 255.0
X_test_normalize = X_test.reshape(X_test.shape[0], 38, 18,
1).astype("float") / 255.0
#轉換標籤為 one-hot
lb = LabelBinarizer().fit(Y_train)
Y_train_OneHot = lb.transform(Y_train)
Y_test_OneHot = lb.transform(Y_test)

#建立模型
model = Sequential()
#神經網路
model.add(Conv2D(20, (5, 5), padding="same", input_shape=(38, 18,
1), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strices=(2, 2)))
model.add(Conv2D(50, (5, 5), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strices=(2, 2)))
model.add(Flatten())
model.add(Dense(500, activation="relu"))
model.add(Dense(34, activation="softmax"))
model.compile(loss="categorical_crossentropy", optimizer="adam",
metrics=["accuracy"])
#開始訓練
model.fit(X_train_normalize, Y_train_OneHot, validation_split=0.2,
batch_size=32, epochs=10, verbose=1)
model.save(modelname) #儲存模型
```

```
#準確率
scores = model.evaluate(X_train_normalize, Y_train_OneHot)
print(scores[1])
scores2 = model.evaluate(X_test_normalize, Y_test_OneHot)
print(scores2[1])
```

九、 recogPlate

```
def emptydir(dirname):   #清空資料夾
    if os.path.isdir(dirname):   #資料夾存在就刪除
        shutil.rmtree(dirname)
        sleep(2) #需延遲否則會出錯
    os.mkdir(dirname) #建立資料夾

from keras.models import load_model
from PIL import Image
import numpy as np
import cv2
import shutil, os
from time import sleep

labels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
'Z']
#擷取車牌
imgname = '3M6605.jpg' #預測錯誤
imgname = '1710YC.jpg' #預測正確
print('車牌號碼(真實)：', imgname)
dirname = 'recogdata'
emptydir(dirname)
img = cv2.imread('predictPlate/' + imgname)
#img = cv2.imread('predictPlate/' + imgname,encoding='utf-8')
detector = cv2.CascadeClassifier('haar_carplate.xml')
signs = detector.detectMultiScale(img, scaleFactor=1.1,
minNeighbors=4, minSize=(20, 20))
if len(signs) > 0 :
    for (x, y, w, h) in signs:
        image1 = Image.open('predictPlate/' + imgname)
        image2 = image1.crop((x, y, x+w, y+h))
```

```
            #image3 = image2.resize((140,40), Image.ANTIALIAS)
            image3 = image2.resize((140, 40), Image.LANCZOS)
            image3.save('tem.jpg')
            image4 = cv2.imread('tem.jpg')
            gray = cv2.cvtColor(image4, cv2.COLOR_RGBA2GRAY)
            _, img_thre = cv2.threshold(gray, 100, 255,
cv2.THRESH_BINARY)
            cv2.imwrite('tem.jpg', img_thre)
      #分割文字
      img_tem = cv2.imread('tem.jpg')
      gray = cv2.cvtColor(img_tem, cv2.COLOR_BGR2GRAY)
      _, thresh = cv2.threshold(gray, 127, 255,
cv2.THRESH_BINARY_INV)
      contours1 = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
      contours = contours1[0]
      letter_image_regions = []
      for contour in contours:
            (x, y, w, h) = cv2.boundingRect(contour)
            letter_image_regions.append((x, y, w, h))
      letter_image_regions = sorted(letter_image_regions, key=lambda
x: x[0])
      #存檔
      i = 1
      for letter_bounding_box in letter_image_regions:
            x, y, w, h = letter_bounding_box
            if w>=5 and h>32 and h<40:
                  letter_image = gray[y:y+h, x:x+w]
                  letter_image = cv2.resize(letter_image, (18, 38))
                  cv2.imwrite(dirname + '/{}.jpg'.format(i), letter_image)
                  i += 1
      #辨識車牌
      detan = 0
      for fname in os.listdir(dirname):
            if os.path.isfile(os.path.join(dirname, fname)):
                  detan += 1
      tem_data = []
```

```
    for index in range(1, (detan+1)):
        tem_data.append((np.array(Image.open("recogdata/" +
str(index) + ".jpg")))/255.0)
    real_data = np.stack(tem_data)
    real_data1 = np.expand_dims(real_data, axis=3)
    model = load_model("carplate_model.hdf5")

    #翁老師修改 開始
    #predictions = model.predict_classes(real_data1)
    predictions = model.predict(real_data1)
    pred = predictions
    #Mutli-class Classfication
    #pred = model.predict(X[val_index])
    pred = np.argmax(pred, axis=1)
    print(pred)
    predictions = pred
    #翁老師修改結束

    print('車牌號碼為(預測):')
    for i in range(len(predictions)):
        print(labels[int(predictions[i])], end='')
```

十、 recogPlate_weng

```
def emptydir(dirname):   #清空資料夾
    if os.path.isdir(dirname):   #資料夾存在就刪除
        shutil.rmtree(dirname)
        sleep(2) #需延遲否則會出錯
    os.mkdir(dirname) #建立資料夾

from keras.models import load_model
from PIL import Image
import numpy as np
import cv2
import shutil, os
from time import sleep

labels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
```

```
'Z']
#擷取車牌
imgname = '3M6605.jpg' #預測錯誤
imgname = '1710YC.jpg' #預測正確
print('車牌號碼(真實)：', imgname)
dirname = 'recogdata'
emptydir(dirname)
img = cv2.imread('predictPlate/' + imgname)
#img = cv2.imread('predictPlate/' + imgname,encoding='utf-8')
detector = cv2.CascadeClassifier('haar_carplate.xml')
signs = detector.detectMultiScale(img, scaleFactor=1.1,
minNeighbors=4, minSize=(20, 20))
if len(signs) > 0 :
    for (x, y, w, h) in signs:
        image1 = Image.open('predictPlate/' + imgname)
        image2 = image1.crop((x, y, x+w, y+h))
        #image3 = image2.resize((140,40), Image.ANTIALIAS)
        image3 = image2.resize((140, 40), Image.LANCZOS)
        image3.save('tem.jpg')
        image4 = cv2.imread('tem.jpg')
        gray = cv2.cvtColor(image4, cv2.COLOR_RGBA2GRAY)
        _, img_thre = cv2.threshold(gray, 100, 255,
cv2.THRESH_BINARY)
        cv2.imwrite('tem.jpg', img_thre)
    #分割文字
    img_tem = cv2.imread('tem.jpg')
    gray = cv2.cvtColor(img_tem, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 127, 255,
cv2.THRESH_BINARY_INV)
    contours1 = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours = contours1[0]
    letter_image_regions = []
    for contour in contours:
        (x, y, w, h) = cv2.boundingRect(contour)
        letter_image_regions.append((x, y, w, h))
    letter_image_regions = sorted(letter_image_regions, key=lambda
```

```
x: x[0])
    #存檔
    i = 1
    for letter_bounding_box in letter_image_regions:
        x, y, w, h = letter_bounding_box
        if w>=5 and h>32 and h<40:
            letter_image = gray[y:y+h, x:x+w]
            letter_image = cv2.resize(letter_image, (18, 38))
            cv2.imwrite(dirname + '/{}.jpg'.format(i), letter_image)
            i += 1
    #辨識車牌
    detan = 0
    for fname in os.listdir(dirname):
        if os.path.isfile(os.path.join(dirname, fname)):
            detan += 1
    tem_data = []
    for index in range(1, (detan+1)):
        tem_data.append((np.array(Image.open("recogdata/" +
str(index) + ".jpg")))/255.0)
    real_data = np.stack(tem_data)
    real_data1 = np.expand_dims(real_data, axis=3)
    model = load_model("carplate_model.hdf5")

    #翁老師修改 開始
    #predictions = model.predict_classes(real_data1)
    predictions = model.predict(real_data1)
    pred = predictions
    #Mutli-class Classfication
    #pred = model.predict(X[val_index])
    pred = np.argmax(pred, axis=1)
    print(pred)
    predictions = pred
    #翁老師修改結束

    print('車牌號碼為(預測):')
    for i in range(len(predictions)):
        print(labels[int(predictions[i])], end='')
```

```
else:
    print('無法擷取車牌！')
os.remove('tem.jpg')
```

20240923. DNN、最佳參數與損失函數

壹、 相關知識

一、 深度學習：基於人工神經網絡的機器學習方法，透過多層結構自動提取和學習數據的高層次特徵，以實現複雜的模式辨識與預測。

二、 單層感知器：基本的人工神經網絡，由一個輸入層和一個輸出層組成，透過加權求和和激活函數進行二元分類。

三、 多層感知器：前饋式神經網絡，由輸入層、隱藏層和輸出層組成，透過非線性激活函數和反向傳播算法實現更複雜的數據模式學習與分類。

四、 梯度下降法：一種優化算法，通過沿著損失函數梯度下降的方向調整參數，以最小化模型的誤差或損失。

五、 深度網路：擁有多層結構的人工神經網絡，透過逐層學習數據的抽象特徵，用於處理複雜的非線性問題，如圖像識別與自然語言處理。

六、 激活函數：一種數學函數，應用於神經網絡中每個節點的輸出，用於引入非線性，使網絡能學習和表示複雜的數據模式。

七、 Loss Function：損失函數，是一種評估模型預測結果與實際目標值之間差異的數學函數，用於指導模型參數的優化與更新。

八、 DNN：深度神經網路，擁有多個隱藏層的人工神經網絡，能透過逐層提取數據特徵來處理複雜的模式辨識與預測問題。

九、 Predictor：預測器，是一個模型或算法，用於根據輸入數據生成輸出的預測結果，廣泛應用於分類、回歸等任務中。

十、 如何固定抽樣一樣：要在使用 Keras 或 TensorFlow 時，確保每次執行模型時的抽樣結果一致（例如，數據的分割、權重初始化、隨機操作等），需要設定隨機種子。透過設定隨機種子，可以保證每次抽樣和模型訓練過程都相同，從而得到可重複的結果。

十一、SGD（Stochastic Gradient Descent，隨機梯度下降）是一種優化算法，透過在每次迭代中隨機選取一小部分數據來計算梯度並更新模型參數，以加速訓練並降低計算成本。

十二、Adam 是目前最廣泛使用的優化器，對大多數模型和數據集都有良好的效果，並且不需要太多的超參數調整。

十三、RMSProp 非常適合於 RNN 或 LSTM 等需要處理時間序列數據的任務。

十四、SGD with Momentum 適合較簡單的模型，但需要更謹慎的學習率調整。

貳、程式碼解讀

一、 DNN 訓練與驗證模型性能

1. 讀取與處理資料：
   (1) 使用 pandas 讀取 CSV 資料集。
   (2) 分離特徵與標籤資料，進行訓練集與測試集的劃分（train_test_split）。
2. 建立與編譯模型：
   (1) 使用 Keras 建立一個 Sequential 深度神經網路模型。
   (2) 配置隱藏層，激活函數為 ReLU，輸出層為 Sigmoid（二元分類）。
   (3) 編譯模型，損失函數為 binary_crossentropy，優化器可選 Adam 或 SGD。
3. 模型訓練與預測：
   (1) 訓練模型（含驗證集劃分）。
   (2) 使用訓練資料與測試資料進行預測，設定二元分類的門檻值 0.5。
4. 性能評估：
   (1) 計算並輸出混淆矩陣與分類報告。
   (2) 繪製損失（Loss）與準確率（Accuracy）隨 Epoch 變化的曲線。

二、 模型性能的詳細可視化

1. ROC 曲線與 AUC 分析：
   (1) 使用 sklearn 計算 ROC 曲線的假正率（FPR）與真正率（TPR）。
   (2) 繪製 ROC 曲線，顯示曲線下的面積（AUC）。
2. PRC 曲線與指標：
   (1) 計算 Precision-Recall 曲線並計算 AUC。
   (2) 繪製 PRC 曲線，展示模型在不同召回率下的精確度。
3. 預測結果輸出：
   (1) 整合真實值、預測值與機率值為 DataFrame，並將結果匯出為 CSV 檔。

三、 最佳化超參數（Grid Search）

1. 模型包裝：

(1) 使用 Scikeras 將 Keras 模型包裝為 Scikit-learn 兼容的模型。

(2) 定義函數 create_model()，生成模型。

2. 超參數網格搜索：

(1) 設定超參數組合（batch_size 與 epochs）。

(2) 使用 GridSearchCV 進行交叉驗證，尋找最佳參數組合。

3. 結果輸出：

(1) 輸出每組超參數的平均準確率與標準差。

(2) 顯示最佳準確率及其對應的參數組合。

參、 附錄 – 程式碼

一、 DNN_loss-val_loss

```
#機器學習演算法(DNN)範例 應用於 parkinsons 資料集
#designed by 翁政雄 博士/教授
#看輸出幾次最好
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings("ignore") #取消 warnings

filename="D:\學校\國立彰化師範大學\大三上\人工智慧\課程資料\ParkinsonsDNN.csv"
#filename="ParkinsonsDNN.csv"
df = pd.read_csv(filename)
features = list(df.columns[:22])

df_X = df[features]
df_y = df["status"]

from sklearn.model_selection import train_test_split
X,y = df_X.values, df_y.values

test_size = 0.2
#X_trian, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
X_trian, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=test_size)

#建立模型
from keras.models import Sequential
from keras.layers import Dense
#import tensorflow as tf #for optimizer =
tf.keras.optimizers.Adam(0.001)
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.initializers import Zeros, RandomNormal,
glorot_normal, glorot_uniform
from keras.layers import Dropout

model = Sequential()
model.add(Dense(256, input_dim = X_trian.shape[1], activation =
'relu'))

#model.add(Dense(64, activation = 'relu'))
#model.add(BatchNormalization())
#model.add(Dense(32, activation = 'relu'))
#model.add(Dense(256, activation = 'relu'))
#from keras.layers import Dropout
model.add(Dropout(0.2))
#model.add(Dense(9, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid')) #從二元改多元
softmax

# compile the keras model
#model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.compile(loss='binary_crossentropy', optimizer='sgd',
metrics=['accuracy']) #binary_accuracy #二元
#model.compile(loss='categorical_crossentropy', optimizer='sgd',
metrics=['categorical_accuracy']) #多元

#訓練
history = model.fit(X_trian, y_train, validation_split = 0.1, epochs
= 5, verbose = 1)# batch_size = 5, verbose = 2  顯示訓練結果
```

```
#epochs 是訓練週期 validation_split 是將一部分訓練數據劃分為驗證集的參數（關係到訓練幾次） verbose 是控制訓練或評估模型時輸出的日誌資訊量

#預測
predProb = model.predict(X_trian)
#Binary Classification
pred = np.where(predProb > 0.5, 1, 0) #門檻值(機率)0.5
#Muti-class Classification
#pred = model.predict(X[val_index])
#pred = np.argmax(pred, axis=1)#依照機率值高低，決定 label

#輸出混亂矩陣，顯示準確率
from sklearn.metrics import confusion_matrix, classification_report
print("輸出混亂矩陣，顯示準確率(訓練資料):")
print(confusion_matrix(y_train, pred))
print(classification_report(y_train, pred))

#預測，評估模型好壞；使用訓練資料當測試資料
#pred = model.predict_classes(X_test) #DNN 不支援
predProb = model.predict(X_test)
pred = np.where(predProb > 0.5, 1, 0) #門檻值(機率)0.5
#pred = np.argmax(predProb, axis=1) #依據機率值高低，決定 label

print("輸出混亂矩陣，顯示準確率(測試資料):")
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

import matplotlib.pyplot as plt

def plot_loss(history):
    plt.plot(history.history['loss'], label='loss') #validation_split 的部分
    plt.plot(history.history['val_loss'], label='val_loss') #剩餘部分
    #plt.ylim([0, 2])
    plt.xlabel('Epoch')
```

```
        plt.ylabel('Error')
        plt.legend()
        plt.grid(True)
        plt.show()
plot_loss(history)

def plot_accuracy(history):
        plt.plot(history.history['accuracy'], label='accuracy')
#validation_split 的部分
        plt.plot(history.history['val_accuracy'], label='val_accuracy') #
剩餘部分
        #plt.ylim([0, 2])
        plt.xlabel('Epoch')
        plt.ylabel('accuracy')
        plt.legend()
        plt.grid(True)
        plt.show()
plot_accuracy(history)
```

二、 Parkinsons_DNN

```
#機器學習演算法(DNN)範例 應用於 parkinsons 資料集
#designed by 翁政雄 博士/教授
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings("ignore") #取消 warnings

#filename="D:\學校\國立彰化師範大學\大三上\人工智慧\課程
資料\ParkinsonsDNN.csv"
filename="D:/學校/國立彰化師範大學/大三上/人工智慧
/20240923/ParkinsonsDNN.csv"
df = pd.read_csv(filename)
features = list(df.columns[:22])

df_X = df[features]
df_y = df["status"]
```

```
from sklearn.model_selection import train_test_split
X,y = df_X.values, df_y.values

test_size = 0.2
#X_trian, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
X_trian, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size)

#建立模型
from keras.models import Sequential
from keras.layers import Dense
#import tensorflow as tf #for optimizer =
tf.keras.optimizers.Adam(0.001)
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.initializers import Zeros, RandomNormal,
glorot_normal, glorot_uniform
from keras.layers import Dropout

model = Sequential()
model.add(Dense(256, input_dim = X_trian.shape[1], activation =
'relu'))

#model.add(Dense(64, activation = 'relu'))
#model.add(BatchNormalization())
#model.add(Dense(32, activation = 'relu'))
#model.add(Dense(256, activation = 'relu'))
#from keras.layers import Dropout
model.add(Dropout(0.2))
#model.add(Dense(9, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid')) #從二元改多元
softmax

# compile the keras model
#model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.compile(loss='binary_crossentropy', optimizer='sgd',
```

```
metrics=['accuracy']) #binary_accuracy #二元
#model.compile(loss='categorical_crossentropy', optimizer='sgd',
metrics=['categorical_accuracy']) #多元

#訓練
history = model.fit(X_trian, y_train, validation_split = 0.1, epochs
= 5, verbose = 1)# batch_size = 5, verbose = 2 顯示訓練結果
#epochs 是訓練週期 validation_split 是將一部分訓練數據劃分
為驗證集的參數（關係到訓練幾次） verbose 是控制訓練或評
估模型時輸出的日誌資訊量

#預測
predProb = model.predict(X_trian)
#Binary Classification
pred = np.where(predProb > 0.5, 1, 0) #門檻值(機率)0.5
#Muti-class Classification
#pred = model.predict(X[val_index])
#pred = np.argmax(pred, axis=1)#依照機率值高低，決定 label

#輸出混亂矩陣，顯示準確率
from sklearn.metrics import confusion_matrix, classification_report
print("輸出混亂矩陣，顯示準確率(訓練資料):")
print(confusion_matrix(y_train, pred))
print(classification_report(y_train, pred))

#預測，評估模型好壞；使用訓練資料當測試資料
#pred = model.predict_classes(X_test) #DNN 不支援
predProb = model.predict(X_test)
pred = np.where(predProb > 0.5, 1, 0) #門檻值(機率)0.5
#pred = np.argmax(predProb, axis=1) #依據機率值高低，決定
label

print("輸出混亂矩陣，顯示準確率(測試資料):")
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

#將真實資料，預測結果與預測機率合併成 DataFrame
```

```
from pandas import DataFrame
#因為 DNN 的 pred = model.predict_classes(X_test) 的輸出 非 1 維
#df_PredProb = DataFrame({'Prob':predProb}) #各標籤的機率值
prob = predProb.flatten() #2D->1D
pred = pred.flatten() #2D->1D

#輸出資料，將 Real, Pred, Prob 合併成 DataFrame
df_PredResult = DataFrame({'Real':y_test, 'Pred': pred,
'prob':prob})
filename='PredResult_DNN.csv'
df_PredResult.to_csv(filename)
#print(df_PredResult)

''' #debug
#axis=1 水平合併
#df_myResult = pd.concat([df_PredResult, df_PredProb], axis=1)
print(prob)
print(predProb)
print(df_PredResult)
print(pred)
print(pred)
print(y_test)
#df_PredResult = DataFrame({'Real':y_test, 'Pred':pred})
'''

#Compute ROC curve and ROC area for each class
#(y_test, pred)必須是[0,1]轉換；N->0,Y->1
#df = DataFrame({"Pred": pred, "Real":y_test})
#df["PredNew"]=0
#df.loc[df["Pred"] == "Y",["PredNew"]] = 1
#df["RealNew"]=0
#df.loc[df["Real"] == "Y",["RealNew"]] = 1

#Compute ROC curve and ROC area for each class
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

```
fpr,tpr,threshold = roc_curve(y_test, pred)#計算真正率及假正率
roc_auc = auc(fpr, tpr)#計算 auc 的值
plt.figure()
lw = 2
#plt.figure(figsize=(10, 10))
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.4f)' % roc_auc)#
假正率為横坐標，真正率為縱座標作曲線
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('Ture Positive Rate')
plt.title('ROC Curve')

#plt.legend(loc="lower right")#標籤位置
plt.legend()
plt.show()
print("\nROC_auc area=%.4f" % (roc_auc))

#lr_probs = model.predict_proba(X_test)
#print(lr_probs)
#keep probabilities for the positive outcome only
#lr_probs = lr_probs[:, 1]
#prdict class values
#yhat = model.predict(X_test)
from sklearn.metrics import precision_recall_curve
#lr_precision, lr_recall, = precision recall_curve(y_test, lr_probs)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)

from sklearn.metrics import f1_score
lr_f1, lr_auc = f1_score(y_test, pred), auc(lr_recall, lr_precision)
# summarize scores
#print('MLP(ANN): f1-%.3f PRC_auc area=%.3f' % (lr_f1,
lr_auc))# f1 乃是 label-1 的 f1
print("PRC_auc area=%.4f" % (lr_auc))
```

```
# plot the precision-recall curves
no_skill = len(y_test [y_test==1]) / len(y_test)


#plt.figure(figsize=(10,10)) 151
#plt.plot([0, 1], [no_skill, no_skill], linestyle="--", label= 'No
Skill')
plt.plot([0, 1], [1, 0], color= 'navy', lw=lw, linestyle='--')
plt.plot(lr_recall, lr_precision, color='darkorange',
            lw=lw, label= 'PRC curve (area = %0.4f)' % lr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

# axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
# show the legend
plt.legend()
# show the plot
plt.show()
```

三、 Parkinsons_DNN_bestPara

```
# Use scikit-learn to grid search the batch size and epochs
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
#from keras.wrappers.scikit_learn import KerasClassifier
from scikeras.wrappers import KerasClassifier
import pandas as pd

# Function to create model, required for KerasClassifier
def create_model():
    # create model
    model = Sequential()
```

```
        model.add(Dense(64, input_dim=22, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))
        # Compile model
        model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
        return model

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

# load dataset
#filename="D:\wekadata\Parkinsons_TrainDNN.csv"
filename="D:/學校/國立彰化師範大學/大三上/人工智慧
/20240923/ParkinsonsDNN.csv"
df_Train = pd.read_csv(filename)
#split into input (x) and output (Y) variables
features = list(df_Train.columns[:22])
X = df_Train[features]
Y = df_Train["status"]

# create model
model = KerasClassifier(build_fn=create_model, verbose=0)

# define the grid search parameters
batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid,
n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# summarize results

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
```

```
        print("%f (%f) with: %r" % (mean, stdev, param))
#輸出最佳參數
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
```

20240930. kFold 於 DNN 的應用

壹、 相關知識

一、 相同準確率下，層數越少越好，神經元個數亦是。

二、 L1 L2 是懲罰參數使程式不要這麼快抵達模型以免 overfitting。

三、 深度學習如何提高準確率：增加訓練數據、調整超參數、改變模型結構、訓練更長時間、使用不同的模型。

四、 參數是模型在訓練過程中自動學習和更新的變量，通常包括神經網絡中的權重和偏置。而超參數則是由使用者預先設定的參數，用來控制模型訓練過程的配置，如學習率、批次大小、層數等，並不會通過訓練數據自動調整。

五、 Precision（精確率）：被模型預測為正類（Positive）的樣本中，實際為正類的比例。

六、 Recall（召回率）：實際為正類的樣本中，模型正確預測為正類的比例。

七、 F1-Score：Precision 和 Recall 的加權平均，專門用於平衡 Precision 和 Recall 的場景。

八、 Accuracy（準確率）：模型的正確預測佔所有樣本的比例。

九、 K-fold 交叉驗證能提供更穩定、更準確的模型評估，因為它考慮了多次不同的訓練和測試情境，相較於單純的十次取平均，能有效降低隨機性和過擬合的風險。

貳、 程式碼解讀

一、 模型性能改進：

1. 在每個 DNN 模型中嘗試不同的優化器（如 Adam 或 RMSprop）和學習率。
2. 使用更多層或者改變每層神經元的數量，並測試 Dropout 的最佳比例。

二、 數據處理：

1. 檢查數據是否需要正規化或標準化，特別是輸入的特徵值分佈是否均勻。

三、 交叉驗證的穩定性：

1. 在使用 KFold 時，確保 random_state 設定固定，便於結果重現。
2. 如果資料集不平衡，可嘗試加入類別權重（class_weight）或使用 SMOTE 技術進行過採樣。

四、 圖像化與報告：

1. 繪製不同實驗的損失曲線和準確率曲線以進一步分析模型的收斂情況。
2. 比較多次訓練結果的變化，檢查平均 ROC AUC 和 PRC AUC 的標準差。

五、 最佳化參數搜尋：

1. 在 GridSearchCV 中加入更多參數的組合，比如隱藏層數量、神經元數和學習率。
2. 考慮使用更高效的超參數調整方法，例如 Bayesian Optimization 或 Random Search。

參、 附錄 – 程式碼

一、 Parkinsons_DNN

```
#機器學習演算法(DNN)範例 應用於 parkinsons 資料集
#designed by 翁政雄 博士/教授
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings("ignore") #取消 warnings

#filename="D:\學校\國立彰化師範大學\大三上\人工智慧\課程資料\ParkinsonsDNN.csv"
filename="D:/學校/國立彰化師範大學/大三上/人工智慧/20240923/ParkinsonsDNN.csv"
df = pd.read_csv(filename)
features = list(df.columns[:22]) #22 個欄位

df_X = df[features]
df_y = df["status"]

from sklearn.model_selection import train_test_split
X,y = df_X.values, df_y.values

test_size = 0.2
#X_trian, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
```

```
X_trian, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size)

#建立模型
from keras.models import Sequential
from keras.layers import Dense
#import tensorflow as tf #for optimizer =
tf.keras.optimizers.Adam(0.001)
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.initializers import Zeros, RandomNormal,
glorot_normal, glorot_uniform
from keras.layers import Dropout

model = Sequential()
model.add(Dense(256, input_dim = X_trian.shape[1], activation =
'relu'))#機活函數 relu  第一層隱藏層 256 input_dim22 個欄位

#model.add(Dense(64, activation = 'relu'))
#model.add(BatchNormalization())
#model.add(Dense(32, activation = 'relu'))
#model.add(Dense(256, activation = 'relu'))
#from keras.layers import Dropout
model.add(Dropout(0.2))
#model.add(Dense(9, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid')) #從二元改多元
softmax

# compile the keras model
#model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.compile(loss='binary_crossentropy', optimizer='sgd',
metrics=['accuracy']) #binary_accuracy #二元
#model.compile(loss='categorical_crossentropy', optimizer='sgd',
metrics=['categorical_accuracy']) #多元

#訓練
history = model.fit(X_trian, y_train, validation_split = 0.1, epochs
```

```
= 5, verbose = 1)# batch_size = 5, verbose = 2 顯示訓練結果
#epochs 是訓練週期 validation_split 是將一部分訓練數據劃分為驗證集的參數（關係到訓練幾次） verbose 是控制訓練或評估模型時輸出的日誌資訊量

#預測
predProb = model.predict(X_trian)
#Binary Classification
pred = np.where(predProb > 0.5, 1, 0) #門檻值(機率)0.5
#Muti-class Classification
#pred = model.predict(X[val_index])
#pred = np.argmax(pred, axis=1)#依照機率值高低，決定 label

#輸出混亂矩陣，顯示準確率
from sklearn.metrics import confusion_matrix, classification_report
print("輸出混亂矩陣，顯示準確率(訓練資料):")
print(confusion_matrix(y_train, pred))
print(classification_report(y_train, pred))

#預測，評估模型好壞；使用訓練資料當測試資料
#pred = model.predict_classes(X_test) #DNN 不支援
predProb = model.predict(X_test)
pred = np.where(predProb > 0.5, 1, 0) #門檻值(機率)0.5
#pred = np.argmax(predProb, axis=1) #依據機率值高低，決定 label

print("輸出混亂矩陣，顯示準確率(測試資料):")
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

#將真實資料，預測結果與預測機率合併成 DataFrame
from pandas import DataFrame
#因為 DNN 的 pred = model.predict_classes(X_test) 的輸出 非 1 維
#df_PredProb = DataFrame({'Prob':predProb}) #各標籤的機率值
prob = predProb.flatten() #2D->1D
pred = pred.flatten() #2D->1D
```

```
#輸出資料，將 Real, Pred, Prob 合併成 DataFrame
df_PredResult = DataFrame({'Real':y_test, 'Pred': pred,
'prob':prob})
filename='PredResult_DNN.csv'
df_PredResult.to_csv(filename)
#print(df_PredResult)

''' #debug
#axis=1 水平合併
#df_myResult = pd.concat([df_PredResult, df_PredProb], axis=1)
print(prob)
print(predProb)
print(df_PredResult)
print(pred)
print(pred)
print(y_test)
#df_PredResult = DataFrame({'Real':y_test, 'Pred':pred})
'''

#Compute ROC curve and ROC area for each class
#(y_test, pred)必須是[0,1]轉換；N->0,Y->1
#df = DataFrame({"Pred": pred, "Real":y_test})
#df["PredNew"]=0
#df.loc[df["Pred"] == "Y",["PredNew"]] = 1
#df["RealNew"]=0
#df.loc[df["Real"] == "Y",["RealNew"]] = 1

#Compute ROC curve and ROC area for each class
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr,tpr,threshold = roc_curve(y_test, pred)#計算真正率及假正率
roc_auc = auc(fpr, tpr)#計算 auc 的值
plt.figure()
lw = 2
#plt.figure(figsize=(10, 10))
```

```
plt.plot(fpr, tpr, color='darkorange',
                lw=lw, label='ROC curve (area = %0.4f)' % roc_auc)#
假正率為横坐標，真正率為縱座標作曲線
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('Ture Positive Rate')
plt.title('ROC Curve')

#plt.legend(loc="lower right")#標籤位置
plt.legend()
plt.show()
print("\nROC_auc area=%.4f" % (roc_auc))

#lr_probs = model.predict_proba(X_test)
#print(lr_probs)
#keep probabilities for the positive outcome only
#lr_probs = lr_probs[:, 1]
#prdict class values
#yhat = model.predict(X_test)
from sklearn.metrics import precision_recall_curve
#lr_precision, lr_recall, = precision recall_curve(y_test, lr_probs)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)

from sklearn.metrics import f1_score
lr_f1, lr_auc = f1_score(y_test, pred), auc(lr_recall, lr_precision)
# summarize scores
#print('MLP(ANN): f1-%.3f PRC_auc area=%.3f' % (lr_f1,
lr_auc))# f1 乃是 label-1 的 f1
print("PRC_auc area=%.4f" % (lr_auc))



# plot the precision-recall curves
no_skill = len(y_test [y_test==1]) / len(y_test)
```

```
#plt.figure(figsize=(10,10)) 151
#plt.plot([0, 1], [no_skill, no_skill], linestyle="--", label= 'No
Skill')
plt.plot([0, 1], [1, 0], color= 'navy', lw=lw, linestyle='--')
plt.plot(lr_recall, lr_precision, color='darkorange',
         lw=lw, label= 'PRC curve (area = %0.4f)' % lr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

# axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
# show the legend
plt.legend()
# show the plot
plt.show()
```

二、 Parkinsons_DNN_avg10

```
import numpy as np
import pandas as pd
import warnings
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Dropout
from tensorflow.keras.layers import BatchNormalization
from sklearn.metrics import confusion_matrix,
classification_report, roc_curve, auc, precision_recall_curve,
f1_score
import matplotlib.pyplot as plt

warnings.filterwarnings("ignore") #取消  warnings

filename = "D:/學校/國立彰化師範大學/大三上/人工智慧
/20240923/ParkinsonsDNN.csv"
df = pd.read_csv(filename)
features = list(df.columns[:22])    # 22 個欄位
```

```
df_X = df[features]
df_y = df["status"]

X, y = df_X.values, df_y.values
test_size = 0.2

# 记录各次迭代的 ROC AUC 和 PRC AUC
roc_aucs = []
prc_aucs = []

# 重复 10 次训练
for i in range(10):
    # 分割資料集
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size)

    # 建立模型
    model = Sequential()
    model.add(Dense(256, input_dim=X_train.shape[1],
activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))  # 二元分類

    # 編譯模型
    model.compile(loss='binary_crossentropy', optimizer='sgd',
metrics=['accuracy'])

    # 訓練模型
    model.fit(X_train, y_train, validation_split=0.1, epochs=5,
verbose=2)

    # 預測並計算 ROC 和 PRC 曲線
    predProb = model.predict(X_test)
    pred = np.where(predProb > 0.5, 1, 0)

    # 計算 ROC AUC
    fpr, tpr, _ = roc_curve(y_test, pred)
```

```
        roc_auc = auc(fpr, tpr)
        roc_aucs.append(roc_auc)

        # 計算 PRC AUC
        lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)
        prc_auc = auc(lr_recall, lr_precision)
        prc_aucs.append(prc_auc)

        print(f"第 {i+1} 次訓練完成, ROC AUC: {roc_auc:.4f},
PRC AUC: {prc_auc:.4f}")

# 计算平均 ROC AUC 和 PRC AUC
mean_roc_auc = np.mean(roc_aucs)
mean_prc_auc = np.mean(prc_aucs)

print(f"\n10 次訓練的平均 ROC AUC: {mean_roc_auc:.4f}")
print(f"10 次訓練的平均 PRC AUC: {mean_prc_auc:.4f}")

# 繪製 ROC 曲線和 PRC 曲線（取最後一次的結果）
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area
= %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

plt.figure()
plt.plot(lr_recall, lr_precision, color='darkorange', lw=2,
label='PRC curve (area = %0.4f)' % prc_auc)
plt.plot([0, 1], [1, 0], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
plt.legend(loc="lower right")
plt.show()
```

三、 Parkinsons_DNN_avg10_kFold(same_resource)

```
import numpy as np
import pandas as pd
import warnings
from sklearn.model_selection import KFold
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf

warnings.filterwarnings("ignore")  # 取消 warnings

# 讀取資料
filename = "D:/學校/國立彰化師範大學/大三上/人工智慧
/20240923/ParkinsonsDNN.csv"
df = pd.read_csv(filename)
features = list(df.columns[:22])  # 22 個欄位
X, y = df[features].values, df["status"].values

# 建立 DNN 模型函數
def create_model():
    model = Sequential()
    model.add(Dense(256, input_dim=X.shape[1],
activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='sgd',
metrics=['accuracy'])
    return model

# KFold 交叉驗證（設置 random_state 以保證每次分割相同）
n_splits = 10
```

```
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

list_report = []
i = 1

for train_index, val_index in kf.split(X):
    # 建立模型
    model = create_model()

    # 訓練模型
    model.fit(X[train_index], y[train_index], validation_split=0.1,
epochs=5, verbose=2)

    # 預測
    predProb = model.predict(X[val_index])
    pred = np.where(predProb > 0.5, 1, 0)    # Binary classification

    print(f"\nCross validation #{i}")
    print(confusion_matrix(y[val_index], pred))

    # 計算 classification report
    report = classification_report(y[val_index], pred,
output_dict=True)
    df_report = pd.DataFrame(report).transpose()

    precision = df_report.loc['1']['precision']
    recall = df_report.loc['1']['recall']
    f1 = df_report.loc['1']['f1-score']
    accuracy = df_report.loc['accuracy']['f1-score']

    # 存入 list
    tupleOne = [round(precision, 4), round(recall, 4), round(f1, 4),
round(accuracy, 4)]
    list_report.append(tupleOne)

    i += 1
```

```
# 計算總平均
array_report = np.array(list_report)
avg_precision = round(np.average(array_report[:, 0]), 4)
avg_recall = round(np.average(array_report[:, 1]), 4)
avg_f1_score = round(np.average(array_report[:, 2]), 4)
avg_accuracy = round(np.average(array_report[:, 3]), 4)

# 將平均結果存入列表
tupleAvg = [avg_precision, avg_recall, avg_f1_score,
avg_accuracy]
list_report.append(tupleAvg)

# 將結果轉為 DataFrame
df_final_report = pd.DataFrame(list_report, columns=["precision",
"recall", "f1-score", "accuracy"])
print(f'KFold = {n_splits}. Cross validation 的結果，最後一列
(row)=總平均')
print(df_final_report)

# 輸出結果至 CSV
output_filename = "D:/學校/國立彰化師範大學/大三上/人工智慧
/df_report_DNN.csv"
df_final_report.to_csv(output_filename, index=True)
print(f"輸出至檔案: {output_filename}")
```

四、 Parkinsons_DNN_avg10_kFold(teacher)

```
#機器學習演算法（DNN） 範例 應用於 Parkinsons 資料集
#designed by 翁政雄 博士/教授
import numpy as np
import pandas as pd

import warnings

warnings.filterwarnings ("ignore")#取消 warnings

#filename="D:\wekadata\Parkinsons_TrainDNN.CSV"
filename="D:/學校/國立彰化師範大學/大三上/人工智慧
/20240923/ParkinsonsDNN.csv"
```

```
df = pd.read_csv(filename)
features = list(df.columns[:22])

df_X = df[features]
df_y = df["status"]

from sklearn.model_selection import train_test_split
X, y=df_X.values, df_y.values

test_size=0.2
#X_train, X_test,y_train, y_test=train_test_split(X,y, test_size=0.3,
random_state=0)
X_train, X_test,y_train, y_test=train_test_split(X,y,
test_size=test_size)

#建立模型
from keras.models import Sequential
from keras.layers import Dense
#import tensorflow as tf #for optimizer=tf.keras.optimizers.Adam
(0.001)
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras. initializers import Zeros, RandomNormal,
glorot_normal, glorot_uniform
from keras. layers import Dropout

model=Sequential()
model.add(Dense (256, input_dim=X_train.shape[1],
activation='relu')) # input_dim=X_train.shape[1] -> 22 個欄位

#model.add(Dense(64, activation='relu'))
#model.add(BatchNormalization())
#model.add(Dense(32, activation='relu'))
#model.add(Dense(256, activation='relu'))
#from keras.layers import Dropout
model.add(Dropout (0.2))
model.add(Dense(9, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
# compile the keras model
#mode1.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy*])
model.compile(loss='binary_crossentropy', optimizer='sgd',
metrics=['accuracy'])


#輸出混亂矩陣，顯示準確率
from sklearn.metrics import confusion_matrix, classification_report
k=11
list_report = list()

for i in range(1,k):
    #訓練
    model.fit(X_train,y_train, validation_split=0.1,epochs=5,
verbose=0)#batch_size=5, verbose=2 顯示訓練結果
    #預測，評估模型好壞；使用訓練資料當測試資料
    predProb = model.predict(X_test)
    #Binary Classification
    pred = np.where(predProb > 0.5, 1,0)#門檻值（機率）0.5
    #Mutli-class Classification
    #pred= model.predict(X[val_index])
    #pred= np.argmax(pred,axis=1)#依據機率值高低，決定
1abel
    print("\nCross validation #",i)
    print(confusion_matrix(y_test, pred))
    print(classification_report(y_test,pred))

    #classification report 取值
    report=classification_report(y_test, pred, output_dict=True)
    df=pd.DataFrame(report).transpose()
    precision=np.array(df[4:5]["precision"])
    recall=np.array(df[4:5]["recall"])
    f1=np.array(df[4:5]["f1-score"])
    accuracy=np.array(df[2:3]["f1-score"])
    tupleOne=[round(precision[0],4), round(recall[0],4),
```

```
round(f1[0],4), round(accuracy[0],4)]
        list_report.append(tupleOne)

        i=i+1

#print(list_report)
#list 轉陣列，用於平均值計算
array_report = np.array(list_report)
#print(array_report)
#print(array_report[:,0])
#計算總平均(Cross validation)放置於 最後一列（row）
avg_precision=round(np.average(array_report[:,0]),4)
avg_recall=round(np.average(array_report[:,1]),4)
avg_f1_score=round(np.average(array_report[:,2]),4)
avg_accuracy=round(np.average(array_report[:,3]),4)

tupleAvg=[avg_precision, avg_recall, avg_f1_score ,
avg_accuracy]
list_report.append(tupleAvg)

from pandas import DataFrame
df_report=DataFrame(list_report)
df_report.columns=("precision", "recall", "f1-score", "accuracy")
print('kfold =',k-1,". Cross validation 的結果。最後一列（row）=
總平均")
print(df_report)



'''
filename="D:/data2/人臉辨識/report/df_DNNavg.csv"
print("輸出至檔案",filename)
df_report.to_csv(filename, index=True)'''
```

五、 Parkinsons_DNN_avg10_kFold

```
import numpy as np
import pandas as pd
import warnings
from sklearn.model_selection import KFold
```

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf

warnings.filterwarnings("ignore")  # 取消 warnings

# 讀取資料
filename = "D:/學校/國立彰化師範大學/大三上/人工智慧
/20240923/ParkinsonsDNN.csv"
df = pd.read_csv(filename)
features = list(df.columns[:22])  # 22 個欄位
X, y = df[features].values, df["status"].values

# 建立 DNN 模型函數
def create_model():
    model = Sequential()
    model.add(Dense(256, input_dim=X.shape[1],
activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='sgd',
metrics=['accuracy'])
    return model

# KFold 交叉驗證
n_splits = 10
kf = KFold(n_splits=n_splits, shuffle=True)

list_report = []
i = 1

for train_index, val_index in kf.split(X):
    # 建立模型
    model = create_model()

    # 訓練模型
```

```
        model.fit(X[train_index], y[train_index], validation_split=0.1,
epochs=5, verbose=2)

        # 預測
        predProb = model.predict(X[val_index])
        pred = np.where(predProb > 0.5, 1, 0)    # Binary classification

        print(f"\nCross validation #{i}")
        print(confusion_matrix(y[val_index], pred))

        # 計算 classification report
        report = classification_report(y[val_index], pred,
output_dict=True)
        df_report = pd.DataFrame(report).transpose()

        precision = df_report.loc['1']['precision']
        recall = df_report.loc['1']['recall']
        f1 = df_report.loc['1']['f1-score']
        accuracy = df_report.loc['accuracy']['f1-score']

        # 存入 list
        tupleOne = [round(precision, 4), round(recall, 4), round(f1, 4),
round(accuracy, 4)]
        list_report.append(tupleOne)

        i += 1

# 計算總平均
array_report = np.array(list_report)
avg_precision = round(np.average(array_report[:, 0]), 4)
avg_recall = round(np.average(array_report[:, 1]), 4)
avg_f1_score = round(np.average(array_report[:, 2]), 4)
avg_accuracy = round(np.average(array_report[:, 3]), 4)

# 將平均結果存入列表
tupleAvg = [avg_precision, avg_recall, avg_f1_score,
avg_accuracy]
```

```
list_report.append(tupleAvg)

# 將結果轉為 DataFrame
df_final_report = pd.DataFrame(list_report, columns=["precision",
"recall", "f1-score", "accuracy"])
print(f'KFold = {n_splits}. Cross validation 的結果，最後一列
(row)=總平均')
print(df_final_report)

# 輸出結果至 CSV
output_filename = "D:/學校/國立彰化師範大學/大三上/人工智慧
/df_report_DNN.csv"
df_final_report.to_csv(output_filename, index=True)
print(f"輸出至檔案: {output_filename}")
```

六、 Parkinsons_DNN_bestPara

```
# Use scikit-learn to grid search the batch size and epochs
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
#from keras.wrappers.scikit_learn import KerasClassifier
from scikeras.wrappers import KerasClassifier
import pandas as pd

# Function to create model, required for KerasClassifier
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(64, input_dim=22, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model

# fix random seed for reproducibility
seed = 7
```

```
numpy.random.seed(seed)

# load dataset
#filename="D:\wekadata\Parkinsons_TrainDNN.csv"
filename="D:/學校/國立彰化師範大學/大三上/人工智慧
/20240923/ParkinsonsDNN.csv"
df_Train = pd.read_csv(filename)
#split into input (x) and output (Y) variables
features = list(df_Train.columns[:22])
X = df_Train[features]
Y = df_Train["status"]

# create model
model = KerasClassifier(build_fn=create_model, verbose=0)

# define the grid search parameters
batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid,
n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# summarize results

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
      print("%f (%f) with: %r" % (mean, stdev, param))
#輸出最佳參數
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
```

七、 Parkinsons_DNN_bestWeight

```
#機器學習演算法(DNN) 範例 應用於 Parkisons 資料集
#designed by 翁政雄 博士/教授
import numpy as np
import pandas as pd
```

```
import warnings
warnings.filterwarnings("ignore")

#filename="D:/學校/國立彰化師範大學/大三上/人工智慧
/20240923/ParkinsonsDNN.csv"
filename = "D:/學校/國立彰化師範大學/大三上/人工智慧
/20240923/ParkinsonsDNN.csv"
df = pd.read_csv(filename)
features = list(df.columns[:22])

df_X = df[features]
df_y = df["status"]

from sklearn.model_selection import train_test_split
X,y = df_X.values, df_y.values

test_size = 0.2
#X_trian, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size)

#建立模型
from keras.models import Sequential
from keras.layers import Dense
#import tensorflow as tf #for optimizer =
tf.keras.optimizers.Adam(0.001)
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.initializers import Zeros, RandomNormal,
glorot_normal, glorot_uniform
from keras.layers import Dropout

model = Sequential()
model.add(Dense(256, input_dim=X_train.shape[1],
activation='relu'))
```

```
#model.add(Dense(64, activation = 'relu'))
#model.add(BatchNormalization())
#model.add(Dense(32, activation = 'relu'))
#model.add(Dense(256, activation = 'relu'))
#from keras.layers import Dropout
model.add(Dropout(0.2))
#model.add(Dense(9, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid')) #從二元改多元
softmax

# compile the keras model
#model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.compile(loss='binary_crossentropy', optimizer='sgd',
metrics=['accuracy']) #binary_accuracy #二元

#訓練
#model.fit(X_train, y_train, validation_split=0.1, epochs=5,
verbose=0)

#從訓練週期中，尋找最佳權重
from tensorflow.keras.callbacks import ModelCheckpoint
#checkpoint_name = 'DNN_Weights-{epochs:03d}--
{val_loss:.5f}.hdf5'
checkpoint_name = 'DNN_Weights-{epoch:03d}--
{val_loss:.5f}.keras' #新版要求
checkpoint = ModelCheckpoint(checkpoint_name,
monitor='val_loss', verbose = 1,
                                        save_best_only = True, mode =
'auto')
callbacks_list = [checkpoint]
model.fit(X_train, y_train, validation_split=0.2, epochs=50,
batch_size=32, callbacks=callbacks_list)

#預測
predProb = model.predict(X_train)
#Binary Classification
```

```
pred = np.where(predProb > 0.5, 1, 0) #門檻值(機率)0.5
#Muti-class Classification
#pred = model.predict(X[val_index])
#pred = np.argmax(pred, axis=1)#依照機率值高低，決定 label

#輸出混亂矩陣，顯示準確率
from sklearn.metrics import confusion_matrix, classification_report
print("輸出混亂矩陣，顯示準確率(訓練資料):")
print(confusion_matrix(y_train, pred))
print(classification_report(y_train, pred))

#預測，評估模型好壞；使用訓練資料當測試資料
#pred = model.predict_classes(X_test) #DNN 不支援
predProb = model.predict(X_test)
pred = np.where(predProb > 0.5, 1, 0) #門檻值(機率)0.5
#pred = np.argmax(predProb, axis=1) #依據機率值高低，決定
label

print("輸出混亂矩陣，顯示準確率(測試資料):")
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

#將真實資料，預測結果與預測機率合併成 DataFrame
from pandas import DataFrame
#因為 DNN 的 pred = model.predict_classes(X_test) 的輸出 非
1 維
#df_PredProb = DataFrame({'Prob':predProb}) #各標籤的機率值
prob = predProb.flatten() #2D->1D
pred = pred.flatten() #2D->1D

#輸出資料，將 Real, Pred, Prob 合併成 DataFrame
df_PredResult = DataFrame({'Real':y_test, 'Pred': pred,
'prob':prob})
filename='PredResult_DNN.csv'
df_PredResult.to_csv(filename)
#print(df_PredResult)
```

八、 Parkinsons_DNN_bestWeight_LoadIt

```
#機器學習演算法(DNN) 範例 應用於 Parkisons 資料集
#designed by 翁政雄 博士/教授
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

#filename="D:/學校/國立彰化師範大學/大三上/人工智慧
/20240923/ParkinsonsDNN.csv"
filename = "D:/學校/國立彰化師範大學/大三上/人工智慧
/20240923/ParkinsonsDNN.csv"
df = pd.read_csv(filename)
features = list(df.columns[:22])

df_X = df[features]
df_y = df["status"]

from sklearn.model_selection import train_test_split
X,y = df_X.values, df_y.values

test_size = 0.2
#X_trian, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size)

#建立模型
from keras.models import Sequential
from keras.layers import Dense
#import tensorflow as tf #for optimizer =
tf.keras.optimizers.Adam(0.001)
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.initializers import Zeros, RandomNormal,
glorot_normal, glorot_uniform
from keras.layers import Dropout
```

```
model = Sequential()
model.add(Dense(256, input_dim=X_train.shape[1],
activation='relu'))

#model.add(Dense(64, activation = 'relu'))
#model.add(BatchNormalization())
#model.add(Dense(32, activation = 'relu'))
#model.add(Dense(256, activation = 'relu'))
#from keras.layers import Dropout
model.add(Dropout(0.2))
#model.add(Dense(9, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid')) #從二元改多元
softmax

# compile the keras model
#model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.compile(loss='binary_crossentropy', optimizer='sgd',
metrics=['accuracy']) #binary_accuracy #二元

'''
#訓練
#model.fit(X_train, y_train, validation_split=0.1, epochs=5,
verbose=0)

#從訓練週期中，尋找最佳權重
from tensorflow.keras.callbacks import ModelCheckpoint
#checkpoint_name = 'DNN_Weights-{epochs:03d}--
{val_loss:.5f}.hdf5'
checkpoint_name = 'DNN_Weights-{epoch:03d}--
{val_loss:.5f}.keras' #新版要求
checkpoint = ModelCheckpoint(checkpoint_name,
monitor='val_loss', verbose = 1,
                                        save_best_only = True, mode =
'auto')
callbacks_list = [checkpoint]
```

```
model.fit(X_train, y_train, validation_split=0.2, epochs=50,
batch_size=32, callbacks=callbacks_list)
'''

#載入最佳權重
# Load weights file of the best model:
#weights_file='DNN_Weights-009--0.36642.hdf5'
weights_file = 'DNN_Weights-009--0.36642.keras'

model.load_weights(weights_file) #load it

model.compile(loss='mean_absolute_error', optimizer='adam',
metrics=['mean_absolute_error'])

#預測
predProb = model.predict(X_train)
#Binary Classification
pred = np.where(predProb > 0.5, 1, 0) #門檻值(機率)0.5
#Muti-class Classification
#pred = model.predict(X[val_index])
#pred = np.argmax(pred, axis=1)#依照機率值高低，決定 label

#輸出混亂矩陣，顯示準確率
from sklearn.metrics import confusion_matrix, classification_report
print("輸出混亂矩陣，顯示準確率(訓練資料):")
print(confusion_matrix(y_train, pred))
print(classification_report(y_train, pred))

#預測，評估模型好壞；使用訓練資料當測試資料
#pred = model.predict_classes(X_test) #DNN 不支援
predProb = model.predict(X_test)
pred = np.where(predProb > 0.5, 1, 0) #門檻值(機率)0.5
#pred = np.argmax(predProb, axis=1) #依據機率值高低，決定
label

print("輸出混亂矩陣，顯示準確率(測試資料):")
print(confusion_matrix(y_test, pred))
```

```
print(classification_report(y_test, pred))

#將真實資料，預測結果與預測機率合併成 DataFrame
from pandas import DataFrame
#因為 DNN 的 pred = model.predict_classes(X_test) 的輸出 非 1 維
#df_PredProb = DataFrame({'Prob':predProb}) #各標籤的機率值
prob = predProb.flatten() #2D->1D
pred = pred.flatten() #2D->1D

#輸出資料，將 Real, Pred, Prob 合併成 DataFrame
df_PredResult = DataFrame({'Real':y_test, 'Pred': pred,
'prob':prob})
filename='PredResult_DNN.csv'
df_PredResult.to_csv(filename)
#print(df_PredResult)
```

20241007. 激活函數與更替模型運用

壹、 相關知識

一、 Dense 層的激活函數如果沒有明確指定，預設是線性激活函數。

二、 回歸分析最適合的激活函數：ReLU 是最常用的隱藏層激活函數，特別是在深度神經網路和回歸問題；非線性：ReLU 引入了非線性，使得神經網路可以學習到複雜的非線性關係，這對解決迴歸任務中的複雜資料關係至關重要；運算高效：ReLU 的運算非常簡單，公式為 f(x) = max(0, x)，因此運算效率高，尤其是在深度網路中，能夠加速訓練；梯度消失問題的緩解：與 Sigmoid 或 Tanh 活化函數相比，ReLU 在正值區域不會產生梯度消失問題。這使得在深層網路中，梯度能夠順暢地反向傳播，從而加速訓練。

貳、 程式碼解讀

一、 Loss

1. 輸入參數：
   (1) history:由 Keras 模型的 fit()方法返回的結果物件。該物件包含模型在每個訓練週期的損失和驗證損失數據。
2. 主要功能：
   (1) 繪製 loss 和 val_loss 的折線圖。
   (2) x 軸代表訓練週期數（Epoch），y 軸代表損失值。
   (3) 圖表中使用標籤區分損失和驗證損失曲線。
3. 可視化優勢：
   (1) 直觀了解模型的訓練過程。
   (2) 檢查是否出現過擬合（如：訓練損失持續下降但驗證損失上升）。
   (3) 確定模型是否收斂（損失曲線趨於平緩）。

參、 附錄 – 程式碼

一、 StockTSMC_DNN

```
#機器學習演算法(DNN) 範例 應用於 StockTSMC 資料集
#designed by 翁政雄 博士/教授


import datetime
import pandas as pd
```

```
#紀錄開始時間
starttime = datetime.datetime.now()
print('開始時間: ', starttime)
#預設檔名
filename='Datasets\StockTsmc.csv'
filenameOut='Datasets\StockTsmcPredDNN.csv'
filenameModel='Datasets\StockTsmcDNN.h5'

df=pd.read_csv(filename)
print("資料筆數(原始): ",df.shape[0])
#刪除 有 na 的樣本，刪除該筆資料
df.dropna(axis=0,inplace=True)
#設定特徵屬性(輸入欄位)
features=list(df.columns[2:6])
#設定特徵屬性(輸入欄位):OpeningPrice, HighestPrice,
LowestPrice, Volume
X=df[features]
y=df["ClosingPrice"]

recnum=len(X) #資料總筆數
#print(data_dum_DF)
print("資料筆數(處理): ",X.shape[0])
Algorithmtime=datetime.datetime.now()
print("演算時間: ",Algorithmtime)

f_dim=len(features) #X.shape[1]
#建立模型
from keras.models import Sequential
from keras.layers import Dense
#import tensorflow as tf #for
optimizer=tf.keras.optimizers.Adam(0.001)
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.initializers import Zeros, RandomNormal,
glorot_normal, glorot_uniform
#define the keras model
#sigmoid{0~1},tanh{-1~1},relu{0~z}
```

```
model=Sequential()
model.add(Dense(512, input_dim=X.shape[1], activation='relu'))
#model.add(Dense(512, input_dim=X.shape[1],
activation='relu',kernel_initializer=glorot_uniform()))
#model.add(Dense(512, input_dim=X.shape[1],
activation='relu',kernel_initializer=glorot_uniform(),bias_initializer
=Zeros()))

#model.add(BatchNormalization()) #不可使用在 Regressor 模
型，若要正規化，使用 MinMaxScaler，並反正規化預測值
#model.add(Dense(64, activation='relu'))
#model.add(BatchNormalization())
#model.add(Dense(32, activation='relu'))
#from keras.layers import Dropout
#model.add(Dropout(0.2))
#model.add(Dense(8, activation='relu'))
#model.add(Dense(1, activation='linear'))
model.add(Dense(1))

#tf.keras.optimizers.Adam(0.0001)
#compile the keras model
#model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
#model.compile(loss='mean_absolute_error',
optimizer=tf.keras.optimizers.Adam(0.001), metrics=['accuracy'])
#rmsprop
model.compile(loss='mean_absolute_error', optimizer='adam',
metrics=['accuracy'])
#fit the keras model on the dataset
#model.fit(X_Train, y_Train, validation_split=0.2, epoch=15,
batch_size=10)

#Dataframe to ndarray
X=X.to_numpy()
y=y.to_numpy()

n_train = recnum-100 #renum-100 筆作為訓練資料，最後 100 做
```

```
測試
#訓練模型
model.fit(X[:n_train], y[:n_train]) #訓練模型
model.fit(X[:n_train], y[:n_train], epochs=50, validation_split=0.2,
verbose=0)
#預測資料
pred=model.predict(X[n_train:]) #預測資料

#儲存模型
#from keras.models import load_model
#model.save(filenameModel)
#print("儲存 model",filenameModel)

#統計執行時間
endtime=datetime.datetime.now()
print('開始時間: ',starttime)
print('演算時間: ',Algorithmtime)
print('結束時間: ', endtime)
print('花費時間: ',endtime-starttime)

#顯示準確率，評價指標(loss function)
#SSE、MSE、RMSE、MAE 越低越好
real=y[n_train:] #真實資料

#calculate 均方根誤差 RMSE (root mean squared error)
from math import sqrt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

rmse=sqrt(mean_squared_error(real, pred))
print('Test RMSE: %.4f' % rmse)
#calculate 平均絕對誤差 (MAE)(mean absolut error)
mae=mean_absolute_error(real, pred)
print('Test MAE: %.4f' % mae)

#print(pred)
pred=pred.reshape((len(pred))) #2D->1D
```

```
#print(pred)

#將預測結果與真實資料合併成 DataFrame
from pandas import DataFrame

real=y[n_train:]
df_PredResult=DataFrame({"Real":real,"Pred":pred})
#df_PredResult=DataFrame(pred[:,0])
#print("DataFrame: Pred, Real")
print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)

import matplotlib.pyplot as plt

plt.plot(real, 'r-')#真實
plt.plot(pred, 'b:')#預測
plt.legend(['real','pred'])
plt.xlabel("Ti")
plt.ylabel("Price")
plt.title("StockTSMC-ClosingPrice (DNN)")
plt.show()
```

二、 StockTSMC_DNN_bestPara

```
# 機器學習演算法(DNN) 範例 應用於 StockTSMC 資料集
# designed by 翁政雄 博士/教授

import datetime
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV

# 紀錄開始時間
starttime = datetime.datetime.now()
```

```
print('開始時間: ', starttime)

# 預設檔名
filename = 'Datasets/StockTsmc.csv'
filenameOut = 'Datasets/StockTsmcPredDNN.csv'
filenameModel = 'Datasets/StockTsmcDNN.h5'

df = pd.read_csv(filename)
print("資料筆數(原始): ", df.shape[0])
# 刪除有 na 的樣本
df.dropna(axis=0, inplace=True)
# 設定特徵屬性(輸入欄位)
features = list(df.columns[2:6])
X = df[features]
y = df["ClosingPrice"]

recnum = len(X) # 資料總筆數
print("資料筆數(處理): ", X.shape[0])

# 設定模型
def create_model():
    model = Sequential()
    model.add(Dense(512, input_dim=X.shape[1],
activation='relu'))
    model.add(Dense(1))  # 線性回歸模型
    model.compile(loss='mean_absolute_error', optimizer='adam',
metrics=['accuracy'])
    return model

# 包裝成 KerasClassifier
model = KerasClassifier(build_fn=create_model, verbose=0)

# 定義網格搜索參數
batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
```

```
# 建立 GridSearchCV 物件
grid = GridSearchCV(estimator=model, param_grid=param_grid,
n_jobs=-1, cv=3)

# 轉換 Dataframe 至 ndarray
X = X.to_numpy()
y = y.to_numpy()

# 定義訓練資料
n_train = recnum - 100   # 最後 100 筆作為測試

# 執行 GridSearch
grid_result = grid.fit(X[:n_train], y[:n_train])

# 顯示最佳參數
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))

# 用最佳參數訓練最終模型
best_batch_size = grid_result.best_params_['batch_size']
best_epochs = grid_result.best_params_['epochs']
model = create_model()
model.fit(X[:n_train], y[:n_train], batch_size=best_batch_size,
epochs=best_epochs, validation_split=0.2, verbose=0)

# 預測資料
pred = model.predict(X[n_train:])

# 統計執行時間
endtime = datetime.datetime.now()
print('開始時間: ', starttime)
print('演算時間: ', datetime.datetime.now())
print('結束時間: ', endtime)
print('花費時間: ', endtime - starttime)

# 顯示準確率與誤差
from math import sqrt
```

```
from sklearn.metrics import mean_squared_error,
mean_absolute_error

real = y[n_train:]
rmse = sqrt(mean_squared_error(real, pred))
mae = mean_absolute_error(real, pred)
print('Test RMSE: %.4f' % rmse)
print('Test MAE: %.4f' % mae)

# 將預測結果與真實資料合併成 DataFrame
from pandas import DataFrame

pred = pred.reshape((len(pred)))   # 2D -> 1D
df_PredResult = DataFrame({"Real": real, "Pred": pred})

print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案", filenameOut)
df_PredResult.to_csv(filenameOut)

# 顯示結果圖表
import matplotlib.pyplot as plt

plt.plot(real, 'r-')   # 真實資料
plt.plot(pred, 'b:')   # 預測資料
plt.legend(['real', 'pred'])
plt.xlabel("Ti")
plt.ylabel("Price")
plt.title("StockTSMC-ClosingPrice (DNN)")
plt.show()
```

三、 StockTSMC_DNN_Rege_valLoss

```
#機器學習演算法(DNN) 範例 應用於 StockTSMC 資料集
#designed by 翁政雄 博士/教授

import datetime
import pandas as pd
```

```
#紀錄開始時間
starttime = datetime.datetime.now()
print('開始時間: ', starttime)
#預設檔名
filename='Datasets\StockTsmc.csv'
filenameOut='Datasets\StockTsmcPredDNN.csv'
filenameModel='Datasets\StockTsmcDNN.h5'

df=pd.read_csv(filename)
print("資料筆數(原始): ",df.shape[0])
#刪除 有 na 的樣本，刪除該筆資料
df.dropna(axis=0,inplace=True)
#設定特徵屬性(輸入欄位)
features=list(df.columns[2:6])
#設定特徵屬性(輸入欄位):OpeningPrice, HighestPrice,
LowestPrice, Volume
X=df[features]
y=df["ClosingPrice"]

recnum=len(X) #資料總筆數
#print(data_dum_DF)
print("資料筆數(處理): ",X.shape[0])
Algorithmtime=datetime.datetime.now()
print("演算時間: ",Algorithmtime)

f_dim=len(features) #X.shape[1]
#建立模型
from keras.models import Sequential
from keras.layers import Dense
#import tensorflow as tf #for
optimizer=tf.keras.optimizers.Adam(0.001)
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.initializers import Zeros, RandomNormal,
glorot_normal, glorot_uniform
#define the keras model
#sigmoid{0~1},tanh{-1~1},relu{0~z}
model=Sequential()
```

```
model.add(Dense(512, input_dim=f_dim, activation='relu'))
#model.add(Dense(512, input_dim=X.shape[1],
activation='relu',kernel_initializer=glorot_uniform()))
#model.add(Dense(512, input_dim=X.shape[1],
activation='relu',kernel_initializer=glorot_uniform(),bias_initializer
=Zeros()))

#model.add(BatchNormalization()) #不可使用在 Regressor 模
型，若要正規化，使用 MinMaxScaler，並反正規化預測值
#model.add(Dense(64, activation='relu'))
#model.add(BatchNormalization())
#model.add(Dense(32, activation='relu'))
#from keras.layers import Dropout
#model.add(Dropout(0.2))
#model.add(Dense(8, activation='relu'))
#model.add(Dense(1, activation='linear'))
model.add(Dense(1))

#tf.keras.optimizers.Adam(0.0001)
#compile the keras model
#model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
#model.compile(loss='mean_absolute_error',
optimizer=tf.keras.optimizers.Adam(0.001), metrics=['accuracy'])
#rmsprop
model.compile(loss='mean_absolute_error', optimizer='adam',
metrics=['mean_absolute_error'])
#fit the keras model on the dataset
#model.fit(X_Train, y_Train, validation_split=0.2, epoch=15,
batch_size=10)

#Dataframe to ndarray
X=X.to_numpy()
y=y.to_numpy()

n_train = recnum-100 #renum-100 筆作為訓練資料，最後 100 做
測試
```

```
#訓練模型
#model.fit(X[:n_train], y[:n_train]) #訓練模型
history=model.fit(X[:n_train], y[:n_train], epochs=20,
validation_split=0.2, verbose=0)
#預測資料
pred=model.predict(X[n_train:]) #預測資料

#儲存模型
#from keras.models import load_model
#model.save(filenameModel)
#print("儲存 model",filenameModel)

#統計執行時間
endtime=datetime.datetime.now()
print('開始時間: ',starttime)
print('演算時間: ',Algorithmtime)
print('結束時間: ', endtime)
print('花費時間: ',endtime-starttime)

#顯示準確率，評價指標(loss function)
#SSE、MSE、RMSE、MAE 越低越好
real=y[n_train:] #真實資料

#calculate 均方根誤差 RMSE (root mean squared error)
from math import sqrt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

rmse=sqrt(mean_squared_error(real, pred))
print('Test RMSE: %.4f' % rmse)
#calculate 平均絕對誤差 (MAE)(mean absolut error)
mae=mean_absolute_error(real, pred)
print('Test MAE: %.4f' % mae)

#print(pred)
pred=pred.reshape((len(pred))) #2D->1D
#print(pred)
```

```
#將預測結果與真實資料合併成 DataFrame
from pandas import DataFrame

real=y[n_train:]
df_PredResult=DataFrame({"Real":real,"Pred":pred})
#df_PredResult=DataFrame(pred[:,0])
#print("DataFrame: Pred, Real")
print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)

import matplotlib.pyplot as plt

plt.plot(real, 'r-')#真實
plt.plot(pred, 'b:')#預測
plt.legend(['real','pred'])
plt.xlabel("Ti")
plt.ylabel("Price")
plt.title("StockTSMC-ClosingPrice (DNN)")
plt.show()



def plot_loss(history):
    plt.plot(history.history['loss'], label='loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    #plt.ylim([0, 10])
    plt.xlabel('Epoch')
    plt.ylabel('Error')
    plt.legend()
    plt.grid(True)

plot_loss(history)
```

四、 StockTSMC_LSTM

```
#機器學習演算法(LSTM)範例 應用於 TSMC 資料集
#designed by 翁政雄 博士/教授
```

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
#from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import LSTM
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df=DataFrame(data)
	cols, names = list(), list()
	#input sequense(t-n, ...t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	#forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i ==0:
			names+=[('var%d(t)' % (j+1)) for j in range(n_vars)]
		else:
			names+=[('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
	#put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	if dropnan:
```

```
            agg.dropna(inplace=True)
    return agg

#設定檔名(資料集、預測值結果、模型輸出)
filename='Datasets\StockTsmc.csv'
filenameOut='Datasets\StockTsmcPred_LSTM.csv'
filenameModel='Datasets\StockTsmc_LSTM.h5'

#load dataset
dataset=read_csv(filename, header=0, index_col=0)
values=dataset.values
'''
#integer encode direction
encoder = LabelEncoder()
values[:,4]=encoder.fit_transform(values[:,4])
#ensure all data is float
'''

#資料型態轉換
values=values.astype('float32')
#normalize features
scaler=MinMaxScaler(feature_range=(0,1))
scaled=scaler.fit_transform(values)

#specify the number of lag hours
n_ts=3#用前 n_ts 天預測第 n_ts 天的結果，即第 n_ts+1 天的結果
n_features=5#特徵數
#frame as supervised learing
reframed=series_to_supervised(scaled, n_ts, 1)
#(1220,20),用前 3 天預測第四天的結果，必須 4 天資料(4x5=20 特徵欄位)，合併成一列(row)
print("reframed.shape:",reframed.shape)
#print(type(reframed))#DataFrame
#print(type(values))#numpy.ndarray

#split into train and test sets
```

```
values=reframed.values

##取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資料做為測試資料
n_train_ts=200*5 #1220=1000+220
train=values[:n_train_ts,:]
test=values[n_train_ts:,:]

#切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設 timesteps=5)
# x1,x2,x3...xn 要變成 timesteps 組成的 3D 數據(x1,x2,x3)...y)
#n_ts = 3, 為每個 timesteps, 每個包含 5 個訓練例數據作為其預測輸入(x1,x2...n_features)
n_obs = n_ts * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print("train_X.shape:", train_X.shape)
print("len(train_X):",len(train_X))
print("train_y.shape:", train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_ts, n_features))
test_X = test_X.reshape((test_X.shape[0], n_ts, n_features))
print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1],
train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                                validation_data=(test_X, test_y),
verbose=0, shuffle=False)
```

```
# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_ts*n_features))

# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
print("inv_yhat.shpe=",inv_yhat.shape)
inv_yhat=scaler.inverse_transform(inv_yhat)
inv_yhat=inv_yhat[:,0]

#真實資料(test_y)還原成原始數值，先前用 minmaxscaler 將數值正規化(0~1)
#維度轉換:1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#1D(220,)
test_y=test_y.reshape((len(test_y),1)) #1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#2D(220,1)

#合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y=concatenate((test_y, test_X[:,-(n_features-1):]),axis=1)
#真實資料(inv_y)還原成原始數值(test_y)
inv_y=scaler.inverse_transform(inv_y)
#只取第一欄的值，即標籤欄(真實)的值(value)
inv_y=inv_y[:,0]

#calculate 均方根誤差 RMSE(root mean squared error)
rmse=sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
#calculate 平均絕對誤差 MAE(mean absolute error)
```

```
mae=mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)



test_y,predict_y=inv_y,inv_yhat

import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:') #預測
plt.plot(test_y, 'r-') #real
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("ClosingPrice")
plt.title("Prediction(LSTM)")
plt.show()

#from keras.model import load_model
model.save(filenameModel)
print("儲存 Model",filenameModel)

#將預測結果與真實資料合併成 DataFrame
#from pandas import DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred,Real")
print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
```

五、 StockTSMC_RNN

```
#機器學習演算法(RNN)範例 應用於 TSMC 資料集
#designed by 翁政雄 博士/教授
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
```

```
from sklearn.preprocessing import MinMaxScaler
#from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import LSTM
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df=DataFrame(data)
	cols, names = list(), list()
	#input sequense(t-n, ...t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	#forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i ==0:
			names+=[('var%d(t)' % (j+1)) for j in range(n_vars)]
		else:
			names+=[('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
	#put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	if dropnan:
		agg.dropna(inplace=True)
	return agg

#設定檔名(資料集、預測值結果、模型輸出)
filename='Datasets\StockTsmc.csv'
filenameOut='Datasets\StockTsmcPred_RNN.csv'
```

```
filenameModel='Datasets\StockTsmc_RNN.h5'

#load dataset
dataset=read_csv(filename, header=0, index_col=0)
values=dataset.values
'''
#integer encode direction
encoder = LabelEncoder()
values[:,4]=encoder.fit_transform(values[:,4])
#ensure all data is float
'''

#資料型態轉換
values=values.astype('float32')
#normalize features
scaler=MinMaxScaler(feature_range=(0,1))
scaled=scaler.fit_transform(values)

#specify the number of lag hours
n_ts=3#用前 n_ts 天預測第 n_ts 天的結果，即第 n_ts+1 天的結果
n_features=5#特徵數
#frame as supervised learing
reframed=series_to_supervised(scaled, n_ts, 1)
#(1220,20),用前 3 天預測第四天的結果，必須 4 天資料(4x5=20 特徵欄位)，合併成一列(row)
print("reframed.shape:",reframed.shape)
#print(type(reframed))#DataFrame
#print(type(values))#numpy.ndarray

#split into train and test sets
values=reframed.values

##取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資料做為測試資料
n_train_ts=200*5 #1220=1000+220
train=values[:n_train_ts,:]
```

```
test=values[n_train_ts:,:]

#切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設
timesteps=5)
# x1,x2,x3...xn 要變成 timesteps 組成的 3D 數據(x1,x2,x3)...y)
#n_ts = 3, 為每個 timesteps, 每個包含 5 個訓練例數據作為其
預測輸入(x1,x2...n_features)
n_obs = n_ts * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print("train_X.shape:", train_X.shape)
print("len(train_X):",len(train_X))
print("train_y.shape:", train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_ts, n_features))
test_X = test_X.reshape((test_X.shape[0], n_ts, n_features))
print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design network
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1],
train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                                validation_data=(test_X, test_y),
verbose=0, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

```
# Make a prediction
yhat = model.predict(test_X)

# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_ts*n_features))

# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
print("inv_yhat.shpe=",inv_yhat.shape)
inv_yhat=scaler.inverse_transform(inv_yhat)
inv_yhat=inv_yhat[:,0]

#真實資料(test_y)還原成原始數值，先前用 minmaxscaler 將數值正規化(0~1)
#維度轉換:1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#1D(220,)
test_y=test_y.reshape((len(test_y),1)) #1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#2D(220,1)

#合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y=concatenate((test_y, test_X[:,-(n_features-1):]),axis=1)
#真實資料(inv_y)還原成原始數值(test_y)
inv_y=scaler.inverse_transform(inv_y)
#只取第一欄的值，即標籤欄(真實)的值(value)
inv_y=inv_y[:,0]

#calculate 均方根誤差 RMSE(root mean squared error)
rmse=sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
#calculate 平均絕對誤差 MAE(mean absolute error)
mae=mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)



test_y,predict_y=inv_y,inv_yhat
```

```
import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:') #預測
plt.plot(test_y, 'r-') #real
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("ClosingPrice")
plt.title("Prediction(RNN)")
plt.show()

#from keras.model import load_model
model.save(filenameModel)
print("儲存 Model",filenameModel)

#將預測結果與真實資料合併成 DataFrame
#from pandas import DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred,Real")
print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
```

20241014. 深度學習的雙向與堆疊

壹、 相關知識

一、 遞迴式神經網路（RNN）是一種深度學習模型，專為處理具有時序或序列關係的資料而設計，通過隱藏層的循環結構，利用當前輸入和短期記憶來預測後續輸出，常用於語音辨識、語言翻譯和時間序列分析等任務。

貳、 程式碼解讀

一、 -n_features 的含意是由往前後面刪掉不是 label 的欄位（特徵欄）

二、 將時間序列資料轉換為適合監督學習模型的資料集，尤其是用於時間序列預測任務。通過這個函數，可以生成過去和未來的多個時間步的資料，作為模型的輸入和輸出。

三、 Bidirectional

1. 定義：
   (1) 一種增強型的循環神經網路（RNN）結構，處理序列數據時同時考慮「過去」和「未來」的上下文資訊。
2. 核心特性：
   (1) 雙向處理：正向（從頭到尾）與反向（從尾到頭）同時進行。
   (2) 上下文結合：將兩個方向的輸出合併（如拼接、相加等）。
3. 優勢：
   (1) 捕捉全局資訊：全面理解序列的前後文。
   (2) 提升性能：對 NLP 和語音處理等序列相關任務效果顯著。
4. 使用場景：
   (1) 自然語言處理(NLP)：
      i. 文本分類
      ii. 序列標註（如命名實體識別）
      iii. 機器翻譯
   (2) 語音處理：
      i. 語音到文本轉錄
      ii. 語音情緒分析
   (3) 生物資訊學：
      i. DNA 序列分析

四、 Stacked

1. 定義：
   (1) 在神經網路中，將多層相同或不同類型的層（如 LSTM、GRU、RNN）垂直堆疊，形成更深的結構，提升模型表現力和學習能力。
2. 核心特性：
   (1) 多層結構：每一層的輸出作為下一層的輸入。
   (2) 特徵提取：下層捕捉簡單特徵，上層捕捉更高級的抽象特徵。
   (3) 深度學習：通過增加層數，提升模型學習複雜模式的能力。
3. 優勢：
   (1) 增強模型能力：捕捉更深層次的序列關係。
   (2) 解決複雜問題：特別適用於長序列或結構複雜的數據。
4. 使用場景：
   (1) 自然語言處理(NLP)：
      i. 長文本分析
      ii. 機器翻譯
   (2) 語音處理：
      i. 複雜語音信號分析
   (3) 時間序列分析：
      i. 股票價格預測
      ii. 氣象預測

五、 當 dense 層越來越小的情況

1. 模型壓縮：為了減少模型的計算複雜度和存儲需求，通常會對層進行剪枝或量化，這會導致 dense 層的參數數量減少。
2. 過擬合控制：在訓練過程中，如果發現模型在訓練集上表現良好但在驗證集上表現不佳，則可能會選擇減少 dense 層的大小，以降低模型的複雜度，從而減少過擬合的風險。
3. 特徵提取：在一些應用中，隨著網絡層數的增加，前面的卷積層已經提取了足夠的特徵，因此後續的 dense 層可以設計得較小，以專注於這些特徵的整合。

六、 當 dense 層越來越大的情況

1. 增加模型容量：如果模型在訓練集和驗證集上的表現都不佳，可能需要增加 dense 層的大小，以提高模型的學習能力和表達能力。

2. 複雜任務需求：對於需要更高預測準確度的任務（如圖像分類、語音識別等），可能會設計更大的 dense 層，以便捕捉更多的特徵和模式。
3. 多任務學習：在多任務學習中，可能需要為不同的輸出設計較大的 dense 層，以便同時處理多個任務並共享學習到的信息。

參、 附錄 – 程式碼

一、 StockTsmc_BiRNN

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
#from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN, Bidirectional
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df = DataFrame(data)
	cols, names = list(), list()
	# input sequence (t-n, ...t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
```

```
                names += [('var%d(t)' % (j+1)) for j in
range(n_vars)]
            else:
                names += [('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    if dropnan:
        agg.dropna(inplace=True)
    return agg

# 設定檔名(資料集、預測值結果、模型輸出)
filename = 'Datasets\StockTsmc.csv'
filenameOut = 'Datasets\StockTsmcPred_BiRNN.csv'
filenameModel = 'Datasets\StockTsmc_BiRNN.h5'

# load dataset
dataset = read_csv(filename, header=0, index_col=0)
values = dataset.values
'''
# integer encode direction
encoder = LabelEncoder()
values[:, 4] = encoder.fit_transform(values[:, 4])
# ensure all data is float
'''

# 資料型態轉換
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

# specify the number of lag hours
n_ts = 3  # 用前 n_ts 天預測第 n_ts 天的結果，即第 n_ts+1 天
的結果
n_features = 5  # 特徵數
```

```
# frame as supervised learning
reframed = series_to_supervised(scaled, n_ts, 1)
# (1220,20), 用前 3 天預測第四天的結果，必須 4 天資料
(4x5=20 特徵欄位)，合併成一列(row)
print("reframed.shape:", reframed.shape)

# split into train and test sets
values = reframed.values

# 取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資
料做為測試資料
n_train_ts = 200 * 5   # 1220=1000+220
train = values[:n_train_ts, :]
test = values[n_train_ts:, :]

# 切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假
設 timesteps=5)
n_obs = n_ts * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print("train_X.shape:", train_X.shape)
print("len(train_X):", len(train_X))
print("train_y.shape:", train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_ts, n_features))
test_X = test_X.reshape((test_X.shape[0], n_ts, n_features))
print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design network with Bidirectional RNN
model = Sequential()
model.add(Bidirectional(SimpleRNN(50,
input_shape=(train_X.shape[1], train_X.shape[2]))))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
```

```
# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                                validation_data=(test_X, test_y),
verbose=0, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_ts * n_features))

# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:, 0]

# 真實資料(test_y)還原成原始數值
test_y = test_y.reshape((len(test_y), 1))

# 合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y = concatenate((test_y, test_X[:, -(n_features-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:, 0]

# calculate RMSE and MAE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
mae = mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)

# Plot prediction vs real
```

```
test_y, predict_y = inv_y, inv_yhat
import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:')  # 預測
plt.plot(test_y, 'r-')  # 真實資料
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("ClosingPrice")
plt.title("Prediction(BiRNN)")
plt.show()

# Save the model
model.save(filenameModel)
print("儲存 Model", filenameModel)

# Save prediction results to CSV
df_PredResult = DataFrame({"Real": inv_y, "Pred": inv_yhat})
print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案", filenameOut)
df_PredResult.to_csv(filenameOut)
```

二、 StockTSMC_RNN

```
#機器學習演算法(RNN)範例 應用於 TSMC 資料集
#designed by 翁政雄 博士/教授
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
#from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import LSTM
from sklearn.metrics import mean_absolute_error
```

```
from sklearn.metrics import mean_squared_error

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1,
dropnan=True):#out 是天數
	n_vars = 1 if type(data) is list else data.shape[1]
	df=DataFrame(data)
	cols, names = list(), list()
	#input sequense(t-n, ...t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	#forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i ==0:
			names+=[('var%d(t)' % (j+1)) for j in range(n_vars)]
		else:
			names+=[('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
	#put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	if dropnan:
		agg.dropna(inplace=True)
	return agg

#設定檔名(資料集、預測值結果、模型輸出)
filename='Datasets\StockTsmc.csv'
filenameOut='Datasets\StockTsmcPred_RNN.csv'
filenameModel='Datasets\StockTsmc_RNN.h5'

#load dataset
dataset=read_csv(filename, header=0, index_col=0)
values=dataset.values
'''
```

```
#integer encode direction
encoder = LabelEncoder()
values[:,4]=encoder.fit_transform(values[:,4])
#ensure all data is float
'''

#資料型態轉換
values=values.astype('float32')
#normalize features
scaler=MinMaxScaler(feature_range=(0,1))
scaled=scaler.fit_transform(values)

#specify the number of lag hours
n_ts=3#用前 n_ts 天預測第 n_ts 天的結果，即第 n_ts+1 天的結果
n_features=5#特徵數
#frame as supervised learing
reframed=series_to_supervised(scaled, n_ts, 1)
#(1220,20),用前 3 天預測第四天的結果，必須 4 天資料(4x5=20 特徵欄位)，合併成一列(row)
print("reframed.shape:",reframed.shape)
#print(type(reframed))#DataFrame
#print(type(values))#numpy.ndarray

#split into train and test sets
values=reframed.values

##取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資料做為測試資料
n_train_ts=200*5 #1220=1000+220
train=values[:n_train_ts,:]
test=values[n_train_ts:,:]

#切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設 timesteps=5)
# x1,x2,x3...xn 要變成 timesteps 組成的 3D 數據(x1,x2,x3)...y)
#n_ts = 3, 為每個 timesteps, 每個包含 5 個訓練例數據作為其
```

```
預測輸入(x1,x2...n_features)
n_obs = n_ts * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print("train_X.shape:", train_X.shape)
print("len(train_X):",len(train_X))
print("train_y.shape:", train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_ts, n_features))
test_X = test_X.reshape((test_X.shape[0], n_ts, n_features))
print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design network
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1],
train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                                validation_data=(test_X, test_y),
verbose=0, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_ts*n_features))
```

```
# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
print("inv_yhat.shpe=",inv_yhat.shape)
inv_yhat=scaler.inverse_transform(inv_yhat)#轉換為真正的數值
inv_yhat=inv_yhat[:,0]

#真實資料(test_y)還原成原始數值，先前用 minmaxscaler 將數值正規化(0~1)
#維度轉換:1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#1D(220,)
test_y=test_y.reshape((len(test_y),1)) #1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#2D(220,1)

#合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y=concatenate((test_y, test_X[:,-(n_features-1):]),axis=1)
#真實資料(inv_y)還原成原始數值(test_y)
inv_y=scaler.inverse_transform(inv_y)
#只取第一欄的值，即標籤欄(真實)的值(value)
inv_y=inv_y[:,0]

#calculate 均方根誤差 RMSE(root mean squared error)
rmse=sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
#calculate 平均絕對誤差 MAE(mean absolute error)
mae=mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)



test_y,predict_y=inv_y,inv_yhat

import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:') #預測
plt.plot(test_y, 'r-') #real
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
```

```
plt.ylabel("ClosingPrice")
plt.title("Prediction(RNN)")
plt.show()

#from keras.model import load_model
model.save(filenameModel)
print("儲存 Model",filenameModel)

#將預測結果與真實資料合併成 DataFrame
#from pandas import DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred,Real")
print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
```

三、 StockTSMC_RNN_OpeningPrice

```
#機器學習演算法(RNN)範例 應用於 TSMC 資料集
#designed by 翁政雄 博士/教授
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
#from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import LSTM
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
```

```
    df=DataFrame(data)
    cols, names = list(), list()
    #input sequense(t-n, ...t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
    #forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i ==0:
            names+=[('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names+=[('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
    #put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    if dropnan:
        agg.dropna(inplace=True)
    return agg

#設定檔名(資料集、預測值結果、模型輸出)
filename='Datasets\StockTsmc.csv'
filenameOut='Datasets\StockTsmcPred_RNN.csv'
filenameModel='Datasets\StockTsmc_RNN.h5'

#load dataset
dataset=read_csv(filename, header=0, index_col=0)

#預測開盤價，調整欄位:將'OpeningPrice'放在第一個欄位
dataset=dataset[['OpeningPrice','ClosingPrice','HighestPrice','Lowe
stPrice','Volume']]
columnName=list(dataset.columns[0:6])
print(columnName)

values=dataset.values
```

```
'''
#integer encode direction
encoder = LabelEncoder()
values[:,4]=encoder.fit_transform(values[:,4])
#ensure all data is float
'''

#資料型態轉換
values=values.astype('float32')
#normalize features
scaler=MinMaxScaler(feature_range=(0,1))
scaled=scaler.fit_transform(values)

#specify the number of lag hours
n_ts=3#用前 n_ts 天預測第 n_ts 天的結果，即第 n_ts+1 天的結果
n_features=5#特徵數
#frame as supervised learing
reframed=series_to_supervised(scaled, n_ts, 1)
#(1220,20),用前 3 天預測第四天的結果，必須 4 天資料(4x5=20 特徵欄位)，合併成一列(row)
print("reframed.shape:",reframed.shape)
#print(type(reframed))#DataFrame
#print(type(values))#numpy.ndarray

#split into train and test sets
values=reframed.values

##取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資料做為測試資料
n_train_ts=200*5 #1220=1000+220
train=values[:n_train_ts,:]
test=values[n_train_ts:,:]

#切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設 timesteps=5)
```

```
# x1,x2,x3...xn 要變成 timesteps 組成的 3D 數據(x1,x2,x3)...y)
#n_ts = 3, 為每個 timesteps, 每個包含 5 個訓練例數據作為其
預測輸入(x1,x2...n_features)
n_obs = n_ts * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print("train_X.shape:", train_X.shape)
print("len(train_X):",len(train_X))
print("train_y.shape:", train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_ts, n_features))
test_X = test_X.reshape((test_X.shape[0], n_ts, n_features))
print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design network
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1],
train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                                validation_data=(test_X, test_y),
verbose=0, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)
```

```
# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_ts*n_features))

# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
print("inv_yhat.shpe=",inv_yhat.shape)
inv_yhat=scaler.inverse_transform(inv_yhat)
inv_yhat=inv_yhat[:,0]

#真實資料(test_y)還原成原始數值，先前用 minmaxscaler 將數值正規化(0~1)
#維度轉換:1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#1D(220,)
test_y=test_y.reshape((len(test_y),1)) #1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#2D(220,1)

#合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y=concatenate((test_y, test_X[:,-(n_features-1):]),axis=1)
#真實資料(inv_y)還原成原始數值(test_y)
inv_y=scaler.inverse_transform(inv_y)
#只取第一欄的值，即標籤欄(真實)的值(value)
inv_y=inv_y[:,0]

#calculate 均方根誤差 RMSE(root mean squared error)
rmse=sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
#calculate 平均絕對誤差 MAE(mean absolute error)
mae=mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)



test_y,predict_y=inv_y,inv_yhat

import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:') #預測
plt.plot(test_y, 'r-') #real
```

```
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("OpeningPrice")
plt.title("Prediction(RNN)")
plt.show()

#from keras.model import load_model
model.save(filenameModel)
print("儲存 Model",filenameModel)

#將預測結果與真實資料合併成 DataFrame
#from pandas import DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred,Real")
print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
```

四、 StockTsmc_StackedBiRNN

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN, Bidirectional
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
```

```
	# input sequence (t-n, ...t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
			names += [('var%d(t)' % (j+1)) for j in
range(n_vars)]
		else:
			names += [('var%d(t+%d)' % (j+1), i) for j in
range(n_vars)]
	# put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	if dropnan:
		agg.dropna(inplace=True)
	return agg

# 設定檔名(資料集、預測值結果、模型輸出)
filename = 'Datasets\StockTsmc.csv'
filenameOut = 'Datasets\StockTsmcPred_BiStackedRNN.csv'
filenameModel = 'Datasets\StockTsmc_BiStackedRNN.h5'

# load dataset
dataset = read_csv(filename, header=0, index_col=0)
values = dataset.values

# 資料型態轉換
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

# specify the number of lag hours
```

```
n_ts = 3  # 用前 n_ts 天預測第 n_ts 天的結果，即第 n_ts+1 天的結果
n_features = 5  # 特徵數
# frame as supervised learning
reframed = series_to_supervised(scaled, n_ts, 1)
print("reframed.shape:", reframed.shape)

# split into train and test sets
values = reframed.values

# 取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資料做為測試資料
n_train_ts = 200 * 5  # 1220=1000+220
train = values[:n_train_ts, :]
test = values[n_train_ts:, :]

# 切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設 timesteps=5)
n_obs = n_ts * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print("train_X.shape:", train_X.shape)
print("len(train_X):", len(train_X))
print("train_y.shape:", train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_ts, n_features))
test_X = test_X.reshape((test_X.shape[0], n_ts, n_features))
print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design bidirectional stacked RNN network
model = Sequential()
# 第一層雙向 RNN, return_sequences=True
model.add(Bidirectional(SimpleRNN(50, return_sequences=True),
input_shape=(train_X.shape[1], train_X.shape[2])))
# 第二層雙向 RNN, return_sequences=False 默認
```

```
model.add(Bidirectional(SimpleRNN(50)))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                                validation_data=(test_X, test_y),
verbose=0, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_ts * n_features))

# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:, 0]

# 真實資料(test_y)還原成原始數值
test_y = test_y.reshape((len(test_y), 1))

# 合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y = concatenate((test_y, test_X[:, -(n_features-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:, 0]

# calculate RMSE and MAE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

```
mae = mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)

# Plot prediction vs real
test_y, predict_y = inv_y, inv_yhat
import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:')  # 預測
plt.plot(test_y, 'r-')  # 真實資料
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("ClosingPrice")
plt.title("Prediction(Bidirectional Stacked RNN)")
plt.show()

# Save the model
model.save(filenameModel)
print("儲存 Model", filenameModel)

# Save prediction results to CSV
df_PredResult = DataFrame({"Real": inv_y, "Pred": inv_yhat})
print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案", filenameOut)
df_PredResult.to_csv(filenameOut)
```

五、 StockTsmc_StackedRNN

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
#from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
```

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df = DataFrame(data)
	cols, names = list(), list()
	# input sequence (t-n, ...t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
			names += [('var%d(t)' % (j+1)) for j in
range(n_vars)]
		else:
			names += [('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
	# put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	if dropnan:
		agg.dropna(inplace=True)
	return agg

# 設定檔名(資料集、預測值結果、模型輸出)
filename = 'Datasets\StockTsmc.csv'
filenameOut = 'Datasets\StockTsmcPred_StackedRNN.csv'
filenameModel = 'Datasets\StockTsmc_StackedRNN.h5'

# load dataset
dataset = read_csv(filename, header=0, index_col=0)
values = dataset.values
```

```
'''
# integer encode direction
encoder = LabelEncoder()
values[:, 4] = encoder.fit_transform(values[:, 4])
# ensure all data is float
'''

# 資料型態轉換
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

# specify the number of lag hours
n_ts = 3  # 用前 n_ts 天預測第 n_ts 天的結果，即第 n_ts+1 天的結果
n_features = 5  # 特徵數
# frame as supervised learning
reframed = series_to_supervised(scaled, n_ts, 1)
# (1220,20), 用前 3 天預測第四天的結果，必須 4 天資料(4x5=20 特徵欄位)，合併成一列(row)
print("reframed.shape:", reframed.shape)

# split into train and test sets
values = reframed.values

# 取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資料做為測試資料
n_train_ts = 200 * 5  # 1220=1000+220
train = values[:n_train_ts, :]
test = values[n_train_ts:, :]

# 切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設 timesteps=5)
n_obs = n_ts * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
```

```
print("train_X.shape:", train_X.shape)
print("len(train_X):", len(train_X))
print("train_y.shape:", train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_ts, n_features))
test_X = test_X.reshape((test_X.shape[0], n_ts, n_features))
print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design stacked RNN network
model = Sequential()
model.add(SimpleRNN(50, return_sequences=True,
input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(SimpleRNN(50))   # 第二層 RNN,
return_sequences=False 默認的
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                                validation_data=(test_X, test_y),
verbose=0, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_ts * n_features))

# Invert scaling for forecast
```

```
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:, 0]

# 真實資料(test_y)還原成原始數值
test_y = test_y.reshape((len(test_y), 1))

# 合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y = concatenate((test_y, test_X[:, -(n_features-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:, 0]

# calculate RMSE and MAE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
mae = mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)

# Plot prediction vs real
test_y, predict_y = inv_y, inv_yhat
import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:')  # 預測
plt.plot(test_y, 'r-')  # 真實資料
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("ClosingPrice")
plt.title("Prediction(Stacked RNN)")
plt.show()

# Save the model
model.save(filenameModel)
print("儲存 Model", filenameModel)

# Save prediction results to CSV
df_PredResult = DataFrame({"Real": inv_y, "Pred": inv_yhat})
print('顯示前 5 筆資料')
```

```
print(df_PredResult.head())
print("儲存檔案", filenameOut)
df_PredResult.to_csv(filenameOut)
```

20241021. 優化器與人工智慧的意義

壹、 相關知識

一、 Python 支援的優化器有哪些？

在 Python 中，特別是使用深度學習框架如 TensorFlow、PyTorch 和 Keras 時，支援的優化器種類相當多。有梯度下降法（Gradient Descent）、動量法（Momentum-based Optimizer）、自適應優化器（Adaptive Optimizers）、二階優化法、FTRL（Follow The Regularized Leader）、Amsgrad 等，其中每種皆還有細分及相關演化。

這些優化器可以根據不同的模型和數據集進行選擇和調整，許多都允許設定超參數，如學習率（learning rate）和權重衰減（weight decay）。

貳、 附錄 – Excel 截圖

一、

神經元	4			
x	1	0	1	Θ4
w	0.2	0.4	-0.5	
x*w	0.2	0	-0.5	-0.4
SUM	-0.7			
Logistic(x)	0.331812228			
神經元	5			
x	1	0	1	Θ5
w	-0.3	0.1	0.2	
x*w	-0.3	0	0.2	0.2
SUM	0.1			
Logistic(x)	0.524979187			
神經元	6			
x	0.331812228	0.524979187		Θ6
w	-0.3	-0.2		
x*w	-0.099543668	-0.104995837		0.1
SUM	-0.104539506			
Logistic(x)	0.473888899			

$$Error_j = O_j \times (1 - O_j) \times (T_j - O_j)$$

神經元	6	誤差值
O_j	0.473888899	
$1-O_j$	0.526111101	
T_j-O_j	0.526111101	
$Error_j$	0.131169078	

二、 Tj是目標(Target)

神經元	4	誤差值
O_j	0.331812228	
$1-O_j$	0.668187772	
$Error_k * W_{jk}$	-0.039350723	j=4,k=6
$Error_j$	-0.008724562	

神經元	5	誤差值
O_j	0.524979187	
$1-O_j$	0.475020813	
$Error_k * W_{jk}$	-0.026233816	j=5,k=6
$Error_j$	-0.006542085	

三、
$$Error_j = O_j \times (1 - O_j) \times \sum_k Error_k \times w_{jk}$$

權重	W(R)	W	ΔW	*l(learning rate)*	Errorj	Oi
w56	-0.138025068	-0.2	0.061974932	0.9	0.131169078	0.524979187
w46	-0.260828846	-0.3	0.039171154	0.9	0.131169078	0.331812228
w35	0.194112123	0.2	-0.005887877	0.9	-0.006542085	1
w25	0.1	0.1	0	0.9	-0.006542085	0
w15	-0.305887877	-0.3	-0.005887877	0.9	-0.006542085	1
w34	-0.507852106	-0.5	-0.007852106	0.9	-0.008724562	1
w24	0.4	0.4	0	0.9	-0.008724562	0
w14	0.192147894	0.2	-0.007852106	0.9	-0.008724562	1

偏差項	Θ (R)	Θ	ΔΘ	*l*	Errorj
Θ6	0.21805217	0.1	0.11805217	0.9	0.131169078
Θ5	0.194112123	0.2	-0.005887877	0.9	-0.006542085
Θ4	-0.407852106	-0.4	-0.007852106	0.9	-0.008724562

四、

w14	w15	w24	w25	w34	w35	w46	w56	Θ4	Θ5	Θ6	
0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1	原始
0.192147894	-0.305887877	0.4	0.1	-0.507852106	0.194112123	-0.260828846	-0.14	-0.407852106	0.194112123	0.21805217	R1

(X1,X2,X3,Y)=(1,0,1,1)

$$I_j = \sum_i w_{ij} o_i + \theta_j$$

$$\Delta W_{ij} = l \times Error_j \times O_i$$

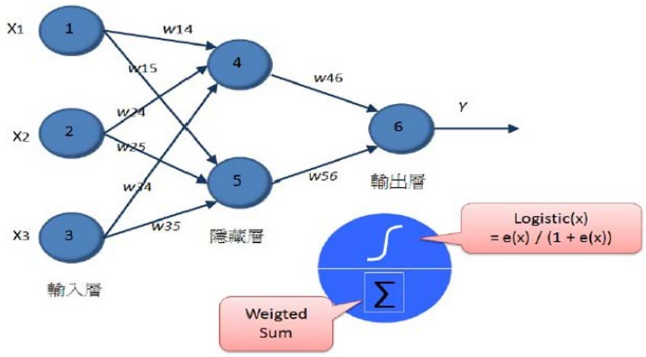
$$\Delta \theta_j = l \times Error_j$$

常為隨機產生

I4是神經元4接收到的訊息

五、
$$\Delta W_{ij} = l \times Error_j \times O_i$$

六、


X1
X2
X3
1
2
3
4
5
6
w14
w15
w24
w25
w34
w35
w46
w56
Y
輸入層
隱藏層
輸出層
Logistic(x)
= e(x) / (1 + e(x))
Weigted
Sum

20241028. 比較 DNN、RNN、CNN 並運用 RNN、LSTM、GRU

壹、 相關知識

一、 LSTM（長短期記憶網絡）是對 RNN（遞迴式神經網路）的改進，旨在解決 RNN 處理長序列時容易出現的梯度消失問題。LSTM 通過引入「門控機制」（遺忘門、輸入門、輸出門）和細胞狀態，選擇性地記住或遺忘信息，能有效捕捉長期依賴關係，尤其適用於語言翻譯、時間序列預測和語音辨識等任務。雖然 LSTM 結構更複雜且計算成本較高，但其長期記憶能力顯著優於傳統 RNN，是深度學習中應用廣泛的模型之一。

二、 DNN、RNN、CNN 三者比較：

特徵	DNN	RNN	CNN
數據類型	通用型數據	序列數據（時序、文本）	格網數據（圖像、視頻）
結構	完全連接層	循環結構，具有記憶能力	卷積層、池化層、全連接層
權重共享	否	是	是（局部共享）
依賴關係	無依賴	時序依賴	空間依賴
應用範圍	分類回歸	語言、時間序列	圖像識別、目標檢測
優點	通用型模型，結構簡單，適用於多種任務	適合序列和時序任務，能記住上下文信息	對圖像任務優化，特徵提取能力強，計算效率高
缺點	無法處理時序或空間依賴，易過擬合，需大量數據支持	梯度消失/爆炸問題，長序列性能差，計算成本高	僅適用於格網數據，對輸入形狀要求高，需大量數據訓練

三、 GRU 為什麼可以解決 LSTM 的問題

1. 簡化 LSTM
2. 速度較快
3. 將輸入門、遺忘門、輸出門，簡化為更新門跟重設門

貳、 程式碼解讀

一、 train = values[:n_train_hours, :]：選擇資料集 values 的前 8760 小時作為訓練資料集 train

二、 test = values[n_train_hours:, :]：選擇剩下的資料作為測試資料集 test

三、 n_obs = n_hours * n_features：每個觀察值（即每個時間點）會包含 n_hours（假設時間步長）和 n_features（特徵數）的資料，這樣就得到了每個時間步長的觀察資料的總數。

四、 train_X, train_y = train[:, :n_obs], train[:, -n_features]：這裡將訓練資料 train 分成兩部分：

1. train_X：選擇 train 的前 n_obs 欄位，這些是作為輸入特徵的資料。
2. train_y：選擇 train 的最後 n_features 欄位，這些是作為目標變量（標籤）的資料。

五、 test_X, test_y = test[:, :n_obs], test[:, -n_features]：同樣的，將測試資料 test 分成特徵 test_X 和目標變量 test_y。

六、 LSTM 或 GRU 類神經網絡需要 3D 輸入資料，格式為[樣本數, 時間步長, 特徵數]。

1. train_X.shape[0]：這是訓練資料的樣本數。
2. n_hours：每個時間步的資料數量（例如，若 n_hours = 5，則每次模型將處理 5 小時的資料）。
3. n_features：每個時間步的特徵數量（例如，PM2.5、風速等）。

七、 train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))：將訓練資料 train_X 重新排列成 3D 格式，變為[樣本數, 時間步長, 特徵數]。

八、 test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))：將測試資料 test_X 也重新排列成相同的 3D 格式

參、 附錄 – 程式碼

一、 Airpollution_Preprocess

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 28 11:28:39 2024

@author: ASUS
```

```
"""

from pandas import read_csv
from datetime import datetime

import warnings
warnings.filterwarnings("ignore")

def parse(x):
    return datetime.strptime(x, '%Y %m %d %H')

filename='Datasets/Beijing_PM2.5_Data_Set.csv'
dataset=read_csv(filename, parse_dates=[['year', 'month', 'day',
'hour']],
                 index_col=0, date_parser=parse)
print('\n 顯示原始資料_組合(年月日時)')
print(dataset.head(5))

dataset.drop('No', axis=1, inplace=True)
dataset.colmuns=['pm2.5','dew','temp','press','wnd_dir','wnd_spd',
                 'snow','rain']
dataset.index.name='date'
dataset['pm2.5'].fillna(0,inplace=True)

print('\n 顯示前處理資料_0 取代 NA&新欄位名稱')
print(dataset.head(5))

dataset=dataset[24:]
print('\n 顯示前處理資料_drop 第一天 24 小時資料，因為
pm2.5 為 0')
print(dataset.head(5))
filename='Datasets\pollution.csv'
dataset.to_csv(filename)
print('資料前處理完成，儲存檔案:',filename)
```

二、 Beijing PM2.5 Data Set_GRU

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Mon Oct 28 11:04:53 2024

@author: ASUS
"""

from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, LSTM, GRU
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df=DataFrame(data)
	cols, names = list(), list()
	#input sequense(t-n, ...t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	#forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i ==0:
			names+=[('var%d(t)' % (j+1)) for j in range(n_vars)]
		else:
			names+=[('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
	#put it all together
```

```
    agg = concat(cols, axis=1)
    agg.columns = names
    if dropnan:
        agg.dropna(inplace=True)
    return agg

#設定檔名(資料集、預測值結果、模型輸出)
filename='Datasets\pollution.csv'
filenameOut='Datasets\pollution_RNN.csv'
filenameModel='Datasets\pollution6_RNN.h5'

#load dataset
dataset=read_csv(filename, header=0, index_col=0)
values=dataset.values

#integer encode direction
encoder = LabelEncoder()
values[:,4]=encoder.fit_transform(values[:,4])
#ensure all data is float



#資料型態轉換
values=values.astype('float32')
#normalize features
scaler=MinMaxScaler(feature_range=(0,1))
scaled=scaler.fit_transform(values)

#specify the number of lag hours
n_hours=3#用前 n_ts 天預測第 n_ts 天的結果，即第 n_ts+1 天
的結果
n_features=8#特徵數
#frame as supervised learing
reframed=series_to_supervised(scaled, n_hours, 1)
#(1220,20),用前 3 天預測第四天的結果，必須 4 天資料(4x5=20
特徵欄位)，合併成一列(row)
print("reframed.shape:",reframed.shape)
#print(type(reframed))#DataFrame
```

```
#print(type(values))#numpy.ndarray

#split into train and test sets
values=reframed.values

##取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資料做為測試資料
n_train_hours=365*24
train=values[:n_train_hours,:]
test=values[n_train_hours:,:]

#切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設 timesteps=5)
# x1,x2,x3...xn 要變成 timesteps 組成的 3D 數據(x1,x2,x3)...y)
#n_ts = 3, 為每個 timesteps, 每個包含 5 個訓練例數據作為其預測輸入(x1,x2...n_features)
n_obs = n_hours * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print("train_X.shape:", train_X.shape, len(train_X), train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))
print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design network
model = Sequential()
model.add(GRU(50, input_shape=(train_X.shape[1],
train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                                validation_data=(test_X, test_y),
```

```
verbose=0, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_hours*n_features))

# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
print("inv_yhat.shpe=",inv_yhat.shape)
inv_yhat=scaler.inverse_transform(inv_yhat)
inv_yhat=inv_yhat[:,0]

#真實資料(test_y)還原成原始數值，先前用 minmaxscaler 將數值正規化(0~1)
#維度轉換:1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#1D(220,)
test_y=test_y.reshape((len(test_y),1)) #1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#2D(220,1)

#合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y=concatenate((test_y, test_X[:,-(n_features-1):]),axis=1)
#真實資料(inv_y)還原成原始數值(test_y)
inv_y=scaler.inverse_transform(inv_y)
#只取第一欄的值，即標籤欄(真實)的值(value)
inv_y=inv_y[:,0]

#calculate 均方根誤差 RMSE(root mean squared error)
rmse=sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

```
#calculate 平均絕對誤差 MAE(mean absolute error)
mae=mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)



test_y,predict_y=inv_y,inv_yhat

import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:') #預測
plt.plot(test_y, 'r-') #real
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("ClosingPrice")
plt.title("Prediction(RNN)")
plt.show()

#from keras.model import load_model
model.save(filenameModel)
print("儲存 Model",filenameModel)

#將預測結果與真實資料合併成 DataFrame
#from pandas import DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred,Real")
print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
```

三、 Beijing PM2.5 Data Set_LSTM

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 28 11:04:53 2024

@author: ASUS
"""
```

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import LSTM
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df=DataFrame(data)
	cols, names = list(), list()
	#input sequense(t-n, ...t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	#forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i ==0:
			names+=[('var%d(t)' % (j+1)) for j in range(n_vars)]
		else:
			names+=[('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
	#put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	if dropnan:
```

```
            agg.dropna(inplace=True)
    return agg

#設定檔名(資料集、預測值結果、模型輸出)
filename='Datasets\pollution.csv'
filenameOut='Datasets\pollution_LSTM.csv'
filenameModel='Datasets\pollution6_LSTM.h5'

#load dataset
dataset=read_csv(filename, header=0, index_col=0)
values=dataset.values

#integer encode direction
encoder = LabelEncoder()
values[:,4]=encoder.fit_transform(values[:,4])
#ensure all data is float


#資料型態轉換
values=values.astype('float32')
#normalize features
scaler=MinMaxScaler(feature_range=(0,1))
scaled=scaler.fit_transform(values)

#specify the number of lag hours
n_hours=3#用前 n_ts 天預測第 n_ts 天的結果，即第 n_ts+1 天的結果
n_features=8#特徵數
#frame as supervised learing
reframed=series_to_supervised(scaled, n_hours, 1)
#(1220,20),用前 3 天預測第四天的結果，必須 4 天資料(4x5=20 特徵欄位)，合併成一列(row)
print("reframed.shape:",reframed.shape)
#print(type(reframed))#DataFrame
#print(type(values))#numpy.ndarray

#split into train and test sets
```

```
values=reframed.values

##取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資料做為測試資料
n_train_hours=365*24
train=values[:n_train_hours,:]
test=values[n_train_hours:,:]

#切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設 timesteps=5)
# x1,x2,x3...xn 要變成 timesteps 組成的 3D 數據(x1,x2,x3)...y)
#n_ts = 3, 為每個 timesteps, 每個包含 5 個訓練例數據作為其預測輸入(x1,x2...n_features)
n_obs = n_hours * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print("train_X.shape:", train_X.shape, len(train_X), train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))
print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1],
train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                                validation_data=(test_X, test_y),
verbose=0, shuffle=False)

# Plot training and validation loss
```

```
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_hours*n_features))

# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
print("inv_yhat.shpe=",inv_yhat.shape)
inv_yhat=scaler.inverse_transform(inv_yhat)
inv_yhat=inv_yhat[:,0] #0

#真實資料(test_y)還原成原始數值，先前用 minmaxscaler 將數值正規化(0~1)
#維度轉換:1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#1D(220,)
test_y=test_y.reshape((len(test_y),1)) #1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#2D(220,1)

#合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y=concatenate((test_y, test_X[:,-(n_features-1):]),axis=1)
#真實資料(inv_y)還原成原始數值(test_y)
inv_y=scaler.inverse_transform(inv_y)
#只取第一欄的值，即標籤欄(真實)的值(value)
inv_y=inv_y[:,0] #0

#calculate 均方根誤差 RMSE(root mean squared error)
rmse=sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
#calculate 平均絕對誤差 MAE(mean absolute error)
mae=mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)
```

```
test_y,predict_y=inv_y,inv_yhat

import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:') #預測
plt.plot(test_y, 'r-') #real
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("ClosingPrice")
plt.title("Prediction(RNN)")
plt.show()

#from keras.model import load_model
model.save(filenameModel)
print("儲存 Model",filenameModel)

#將預測結果與真實資料合併成 DataFrame
#from pandas import DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred,Real")
print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
```

四、 Beijing PM2.5 Data Set_RNN

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 28 11:04:53 2024

@author: ASUS
"""

from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
```

```
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import LSTM
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df=DataFrame(data)
	cols, names = list(), list()
	#input sequense(t-n, ...t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	#forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i ==0:
			names+=[('var%d(t)' % (j+1)) for j in range(n_vars)]
		else:
			names+=[('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
	#put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	if dropnan:
		agg.dropna(inplace=True)
	return agg
```

```
#設定檔名(資料集、預測值結果、模型輸出)
filename='Datasets\pollution.csv'
filenameOut='Datasets\pollution_RNN.csv'
filenameModel='Datasets\pollution6_RNN.h5'

#load dataset
dataset=read_csv(filename, header=0, index_col=0)
values=dataset.values

#integer encode direction
encoder = LabelEncoder()
values[:,4]=encoder.fit_transform(values[:,4])
#ensure all data is float



#資料型態轉換
values=values.astype('float32')
#normalize features
scaler=MinMaxScaler(feature_range=(0,1))
scaled=scaler.fit_transform(values)

#specify the number of lag hours
n_hours=3#用前 n_ts 天預測第 n_ts 天的結果，即第 n_ts+1 天
的結果
n_features=8#特徵數
#frame as supervised learing
reframed=series_to_supervised(scaled, n_hours, 1)
#(1220,20),用前 3 天預測第四天的結果，必須 4 天資料(4x5=20
特徵欄位)，合併成一列(row)
print("reframed.shape:",reframed.shape)
#print(type(reframed))#DataFrame
#print(type(values))#numpy.ndarray

#split into train and test sets
values=reframed.values

##取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資
```

```
料做為測試資料
n_train_hours=365*24
train=values[:n_train_hours,:]
test=values[n_train_hours:,:]

#切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設
timesteps=5)
# x1,x2,x3...xn 要變成 timesteps 組成的 3D 數據(x1,x2,x3)...y)
#n_ts = 3, 為每個 timesteps, 每個包含 5 個訓練例數據作為其
預測輸入(x1,x2...n_features)
n_obs = n_hours * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print("train_X.shape:", train_X.shape, len(train_X), train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))
print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design network
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1],
train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                                validation_data=(test_X, test_y),
verbose=0, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
```

```
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_hours*n_features))

# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
print("inv_yhat.shpe=",inv_yhat.shape)
inv_yhat=scaler.inverse_transform(inv_yhat)
inv_yhat=inv_yhat[:,0]

#真實資料(test_y)還原成原始數值，先前用 minmaxscaler 將數值正規化(0~1)
#維度轉換:1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#1D(220,)
test_y=test_y.reshape((len(test_y),1)) #1D(220,)=>2D(220,1)
print("test_y.shape=", test_y.shape,len(test_y))#2D(220,1)

#合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y=concatenate((test_y, test_X[:,-(n_features-1):]),axis=1)
#真實資料(inv_y)還原成原始數值(test_y)
inv_y=scaler.inverse_transform(inv_y)
#只取第一欄的值，即標籤欄(真實)的值(value)
inv_y=inv_y[:,0]

#calculate 均方根誤差 RMSE(root mean squared error)
rmse=sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
#calculate 平均絕對誤差 MAE(mean absolute error)
mae=mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)



test_y,predict_y=inv_y,inv_yhat
```

```
import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:') #預測
plt.plot(test_y, 'r-') #real
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("ClosingPrice")
plt.title("Prediction(RNN)")
plt.show()

#from keras.model import load_model
model.save(filenameModel)
print("儲存 Model",filenameModel)

#將預測結果與真實資料合併成 DataFrame
#from pandas import DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred,Real")
print('顯示前 5 筆資料')
print(df_PredResult.head())
print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
```

20241031. 多對多預測

壹、 程式碼解讀

一、 數據格式與預處理：

1. LSTM 模型處理多維時間序列數據，將歷史資料轉換為可用於預測的時間序列格式（前 3 天預測第 4 天），而非單一特徵。

二、 特徵數量：

1. LSTM 預測多個特徵（如收盤價、開盤價等），而單一數據預測通常只處理一個特徵。

三、 模型架構：

1. LSTM 使用循環神經網絡來捕捉時間序列的長期依賴性，而傳統的模型（如回歸、簡單神經網絡）通常用於處理較簡單的數據。

四、 訓練過程：

1. LSTM 訓練過程需要更多計算資源，並且處理時間序列的複雜性，訓練時間也更長。

五、 預測結果：

1. LSTM 同時預測多個特徵，並根據多個時間步進行預測，而單一數據預測只有一個輸出。

六、 模型評估：

1. LSTM 模型不僅評估總體預測的誤差（如 RMSE、MAE），還針對每個特徵分別評估預測效果。

七、 視覺化：

1. LSTM 會繪製每個特徵的預測與真實值對比圖，單一數據預測只關注一個特徵的結果。

八、 模型保存與結果輸出：

1. LSTM 模型會保存訓練過的模型並輸出詳細的預測結果和真實值，並將結果儲存為 CSV 檔案。

貳、 附錄 – 程式碼

一、 StockTSMC_LSTM_M2M

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
```

```
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df = DataFrame(data)
	cols, names = list(), list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j + 1, i)) for j in
range(n_vars)]
	# forecast sequence (t, t+1, ...t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
			names += [('var%d(t)' % (j + 1)) for j in
range(n_vars)]
		else:
			names += [('var%d(t+%d)' % (j + 1, i)) for j in
range(n_vars)]
	# put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	# drop rows with NaN values
	if dropnan:
		agg.dropna(inplace=True)
	return agg

# 設定檔名(資料集、預測值結果、模型輸出)
```

```
filename = r"C:\Users\OHYA\Desktop\AI_DNN\StockTSMC.csv"
filenameOut =
r"C:\Users\OHYA\Desktop\AI_DNN\StockTSMCPredLSTM_M2
M_me.csv"
filenameModel =
r"C:\Users\OHYA\Desktop\AI_DNN\StockTSMLSTM_M2M_me.
h5"

# load dataset
dataset = read_csv(filename, header=0, index_col=0)
values = dataset.values
values = values.astype('float32')

# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

# specify the number of lag days
n_ts = 3  # 用前 3 天預測第 4 天的結果
n_features = 5  # 特徵數
# frame as supervised learning
reframed = series_to_supervised(scaled, n_ts, 1)
print("reframed.shape:", reframed.shape)

# split into train and test sets
values = reframed.values
n_train_ts = 200 * 5  # 訓練集取前 200 週的資料(5 天/週)
train = values[:n_train_ts, :]
test = values[n_train_ts:, :]

n_obs = n_ts * n_features

# many to many
train_X, train_y = train[:, :n_obs], train[:, n_obs:]
test_X, test_y = test[:, :n_obs], test[:, n_obs:]

# reshape input to be 3D [samples, timesteps, features]
```

```
train_X = train_X.reshape((train_X.shape[0], n_ts, n_features))
test_X = test_X.reshape((test_X.shape[0], n_ts, n_features))

# design network.
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1],
train_X.shape[2])))   # LSTM layer
model.add(Dense(5))
model.compile(loss='mae', optimizer='adam')

# 訓練模型
history = model.fit(train_X, train_y, epochs=5, batch_size=10,
                                validation_data=(test_X, test_y),
verbose=0, shuffle=False)

# 顯示模型訓練結果
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# 預測結果
yhat = model.predict(test_X)

# 反轉縮放比例以得到原始數據，保留所有預測值
inv_yhat = scaler.inverse_transform(yhat)   # 將預測值反標準化

# 真實資料還原成原始數據
inv_y = scaler.inverse_transform(test_y)   # 將真實值反標準化

# 計算針對全部預測值的 RMSE 和 MAE
rmse_all = sqrt(mean_squared_error(inv_y, inv_yhat))
mae_all = mean_absolute_error(inv_y, inv_yhat)
print('Test RMSE for All Variables: %.3f' % rmse_all)
print('Test MAE for All Variables: %.3f' % mae_all)

# 針對各欄位分別計算 RMSE 和 MAE
```

```
columns = ['ClosingPrice', 'OpeningPrice', 'HighestPrice',
'LowestPrice', 'Volume']
for i, column in enumerate(columns):
        rmse = sqrt(mean_squared_error(inv_y[:, i], inv_yhat[:, i]))
        mae = mean_absolute_error(inv_y[:, i], inv_yhat[:, i])
        print(f'Test RMSE for {column}: %.3f' % rmse)
        print(f'Test MAE for {column}: %.3f' % mae)

# 繪製預測與真實值
import matplotlib.pyplot as plt

# 繪製所有欄位的預測與真實值
plt.figure(figsize=(15, 10))

# Closing Price
plt.subplot(3, 2, 1)
plt.plot(inv_yhat[:, 0], 'b:', label='Pred Closing Price')  # 預測值
plt.plot(inv_y[:, 0], 'r-', label='Real Closing Price')  # 真實值
plt.legend()
plt.xlabel("Time")
plt.ylabel("Closing Price")
plt.title("Closing Price Prediction (LSTM)")

# Opening Price
plt.subplot(3, 2, 2)
plt.plot(inv_yhat[:, 1], 'b:', label='Pred Opening Price')  # 預測值
plt.plot(inv_y[:, 1], 'r-', label='Real Opening Price')  # 真實值
plt.legend()
plt.xlabel("Time")
plt.ylabel("Opening Price")
plt.title("Opening Price Prediction (LSTM)")

# Highest Price
plt.subplot(3, 2, 3)
plt.plot(inv_yhat[:, 2], 'b:', label='Pred Highest Price')  # 預測值
plt.plot(inv_y[:, 2], 'r-', label='Real Highest Price')  # 真實值
plt.legend()
```

```
plt.xlabel("Time")
plt.ylabel("Highest Price")
plt.title("Highest Price Prediction (LSTM)")

# Lowest Price
plt.subplot(3, 2, 4)
plt.plot(inv_yhat[:, 3], 'b:', label='Pred Lowest Price')  # 預測值
plt.plot(inv_y[:, 3], 'r-', label='Real Lowest Price')  # 真實值
plt.legend()
plt.xlabel("Time")
plt.ylabel("Lowest Price")
plt.title("Lowest Price Prediction (LSTM)")

# Volume
plt.subplot(3, 2, 5)
plt.plot(inv_yhat[:, 4], 'b:', label='Pred Volume')  # 預測值
plt.plot(inv_y[:, 4], 'r-', label='Real Volume')  # 真實值
plt.legend()
plt.xlabel("Time")
plt.ylabel("Volume")
plt.title("Volume Prediction (LSTM)")

# 顯示圖形
plt.tight_layout()
plt.show()

# 儲存模型
model.save(filenameModel)
print("儲存 model", filenameModel)

# 將預測結果與真實資料合併成 DataFrame 並儲存
df_PredResult = DataFrame({
    "Real_ClosingPrice": inv_y[:, 0],
    "Pred_ClosingPrice": inv_yhat[:, 0],
    "Real_OpeningPrice": inv_y[:, 1],
    "Pred_OpeningPrice": inv_yhat[:, 1],
    "Real_HighestPrice": inv_y[:, 2],
```

```
        "Pred_HighestPrice": inv_yhat[:, 2],
        "Real_LowestPrice": inv_y[:, 3],
        "Pred_LowestPrice": inv_yhat[:, 3],
        "Real_Volume": inv_y[:, 4],
        "Pred_Volume": inv_yhat[:, 4]
})

print("儲存檔案", filenameOut)
df_PredResult.to_csv(filenameOut)   # 儲存為 CSV 檔案，
index=False 以不儲存索引
# 輸出前五筆資料
print("預測與真實資料的前五筆:")
print(df_PredResult.head())
```

20241117. 資料前處理　台灣空汙資料處理

壹、 相關知識

一、 VBA 是 Visual Basic 語言的應用版本，它能夠讓使用者撰寫宏、定製功能、操作資料以及自動化日常任務

1. 主要功能
   (1) 自動化任務：可以自動化大量重複性的工作流程，比如在 Excel 中處理數據、創建報告等。
   (2) 創建自訂功能：用戶可以根據需求創建自定義函數和工具，擴展 Office 應用程式的內建功能。
   (3) 與外部應用程式互動：VBA 能夠與其他 Office 應用程式或外部系統（如資料庫、網絡服務）進行數據交互。
   (4) 事件處理：VBA 支援事件驅動編程，根據用戶操作或其他觸發條件執行特定的程式碼。
2. 優點
   (1) 易學易用：VBA 語法簡單，對於不具備程式設計經驗的用戶也比較友好。
   (2) 強大的自動化能力：可以大量減少人工操作，提高效率。
   (3) Office 集成：直接與 Microsoft Office 應用集成，操作和數據管理更加靈活。

貳、 程式碼解讀

一、 swapaxes 是 NumPy 中的一個方法，用於交換數組的兩個軸（維度）。它可以用來改變一個陣列的維度順序，通常用於多維數組的操作。這個方法傳回一個新數組，原始數組的軸順序被調換了，但不修改原數組。

參、 附錄 – 程式碼

一、 資料前處理 大里空汙資料轉換 及 資料前處理

```
#資料前處理 大里空汙資料轉換 及 資料前處理
#每個特徵-每個小時　轉換成 每天每個小時-每個特徵
#designed by 翁政雄 博士/教授

import pandas as pd
```

```
filename='D:\wekadata\pollution\myArrAllDate.csv'
#df = pd.read_csv(filename, skiprows = 1)
df = pd.read_csv(filename)
print("讀取檔案：",filename)
print("原始資料筆數:",len(df))
DayNo = int((len(df)) / 18)
print("           天數:",DayNo)


rn=18 #特徵數
#設定特徵屬性(欄位名稱)
cName=[
"AMB_TEMP","CH4","CO","NMHC","NO","NO2","Nox",
"O3","PM10","PM2.5","RAINFALL","RH","SO2","THC",
"WD_HR","WIND_DIREC","WIND_SPEED","WS_HR"]

i=0
#print(df.iat[0,0])
df_temp=df.iloc[(i*rn):(i*rn+rn),1:] #擷取一天資料 (y=特徵/測項)-(x=小時)
df_all=df_temp.swapaxes(0,1) #矩陣轉置，(y=小時)-(x=特徵/測項)
df_all.columns=cName
#增加 2 個欄位(Date,Time)，合計 18+2=20 欄位
df_all["Date"]=df.iat[0,0]
df_all["Time"]=range(0,24)

#print(df_all)
#合併時，欄位名稱必須相同；(1)每天資料轉置；(2)合併至原有資料
for i in range(1,DayNo):
#for i in range(1,2):
   df_temp=df.iloc[(i*rn):(i*rn+rn),1:] #擷取一天資料 (y=特徵/測項)-(x=小時)
   df_swap=df_temp.swapaxes(0,1) #矩陣轉置，(y=小時)-(x=特徵/測項)
   #print(df_swap)
```

```
    df_swap.columns=cName
    #增加 2 個欄位(Date,Time)，合計 18+2=20 欄位
    df_swap["Date"]=df.iat[(i*rn),0]
    df_swap["Time"]=range(0,24)
    df_all=pd.concat([df_all, df_swap])
    #df_all=df_all.append(df_swap)

print("轉換資料筆數:",len(df_all))

#print(df_all)
filename='D:\wekadata\pollution\myArrAllDateSwap.csv'
df_all.to_csv(filename, index=False)
print("另存新檔：",filename)
```

二、 每個特徵-每個小時　轉換成 每天每個小時-每個特徵

```
#資料前處理
#先執行 PollutionDFCombine, 每個特徵-每個小時　轉換成 每天每個小時-每個特徵
#刪除 na
#過濾 "雜質"(#,NA,x,...)
#另存新檔(myArrAllOKdropNA.csv), 可以跑 LSTM
#designed by 翁政雄 博士/教授

import pandas as pd
#filename='D:\wekadata\pollution\myArrAllOK.csv'
filename='AirpolutionDali\myArrAllOKDate.csv'
df = pd.read_csv(filename)
print('原始筆數:',len(df))

#刪除 有 na 的樣本，刪除該筆資料
df.dropna(axis=0,inplace=True)

filtA1 = (df['A'] != '#                                    ')
filtA2 = (df['A'] != 'NA                                    ')
filtA3 = (df['A'] != 'x                                    ')
filtA4 = (df['A'] != 'A                                    ')
filtA5 = (df['A'] != '*                                    ')
filtA = filtA1 & filtA2 & filtA3 & filtA4 & filtA5
```

```
filtB1 = (df['B'] != '#                                    ')
filtB2 = (df['B'] != 'NA                                    ')
filtB3 = (df['B'] != 'x                                    ')
filtB4 = (df['B'] != 'A                                    ')
filtB5 = (df['B'] != '*                                    ')
filtB = filtB1 & filtB2 & filtB3 & filtB4 & filtB5

filtC1 = (df['C'] != '#                                    ')
filtC2 = (df['C'] != 'NA                                    ')
filtC3 = (df['C'] != 'x                                    ')
filtC4 = (df['C'] != 'A                                    ')
filtC5 = (df['C'] != '*                                    ')
filtC = filtC1 & filtC2 & filtC3 & filtC4 & filtC5

filtD1 = (df['D'] != '#                                    ')
filtD2 = (df['D'] != 'NA                                    ')
filtD3 = (df['D'] != 'x                                    ')
filtD4 = (df['D'] != 'A                                    ')
filtD5 = (df['D'] != '*                                    ')
filtD = filtD1 & filtD2 & filtD3 & filtD4 & filtD5

filtE1 = (df['E'] != '#                                    ')
filtE2 = (df['E'] != 'NA                                    ')
filtE3 = (df['E'] != 'x                                    ')
filtE4 = (df['E'] != 'A                                    ')
filtE5 = (df['E'] != '*                                    ')
filtE = filtE1 & filtE2 & filtE3 & filtE4 & filtE5

filtF1 = (df['F'] != '#                                    ')
filtF2 = (df['F'] != 'NA                                    ')
filtF3 = (df['F'] != 'x                                    ')
filtF4 = (df['F'] != 'A                                    ')
filtF5 = (df['F'] != '*                                    ')
filtF = filtF1 & filtF2 & filtF3 & filtF4 & filtF5

filtG1 = (df['G'] != '#                                    ')
```

```
filtG2 = (df['G'] != 'NA                                    ')
filtG3 = (df['G'] != 'x                                     ')
filtG4 = (df['G'] != 'A                                     ')
filtG5 = (df['G'] != '*                                     ')
filtG = filtG1 & filtG2 & filtG3 & filtG4 & filtG5

filtH1 = (df['H'] != '#                                     ')
filtH2 = (df['H'] != 'NA                                    ')
filtH3 = (df['H'] != 'x                                     ')
filtH4 = (df['H'] != 'A                                     ')
filtH5 = (df['H'] != '*                                     ')
filtH = filtH1 & filtH2 & filtH3 & filtH4 & filtH5

filtI1 = (df['I'] != '#                                   ')
filtI2 = (df['I'] != 'NA                                    ')
filtI3 = (df['I'] != 'x                                   ')
filtI4 = (df['I'] != 'A                                   ')
filtI5 = (df['I'] != '*                                   ')
filtI = filtI1 & filtI2 & filtI3 & filtI4 & filtI5

filtJ1 = (df['J'] != '#                                   ')
filtJ2 = (df['J'] != 'NA                                    ')
filtJ3 = (df['J'] != 'x                                   ')
filtJ4 = (df['J'] != 'A                                   ')
filtJ5 = (df['J'] != '*                                   ')
filtJ = filtJ1 & filtJ2 & filtJ3 & filtJ4 & filtJ5

filtK1 = (df['K'] != '#                                     ')
filtK2 = (df['K'] != 'NA                                    ')
filtK3 = (df['K'] != 'x                                     ')
filtK4 = (df['K'] != 'A                                     ')
filtK5 = (df['K'] != '*                                     ')
filtK = filtK1 & filtK2 & filtK3 & filtK4 & filtK5

filtL1 = (df['L'] != '#                                   ')
filtL2 = (df['L'] != 'NA                                    ')
filtL3 = (df['L'] != 'x                                   ')
```

```
filtL4 = (df['L'] != 'A                                    ')
filtL5 = (df['L'] != '*                                   ')
filtL = filtL1 & filtL2 & filtL3 & filtL4 & filtL5

filtM1 = (df['M'] != '#                                    ')
filtM2 = (df['M'] != 'NA                                    ')
filtM3 = (df['M'] != 'x                                    ')
filtM4 = (df['M'] != 'A                                    ')
filtM5 = (df['M'] != '*                                    ')
filtM = filtM1 & filtM2 & filtM3 & filtM4 & filtM5

filtN1 = (df['N'] != '#                                    ')
filtN2 = (df['N'] != 'NA                                    ')
filtN3 = (df['N'] != 'x                                    ')
filtN4 = (df['N'] != 'A                                    ')
filtN5 = (df['N'] != '*                                    ')
filtN = filtN1 & filtN2 & filtN3 & filtN4 & filtN5

filtO1 = (df['O'] != '#                                    ')
filtO2 = (df['O'] != 'OA                                    ')
filtO3 = (df['O'] != 'x                                    ')
filtO4 = (df['O'] != 'A                                    ')
filtO5 = (df['O'] != '*                                    ')
filtO = filtO1 & filtO2 & filtO3 & filtO4 & filtO5

filtP1 = (df['P'] != '#                                   ')
filtP2 = (df['P'] != 'PA                                   ')
filtP3 = (df['P'] != 'x                                   ')
filtP4 = (df['P'] != 'A                                   ')
filtP5 = (df['P'] != '*                                   ')
filtP = filtP1 & filtP2 & filtP3 & filtP4 & filtP5

filtQ1 = (df['Q'] != '#                                    ')
filtQ2 = (df['Q'] != 'QA                                    ')
filtQ3 = (df['Q'] != 'x                                    ')
filtQ4 = (df['Q'] != 'A                                    ')
filtQ5 = (df['Q'] != '*                                    ')
```

```
filtQ = filtQ1 & filtQ2 & filtQ3 & filtQ4 & filtQ5

filtR1 = (df['R'] != '#                                                    ')
filtR2 = (df['R'] != 'RA                                                    ')
filtR3 = (df['R'] != 'x                                                    ')
filtR4 = (df['R'] != 'A                                                    ')
filtR5 = (df['R'] != '*                                                    ')
filtR = filtR1 & filtR2 & filtR3 & filtR4 & filtR5

#合併篩選條件
filt_str1=filtA & filtB & filtC & filtD & filtE & filtF & filtG &
filtH & filtI & filtJ
filt_str2=filtK & filtL & filtM & filtN & filtO & filtP & filtQ &
filtR

filt_str= filt_str1 & filt_str2
df_filter=df.loc[filt_str]

print('轉換筆數:',len(df_filter))
#print(df_filter)

#filename='D:\wekadata\pollution\myArrAllOKdropNA.csv'
filename='AirpolutionDali\myArrAllOKDatedropNA.csv'
df_filter.to_csv(filename, index=False)



df_temp = df_filter.values[:,:18] #僅篩選 前 18 欄位
from pandas import DataFrame

#設定特徵屬性(欄位名稱)
cName=[
"AMB_TEMP","CH4","CO","NMHC","NO","NO2","Nox",
"O3","PM10","PM2.5","RAINFALL","RH","SO2","THC",
"WD_HR","WIND_DIREC","WIND_SPEED","WS_HR"]
df_OK=DataFrame(df_temp)
filename='AirpolutionDali\DaliPM25OK.csv'
```

```
df_OK.columns=cName
df_OK.to_csv(filename, index=False)
```

三、 predictionGRU

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from keras.layers import GRU, Activation, Dense

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df = DataFrame(data)
	cols, names = list(), list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
			names += [('var%d(t)' % (j+1)) for j in
range(n_vars)]
		else:
			names += [('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
```

```
        # put it all together
        agg = concat(cols, axis=1)
        agg.columns = names
        # drop rows with NaN values
        if dropnan:
                agg.dropna(inplace=True)
        return agg

# load dataset
filename='AirpolutionDali\myArrAllOKDatedropNA.csv'
#filename='AirpolutionDali\DaliPM25OK.csv'    #有欄位名稱
#dataset = read_csv(filename, header=0, index_col=0)
dataset = read_csv(filename, header=0)
values = dataset.values[:,:18] #僅篩選 前 18 欄位
'''
# integer encode direction
encoder = LabelEncoder()
values[:,4] = encoder.fit_transform(values[:,4])
# ensure all data is float
'''



values = values.astype('float32')    #(43800, 8)
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# specify the number of lag hours
n_hours = 3
n_features = 18
# frame as supervised learning
reframed = series_to_supervised(scaled, n_hours, 1)
print(reframed.shape) #(43797, 32), 用前 3 小時預測第 4 小時的結果，必須 4 小時的資料(32 特徵欄位)，合併成 1 列(row)

#print(type(reframed))#DataFrame
#print(type(values)) #numpy.ndarray
```

```
# split into train and test sets
values = reframed.values
#n_train_hours = 365 * 24
n_train_hours = 200 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs

n_obs = n_hours * n_features
#n_hours = 3, 用前 3 小時預測第 4 小時的結果，必須 4 小時的資料，合併成 1 列(row)
#series_to_supervised()產生 1 筆資料有 4*8=32 欄位，
#n_obs=3*8=24，取 0~23 欄位當特徵欄位(即前 3 小時的資料)；標籤欄為第 4 小時的第 1 個欄位(即-8=-n_features)
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print(train_X.shape, len(train_X), train_y.shape)
#(8760, 24) 8760 (8760,)
# reshape input to be 3D [samples, timesteps, features]
#(8760, 24) => (8760, 3, 8);   (35037, 24) =>(35037, 3, 8)
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
#(8760, 3, 8) (8760,) (35037, 3, 8) (35037,)

# design network
model = Sequential()
#model.add(LSTM(50, input_shape=(train_X.shape[1],
train_X.shape[2])))
#換成 GRU
model.add(GRU(50, input_shape=(train_X.shape[1],
train_X.shape[2])))
#stack LTM
#model.add(LSTM(50, input_shape=(train_X.shape[1],
train_X.shape[2]),unroll=False, return_sequences=True))
```

```
#model.add(LSTM(50, input_shape=(train_X.shape[1],
train_X.shape[2]),unroll=False))

#model.add(Dense(256, activation='relu'))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=5, batch_size=72,
validation_data=(test_X, test_y), verbose=1, shuffle=False)
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# make a prediction
yhat = model.predict(test_X)
#(35037, 3, 8) => (35037, 24)
test_X = test_X.reshape((test_X.shape[0], n_hours*n_features))
# invert scaling for forecast
#預測值(yhat) (1 個欄位) 合併 test_X[:, -7:] (7 個欄位)，即 第
4 小時的特徵屬性(-7:)，共 8 個欄位；另外(-8 是第 4 小時的標
籤)
#inv_yhat = concatenate((yhat, test_X[:, -7:]), axis=1)
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
print("inv_yhat.shape=",inv_yhat.shape)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]



# invert scaling for actual
print("test_y.shape=",test_y.shape,len(test_y))
test_y = test_y.reshape((len(test_y), 1)) # 1D (35037,) => 2D
(35037, 1)
print("test_y.shape=",test_y.shape,len(test_y))

#inv_y = concatenate((test_y, test_X[:, -7:]), axis=1)
```

```
inv_y = concatenate((test_y, test_X[:, -(n_features-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calculate 均方根誤差 RMSE (root mean squared error)
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
# calculate 平均絕對誤差（MAE）(mean absolut error)
mae = mean_absolute_error(inv_y, inv_yhat)
print('Test    MAE: %.3f' % mae)

'''
#from keras.models import load_model
model.save(filenameModel)
print("儲存 model",filenameModel)



#將預測結果與真實資料 合併成 DataFrame
#from pandas import    DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred, Real")
print(df_PredResult.head())
print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
'''
```

四、 predictionLSTM

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.metrics import mean_squared_error
```

```
from sklearn.metrics import mean_absolute_error



# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df = DataFrame(data)
	cols, names = list(), list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
			names += [('var%d(t)' % (j+1)) for j in
range(n_vars)]
		else:
			names += [('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
	# put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	# drop rows with NaN values
	if dropnan:
		agg.dropna(inplace=True)
	return agg

# load dataset
filename='AirpolutionDali\myArrAllOKDatedropNA.csv'
filename='AirpolutionDali\DaliPM25OK.csv'   #有欄位名稱

#dataset = read_csv(filename, header=0, index_col=0)
dataset = read_csv(filename, header=0)
values = dataset.values[:,:18]
```

```
'''
# integer encode direction
encoder = LabelEncoder()
values[:,4] = encoder.fit_transform(values[:,4])
# ensure all data is float
'''



values = values.astype('float32')    #(43800, 8)
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# specify the number of lag hours
n_hours = 3
n_features = 18
# frame as supervised learning
reframed = series_to_supervised(scaled, n_hours, 1)
print(reframed.shape) #(43797, 32), 用前 3 小時預測第 4 小時的結果，必須 4 小時的資料(32 特徵欄位)，合併成 1 列(row)

#print(type(reframed))#DataFrame
#print(type(values)) #numpy.ndarray



# split into train and test sets
values = reframed.values
#n_train_hours = 365 * 24
n_train_hours = 200 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs

n_obs = n_hours * n_features
#n_hours = 3, 用前 3 小時預測第 4 小時的結果，必須 4 小時的資料，合併成 1 列(row)
#series_to_supervised()產生 1 筆資料有 4*8=32 欄位，
```

```
#n_obs=3*8=24，取 0~23 欄位當特徵欄位(即前 3 小時的資
料)；標籤欄為第 4 小時的第 1 個欄位(即-8=-n_features)
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print(train_X.shape, len(train_X), train_y.shape)
#(8760, 24) 8760 (8760,)
# reshape input to be 3D [samples, timesteps, features]
#(8760, 24) => (8760, 3, 8);    (35037, 24) =>(35037, 3, 8)
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
#(8760, 3, 8) (8760,) (35037, 3, 8) (35037,)

# design network
model = Sequential()
model.add(LSTM(64, input_shape=(train_X.shape[1],
train_X.shape[2])))

#stack LTM
#model.add(LSTM(128, input_shape=(train_X.shape[1],
train_X.shape[2]),unroll=False, return_sequences=True))
#model.add(LSTM(50, input_shape=(train_X.shape[1],
train_X.shape[2]),unroll=False))

#model.add(Dense(32, activation='relu'))
#model.add(Dense(4, activation='relu'))
#model.add(Dense(64, activation='relu'))

model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=5, batch_size=72,
validation_data=(test_X, test_y), verbose=1, shuffle=False)
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
```

```
pyplot.show()

# make a prediction
yhat = model.predict(test_X)
#(35037, 3, 8) => (35037, 24)
test_X = test_X.reshape((test_X.shape[0], n_hours*n_features))
# invert scaling for forecast
#預測值(yhat) (1 個欄位) 合併 test_X[:, -7:] (7 個欄位)，即 第 4 小時的特徵屬性(-7:)，共 8 個欄位；另外(-8 是第 4 小時的標籤)
#inv_yhat = concatenate((yhat, test_X[:, -7:]), axis=1)
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
print("inv_yhat.shape=",inv_yhat.shape)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]



# invert scaling for actual
print("test_y.shape=",test_y.shape,len(test_y))
test_y = test_y.reshape((len(test_y), 1)) # 1D (35037,) => 2D (35037, 1)
print("test_y.shape=",test_y.shape,len(test_y))

#inv_y = concatenate((test_y, test_X[:, -7:]), axis=1)
inv_y = concatenate((test_y, test_X[:, -(n_features-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calculate 均方根誤差 RMSE (root mean squared error)
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
# calculate 平均絕對誤差（MAE）(mean absolut error)
mae = mean_absolute_error(inv_y, inv_yhat)
print('Test  MAE: %.3f' % mae)

import matplotlib.pyplot as plt
#註解
predict_y=inv_yhat
```

```
test_y=inv_y

plt.plot(predict_y, 'b:') #預測
plt.plot(test_y, 'r-')       #real
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("PM2.5")
plt.title("PM2.5 (LSTM)")
plt.show()
'''
import pandas as pd
import numpy as np
from plotly.graph_objs import Scatter, Layout
from plotly.offline import plot
# 建立 DataFrame，加入 predict_y、test_y，準備以 plotly 繪圖
dd2=pd.DataFrame({"predict":list(predict_y),"real":list(test_y)})
#轉換為 numpy 陣列，並轉為 float
dd2["predict"] = np.array(dd2["predict"]).astype('float64')
dd2["real"] = np.array(dd2["real"]).astype('float64')

data = [
      Scatter(y=dd2["predict"],name='predict'),
      Scatter(y=dd2["real"],name='real')
]

plot({"data": data, "layout": Layout(title='Preditcion
(LSTM)')},auto_open=True)
'''

'''
#from keras.models import load_model
model.save(filenameModel)
print("儲存 model",filenameModel)



#將預測結果與真實資料 合併成 DataFrame
```

```
#from pandas import    DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred, Real")
print(df_PredResult.head())
print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
'''
```

20241118. CNN 汽車辨識與期中報告

壹、 相關知識

一、 將神經元數量設置為 2 的冪次（如 32、64、128 等）是因為計算機內部對於處理這類數字更高效，尤其是當硬體架構優化了對這種數字的處理，但其實不一定要使用 2 的冪次。

二、 當 MAE 提高而 RMSE 降低時：較小誤差減少、離群值或極端誤差的變化、誤差分佈變化（若誤差分佈變得更均勻，極端誤差減少，但整體誤差略微上升）

三、 隱藏層神經元數先大後小／先小後大比較

	先大後小	先小後大
優勢	• 特徵提取和降維 減少過擬合	• 逐步增強特徵表示 數據處理效率
敘述	• 有助於在初始階段學習更多的特徵 • 隨著網路層的深入，逐步濃縮、過濾特徵 • 提取重要信息 限制模型複雜度	• 在需要逐步構建更高級特徵的場合可能有效
小結	防止過擬合	有些特定應用中有助於集中處理特定類型的數據
結論	• 「先大後小」的結構更為常見 ▪ 當輸入數據具有高維特徵 ▪ 先擷取廣泛低階特徵，再逐漸濃縮為高階特徵 ▪ 模擬大腦 在計算和記憶體方面常更加節約	

四、 線性整流（Rectified Linear Unit, ReLU）：>0 維持原值，≤0 改為 0

五、 泛化性能（Generalization Performance）是指一個機器學習模型在新數據（未見過的數據）上的表現能力。它衡量模型從訓練數據中學到的模式在實際應用中的適用性，即模型在測試集或真實數據上的表現如何。

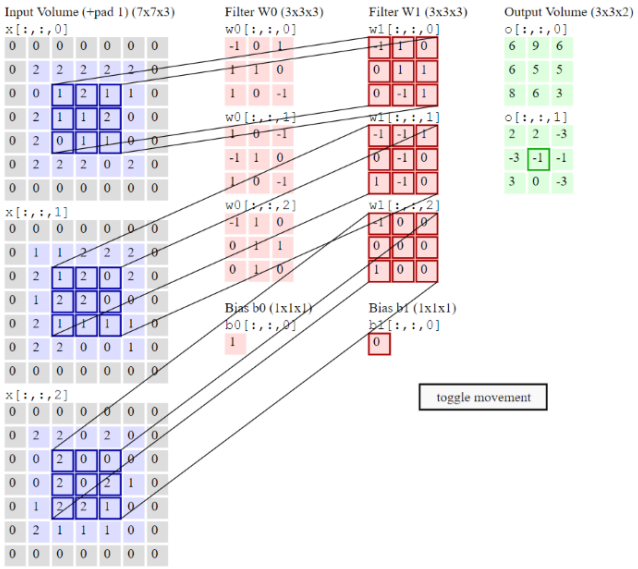
衡量泛化性能的指標包括：

六、 均方誤差（Mean Squared Error, MSE）、均方根誤差（Root Mean Squared Error, RMSE）

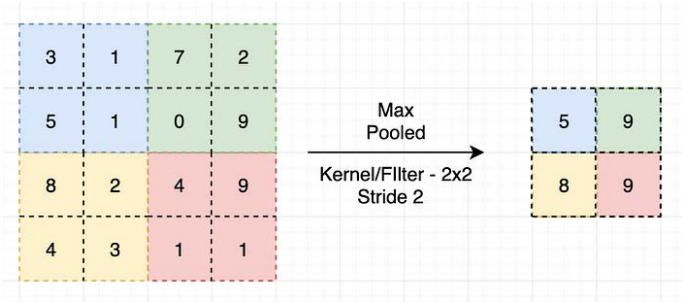
七、 平均絕對誤差（Mean Absolute Error, MAE）

八、 R^2值（決定係數）

九、 卷積層：找特徵，越接近 1 越好（用目標去與原始相乘求出中心值）



十、 池化層：放大特徵，壓縮圖片並保留重點（在指定格數中選擇最佳保留）



貳、 程式碼解讀

一、 ax.imshow(images[i], cmap='binary') 是用來在 Matplotlib 中顯示圖像的程式碼，具體來說，它是將某一張圖像顯示在一個子圖（ax）上，並使用指定的顏色映射（cmap）。

1. ax:是 Matplotlib 中的 Axes 物件，代表子圖區域。通常，你會創建一個或多個 Axes 來顯示多個圖形，或者在一個視窗中顯示不同的圖像。
2. imshow():是 Matplotlib 的函數，用來顯示圖像數據。imshow()會將二維陣列或圖像（如灰階圖像、RGB 圖像等）作為輸入，並將其顯示為圖像。
3. images[i]:這裡的 images 是一個包含多張圖像的陣列（或列表），而 i 是索引，表示你要顯示的第 i 張圖像。

4. cmap='binary':顏色映射（colormap），用來將圖像中的數值映射到顏色。binary 是一種二值的顏色映射，通常用於灰階圖像，它會將數值映射到黑白色調，較小的數值顯示為黑色，較大的數值顯示為白色。

二、 np.zeros():這是 NumPy 的一個函數，用來創建一個指定形狀的數組，並將所有元素初始化為 0。這裡使用 np.zeros 來創建一個全為 0 的數組。

三、 (imageNum, 28, 28): 這是生成數組的形狀。這裡的數組會有 imageNum 個 28x28 的矩陣。這表示每一個 train_feature 中的元素都是一個 28x28 的二維陣列，通常用來表示圖片（例如手寫數字識別中每張圖片的像素）。imageNum 則表示圖片的數量。

參、 附錄 – 程式碼

一、 CNNCarGrey

```
##Readme
#CNN 汽車辨識 Python designed by 翁政雄 博士/教授
#PythonCode_CNNCar CNNCarGrey
#


import numpy as np
from keras.utils import to_categorical
#from keras.utils import np_utils
np.random.seed(10)
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import
Conv2D,MaxPooling2D,Dropout,Flatten,Dense

#自訂函數
#大雄自訂函數，顯示預測結果
def show_labels_prediction(labels, predictions, start_id, num):
    for i in range(0,num):
        #有 AI 預測結果資料，才在標題顯示預測結果
        if(len(predictions)>0):
            title='ai = '+str(predictions[i])
            #預測正確顯示(o),錯誤(x)
            title += ': label = '+str(labels[i])
```

```
                title += (': O' if predictions[i]==labels[i] else ': X')
                #輸出 1:ai=a:label=1:O
                print(i+1, ":", title)
            #沒有 AI 預測結果資料，只在標題顯示真實資料
            else :
                title = 'label = ' + str(labels[i])
            start_id+=1

#大雄自訂函數，計算 TPR,TNR...等指標值
def show_predictions(labels,predictions,start_id,num):
    TP=0
    TN=0
    FP=0
    FN=0
    for i in range(0,num):
        #有 AI 預測結果資料，統計 TP,TN,FP,FN 僅 2 元分類有意義
        if(len(predictions)>0):
            if(predictions[i]==labels[i]):
                if(predictions[i]==1):
                    TP=TP+1
                else:
                    TN=TN+1
            else:
                if(predictions[i]==1):
                    FP=FP+1
                else:
                    FN=FN+1
        else:
            print("\n No Data")
        start_id+=1
    print("\n 各種準確率指標")
    print("TP=",TP)
    print("FN=",FN)
    print("TN=",TN)
    print("FP=",FP)
    print("TPR=",TP/(TP+FN))
```

```
    print("TNR=",TN/(TN+FP))
    print("Accuracy=",(TP+TN)/(TP+FN+TN+FP))
    print("Precision=",TP/(TP+FP))

#顯示圖片、實際標籤、預測標籤、預測結果
def show_images_labels_predictions(images,labels,

predictions,start_id,num=5):
    ax=plt
    plt.figure(figsize=(8,4))

    for i in range(0,num):
        #有 AI 預測結果資料，才在標題顯示預測結果
        if(len(predictions)>0):
            title='ai = '+str(predictions[i])
            #預測正確顯示(o),錯誤(x)
            title += ': label = '+str(labels[i])
            title += (': O' if predictions[i]==labels[i] else ': X')
            #輸出  1:ai=a:label=1:O
            print(i+1, ":", title)
            #顯示黑白圖片
            ax.imshow(images[i], cmap='binary')
            plt.show()
        #沒有 AI 預測結果資料，只在標題顯示真實資料
        else :
            title = 'label = ' + str(labels[i])
        start_id+=1

##########################################################
##############################自訂函數結束



##########讀取訓練資料
#讀取檔案(jpg);圖檔前處理完成
#讀取圖檔標籤(label)
import os
import numpy as np
```

```
from PIL import Image

#設定檔案路徑
filepath='TrainDataCar'

#計算圖檔數量
fn=0
for f in os.listdir(filepath):
    fn=fn+1
print("訓練資料檔案數: ",fn)

#宣告圖片陣列空間
imageNum=fn
train_feature = np.zeros((imageNum, 28, 28), dtype=int) #np.int

#擷取檔案路徑(目錄)的檔案清單
for root,dirs,files in os.walk(filepath):
#print(root)#當前目前路徑
#print(dirs)#當前路徑下所有子目錄
#print(files)#當前路徑下所有非目錄子檔案
    files
#擷取訓練資料 圖檔 及 標籤
i=0
Lablestr=''
for f in files:
    filename=filepath+'/'+f
    im = Image.open(filename)
#from image to array
    train_feature[i,:,:]=np.array(im)
#擷取標籤(label)
    label=os.path.splitext(f)[0][0]
    #print(i+1,':',f,' ',os.path.splitext(f)[0],' ',label)

#紀錄 圖檔的標籤，存取成字串
    if(Lablestr==''):
        Lablestr=str(label)
    else:
```

```
            Lablestr=Lablestr+','+str(label)
        i=i+1
###結束 for f in files:
#標籤字串 轉 陣列
Lablestr=Lablestr.split(',')
train_label = np.array(Lablestr,dtype=int) #np.int
print(train_label)
print("訓練資料 Label: ",train_label)

##########讀取訓練資料
#讀取檔案(jpg);圖檔前處理完成
#讀取圖檔標籤(label)
import os
import numpy as np
from PIL import Image

#設定檔案路徑
filepath='TestDataCar'

#計算圖檔數量
fn=0
for f in os.listdir(filepath):
        fn=fn+1
print("訓練資料檔案數: ",fn)

#宣告圖片陣列空間
imageNum=fn
test_feature = np.zeros((imageNum, 28, 28), dtype=int)

#擷取檔案路徑(目錄)的檔案清單
for root,dirs,files in os.walk(filepath):
#print(root)#當前目前路徑
#print(dirs)#當前路徑下所有子目錄
#print(files)#當前路徑下所有非目錄子檔案
        files
#擷取訓練資料 圖檔 及 標籤
i=0
```

```
Lablestr=''
for f in files:
    filename=filepath+'/'+f
    im = Image.open(filename)
#from image to array
    test_feature[i,:,:]=np.array(im)
#擷取標籤(label)
    label=os.path.splitext(f)[0][0]
    #print(i+1,':',f,' ',os.path.splitext(f)[0],' ',label)

#紀錄 圖檔的標籤，存取成字串
    if(Lablestr==''):
        Lablestr=str(label)
    else:
        Lablestr=Lablestr+','+str(label)
    i=i+1
###結束 for f in files:
#標籤字串 轉 陣列
Lablestr=Lablestr.split(',')
test_label = np.array(Lablestr,dtype=int)
print(test_label)
print("訓練資料 Label: ",test_label)

########################大雄修改 END
### Feature 特徵轉換成 N*28*28*1 的 4 維矩陣
train_feature_vector = train_feature.reshape(len(train_feature), 28,
28, 1).astype('float32')
test_feature_vector = test_feature.reshape(len(test_feature), 28, 28,
1).astype('float32')

train_feature_normalize = train_feature_vector / 255
test_feature_normalize = test_feature_vector / 255

### Label 轉換為 One-Hot Encoding 編碼
train_label_onehot = to_categorical(train_label)
#np_utils.to_categorical
test_label_onehot = to_categorical(test_label)
```

```
#np_utils.to_categorical

#建立模型
model = Sequential()

# 建立卷積層 1
model.add(Conv2D(filters=10,
                 kernel_size=(3, 3),
                 padding='same',
                 input_shape=(28, 28, 1),
                 activation='relu'))

# 建立池化層 1
model.add(MaxPooling2D(pool_size=(2, 2)))    # (28, 28, 1) -> (14,
14, 14)

# 建立卷積層 2
model.add(Conv2D(filters=20,
                 kernel_size=(3, 3),
                 padding='same',
                 activation='relu'))

# 建立池化層 2
model.add(MaxPooling2D(pool_size=(2, 2)))    # (14, 14, 14) -> (7,
7, 7)

# 加入 Dropout 防止過度擬合
model.add(Dropout(0.2))

# 建立平坦層：2D->1D, 7*7*20 變成 784 個神經元
model.add(Flatten())

# 建立全連接層
model.add(Dense(units=256, activation='relu'))

# 建立輸出層，units 設置為分類的類別數
model.add(Dense(units=3, activation='softmax'))
```

```
# 編譯模型
model.compile(loss='categorical_crossentropy',
                    optimizer='adam', metrics=['accuracy'])

#以(train_features_normalize,train_label_onehot)資料訓練，
#訓練資料保留 20%做驗證，訓練 10 次、每批次讀取 200 筆資
練，顯示簡易訓練過程
train_history = model.fit(x=train_feature_normalize,

y=train_label_onehot,validation_split=0.2,

epochs=10,batch_size=30,verbose=0)
#評估準確率
scores=model.evaluate(test_feature_normalize,test_label_onehot)
print('\n 準確率=',scores[1])

#預測
#prediction=moddel.predict_classes(test_feature_normalize)#新版
不支援
prediction=model.predict(test_feature_normalize)
prediction=np.argmax(prediction,axis=-1)#依機率值高低，決定
label

#儲存模型
model.save("CNNCarGrey.h5")
print("儲存模型 成功")

############################顯示結果

#顯示預測值、真實值
show_labels_prediction(test_label,prediction,0,imageNum)

#顯示 measures
show_predictions(test_label,prediction,0,imageNum)

#顯示圖象、預測值、真實值
```

```
show_images_labels_predictions(test_feature,test_label,prediction,
0,imageNum)
```

20241125. 辨別芒果品質

壹、 相關知識

一、 圖片去背步驟：

1. 套用影像分割技術：使用像 OpenCV 的 GrabCut、K-means、或者深度學習分割模型（如 UNet）進行背景移除。
2. 處理結果：將分割結果儲存為具有透明背景的影像（如 PNG 格式）或直接使用遮罩處理訓練影像。
3. 更新訓練流程：將去背影像當作訓練資料進行模型訓練。

貳、 程式碼解讀

一、 np.argmax 用來從模型預測的結果中選擇出最高概率的類別。

二、 圖片去背：

1. rembg 是一個基於深度學習的開源工具，可以快速去除圖片背景。
2. 如果是簡單的圖片（如有清晰背景和前景的情況），可以使用傳統的圖像處理技術進行背景去除。這些方法包括：
   (1) 閾值分割（Thresholding）：將背景與前景根據顏色區分開來。
   (2) 邊緣檢測（Edge Detection）：通過邊緣檢測來識別物體邊界，然後移除背景。
   (3) 色度鍵（Chroma keying）：如果背景是單一顏色（例如綠色屏幕背景），則可以通過色度鍵將該顏色範圍移除。

參、 附錄 – 程式碼

一、 Python_Mango_resize_Grey

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 25 10:56:08 2024

@author: ASUS
"""


#ReadMe Python 程式 designed by 翁政雄 博士/教授
#原始檔案為 jpg
#縮小圖片、彩色/灰階、另存新檔(標籤_序號.jpg)/(原檔名.jpg)
```

```
from os import listdir
from os.path import isfile, isdir, join
import os
#import numpy as np

#指定要列出所有檔案的目錄
mypath = "train"
mypathNew = "trainRGB"

#取得所有檔案與子目錄名稱
files = listdir(mypath)
#匯入 library
from PIL import Image

#以迴圈處理
i = 0
Labelstr = ''
for f in files:
    #產生檔案的絕對路徑
    fullpath = join(mypath, f)
    fullpathNew = join(mypathNew, f)
    #判斷 fullpath 是檔案還是目錄
    if isfile(fullpath) & (os.path.spiltext(f)[1]=='.jpg'):
        #print("檔案:", f)
        #開啟
        im = Image.open(fullpath)
        #顯示原圖片大小(像素)
        print("檔案:", fullpath, "類型:", im.format, im.size,
im.mode)
        #變更大小(像素)
        nim= = im.resize((256, 256), Image.BILINEAR)

        #圖檔格式(GRB)
        image_RGB = nim.convert('RGB')

        #檔名第 2 個字元為 label, E1，1 為標籤，若不是標籤
```

```
則檔名無特殊意義
            label = os.path.splitext(f)[0][1]

            #新檔名(標籤+序號)，副檔名(jpg)
            filenameNew=label+'_'+str(i+1)+".jpg"
            #filenameNew = label+'_'+str(i+1)+os.path.splitext(f)[-1]

            #完整檔名
            #pathfilenameNew = join (mypathNew, filenameNew)
            #使用原檔名
            pathfilenameNew = join (mypathNew, f)
            print(i+1, ':', f, '', os.path.splitext(f)[0], ' ', filenameNew, '
', pathfilenameNew)

            #另存新檔
            image_RGB.save(pathfilenameNew)

            #紀錄圖檔的標籤還有字串
            if (Labelstr==''):
                  Labelstr=str(label)
            else:
                  Labelstr=Labelstr + ',' + str(label)

            i = i + 1

      elif isdir(fullpath):
            print("目標:", f)



#end for for f in files:
#輸出  標籤(label)
#print(Lable)
#Labelstr=Labelstr.split(',')
#train_label = np.array(Labelstr, dtype=int)
#print(train_label)
```

二、 CNNCMango

```
# -*- coding: utf-8 -*-
```

```
"""
Created on Mon Nov 25 09:31:45 2024

@author: ASUS
"""

##ReadMe
#CNN Mango Python designed by 翁政雄 博士/教授
#PythonCode_CNNCMango
#彩色版(RGB)

import csv
import numpy as np
from keras.utils import np_utils
np.random.seed(10)
#from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout,
Flatten, Dense

#自訂函數
#大雄自訂函數，顯示 預測結果
def show_labels_predictions(labels, predictions, start_id, num):
    for i in range (0, num):
        #有 AI 預測結果資料，才在標題顯示預測結果
        if ( len(predictions) > 0):
            title = 'ai = ' + str(predictions[i])
            #預測正確顯示(o)，錯誤顯示(x)
            title += ': label = ' + str(labels[i])
            title +- (' : 0' if predictions[i]==labels[i] else ' : X')
            #輸出  1 : ai = 1 : label = 1 : 0
            print(i+1, ":", title)
        #沒有 AI 預測結果資料，只在標題顯示真實數據
        else :
            title = 'label = ' + str(labels[i])
        start_id += 1
```

```
#大雄自訂函數，計算 TPR, TNR...等指標值
def show_predictions(labels, predictions, start_id, num):
    AA = 0
    BB = 0
    CC = 0
    for i in range(0, num):
        #有 AI 預測結果資料，統計 TP, TN, FP, FN 僅 2 元分類有意義
        if( len(predictions) > 0):
            if(predictions[i]==labels[i]):
                if(labels[i]=='0'):
                    AA = AA+1
                elif (labels[i]=='1'):
                    BB = BB+1
                else :
                    CC = CC+1
        else:
            print("\n No Data")
        start_id += 1
    print("\n 各種準確率指標")
    print("AA = ", AA)
    print("BB = ", BB)
    print("CC = ", CC)
    print("Total = ", start_id)
    print("Accuracy = ", (AA+BB+CC) / (start_id))

#顯示圖片、實際標籤、預測標籤、預測結果
def show_images_labels_predictions(images, labels, predictions,
start_id, num=5):
    ax = plt
    plt.figure(figsize=(8, 4))

    for i in range(0, num):
        #有 AI 預測結果資料，才在標題顯示預測結果
        if ( len(predictions) > 0):
            title = 'ai = ' + str(predictions[i])
```

```
                #預測正確顯示(o)，錯誤顯示(x)
                title += ': label = ' + str(labels[i])
                title +- (' : 0' if predictions[i]==labels[i] else ' : X')
                #輸出  1 : ai = 1 : label = 1 : 0
                print(i+1, ":", title)
                #顯示黑白照片
                ax.imshow(images[i], cmap='binary')
                plt.show()
            #沒有 AI 預測結果資料，只在標題顯示真實數據
            else :
                title = 'label = ' + str(labels[i])
            start_id += 1

############################################################
#########自訂函數結數

###讀取訓練資料
#讀取圖檔(jpg):圖檔前處理完成
#讀取圖檔標籤(label)

import os
import numpy as np
from PIL import Image

#設定檔案路徑(目錄)
filepath='MangoTrain'

#計算圖檔數量
fn = 0
for f in os.listdir(filepath):
    fn = fn + 1
print("訓練資料檔案數: ", fn)

#宣告圖片陣列空間
imageNum = fn
#train_feature = np.zeros((imageNum, 28, 28), dtype=np.int)
#train_feature = np.zeros((imageNum, 28, 28, 3), dtype=np.int)
```

```
train_feature = np.zeros((imageNum, 224, 224, 3), dtype=np.int)

#擷取檔案路徑(目錄)的檔案清單
for root, dirs, files, in os.walk(filepath):
    #print(root)#當前目錄路徑
    #print(dirs)#當前路徑下所有子目錄
    #print(files)#當前路徑下所有非目錄子檔案
    files

#擷取訓練資料 圖檔
i = 0
Labelstr = ''
for f in files:
    filename = filepath + '/' + f
    im = Image.open(filename)
#From image to array
    train_feature[i,:,:,:] = np.array(im)
#擷取標籤(label)
#   label = os.path.splitext(f)[0][0]
    #print(i+1, ':', f, ' ', os.path.splitext(f)[0], ' ', label)
#記錄圖檔標籤，存成字串
#   if (Labelstr == ''):
#       Labelstr = str(label)
#   else:
#       Labelstr = Labelstr + ',' + str(label)

    i+1
###結束 for f in files:

#擷取訓練資料 圖檔標籤
#開啟 csv 檔案
Labelstr = ''
with open('train.csv', newline='') as csvFile:

#轉成一個 dictionary, 讀成 csv 檔內容，將每一列轉成字典
    rows = csv.DictReader(csvFile)
```

```
#迴圈輸出 指定欄位
    for row in rows:
        print(row['image_id'], row['label'])

        if(row['label']=='A'):
            Labelstr = Labelstr + ',' + str(0) #0
        elif (row['label']=='B'):
            Labelstr = Labelstr + ',' + str(1) #1
        else:
            Labelstr = Labelstr + ',' + str(2) #2

#標籤字串 轉 陣列
Labelstr = Labelstr.split(',')
Labelstr = Labelstr[1:]
#print (Labelstr)
train_label = np.array(Labelstr, dtype = int)
print(len(train_label))

#print(Labelstr)
#print("訓練資料 label: ", train_label)

##########################讀測試資料
#讀取圖檔(jpg):圖檔前處理完成
#讀取圖檔標籤
import os
import numpy as np
from PIL import Image

#設定檔案路徑(目錄)
filepath='MangoTrain'

#計算圖檔數量
fn = 0
for f in os.listdir(filepath):
    fn = fn + 1
print("訓練資料檔案數: ", fn)
```

```
#宣告圖片陣列空間
imageNum = fn
test_feature = np.zeros((imageNum, 224, 224, 3), dtype = np.int)
#擷取檔案路徑(目錄)的檔案路徑
for root, dirs, files, in os.walk(filepath):
    #print(root)#當前目錄路徑
    #print(dirs)#當前路徑下所有子目錄
    #print(files)#當前路徑下所有非目錄子檔案
    files

#擷取測試資料 圖檔 及 標籤
i = 0
Labelstr = ''
for f in files:
    filename = filepath + '/' + f
    im = Image.open(filename)
#From image to array
    test_feature[i,:,:,:] = np.array(im)
#擷取標籤(label)
#   label = os.path.splitext(f)[0][0]
    #print(i+1, ':', f, ' ', os.path.splitext(f)[0], ' ', label)
#記錄圖檔標籤，存成字串
#   if (Labelstr == ''):
#       Labelstr = str(label)
#   else:
#       Labelstr = Labelstr + ',' + str(label)

    i+1
###結束 for f in files:

#擷取訓練資料 圖檔標籤
#開啟 csv 檔案
Labelstr = ''
with open('test.csv', newline='') as csvFile:

#轉成一個 dictionary, 讀成 csv 檔內容，將每一列轉成字典
    rows = csv.DictReader(csvFile)
```

```
#迴圈輸出 指定欄位
    for row in rows:
        print(row['image_id'], row['label'])

        if(row['label']=='A'):
            Labelstr = Labelstr + ',' + str(0) #0
        elif (row['label']=='B'):
            Labelstr = Labelstr + ',' + str(1) #1
        else:
            Labelstr = Labelstr + ',' + str(2) #2

#標籤字串 轉 陣列
Labelstr = Labelstr.split(',')
Labelstr = Labelstr[1:]
#print (Labelstr)
test_label = np.array(Labelstr, dtype = int)
print(len(test_label))
#print(Labelstr)
#print(test_label)
#print("訓練資料 label: ", test_label)
##########################################################
大雄修改 END

# Features 特徵值轉換為 60000*28*28*1 的 4 維矩陣
train_feature_vector = train_feature.reshape(len(train_feature), 28,
28, 3).astype('float32')
test_feature_vector = test_feature.reshape(len(test_feature), 28, 28,
3).astype('float32')

# Features 特徵值標準化
train_feature_normalize = train_feature_vector / 255
test_feature_normalize = test_feature_vector / 255

# 標籤為 One-Hot Encoding 編碼
train_label_onehot = np_utils.to_categorical(train_label)
test_label_onehot = np_utils.to_categorical(test_label)
```

```
# 建立模型
model = Sequential()
# 建立卷積層 1
model.add(Conv2D(filters=10,
                 kernel_size=(3, 3),
                 padding='same',
                 input_shape=(28, 28, 3),
                 activation='relu'))

# 建立池化層
model.add(MaxPooling2D(pool_size=(2, 2))) # (10, 14, 14)

# 建立卷積層 2
model.add(Conv2D(filters=20,
                 kernel_size=(3, 3),
                 padding='same',
                 activation='relu'))

# 建立卷積層 2.1
model.add(Conv2D(filters=20,
                 kernel_size=(3, 3),
                 padding='same',
                 activation='relu'))

# 建立池化層 2
model.add(MaxPooling2D(pool_size=(2, 2)))    # (20, 7, 7)

# Dropout 層防止過度擬合，斷開比例:0.2
model.add(Dropout(0.2))

# 建立平坦層：20*7*7=980 個神經元
model.add(Flatten())

# 建立隱藏層
model.add(Dense(units=256, activation='relu'))
```

```
# 建立輸出層：units=3，只有 3 種輸出
model.add(Dense(units=3, activation='softmax'))

# 定義訓練模型
model.compile(loss='categorical_crossentropy',
                    optimizer='adam', metrics=['accuracy'])

# 訓練模型 (train_feature_normalize, train_label_onehot 資料訓練)
# 訓練資料保留 20% 作為驗證集，包含批次為 200 筆資料，顯示簡易訓練過程
train_history = model.fit(x=train_feature_normalize,
y=train_label_onehot, validation_split=0.2,
                                    epochs=5, batch_size=30,
verbose=2)

# 評估準確率
scores = model.evaluate(test_feature_normalize, test_label_onehot)
print('\n 準確率=', scores[1])

#預測
prediction = np.argmax(model.predict(test_feature_normalize),
axis=-1)

##############################################顯示結果

#顯示預測、真實值
#show_labels_predictions(test_label, prediction, 0, imageNum)

#顯示 measures
show_predictions(test_label, prediction, 0, imageNum)

#顯示圖像、預測值、真實值
#show_image_labels_predictions(test_feature,test_label,prediction,
0,imageNum)
```

三、 CNNCMango - 去背（還可以再調整）

```
#%%
```

```
import cv2
import numpy as np
import os

def remove_background(image_path):
    """
    使用 GrabCut 算法進行背景移除。
    :param image_path: 圖片路徑
    :return: 去背後的影像
    """
    img = cv2.imread(image_path)
    mask = np.zeros(img.shape[:2], np.uint8)

    # 定義初始矩形區域（適合您的影像）
    height, width = img.shape[:2]
    rect = (10, 10, width-10, height-10)  # 可根據需求調整

    # 初始化模型
    bgdModel = np.zeros((1, 65), np.float64)
    fgdModel = np.zeros((1, 65), np.float64)

    # 執行 GrabCut 算法
    cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5,
cv2.GC_INIT_WITH_RECT)

    # 建立遮罩
    mask2 = np.where((mask == 2) | (mask == 0), 0,
1).astype('uint8')

    # 將影像與遮罩結合
    img = img * mask2[:, :, np.newaxis]

    return img

#%%
def process_dataset(input_dir, output_dir):
    """
```

```
    處理整個資料集，對圖片去背後存入新資料夾。
    :param input_dir: 原始圖片資料夾路徑
    :param output_dir: 去背後的圖片儲存路徑
    """
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for filename in os.listdir(input_dir):
        input_path = os.path.join(input_dir, filename)
        output_path = os.path.join(output_dir, filename)

        # 確保處理的是圖片
        if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
            try:
                result = remove_background(input_path)
                cv2.imwrite(output_path, result)
                print(f"Processed: {filename}")
            except Exception as e:
                print(f"Error processing {filename}: {e}")
input_dir = "MangoTrain"  # 原始訓練資料資料夾
output_dir = "MangoTrain_BGRemoved"  # 去背後的資料夾
process_dataset(input_dir, output_dir)

#%%
from PIL import Image

def load_images_and_labels(image_dir, label_file,
target_size=(224, 224)):
    """
    載入圖片及標籤。
    :param image_dir: 去背後的圖片資料夾
    :param label_file: 標籤檔案路徑 (CSV)
    :param target_size: 圖片尺寸
    :return: 圖片陣列與標籤
    """
    images = []
    labels = []
```

```
    # 載入標籤
    label_dict = {}
    with open(label_file, 'r') as csv_file:
        for line in csv.DictReader(csv_file):
            label_dict[line['image_id']] = line['label']

    # 載入圖片
    for filename in os.listdir(image_dir):
        if filename in label_dict:
            img_path = os.path.join(image_dir, filename)
            img = Image.open(img_path).resize(target_size)
            images.append(np.array(img))
            # 將標籤轉為數字
            if label_dict[filename] == 'A':
                labels.append(0)
            elif label_dict[filename] == 'B':
                labels.append(1)
            else:
                labels.append(2)

    return np.array(images), np.array(labels)

# 使用去背後的資料
train_images, train_labels =
load_images_and_labels("MangoTrain_BGRemoved", "train.csv")
test_images, test_labels =
load_images_and_labels("MangoTest_BGRemoved", "test.csv")

# 特徵標準化
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255

# 標籤 One-Hot 編碼
train_labels = np_utils.to_categorical(train_labels)
test_labels = np_utils.to_categorical(test_labels)
```

```
#%%
# 建立模型
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu',
input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(filters=64, kernel_size=(3, 3),
activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=3, activation='softmax'))

# 定義訓練
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# 訓練模型
train_history = model.fit(train_images, train_labels,
validation_split=0.2, epochs=10, batch_size=32, verbose=2)

# 評估準確率
scores = model.evaluate(test_images, test_labels, verbose=0)
print(f"Test Accuracy: {scores[1]:.2f}")
```

20241128. 比賽附加

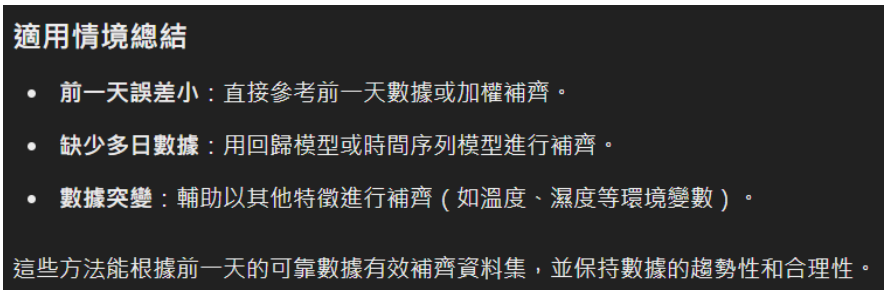
壹、 相關知識

一、 LGBM (LightGBM)、XGBoost 和 CatBoost 是三種常用的梯度提升樹（Gradient Boosting Trees）模型，廣泛用於處理結構化數據的機器學習任務（例如分類與迴歸問題）。

二、 LightGBM (Light Gradient Boosting Machine)：速度快、內存占用低、適用於大數據、支持 GPU 加速：對大數據和高維特徵尤為友好、自動處理缺失值、特徵重要性分析，適用於大規模數據集、高維稀疏數據（如 One-Hot Encoding 特徵）、需要快速模型訓練和預測的場景

三、 XGBoost (Extreme Gradient Boosting)：成熟穩定、強大的正則化、支持多種功能、分佈式訓練、收縮率與列採樣、GPU 支持，適用於中小型數據集（常見 Kaggle 比賽場景）、需要靈活自定義損失函數的情況、傳統數據分析工作流

四、 CatBoost：自動處理類別型特徵、訓練速度快、抗過擬合、缺失值處理、GPU 加速、易用性高，適用於含大量類別型特徵的數據集（例如非數值型特徵）、缺失值多、特徵分布複雜的場景、模型效果敏感場景（如推薦系統）

五、 

20241202. 文本分析和視覺化

壹、 相關知識

一、 將單字向量化後可以做運算，Ex: (king)-(man)+(woman)=(queen)

二、 Word2Vec 是用於將單字轉換為向量的模型，其參數設定對模型的效能和效果有重要影響。以下是 Word2Vec 的主要參數及其說明：

1. sentences:輸入的訓練句子，可以是清單。對於大規模語料，建議使用`BrownCorpus`、`Text8Corpus`或`LineSentence`來建構。
2. size:特徵向量的維度，預設為 100。較大的維度需要更多的訓練數據，但通常會提高效果，建議值在幾十到幾百之間。
3. window:當前字與預測詞在句子中的最大距離，預設為 5。這個參數會影響上下文的考慮範圍，通常設定為 5，但可以根據具體情況調整。
4. alpha:學習率，控制模型學習的速度，預設為 0.025。
5. seed:隨機數產生器的種子，用於初始化詞向量。
6. min_count:字頻的下限，低於此頻率的字會被忽略，預設為 5。這有助於減少稀有詞對模型的影響。
7. max_vocab_size:設定建構詞向量時的記憶體限制。如果獨立單字數量超過該值，將移除頻率最低的單字。每千萬個單字大約需要 1GB 的記憶體。
8. sample:高頻詞彙的隨機降採樣閾值，預設為 1e-3。此參數可以幫助減少高頻詞對模型訓練的影響。
9. workers:控制訓練時使用的平行執行緒數量，預設值為 3。
10. sg:設定訓練演算法，預設為 0（CBOW），如果設為 1，則使用 Skip-gram 演算法。
11. hs:如果為 1，則使用層次 Softmax 技術；如果為 0（預設），則使用負取樣。
12. negative:如果大於 0，則採用負取樣，並設定雜訊單字的數量。
13. cbow_mean:在使用 CBOW 時，如果為 0，則使用上下文詞向量總和；如果為 1（預設），則使用平均值。
14. iter:迭代次數，預設值為 5。增加迭代次數可以提高模型精度，但也會增加訓練時間。

15. trim_rule:用於設定詞彙表整理規則，可以指定哪些單字保留或刪除 sorted_vocab:如果為 1（預設），則在分配單字索引時會先按頻率降序排序。
16. batch_words:每批傳遞給線程的單字數量，預設為 10000，有助於控制記憶體使用和訓練效率。

三、 詞向量（Word Vector）是一種將自然語言中的詞轉換為計算機可處理的數值形式的方法，這對於自然語言處理（NLP）至關重要。詞向量技術使得每個詞可以在高維空間中表示，並且這些向量之間的距離或角度反映了詞與詞之間的語義相似性。

1. 詞向量的基本概念
   (1) 詞向量的定義：詞向量是將詞彙轉換成包含語意訊息的向量表達，通常使用類神經網路進行訓練。這種方法將傳統的獨熱編碼（One-hot Encoding）轉換為低維度的向量，以便於計算和分析。
   (2) 語義相似性：在向量空間中，兩個詞的餘弦相似度越高，表示它們的語意越接近。例如，「男人」和「女人」在某些上下文中可能具有相似的語義，但在性別這一維度上則是相反的。
2. 常見的詞向量模型
   (1) Word2Vec：是一種流行的詞向量計算方法，主要有兩種模型：CBOW（Continuous Bag of Words）：根據上下文來預測當前詞；Skip-gram：根據當前詞來預測上下文中的其他詞。
   (2) GloVe（Global Vectors for Word Representation）：GloVe 是另一種詞向量模型，它通過統計全局共現信息來學習詞向量，強調了整體語料庫中的統計特性。
3. 詞向量的應用

   詞向量在多個 NLP 任務中具有廣泛應用，包括：
   (1) 文本分類：將文本數據自動劃分到預定義類別中。
   (2) 情感分析：評估文本中所表達的情感傾向。
   (3) 機器翻譯：將一種語言的文本自動翻譯成另一種語言。
4. 訓練和評估

   在訓練詞向量時，需要考慮以下因素：
   (1) 維度選擇：增加維度通常能提高效果，但過高的維度會導致計算複雜性增加。
   (2) 語料質量：使用純淨且相關性高的語料庫能顯著提升訓練效果。

(3) 上下文窗口大小：大的上下文窗口能捕捉到更豐富的主題信息，而小窗口則更注重局部語境。

5. 除 word2vec 外的詞向量模型
   (1) Bag-of-Word(BoW)
   (2) GloVe
   (3) FastText

四、 NLP 的應用

1. 情緒分析
2. 聊天機器人
3. 語言翻譯
4. 語音辨識
5. 資訊檢索
6. 電子郵件篩選器
7. 智慧助理
8. chatgpt 與 MLP 的關係

   MLP 通常需要大量標註數據進行訓練，而 ChatGPT 則採用預訓練和微調的方法。ChatGPT 先在大量無標註文本上進行預訓練，然後再根據特定任務進行微調。這種方法使得 ChatGPT 能夠更有效地學習語言模式，而不僅僅是依賴於固定的輸入輸出

貳、 程式碼解讀

一、 punkt 用於斷句和分詞

二、 stopwords 用於處理停用詞（即在文本中頻繁出現但對文本分析無太大意義的詞，例如"the","a","is"等）

三、 lower()將文本中的每個單詞轉換為小寫，以消除大小寫差異

四、 使用 PorterStemmer 進行詞幹提取，將每個單詞轉換為其基本形式（例如"running"轉換為"run"）

五、 二元組（2-gram），即將文本中的兩個相鄰詞組合成一個新詞組

六、 Phrases 和 Phraser 是 gensim 庫用來檢測文本中的常見雙詞搭配的工具

七、 使用 t-SNE 進行降維，將高維的詞向量投影到 2D 空間，並用 Bokeh 庫來可視化這些詞語

八、 coords_df 是包含詞向量降維後座標的 DataFrame，會顯示前 5000 個詞語的可視化結果

參、 附錄 – 程式碼

一、 NLP

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  2 09:29:18 2024

@author: ASUS
"""

import nltk
from nltk import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import *
nltk.download('gutenberg')
nltk.download('punkt')
nltk.download('stopwords')

import numpy as np

import string
#pip install gensim
import gensim
from gensim.models.phrases import Phraser, Phrases
from gensim.models.word2vec import Word2Vec

from sklearn.manifold import TSNE

import pandas as pd
from bokeh.io import output_notebook, output_file
from bokeh.plotting import show, figure
#%matplotlib inline




from nltk.corpus import gutenberg
print('顯示文學作品數量=',len(gutenberg.fileids()))
print('顯示文學作品名稱=',gutenberg.fileids())
print('顯示文學作品總字數=',len(gutenberg.words()))
```

```
gberg_sent_tokens = sent_tokenize(gutenberg.raw())

print ('標題等相關資訊', gberg_sent_tokens[0])
print('第一段')
print(gberg_sent_tokens[1]) #gberg_sent_tokens[1]
print('斷字(第一段)')
print(word_tokenize(gberg_sent_tokens[1]))
#word_tokenize(gberg_sent_tokens[1])[14]

#顯示上述資料改用其他函數
gberg_sents = gutenberg.sents()
#gberg_sents[0:3]
#gberg_sents[4][14]

#11.1.2 將大寫字母轉成小寫(以艾瑪第 1 句為例)原本索引 4 這句的內容
print('原本索引 4 這句的內容')
print(gberg_sents[4])

#將索引 4 這句轉成小寫
print('將索引 4 這句轉成小寫')
print([w.lower() for w in gberg_sents[4]])

#11.1.3 移除停用字與標點符號(以艾瑪第 1 句為例)
stpwrds = stopwords.words('english') + list(string.punctuation)
print('移除停用字與標點符號（以艾瑪第 1 句為例）')
print(stopwords)

print('移除停用字與標點符號 & 轉成小寫')
print([w.lower() for w in gberg_sents[4] if w.lower() not in
stpwrds])


#11.1.4 字根提取(stemming)(以艾瑪第 1 句為例)
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
```

```
print('字根提取(stemming)(以艾瑪第 1 句為例)')
stemming = [stemmer.stem(w.lower()) for w in gberg_sents[4]
                    if w.lower() not in stpwrds]
print(stemming)


#########將語料庫中的 2-grem 詞彙檢測出來
phrases = Phrases(gberg_sents)
bigram = Phraser(phrases)
print('將語料庫中的 2-grem 詞彙檢測出來, 僅顯示前 5 個')
gram_2 = bigram.phrasegrams
print(list(gram_2)[:5])

'''
#測試看看 bigram 物件能否運作
test_sentence = "Miss Taylor has two doughters".split()
test_sentence
bigram[test_sentence]
'''

lower_sents = []
for s in gberg_sents:
      lower_sents.append([w.lower() for w in s if w.lower()
                                  not in list(string.punctuation)])

lower_bigram = Phraser(Phrases(lower_sents))

lower_bigram.phrasegrams #全小寫的 2-gram
lower_bigram = Phraser(Phrases(lower_sents,
                                          min_count=32,
threshold=64))
lower_bigram.phrasegrams
clean_sents = []
for s in lower_sents:
      clean_sents.append(lower_bigram[s])
```

```
print('輸出 clean_sents[6]')
print(clean_sents[6])

print('建立詞向量空間物件模型')
#建立詞向量空間物件模型, size=64 改成 vector_size=64; iter=5
改成 epochs=僅顯示前 5 個
model = Word2Vec(sentences=clean_sents, vector_size=64,
                 sg=1, window=10, epochs=5,
                 min_count=10, workers=4)
filename = 'ch11-clean_gutenberg_model_trian.w2v'
model.save(filename)
#直接取用本書轉換好的檔案，注意，執行此區塊前，請先執
行本頁第一個程式區塊載入 Google 雲端硬碟!!!
#model = gensim.models.Word2Vec.load
words = list(model.wv.index_to_key)
print('出現十次以上的 token 數量=', words)

#評估生成的詞向量空間
print('dog64 的維度空間向量')
print(model.wv['dog'])

print('father 最相似的 3 個單字')
print(model.wv.most_similar('father', topn=3))
'''
model.wv.most_similar('dog', topn=3)
model.wv.most_similar('eat', topn=3)
model.wv.most_similar('day', topn=3)
'''
print('最不相似的單字')
print(model.wv.doesnt_match("mother father sister brother
dog".split()))

print('相似度(father, dog)=',model.wv.similarity('father', 'dog'))

print('向量加減(father, woman)-(man)')
print(model.wv.most_similar(positive=['father', 'woman'],
negative=['man']))
```

```
coords_df = pd.read_csv('ch11-clean_gutenberg_tsne.csv')

coords_df.head()
_ = coords_df.plot.scatter('x', 'y', figsize=(12,12),
                                        marker='.', s=10, alpha=0.2)

subset_df = coords_df.sample(n=5000)
p = figure(width=800, height=800)
_ = p.text(x=subset_df.x, y=subset_df.y, text=subset_df.token)
show(p)
```

二、 NLP_IMDb

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  2 10:31:09 2024

@author: ASUS
"""

from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import
pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.layers import Embedding
from tensorflow.keras.callbacks import ModelCheckpoint
import os
from sklearn.metrics import roc_auc_score, roc_curve
import pandas as pd
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")

'''
#載入 IMDb 影評資料集(原始資料)
(x_train, y_train), (x_test, y_test) = imdb.load_data()
```

```
#查看 IMDB 資料集的內容
for x in x_train[0:6]:
      print(len(x))
#查看每筆資料的內容
x_train[0:6]

word_index = imdb.get_word_index()
word_index = {k:(v+3) for k, v in word_index.item()}
word_index["START"] = 1
index_word = {v:k for k, v in word_index.items()}

text = ' '.join(index_word[id] for id in x_train[0])
print(text)
'''

#正式模型起點
#載入 IMDb 影評資料集，依據資料預處理參數
#預處理（一）：移除停用子（stop words）
#資料預處理參數
n_words_to_skip = 50 #stop words
max_review_length = 100
pad_type = 'pre' #字數不夠，前面補 0
trunc_type = 'pre' #字數太多，刪除前面文字（僅留後面）

#詞向量空間
n_unique_words = 5000
n_dim = 64
(x_train, y_train), (x_test, y_test) = \
imdb.load_data(skip_top = n_words_to_skip, num_words =
n_unique_words)

print('案例數（訓練集） = ', len(y_train))
print('案例數（測試集） = ', len(y_test))

#print(x_train[0:6])

#預處理（二）：統一影評長度
```

```
x_train = pad_sequences(x_train, maxlen = max_review_length,
                                    padding = pad_type, truncating =
trunc_type, value = 0)
x_test = pad_sequences(x_test, maxlen = max_review_length,
                                   padding = pad_type, truncating =
trunc_type, value = 0)

#建立模型
from sklearn import tree
model = tree.DecisionTreeClassifier()

#訓練
model.fit(x_train, y_train)

y_hat = model.predict(x_test)

print('y_hat[0]', y_hat[0])
y_hat[0]

#模型評估（二）：繪製長方圖
plt.hist(y_hat)
_ = plt.axvline(x = 0.5, color = 'orange')

#模型評估（三）：ROC AUC  分數

#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨機性，
#因此底下（或您重跑一次）的結果不會與書中完全一樣，但模型的能力是相近的
pct_auc = roc_auc_score(y_test, y_hat) * 100.0
"{:0.2f}".format(pct_auc)
print('ROC AUC  分數  = ', pct_auc)

##########################################################
######
##############新增程式碼 provided by  翁政雄老師
##############
```

```
##########################################################
######

pred = y_hat #以下用 pred 代替 y_hat

'''
#將預測結果(0~1)轉成 0/1，即 pred 只能是 0 或 1
#方法(1)
import numpy as np
pred = np.where(y_hat > 0.5, 1, 0)
#方法(2)
pred = y_hat.round(0)
print(pred)
'''

#compute ROC curve and ROC area for each class
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, threshold = roc_curve(y_test, pred) ###計算
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
#plt.figure(figsize = (10, 10))
plt.plot(fpr, tpr, color = 'darkorange',
         lw = lw, label = 'ROC curve (area = %0.4f)' % roc_auc)
#假正率為橫坐標，真正率為縱坐標做曲線
plt.plot([0, 1], [0, 1], color = 'navy', lw = lw, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC Curve')

#plt.legend(loc = "lower right") #標籤位置
plt.legend()
plt.show()
```

```
print("ROC_auc area = %.4f" % (roc_auc))

from sklearn.metrics import precision_recall_curve
#lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_probs)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)

#lr_f1, lr_auc = f1_score(y_test, pred), auc(lr_recall, lr_precision)
lr_auc = auc(lr_recall, lr_precision)
#summarize scores
#print('MLP(ANN): f1 = %.3f PRC_auc area = %.3f' % (lr_f1,
lr_auc)) #f1 乃是 label = 1 的 f1
print("PRC_auc area = %.4f" % (lr_auc))
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

#axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
#show the legend
plt.legend()
#show the plot
plt.show()

#輸出混亂矩陣，顯示準確率
pred = y_hat.round(0) #將預測結果(0~1)轉成 0/1，即 pred 只能
是 0 或 1

from sklearn.metrics import confusion_matrix, classification_report
print('輸出混亂矩陣，顯示準確率：使用訓練資料')
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

#設定每周期結束紀錄模型參數
output_dir = 'E:/學校/國立彰化師範大學/大三上/人工智慧
/20241202'
from pandas import DataFrame
```

```
#df = DataFram({"Real": y_test, "Pred": pred.flatten(), "Comment":
x_test.flatten()})
df = DataFrame({"Real": y_test, "Pred": pred.flatten()})
filenameReport = output_dir + '/DT_IMDb.csv'
print('輸出至檔案', filenameReport)
df.to_csv(filenameReport, index = True)
```

20241209. 文字雲、文字語音、撥放音樂

壹、 相關知識

一、 文字雲：使用 wordcloud 庫將文本可視化。
二、 文字語音：使用 pyttsx3 或 gTTS 庫將文本轉換為語音。
三、 音樂播放：使用 pygame 播放音樂。

貳、 程式碼解讀

一、 dictfile: jieba 的字典文件，用來提高分詞精度
   1. 使用 jieba 進行文本分詞，並使用 jieba.analyse.extract_tags 提取關鍵詞（最多 25 個）
   2. 使用 jieba.lcut 進行精確分詞，並將每個詞的出現頻率計算出來
二、 fontpath:字體文件，用於在文字雲中顯示中文
三、 mdfile:要處理的文本文件
四、 pngfile:生成文字雲時的形狀文件（例如，心形或 Python 標誌）
五、 使用 WordCloud 函數來生成文字雲，其中 mask 設定為圖片形狀（如心形），font_path 設定為支持中文的字體文件，並從 freg 字典中生成頻率圖
六、 SpeechRecognition:用來處理語音識別的主要庫
   1. speech_recognition.Recognizer():創建一個語音識別器物件，用於處理音頻
   2. with speech_recognition.Microphone() as source:: 這行代碼會打開麥克風，並讓程序進行音頻錄製。source 是錄音的來源，這裡使用了默認的麥克風設備
七、 PyAudio:是 SpeechRecognition 所依賴的庫，用於錄製來自麥克風的音頻
八、 result= r.recognize_google(audio, language='zh-TW'): 將錄音轉換為文字，並使用 Google 的語音識別 API 來進行語音轉文字，這裡設置語言為繁體中文（zh-TW）
九、 pyttsx3 是一個文本轉語音的庫，能夠將文字轉換為語音並播放出來
   1. engine = pyttsx3.init(): 初始化語音引擎，該引擎用來將文字轉換為語音

2. engine.setProperty('rate', 160): 設置語速，160 是語速的數值。數值越小語速越慢，數值越大語速越快。默認語速大約是 200
3. engine.setProperty('volume', 0.8): 設置音量，0.8 是音量的設置範圍（範圍從 0.0 到 1.0），1.0 表示最大音量
4. engine.say("You've got mail!"): 將這段文字轉換為語音並加入語音播放隊列中
5. engine.say("翁教授"):再次將這段文字加入語音隊列
6. engine.runAndWait():開始執行語音播放，並等待語音播放完成
7. if engine._inloop:: 判斷語音播放是否仍在進行中。如果是，則結束循環（engine.endLoop()）來防止程序過早結束
8. engine.stop(): 停止語音引擎的運行，這會停止任何尚在進行中的語音播放

十、 pygame.mixer.init()：初始化音頻系統。這步驟是設置 pygame 用來播放音樂所需的音頻混音器

1. filename='w59l4-7pph9.wav'：指定要播放的音樂文件名。這裡的音樂檔案是.wav 格式，可以換成其他支持的音頻格式，如.mp3，但需要確保音頻文件在正確的路徑下
2. pygame.mixer.music.load(filename)：加載指定的音樂檔案
3. 播放控制：
   (1) pygame.mixer.music.play()：開始播放音樂
   (2) pygame.mixer.music.pause()：暫停音樂播放
   (3) pygame.mixer.music.unpause()：從暫停處繼續播放音樂
   (4) pygame.mixer.music.stop()：停止音樂播放，並將播放進度設置為開始處
   (5) pygame.mixer.music.play(-1)：無限循環播放音樂，-1 表示音樂將會不斷重複直到手動停止

參、 附錄 – 程式碼

一、 MLP_IMDb_CNN

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 10:21:36 2024

@author: ASUS
"""
```

```
from tensorflow.keras.datasets import imdb #new!
from tensorflow.keras.preprocessing.sequence import
pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout,
Conv1D, Embedding, SpatialDropout1D, GlobalMaxPooling1D
from tensorflow.keras.callbacks import ModelCheckpoint #new!
import os #news!
from sklearn.metrics import roc_auc_score, roc_curve
#import pandas as pd
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

'''
#載入 IMDb 影評資料集(原始資料)
(x_train, t_train), (X_test, y_test) = imdb.load_data()
#查看 IMDb 資料集的內容
for x in x_train[0:6]:
      print(len(x))
#查看每筆資料內容
x_train[0:6]

word_index = imdb.get_word_index()
word_index = {k : (v+3) for k, v in word_index.items()}
word_index['START'] = 1
index_word = {v:k for k, v in word_index.items()}

text = ' '.join(index_word[id] for id in x_train[0])
print(text)
'''

#正式模型起點
#載入 IMDb 影評資料集，依據資料預處理參數
#預處理（一）：移除停用字（stop words）
```

```
#資料預處理參數
#n_words_to_skip = 50
max_review_length = 400
pad_type = 'pre' #字數太少前面補 0
trunc_type = 'pre' #字數太多刪除前面
n_unique_words = 5000
n_dim = 64
#訓練神經網路 DNN
n_dense = 256
dropout = 0.2
epochs = 3
batch_size = 128
drop_embed = 0.2
n_conv = 256
k_conv = 3



#詞向量空間
(x_train, y_train), (x_test, y_test) = \
imdb.load_data(num_words = n_unique_words)

'''
print('案例數（訓練集）=', len(y_train))
print('案例數（測試集）=', len(y_test))
'''

#print(x_train[0:6])

#預處理（二）：統一影評長度
x_train = pad_sequences(x_train, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

x_test = pad_sequences(x_test, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)
```

```
'''
for x in x_train[0:6]:
    print(len(x))
print(x_train[5])
'''

#用密集神經網路區分正評、負評
#密集神經網路的架構 DNN
model = Sequential()
model.add(Embedding(n_unique_words, n_dim, input_length =
max_review_length))
model.add(SpatialDropout1D(drop_embed))
model.add(Conv1D(n_conv, k_conv, activation = 'relu'))
model.add(GlobalMaxPooling1D())
#model.add(Flatten())
model.add(Dense(n_dense, activation = 'relu'))
model.add(Dropout(dropout))
#model.add(Dense(n_dense, activation = 'relu'))
#model.add(Dropout(dropout))
model.add(Dense(1, activation = 'sigmoid'))

#輸出模型
#print(model.summary())

#編譯模型
model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

#設定每周期結束紀錄模型參數
output_dir = 'E:/學校/國立彰化師範大學/大三上/人工智慧
/20241209/weight'
modelcheckpoint = ModelCheckpoint(filepath = output_dir +
'/weightsCNN.{epoch:02d}.keras')

if not os.path.exists(output_dir):
    os.makedirs(output_dir)
```

```
#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型的能力是相近的
model.fit(x_train, y_train,
                batch_size=batch_size, epochs=epochs, verbose=1,
                validation_data=(x_test, y_test),
                callbacks=[modelcheckpoint])
#模型評估(一):挑幾筆來觀察
model.load_weights(output_dir+ "/weightsCNN.03.keras")#請視以上執行結果指定指定較佳的權重
y_hat = model.predict(x_test)

#print('y_hat[0]',y_hat[0])
#y_hat[0]

#模型評估(二):繪製直方圖
plt.hist(y_hat)
_ = plt.axvline(x = 0.5, color = 'orange')

#模型評估(三):ROC AUC  分數
#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型的能力是相近的
pct_auc = roc_auc_score(y_test, y_hat)*100.0
"{:0.2f}".format(pct_auc)
print('ROC AUC  分數=', pct_auc)

################################################
#新增程式碼 provided by  翁政雄老師###############
################################################

pred = y_hat #以下用 pred 代替 y_hat

'''
```

```
#將預測結果(0~1)轉成 0/1，即 pred 只能是 0 或 1
#方法(1)
import numpy as np
pred = np.where(y_hat > 0.5, 1, 0)
#方法(2)
pred = y_hat.round(0)
print(pred)
'''

#Compute ROC curve and ROC area for each class
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, threshold = roc_curve(y_test, pred)
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
#plt.figure(figsize = (10, 10))
plt.plot(fpr, tpr, color = 'darkorange',
         lw = lw, label = 'ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color = 'navy', lw = lw, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('FPR')
plt.ylabel('TFR')
plt.title('ROC Curve')

#plt.legend(loc = 'lower right')
plt.legend()
plt.show()
print('ROC_auc area = %.4f' % (roc_auc))

from sklearn.metrics import precision_recall_curve
#lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_probs)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)

#lr_f1, lr_auc = f1_score(y_test, pred), auc(lr_recall, lr_precision)
```

```
lr_auc = auc(lr_recall, lr_precision)
#summarize scores
#print('MLP(ANN): f1 = %.3f PRC_auc = %.3f' % (lr_f1, lr_auc))
print('PRC_auc area = %.4f' % (lr_auc))

#plot the precision-recall curves
no_skill = len(y_test[y_test == 1]) / len(y_test)

#plt.figure(figsize=(10, 10))
#plt.plot([0, 1], [no_skill, no_skill], linestyle = '--', label = 'No
Skill')
plt.plot([0, 1], [1, 0], color = 'navy', lw = lw, linestyle = '--')
plt.plot(lr_recall, lr_precision, color = 'darkorange',
         lw = lw, label = 'PRC curve (area = %.4f)' % lr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

#axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
#show the legend
plt.legend()
#show the plot
plt.show()

#輸出混亂矩陣，顯示準確率
pred = y_hat.round(0)

from sklearn.metrics import confusion_matrix, classification_report
print('輸出混亂矩陣，顯示準確率：使用訓練資料')
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

from pandas import DataFrame
df = DataFrame({'Real': y_test, 'Pred': pred.flatten()})
filenameReport = output_dir + '/CNN_IMDb.csv'
```

```
print('輸出至檔案', filenameReport)
df.to_csv(filenameReport, index = True)
```

二、 MLP_IMDb_DNN

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 10:21:36 2024

@author: ASUS
"""

from tensorflow.keras.datasets import imdb #new!
from tensorflow.keras.preprocessing.sequence import
pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout,
Conv1D, Embedding, SpatialDropout1D, GlobalMaxPooling1D
from tensorflow.keras.callbacks import ModelCheckpoint #new!
import os #news!
from sklearn.metrics import roc_auc_score, roc_curve
#import pandas as pd
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

'''
#載入 IMDb 影評資料集(原始資料)
(x_train, t_train), (X_test, y_test) = imdb.load_data()
#查看 IMDb 資料集的內容
for x in x_train[0:6]:
      print(len(x))
#查看每筆資料內容
x_train[0:6]

word_index = imdb.get_word_index()
word_index = {k : (v+3) for k, v in word_index.items()}
word_index['START'] = 1
```

```
index_word = {v:k for k, v in word_index.items()}

text = ' '.join(index_word[id] for id in x_train[0])
print(text)
'''

#正式模型起點
#載入 IMDb 影評資料集，依據資料預處理參數
#預處理（一）：移除停用字（stop words）
#資料預處理參數
#n_words_to_skip = 50
max_review_length = 400
pad_type = 'pre' #字數太少前面補 0
trunc_type = 'pre' #字數太多刪除前面
n_unique_words = 5000
n_dim = 64
#訓練神經網路 DNN
n_dense = 256
dropout = 0.2
epochs = 1
batch_size = 128
drop_embed = 0.2
n_conv = 256
k_conv = 3



#詞向量空間
(x_train, y_train), (x_test, y_test) = \
imdb.load_data(num_words = n_unique_words)

'''
print('案例數（訓練集）=', len(y_train))
print('案例數（測試集）=', len(y_test))
'''

#print(x_train[0:6])
```

```
#預處理（二）：統一影評長度
x_train = pad_sequences(x_train, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

x_test = pad_sequences(x_test, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

'''
for x in x_train[0:6]:
      print(len(x))
print(x_train[5])
'''
model = Sequential()
model.add(Embedding(n_unique_words, n_dim, input_length =
max_review_length))
model.add(Flatten())
model.add(Dense(n_dense, activation = 'relu'))
model.add(Dropout(dropout))
#model.add(Dense(n_dense, activation = 'relu'))
#model.add(Dropout(dropout))
model.add(Dense(1, activation = 'sigmoid'))

#輸出模型
print(model.summary())

#編譯模型
model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

#設定每周期結束紀錄模型參數
output_dir = 'E:/學校/國立彰化師範大學/大三上/人工智慧
/20241209/weight'
modelcheckpoint = ModelCheckpoint(filepath = output_dir +
'/weightsDNN.{epoch:02d}.keras')
```

```
if not os.path.exists(output_dir):
      os.makedirs(output_dir)

#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型的能力是相近的
model.fit(x_train, y_train,
              batch_size=batch_size, epochs=epochs, verbose=1,
              validation_data=(x_test,y_test),
              callbacks=[modelcheckpoint])
#模型評估(一):挑幾筆來觀察
model.load_weights(output_dir+ "/weightsDNN.01.keras")#請視以上執行結果指定指定較佳的權重
y_hat = model.predict(x_test)

print('y_hat[0]',y_hat[0])
y_hat[0]

#模型評估(二):繪製直方圖
plt.hist(y_hat)
_ = plt.axvline(x = 0.5, color = 'orange')

#模型評估(三):ROC AUC  分數
#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型的能力是相近的
pct_auc = roc_auc_score(y_test, y_hat)*100.0
"{:0.2f}".format(pct_auc)
print('ROC AUC  分數=', pct_auc)

###############################################
#新增程式碼 provided by  翁政雄老師##############
###############################################
```

```
pred = y_hat #以下用 pred 代替 y_hat

'''
#將預測結果(0~1)轉成 0/1，即 pred 只能是 0 或 1
#方法(1)
import numpy as np
pred = np.where(y_hat > 0.5, 1, 0)
#方法(2)
pred = y_hat.round(0)
print(pred)
'''

#Compute ROC curve and ROC area for each class
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, threshold = roc_curve(y_test, pred)
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
#plt.figure(figsize = (10, 10))
plt.plot(fpr, tpr, color = 'darkorange',
         lw = lw, label = 'ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color = 'navy', lw = lw, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('FPR')
plt.ylabel('TFR')
plt.title('ROC Curve')

#plt.legend(loc = 'lower right')
plt.legend()
plt.show()
print('ROC_auc area = %.4f' % (roc_auc))

from sklearn.metrics import precision_recall_curve
#lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_probs)
```

```
lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)

#lr_f1, lr_auc = f1_score(y_test, pred), auc(lr_recall, lr_precision)
lr_auc = auc(lr_recall, lr_precision)
#summarize scores
#print('MLP(ANN): f1 = %.3f PRC_auc = %.3f' % (lr_f1, lr_auc))
print('PRC_auc area = %.4f' % (lr_auc))

#plot the precision-recall curves
no_skill = len(y_test[y_test == 1]) / len(y_test)

#plt.figure(figsize=(10, 10))
#plt.plot([0, 1], [no_skill, no_skill], linestyle = '--', label = 'No
Skill')
plt.plot([0, 1], [1, 0], color = 'navy', lw = lw, linestyle = '--')
plt.plot(lr_recall, lr_precision, color = 'darkorange',
            lw = lw, label = 'PRC curve (area = %.4f)' % lr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

#axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
#show the legend
plt.legend()
#show the plot
plt.show()

#輸出混亂矩陣，顯示準確率
pred = y_hat.round(0)

from sklearn.metrics import confusion_matrix, classification_report
print('輸出混亂矩陣，顯示準確率：使用訓練資料')
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
from pandas import DataFrame
df = DataFrame({'Real': y_test, 'Pred': pred.flatten()})
filenameReport = output_dir + '/DNN_IMDb.csv'
print('輸出至檔案', filenameReport)
df.to_csv(filenameReport, index = True)
```

三、 NLPWordCloudNews_2024

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 09:21:17 2024

@author: ASUS
"""

#pip install wordcloud
#pip install jieba
from wordcloud import WordCloud #, STOPWORDS
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import jieba
import jieba.analyse
from collections import Counter # 次數統計
import pandas as pd

dictfile = 'dictionary/dict.txt.big.txt'
stopfile = 'dictionary/stopWord_cloud.txt'
fontpath = 'dictionary/LXGWWenKaiMonoTC-Light.ttf' #msyh.ttc

mdfile = 'dictionary/敘利亞阿塞德垮台 路透：中東重大轉折 伊朗俄羅斯影響力消散.txt' #news2024.txt
pngfile = 'dictionary/heart.png' #Python-Symbol.png

alice_mask = np.array(Image.open(pngfile))

jieba.set_dictionary(dictfile)
jieba.analyse.set_stop_words(stopfile)
```

```
text = open(mdfile, "r", encoding="utf-8").read()

tags = jieba.analyse.extract_tags(text, topK=25)

seg_list = jieba.lcut(text, cut_all = False)
dictionary = Counter(seg_list)

freg = {}
for ele in dictionary:
    if ele in tags:
        freg[ele] = dictionary[ele]



wordcloud = WordCloud(background_color="white",
mask=alice_mask, contour_width=3,
                                contour_color='steelblue',
font_path=fontpath).generate_from_frequencies(freg)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()

wordcloud.to_file("news_WordCloud.png")

print(freg)
# 降冪排序
sorted_freg = sorted(freg.items(), key=lambda x: x[1],
reverse=True)
print(sorted_freg)
# 儲存至 CSV
filename = 'freg.csv'
# 使用 pandas 將字典轉為 DataFrame 後儲存
df_freg = pd.DataFrame(sorted_freg, columns=['Word',
'Frequency'])
df_freg.to_csv(filename, index=False, encoding='utf-8-sig')
```

四、 Speech2Text

```
# -*- coding: utf-8 -*-
```

```
"""
Created on Mon Dec  9 10:14:17 2024

@author: ASUS
"""

#pip install SpeechRecognition
#pip install PyAudio

import speech_recognition
r = speech_recognition.Recognizer()
with speech_recognition.Microphone() as source:
    audio = r.listen(source)
result = r.recognize_google(audio, language='zh-TW')
print(result)
```

五、 text2Speech

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 10:06:02 2024

@author: ASUS
"""

#pip install pyttsx3
import pyttsx3
engine = pyttsx3.init()
engine.setProperty('rate', 160) #語速
engine.setProperty('volume', 0.8) #音量
engine.say("You've got mail!")
engine.say("翁教授")
engine.runAndWait()
if engine._inloop:
    engine.endLoop()
engine.stop()
'''
# 停止引擎
if engine.isBusy():  # 檢查是否忙碌
```

```
        engine.stop()
'''
```

六、 WAV Player

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 09:57:27 2024

@author: ASUS
"""

#pip install pygame
import pygame
pygame.mixer.init()
filename='w59l4-7pph9.wav'
pygame.mixer.music.load(filename)




#pygame.mixer.music.play() #播放
#pygame.mixer.music.pause() #暫停
#pygame.mixer.music.unpause() #繼續
#pygame.mixer.music.stop() #停止
#pygame.mixer.music.play(-1) #其中 -1 表示無限循環 #循環
```

20241216. 嵌入式 CNN（並列）+串列後加入其他模型（LSTM、RNN 等）、生成對抗式 AI

壹、 相關知識

一、 嵌入層（Embedding Layer）

1. 功能:
   (1) 將文本中的單詞（經過整數編碼）映射到低維詞向量空間。
   (2) 獲取詞語的語義表示，便於後續層提取語義特徵。
2. 特點:
   (1) 為非 CNN 組件，屬於特徵學習層。

二、 SpatialDropout1D

1. 功能:
   (1) 對嵌入層的輸出施加隨機失活，丟棄整個特徵通道。
2. 作用:
   (1) 防止過擬合，提高模型的泛化能力。
   (2) 與普通 Dropout 的區別:
   (3) 普通 Dropout 作用於單個神經元，SpatialDropout1D 更適合序列數據。

三、 GlobalMaxPooling1D

1. 功能:
   (1) 從卷積層輸出中選取最重要的特徵（如關鍵詞），降維簡化特徵圖。
2. 特點:
   (1) 池化操作，並非 CNN 組件，但與 CNN 協同作用提升效果。

四、 全連接層（Dense Layer）

1. 功能:
   (1) 作為特徵融合與分類層，將提取的高維特徵轉化為分類結果。
2. 結構:
   (1) 第一層：非線性變換，提高模型的表現能力。
   (2) 第二層：壓縮特徵空間，生成最終輸出（如二元分類）。
3. 特點:

(1) Dense 層為深度學習模型的基本組件，與 CNN 並行協同工作。

五、 回調函數（ModelCheckpoint）

1. 功能:
   (1) 保存訓練過程中最佳的模型權重。
2. 特點:
   (1) 與模型結構無關，但可提高訓練效率及實驗效果。
3. 模型擴展建議
   在現有架構的基礎上，可結合其他技術增強模型特性：
   (1) RNN（LSTM/GRU）
      i. 捕捉文本長時依賴性，補充 CNN 在序列建模方面的不足。
   (2) Transformer
      i. 利用自注意力機制捕捉全局語義，適合長文本任務。
   (3) 多頭卷積（Multi-head CNN）
      i. 融合不同感受野的卷積核，增強對文本特徵的捕捉能力。

六、 Capsule Network

整合語義和句子結構信息，提升模型解釋能力。

七、 為什麼在嵌入層後使用 CNN：特徵提取能力、數據降維與高效處理、結構優化與參數效率、計算性能

貳、程式碼解讀

一、 如果需要在不同層中同時處理數據，比如一部分數據經過卷積操作，另一部分數據直接送入全連接層，可以通過 Functional API 或多輸入模型實現並聯結構

1. 適合處理多樣化特徵
2. 提高模型的靈活性
3. concatenate 是 TensorFlow/Keras 中用於將多個張量合併在一起的一種方法，通常用於合併不同層的輸出

二、 GAN（生成對抗網路）

1. 生成器（Generator）：利用隨機噪聲生成類似真實資料的影像
2. 鑑別器（Discriminator）：辨別輸入的影像是真實還是生成的
3. 對抗訓練：生成器與鑑別器相互競爭，提升影像生成效果
4. 使用反卷積（上採樣）方法構建生成器，卷積方法構建鑑別器

5. 使用隨機噪聲作為生成器的輸入

參、 附錄 – 程式碼

一、 Airpollution_CNN_embed

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  3 14:15:56 2024

@author: ASUS
"""



'''
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv, DataFrame, concat
from sklearn.preprocessing import MinMaxScaler, LabelEncoder,
StandardScaler, RobustScaler, MaxAbsScaler, QuantileTransformer
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten,
Dropout
from sklearn.metrics import mean_absolute_error,
mean_squared_error, mean_absolute_percentage_error
import datetime

import warnings
warnings.filterwarnings("ignore")

starttime = datetime.datetime.now()
print('開始時間: ', starttime)

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
```

```
	# input sequence (t-n, ...t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
			names += [('var%d(t)' % (j+1)) for j in
range(n_vars)]
		else:
			names += [('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
	# put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	if dropnan:
		agg.dropna(inplace=True)
	return agg

Algorithmtime = datetime.datetime.now()
print("演算時間: ", Algorithmtime)

# 設定檔名
filename = 'pollution_cleaned.csv'
filenameOut = 'pollution_CNN.csv'
filenameModel = 'pollution_CNN.h5'

# load dataset
dataset = read_csv(filename, header=0, index_col=0)
values = dataset.values

# integer encode direction
encoder = LabelEncoder()
values[:, 4] = encoder.fit_transform(values[:, 4])
# ensure all data is float
```

```
values = values.astype('float32')

# normalize features
scaler = StandardScaler()
scaled = scaler.fit_transform(values)

# specify the number of lag hours
n_hours = 3
n_features = 8
# frame as supervised learning
reframed = series_to_supervised(scaled, n_hours, 1)

# split into train and test sets
values = reframed.values
n_train_hours = len(values) - 7 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]

# split into input and outputs
n_obs = n_hours * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))

# Design network
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu',
input_shape=(n_hours, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dense(1))
```

```
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72,
                                validation_data=(test_X, test_y),
verbose=2, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# Reshape test_X for inverse scaling
test_X = test_X.reshape((test_X.shape[0], n_hours * n_features))

# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:, 0]

# Invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, -(n_features-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:, 0]

# Calculate RMSE, MAE, MAPE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
mae = mean_absolute_error(inv_y, inv_yhat)
mape = mean_absolute_percentage_error(inv_y, inv_yhat) * 100

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

```
print('Test MAPE: %.2f%%' % mape)

# Plot predictions vs actual
pyplot.plot(inv_yhat, 'b:')    # Predicted
pyplot.plot(inv_y, 'r-')    # Actual
pyplot.legend(['predict', 'real'])
pyplot.xlabel("Timesteps")
pyplot.ylabel("PM2.5")
pyplot.title("Prediction (CNN)")
pyplot.show()

# Save model and results
model.save(filenameModel)
print("Model saved as", filenameModel)

# Save predictions and actual values
df_PredResult = DataFrame({"Real": inv_y, "Pred": inv_yhat})
print("Saving results to", filenameOut)
df_PredResult.to_csv(filenameOut)

endtime = datetime.datetime.now()
print('開始時間: ', starttime)
print('演算時間: ', Algorithmtime)
print('結束時間: ', endtime)
print('花費時間: ', endtime - starttime)



'''


from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv, DataFrame, concat
from sklearn.preprocessing import MinMaxScaler, LabelEncoder,
StandardScaler, RobustScaler, MaxAbsScaler, QuantileTransformer
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten,
```

```
Dropout, Embedding
from keras.layers import LSTM
from sklearn.metrics import mean_absolute_error,
mean_squared_error, mean_absolute_percentage_error
import datetime

import warnings
warnings.filterwarnings("ignore")

starttime = datetime.datetime.now()
print('開始時間: ', starttime)

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df = DataFrame(data)
	cols, names = list(), list()
	# input sequence (t-n, ...t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
			names += [('var%d(t)' % (j+1)) for j in
range(n_vars)]
		else:
			names += [('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
	# put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	if dropnan:
		agg.dropna(inplace=True)
	return agg
```

```
Algorithmtime = datetime.datetime.now()
print("演算時間: ", Algorithmtime)

# 設定檔名
filename = 'pollution_cleaned.csv'
filenameOut = 'pollution_CNN.csv'
filenameModel = 'pollution_CNN.h5'

# load dataset
dataset = read_csv(filename, header=0, index_col=0)
values = dataset.values

# integer encode direction
encoder = LabelEncoder()
values[:, 4] = encoder.fit_transform(values[:, 4])
# ensure all data is float
values = values.astype('float32')

# normalize features
scaler = StandardScaler()
scaled = scaler.fit_transform(values)

# specify the number of lag hours
n_hours = 3
n_features = 8
# frame as supervised learning
reframed = series_to_supervised(scaled, n_hours, 1)

# split into train and test sets
values = reframed.values
n_train_hours = len(values) - 7 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]

# split into input and outputs
n_obs = n_hours * n_features
```

```
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))

# Design network (Embedding CNN)
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu',
input_shape=(n_hours, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72,
                                validation_data=(test_X, test_y),
verbose=2, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# Reshape test_X for inverse scaling
test_X = test_X.reshape((test_X.shape[0], n_hours * n_features))

# Invert scaling for forecast
```

```
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:, 0]

# Invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, -(n_features-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:, 0]

# Calculate RMSE, MAE, MAPE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
mae = mean_absolute_error(inv_y, inv_yhat)
mape = mean_absolute_percentage_error(inv_y, inv_yhat) * 100

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
print('Test MAPE: %.2f%%' % mape)

# Plot predictions vs actual
pyplot.plot(inv_yhat, 'b:')    # Predicted
pyplot.plot(inv_y, 'r-')    # Actual
pyplot.legend(['predict', 'real'])
pyplot.xlabel("Timesteps")
pyplot.ylabel("PM2.5")
pyplot.title("Prediction (CNN)")
pyplot.show()

# Save model and results
model.save(filenameModel)
print("Model saved as", filenameModel)

# Save predictions and actual values
df_PredResult = DataFrame({"Real": inv_y, "Pred": inv_yhat})
print("Saving results to", filenameOut)
df_PredResult.to_csv(filenameOut)
```

```
endtime = datetime.datetime.now()
print('開始時間: ', starttime)
print('演算時間: ', Algorithmtime)
print('結束時間: ', endtime)
print('花費時間: ', endtime - starttime)
```

二、 GAN (Generative Adversarial Network)

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 16 11:38:30 2024

@author: ASUS
"""

#GAN (Generative Adversarial Network)

#匯入模組
import numpy as np
import os

#深度學習所需套件
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input,    Dense, Conv2D,
Dropout
from tensorflow.keras.layers import BatchNormalization, Flatten,
Activation
from tensorflow.keras.layers import Reshape, Conv2DTranspose,
UpSampling2D
from tensorflow.keras.optimizers import RMSprop

#繪圖所需套件
import pandas as pd
from matplotlib import pyplot as plt
#%matplotlib inline

#載入資料庫
data_dir = 'D:'
input_images = data_dir + '/apple.npy'
```

```
data = np.load(input_images)

print('shape = ', data.shape)
print('data[4242] = ', data[4242])

#
data = data/255
data = np.reshape(data, (data.shape[0], 28, 28, 1))
img_w, img_h = data.shape[1:3]
print('正規化後，shape = ', data.shape)
#繪圖
plt.imshow(data[4242, :, :, 0], cmap = 'Greys')

#建構鑑別器(discriminator)神經網路
def build_discriminator(depth = 64, p = 0.4):
    #定義輸入
    image = Input((img_w, img_h, 1))
    #卷積層
    conv1 = Conv2D(depth * 1, 5, strides = 2,
                   padding = 'same', activation = 'relu')(image)
    conv1 = Dropout(p)(conv1)
    conv2 = Conv2D(depth * 2, 5, strides = 2,
                   padding = 'same', activation = 'relu')(conv1)
    conv2 = Dropout(p)(conv2)
    conv3 = Conv2D(depth * 4, 5, strides = 2,
                   padding = 'same', activation = 'relu')(conv2)
    conv3 = Dropout(p)(conv3)
    conv4 = Conv2D(depth * 8, 5, strides = 2,
                   padding = 'same', activation = 'relu')(conv3)
    conv4 = Flatten()(Dropout(p)(conv4))
    #輸出層
    prediction = Dense(1, activation = 'sigmoid')(conv4)
    #定義模型
    model = Model(inputs = image, outputs = prediction)

    return model
```

```
discriminator = build_discriminator()
discriminator.summary

discriminator.compile(loss = 'binary_crossentropy',
                      optimizers = RMSprop(lr = 0.0008,
                                           decay = 6e-8,
                                           clipvalue =
1.0),
                      metrics = ['accuracy'])

#建構生成器（generator）神經網路，生成器的架構
z_dimensions = 32

def build_generator(latent_dim = z_dimensions,
                    depth = 64, p = 0.4):
    #定義輸入
    noise = Input((latent_dim,))

    #第一層密集層
    dense1 = Dense(7 * 7 * depth)(noise)
    dense1 = BatchNormalization(momentum = 0.9)(dense1)
    dense1 = Activation(activation = 'relu')(dense1)
    dense1 = Reshape((7, 7, depth))(dense1)
    dense1 = Dropout(p)(dense1)

    #反卷積層
    conv1 = UpSampling2D()(dense1)
    conv1 = Conv2DTranspose(int(depth / 2),
                            kernel_size = 5,
                            padding = 'same',
                            activation = None,)(conv1)
    conv1 = BatchNormalization(momentum = 0.9)(conv1)
    conv1 = Activation(activation = 'relu')(conv1)

    conv2 = UpSampling2D()(conv1)
    conv2 = Conv2DTranspose(int(depth / 4),
                            kernel_size = 5,
```

```
                                    padding = 'same',
                                    activation = None,)(conv2)
    conv2 = BatchNormalization(momentum = 0.9)(conv2)
    conv2 = Activation(activation = 'relu')(conv2)

    conv3 = Conv2DTranspose(int(depth / 8),
                                    kernel_size = 5,
                                    padding = 'same',
                                    activation = None,)(conv2)
    conv3 = BatchNormalization(momentum = 0.9)(conv3)
    conv3 = Activation(activation = 'relu')(conv3)

    #輸出層
    image = Conv2D(1, kernel_size = 5, padding = 'same',
                        activation = 'sigmoid')(conv3)

    #定義模型
    model = Model(inputs = noise, outputs = image)

    return model

generator = build_generator()
generator.summary()

#結合生成器與鑑別器，建構對抗式生成網路，建立對抗式神經網路
z = Input(shape = (z_dimensions, ))
img = generator(z)

discriminator.trainable = False
pred = discriminator(img)
adversarial_model = Model(z, pred)

#編譯模型
adversarial_model.compile(loss = 'binary_crossentropy',
                                    optimizer =
RMSprop(learning_rate = 0.0004,
```

```
                                                    decay =
3e-8,
                                                    clipvalue
= 1.0),
                                metrics = ['accuracy'])

#訓練 GAN
#定義 train()函式並進行 GAN 的訓練
def train(train_round = 2000, batch = 1289, z_dim =
z_dimensions):

    d_metrics = []
    a_metrics = []

    running_d_loss = 0
    running_d_acc = 0
    running_a_loss = 0
    running_a_acc = 0

    for i in range(train_round):

        #從真影像資料集中取樣：
        real_imgs = np.reshape(
            data[np.random.choise(data.shape[0],
                                  batch,
                                  replace = False)],
            (batch, 28, 28, 1))

        #生成假影像：
        fake_imgs = generator.predict(
            np.random.uniform(-1.0, -1.0,
                              size = [batch, z_dim]))

        #將真假影像串起來，方便一併輸入鑑別器：
        x = np.concatenate((real_imgs, fake_imgs))

        #標籤 y，提供給鑑別器：
```

```
        y = np.ones([2 * batch, 1])
        y[batch:, :] = 0

        #訓練鑑別器：
        d_metrics.append(
            discriminator.train_on_batch(x, y)
        )
        running_d_loss += d_metrics[-1][0]
        running_d_acc += d_metrics[-1][1]

        #設定對抗式神經網路的輸入雜訊與標籤 y
        #（生成式希望鑑別器能誤判為真，所以 y = 1）：
        noise = np.random.uniform(-1.0, 1.0,
                                  size = [batch, z_dim])
        y = np.ones([batch, 1])

        #訓練對抗式神經網路：
        a_metrics.append(
            adversarial_model.train_on_batch(noise, y)
        )
        running_a_loss += a_metrics[-1][0]
        running_a_acc += a_metrics[-1][1]

        #定時顯示進度與生成影像：
        if (i + 1) % 100 == 0:

            print('train_round #{}', format(i))
            log_mesg = '%d: [D loss: %f, acc: %f]' % \
                (i, running_d_loss/i, running_d_acc/i)
            log_mesg = '%s [A loss: %f, acc: %f]' % \
                (log_mesg, running_a_loss/i, running_a_acc/i)
            print(log_mesg)

            noise = np.random.uniform(-1.0, 1.0,
                                      size = [16, z_dim])
            gen_imgs = generator.predict(noise)
```

```
                plt.figure(figsize = (5, 5))

                for k in range(gen_imgs.shape[0]):
                    plt.subplot(4, 4, k+1)
                    plt.imshow(gen_imgs[k, :, :, 0],
                               cmap = 'gray')
                    plt.axis('off')

                plt.tight_layout()
                plt.show()

        return a_metrics, d_metrics

#觀察 GAN 的訓練狀況
a_metrics_complete, d_metrics_complete = train()

ax = pd.DataFrame(
        {
         'Generator': [metric[0] for metric in a_metrics_complete],
         'Discriminator': [metric[0] for metric in d_metrics_complete],
         }
).plot(title = 'Training Loss', logy = True)
ax.set_xlabel ("train_round")
ax.set_ylabel("Loss")

ax = pd.DataFrame(
        {
         'Generator': [metric[1] for metric in a_metrics_complete],
         'Discriminator': [metric[1] for metric in d_metrics_complete],
         }
).plot(title = 'Training Accuracy')
ax.set_xlabel ("train_round")
ax.set_ylabel("Accuracy")
```

三、 MLP_IMDb_CNN_embed

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 10:21:36 2024
```

```
@author: ASUS
"""

from tensorflow.keras.datasets import imdb #new!
from tensorflow.keras.preprocessing.sequence import
pad_sequences
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Flatten, Dropout,
Embedding, Input, concatenate, SpatialDropout1D, Conv1D,
GlobalMaxPooling1D
from tensorflow.keras.callbacks import ModelCheckpoint #new!
import os #news!
from sklearn.metrics import roc_auc_score, roc_curve
#import pandas as pd
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

'''
#載入 IMDb 影評資料集(原始資料)
(x_train, t_train), (X_test, y_test) = imdb.load_data()
#查看 IMDb 資料集的內容
for x in x_train[0:6]:
      print(len(x))
#查看每筆資料內容
x_train[0:6]

word_index = imdb.get_word_index()
word_index = {k : (v+3) for k, v in word_index.items()}
word_index['START'] = 1
index_word = {v:k for k, v in word_index.items()}

text = ' '.join(index_word[id] for id in x_train[0])
print(text)
'''
```

```
#正式模型起點
#載入 IMDb 影評資料集，依據資料預處理參數
#預處理（一）：移除停用字（stop words）
#資料預處理參數
#n_words_to_skip = 50
max_review_length = 400
pad_type = 'pre' #字數太少前面補 0
trunc_type = 'pre' #字數太多刪除前面
drop_embed = 0.2

n_unique_words = 5000
n_dim = 64

n_conv_1 = n_conv_2 = n_conv_3 = 256 #濾鏡個數
#濾鏡大小
k_conv_1 = 3
k_conv_2 = 2
k_conv_3 = 4

n_dense = 256
dropout = 0.2

epochs = 2
batch_size = 128

#詞向量空間
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words =
n_unique_words)

#print('案例數（訓練集）=', len(y_train))
#print('案例數（測試集）=', len(y_test))

#print(x_train[0:6])

#預處理（二）：統一影評長度
x_train = pad_sequences(x_train, maxlen = max_review_length,
```

```
                                    padding = pad_type, truncating =
trunc_type, value = 0)

x_test = pad_sequences(x_test, maxlen = max_review_length,
                                    padding = pad_type, truncating =
trunc_type, value = 0)

'''
for x in x_train[0:6]:
      print(len(x))
print(x_train[5])
'''
input_layer = Input(shape = (max_review_length, ), dtype = 'int16',
name = 'input')
embedding_layer = Embedding(n_unique_words, n_dim, name =
'embedding')(input_layer)
drop_embed_layer = SpatialDropout1D(drop_embed, name =
'drop_embed')(embedding_layer)

conv_1 = Conv1D(n_conv_1, k_conv_1, activation = 'relu', name =
'conv_1')(drop_embed_layer)
maxp_1 = GlobalMaxPooling1D(name = 'maxp_1')(conv_1)
conv_2 = Conv1D(n_conv_2, k_conv_2, activation = 'relu', name =
'conv_2')(drop_embed_layer)
maxp_2 = GlobalMaxPooling1D(name = 'maxp_2')(conv_2)
conv_3 = Conv1D(n_conv_3, k_conv_3, activation = 'relu', name =
'conv_3')(drop_embed_layer)
maxp_3 = GlobalMaxPooling1D(name = 'maxp_3')(conv_3)

concat = concatenate([maxp_1, maxp_2, maxp_3])

dense_layer = Dense(n_dense, activation = 'relu', name =
'dense')(concat)
drop_dense_layer = Dropout(dropout, name =
'drop_dense')(dense_layer)
dense_2 = Dense(int(n_dense/4), activation = 'relu', name =
'dense_2')(drop_dense_layer)
```

```
dropout_2 = Dropout(dropout, name = 'drop_dense_2')(dense_2)

predictions = Dense(1, activation = 'sigmoid', name =
'output')(dropout_2)

model = Model(input_layer, predictions)

#輸出模型
print(model.summary())

#編譯模型
model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

#設定每周期結束紀錄模型參數
output_dir = 'E:/學校/國立彰化師範大學/大三上/人工智慧
/20241216/weight'
modelcheckpoint = ModelCheckpoint(filepath = output_dir +
'/weightsCNNembed.{epoch:02d}.keras')

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨
機性，
#因此底下(或您重跑一次)的結果不會與書中完全一样,但模型
的能力是相近的
model.fit(x_train, y_train,
          batch_size=batch_size, epochs=epochs, verbose=1,
          validation_data=(x_test,y_test),
          callbacks=[modelcheckpoint])
#模型評估(一):挑幾筆來觀察
model.load_weights(output_dir+ "/weightsCNNembed.02.keras")#
請視以上執行結果指定指定較佳的權重
y_hat = model.predict(x_test)

print('y_hat[0]',y_hat[0])
```

```
y_hat[0]

#模型評估(二):繪製直方圖
plt.hist(y_hat)
_ = plt.axvline(x = 0.5, color = 'orange')

#模型評估(三):ROC AUC  分數
#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型的能力是相近的
pct_auc = roc_auc_score(y_test, y_hat)*100.0
"{:0.2f}".format(pct_auc)
print('ROC AUC  分數=', pct_auc)

################################################
#新增程式碼 provided by  翁政雄老師###############
################################################

pred = y_hat #以下用 pred 代替 y_hat

'''
#將預測結果(0~1)轉成 0/1，即 pred 只能是 0 或 1
#方法(1)
import numpy as np
pred = np.where(y_hat > 0.5, 1, 0)
#方法(2)
pred = y_hat.round(0)
print(pred)
'''

#Compute ROC curve and ROC area for each class
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, threshold = roc_curve(y_test, pred)
roc_auc = auc(fpr, tpr)
```

```
plt.figure()
lw = 2
#plt.figure(figsize = (10, 10))
plt.plot(fpr, tpr, color = 'darkorange',
         lw = lw, label = 'ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color = 'navy', lw = lw, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('FPR')
plt.ylabel('TFR')
plt.title('ROC Curve')

#plt.legend(loc = 'lower right')
plt.legend()
plt.show()
print('ROC_auc area = %.4f' % (roc_auc))

from sklearn.metrics import precision_recall_curve
#lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_probs)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)

#lr_f1, lr_auc = f1_score(y_test, pred), auc(lr_recall, lr_precision)
lr_auc = auc(lr_recall, lr_precision)
#summarize scores
#print('MLP(ANN): f1 = %.3f PRC_auc = %.3f' % (lr_f1, lr_auc))
print('PRC_auc area = %.4f' % (lr_auc))

#plot the precision-recall curves
no_skill = len(y_test[y_test == 1]) / len(y_test)

#plt.figure(figsize=(10, 10))
#plt.plot([0, 1], [no_skill, no_skill], linestyle = '--', label = 'No
Skill')
plt.plot([0, 1], [1, 0], color = 'navy', lw = lw, linestyle = '--')
plt.plot(lr_recall, lr_precision, color = 'darkorange',
         lw = lw, label = 'PRC curve (area = %.4f)' % lr_auc)
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])

#axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
#show the legend
plt.legend()
#show the plot
plt.show()

#輸出混亂矩陣，顯示準確率
pred = y_hat.round(0)

from sklearn.metrics import confusion_matrix, classification_report
print('輸出混亂矩陣，顯示準確率：使用訓練資料')
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

from pandas import DataFrame
df = DataFrame({'Real': y_test, 'Pred': pred.flatten()})
filenameReport = output_dir + '/DNN_IMDb.csv'
print('輸出至檔案', filenameReport)
df.to_csv(filenameReport, index = True)
```

四、 MLP_IMDb_CNN_GRU_embed

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 10:21:36 2024

@author: ASUS
"""

from tensorflow.keras.datasets import imdb #new!
from tensorflow.keras.preprocessing.sequence import
pad_sequences
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Flatten, Dropout,
```

```
Embedding, Input, concatenate, SpatialDropout1D, Conv1D,
GlobalMaxPooling1D, LSTM, GRU
from tensorflow.keras.callbacks import ModelCheckpoint #new!
import os #news!
from sklearn.metrics import roc_auc_score, roc_curve
#import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf

import warnings
warnings.filterwarnings('ignore')

'''
#載入 IMDb 影評資料集(原始資料)
(x_train, t_train), (X_test, y_test) = imdb.load_data()
#查看 IMDb 資料集的內容
for x in x_train[0:6]:
      print(len(x))
#查看每筆資料內容
x_train[0:6]

word_index = imdb.get_word_index()
word_index = {k : (v+3) for k, v in word_index.items()}
word_index['START'] = 1
index_word = {v:k for k, v in word_index.items()}

text = ' '.join(index_word[id] for id in x_train[0])
print(text)
'''

#正式模型起點
#載入 IMDb 影評資料集，依據資料預處理參數
#預處理（一）：移除停用字（stop words）
#資料預處理參數
#n_words_to_skip = 50
max_review_length = 400
pad_type = 'pre' #字數太少前面補 0
```

```
trunc_type = 'pre' #字數太多刪除前面
drop_embed = 0.2

n_unique_words = 5000
n_dim = 64

n_conv_1 = n_conv_2 = n_conv_3 = 256 #濾鏡個數
#濾鏡大小
k_conv_1 = 3
k_conv_2 = 2
k_conv_3 = 4

n_dense = 256
dropout = 0.2

epochs = 2
batch_size = 128

n_gru = 50

#詞向量空間
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words =
n_unique_words)

#print('案例數（訓練集）=', len(y_train))
#print('案例數（測試集）=', len(y_test))

#print(x_train[0:6])

#預處理（二）：統一影評長度
x_train = pad_sequences(x_train, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

x_test = pad_sequences(x_test, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)
```

```
'''
for x in x_train[0:6]:
      print(len(x))
print(x_train[5])
'''
input_layer = Input(shape = (max_review_length, ), dtype = 'int16',
name = 'input')
embedding_layer = Embedding(n_unique_words, n_dim, name =
'embedding')(input_layer)
drop_embed_layer = SpatialDropout1D(drop_embed, name =
'drop_embed')(embedding_layer)

conv_1 = Conv1D(n_conv_1, k_conv_1, activation = 'relu', name =
'conv_1')(drop_embed_layer)
maxp_1 = GlobalMaxPooling1D(name = 'maxp_1')(conv_1)
conv_2 = Conv1D(n_conv_2, k_conv_2, activation = 'relu', name =
'conv_2')(drop_embed_layer)
maxp_2 = GlobalMaxPooling1D(name = 'maxp_2')(conv_2)
conv_3 = Conv1D(n_conv_3, k_conv_3, activation = 'relu', name =
'conv_3')(drop_embed_layer)
maxp_3 = GlobalMaxPooling1D(name = 'maxp_3')(conv_3)

concat = concatenate([maxp_1, maxp_2, maxp_3])

# 新增 Reshape 層將二維輸出轉換回三維（為了輸入到
LSTM）
reshape_layer = Dense(n_gru * 2)(concat)  # 增加特徵數以適配
LSTM
reshape_layer = tf.expand_dims(reshape_layer, axis=1)  # 插入時
間步維度

# LSTM 層
gru_layer = GRU(n_gru, dropout=dropout, recurrent_dropout=0.2,
return_sequences=False, name='lstm')(reshape_layer)
#lstm_layer_2 = LSTM(n_lstm, dropout=dropout,
recurrent_dropout=0.2, return_sequences=False,
```

```
name='lstm')(lstm_layer)

dense_layer = Dense(n_dense, activation = 'relu', name =
'dense')(gru_layer)
drop_dense_layer = Dropout(dropout, name =
'drop_dense')(dense_layer)
dense_2 = Dense(int(n_dense/4), activation = 'relu', name =
'dense2')(drop_dense_layer)
dropout_2 = Dropout(dropout, name = 'drop_dense_2')(dense_2)

predictions = Dense(1, activation = 'sigmoid', name =
'output')(dropout_2)

model = Model(input_layer, predictions)

#輸出模型
print(model.summary())

#編譯模型
model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

#設定每周期結束紀錄模型參數
output_dir = 'E:/學校/國立彰化師範大學/大三上/人工智慧
/20241216/weight'
modelcheckpoint = ModelCheckpoint(filepath = output_dir +
'/weightsCNNembed.{epoch:02d}.keras')

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨
機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型
的能力是相近的
model.fit(x_train, y_train,
          batch_size=batch_size, epochs=epochs, verbose=1,
```

```
                validation_data=(x_test,y_test),
                callbacks=[modelcheckpoint])
#模型評估(一):挑幾筆來觀察
model.load_weights(output_dir+ "/weightsCNNembed.02.keras")#請視以上執行結果指定指定較佳的權重
y_hat = model.predict(x_test)

print('y_hat[0]',y_hat[0])
y_hat[0]

#模型評估(二):繪製直方圖
plt.hist(y_hat)
_ = plt.axvline(x = 0.5, color = 'orange')

#模型評估(三):ROC AUC  分數
#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型的能力是相近的
pct_auc = roc_auc_score(y_test, y_hat)*100.0
"{:0.2f}".format(pct_auc)
print('ROC AUC  分數=', pct_auc)

################################################
#新增程式碼 provided by  翁政雄老師###############
################################################

pred = y_hat #以下用 pred 代替 y_hat

'''
#將預測結果(0~1)轉成 0/1，即 pred 只能是 0 或 1
#方法(1)
import numpy as np
pred = np.where(y_hat > 0.5, 1, 0)
#方法(2)
pred = y_hat.round(0)
print(pred)
```

```
'''

#Compute ROC curve and ROC area for each class
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, threshold = roc_curve(y_test, pred)
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
#plt.figure(figsize = (10, 10))
plt.plot(fpr, tpr, color = 'darkorange',
         lw = lw, label = 'ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color = 'navy', lw = lw, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('FPR')
plt.ylabel('TFR')
plt.title('ROC Curve')

#plt.legend(loc = 'lower right')
plt.legend()
plt.show()
print('ROC_auc area = %.4f' % (roc_auc))

from sklearn.metrics import precision_recall_curve
#lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_probs)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)

#lr_f1, lr_auc = f1_score(y_test, pred), auc(lr_recall, lr_precision)
lr_auc = auc(lr_recall, lr_precision)
#summarize scores
#print('MLP(ANN): f1 = %.3f PRC_auc = %.3f' % (lr_f1, lr_auc))
print('PRC_auc area = %.4f' % (lr_auc))

#plot the precision-recall curves
no_skill = len(y_test[y_test == 1]) / len(y_test)
```

```
#plt.figure(figsize=(10, 10))
#plt.plot([0, 1], [no_skill, no_skill], linestyle = '--', label = 'No
Skill')
plt.plot([0, 1], [1, 0], color = 'navy', lw = lw, linestyle = '--')
plt.plot(lr_recall, lr_precision, color = 'darkorange',
         lw = lw, label = 'PRC curve (area = %.4f)' % lr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

#axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
#show the legend
plt.legend()
#show the plot
plt.show()

#輸出混亂矩陣，顯示準確率
pred = y_hat.round(0)

from sklearn.metrics import confusion_matrix, classification_report
print('輸出混亂矩陣，顯示準確率：使用訓練資料')
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

from pandas import DataFrame
df = DataFrame({'Real': y_test, 'Pred': pred.flatten()})
filenameReport = output_dir + '/DNN_IMDb.csv'
print('輸出至檔案', filenameReport)
df.to_csv(filenameReport, index = True)
```

五、 MLP_IMDb_CNN_LSTM_embed

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 10:21:36 2024
```

```
@author: ASUS
"""

from tensorflow.keras.datasets import imdb #new!
from tensorflow.keras.preprocessing.sequence import
pad_sequences
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Flatten, Dropout,
Embedding, Input, concatenate, SpatialDropout1D, Conv1D,
GlobalMaxPooling1D, LSTM
from tensorflow.keras.callbacks import ModelCheckpoint #new!
import os #news!
from sklearn.metrics import roc_auc_score, roc_curve
#import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf

import warnings
warnings.filterwarnings('ignore')

'''
#載入 IMDb 影評資料集(原始資料)
(x_train, t_train), (X_test, y_test) = imdb.load_data()
#查看 IMDb 資料集的內容
for x in x_train[0:6]:
      print(len(x))
#查看每筆資料內容
x_train[0:6]

word_index = imdb.get_word_index()
word_index = {k : (v+3) for k, v in word_index.items()}
word_index['START'] = 1
index_word = {v:k for k, v in word_index.items()}

text = ' '.join(index_word[id] for id in x_train[0])
print(text)
'''
```

```
#正式模型起點
#載入 IMDb 影評資料集，依據資料預處理參數
#預處理（一）：移除停用字（stop words）
#資料預處理參數
#n_words_to_skip = 50
max_review_length = 400
pad_type = 'pre' #字數太少前面補 0
trunc_type = 'pre' #字數太多刪除前面
drop_embed = 0.2

n_unique_words = 5000
n_dim = 64

n_conv_1 = n_conv_2 = n_conv_3 = 256 #濾鏡個數
#濾鏡大小
k_conv_1 = 3
k_conv_2 = 2
k_conv_3 = 4

n_dense = 256
dropout = 0.2

epochs = 2
batch_size = 128

n_lstm = 50

#詞向量空間
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words =
n_unique_words)

#print('案例數（訓練集）=', len(y_train))
#print('案例數（測試集）=', len(y_test))

#print(x_train[0:6])
```

```
#預處理（二）：統一影評長度
x_train = pad_sequences(x_train, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

x_test = pad_sequences(x_test, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

'''
for x in x_train[0:6]:
      print(len(x))
print(x_train[5])
'''
input_layer = Input(shape = (max_review_length, ), dtype = 'int16',
name = 'input')
embedding_layer = Embedding(n_unique_words, n_dim, name =
'embedding')(input_layer)
drop_embed_layer = SpatialDropout1D(drop_embed, name =
'drop_embed')(embedding_layer)

conv_1 = Conv1D(n_conv_1, k_conv_1, activation = 'relu', name =
'conv_1')(drop_embed_layer)
maxp_1 = GlobalMaxPooling1D(name = 'maxp_1')(conv_1)
conv_2 = Conv1D(n_conv_2, k_conv_2, activation = 'relu', name =
'conv_2')(drop_embed_layer)
maxp_2 = GlobalMaxPooling1D(name = 'maxp_2')(conv_2)
conv_3 = Conv1D(n_conv_3, k_conv_3, activation = 'relu', name =
'conv_3')(drop_embed_layer)
maxp_3 = GlobalMaxPooling1D(name = 'maxp_3')(conv_3)

concat = concatenate([maxp_1, maxp_2, maxp_3])

# 新增 Reshape 層將二維輸出轉換回三維（為了輸入到
LSTM）
reshape_layer = Dense(n_lstm * 2)(concat)   # 增加特徵數以適配
LSTM
```

```
reshape_layer = tf.expand_dims(reshape_layer, axis=1)  # 插入時間步維度

# LSTM 層
lstm_layer = LSTM(n_lstm, dropout=dropout,
recurrent_dropout=0.2, return_sequences=False,
name='lstm')(reshape_layer)
#lstm_layer_2 = LSTM(n_lstm, dropout=dropout,
recurrent_dropout=0.2, return_sequences=False,
name='lstm')(lstm_layer)

dense_layer = Dense(n_dense, activation = 'relu', name =
'dense')(lstm_layer)
drop_dense_layer = Dropout(dropout, name =
'drop_dense')(dense_layer)
dense_2 = Dense(int(n_dense/4), activation = 'relu', name =
'dense_2')(drop_dense_layer)
dropout_2 = Dropout(dropout, name = 'drop_dense_2')(dense_2)

predictions = Dense(1, activation = 'sigmoid', name =
'output')(dropout_2)

model = Model(input_layer, predictions)

#輸出模型
print(model.summary())

#編譯模型
model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

#設定每周期結束紀錄模型參數
output_dir = 'E:/學校/國立彰化師範大學/大三上/人工智慧
/20241216/weight'
modelcheckpoint = ModelCheckpoint(filepath = output_dir +
'/weightsCNNembed.{epoch:02d}.keras')
```

```
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型的能力是相近的
model.fit(x_train, y_train,
          batch_size=batch_size, epochs=epochs, verbose=1,
          validation_data=(x_test,y_test),
          callbacks=[modelcheckpoint])
#模型評估(一):挑幾筆來觀察
model.load_weights(output_dir+ "/weightsCNNembed.02.keras")#請視以上執行结果指定指定較佳的權重
y_hat = model.predict(x_test)

print('y_hat[0]',y_hat[0])
y_hat[0]

#模型評估(二):繪製直方圖
plt.hist(y_hat)
_ = plt.axvline(x = 0.5, color = 'orange')

#模型評估(三):ROC AUC  分數
#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型的能力是相近的
pct_auc = roc_auc_score(y_test, y_hat)*100.0
"{:0.2f}".format(pct_auc)
print('ROC AUC  分數=', pct_auc)

###############################################
#新增程式碼 provided by  翁政雄老師##############
###############################################

pred = y_hat #以下用 pred 代替 y_hat
```

```
'''
#將預測結果(0~1)轉成 0/1，即 pred 只能是 0 或 1
#方法(1)
import numpy as np
pred = np.where(y_hat > 0.5, 1, 0)
#方法(2)
pred = y_hat.round(0)
print(pred)
'''

#Compute ROC curve and ROC area for each class
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, threshold = roc_curve(y_test, pred)
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
#plt.figure(figsize = (10, 10))
plt.plot(fpr, tpr, color = 'darkorange',
         lw = lw, label = 'ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color = 'navy', lw = lw, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('FPR')
plt.ylabel('TFR')
plt.title('ROC Curve')

#plt.legend(loc = 'lower right')
plt.legend()
plt.show()
print('ROC_auc area = %.4f' % (roc_auc))

from sklearn.metrics import precision_recall_curve
#lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_probs)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)
```

```
#lr_f1, lr_auc = f1_score(y_test, pred), auc(lr_recall, lr_precision)
lr_auc = auc(lr_recall, lr_precision)
#summarize scores
#print('MLP(ANN): f1 = %.3f PRC_auc = %.3f' % (lr_f1, lr_auc))
print('PRC_auc area = %.4f' % (lr_auc))

#plot the precision-recall curves
no_skill = len(y_test[y_test == 1]) / len(y_test)

#plt.figure(figsize=(10, 10))
#plt.plot([0, 1], [no_skill, no_skill], linestyle = '--', label = 'No
Skill')
plt.plot([0, 1], [1, 0], color = 'navy', lw = lw, linestyle = '--')
plt.plot(lr_recall, lr_precision, color = 'darkorange',
            lw = lw, label = 'PRC curve (area = %.4f)' % lr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

#axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
#show the legend
plt.legend()
#show the plot
plt.show()

#輸出混亂矩陣，顯示準確率
pred = y_hat.round(0)

from sklearn.metrics import confusion_matrix, classification_report
print('輸出混亂矩陣，顯示準確率：使用訓練資料')
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

from pandas import DataFrame
```

```
df = DataFrame({'Real': y_test, 'Pred': pred.flatten()})
filenameReport = output_dir + '/DNN_IMDb.csv'
print('輸出至檔案', filenameReport)
df.to_csv(filenameReport, index = True)
```

六、 MLP_IMDb_GRU

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 10:21:36 2024

@author: ASUS
"""

from tensorflow.keras.datasets import imdb #new!
from tensorflow.keras.preprocessing.sequence import
pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout,
Conv1D, Embedding, SpatialDropout1D, GlobalMaxPooling1D,
LSTM, GRU
from tensorflow.keras.callbacks import ModelCheckpoint #new!
import os #news!
from sklearn.metrics import roc_auc_score, roc_curve
#import pandas as pd
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

'''
#載入 IMDb 影評資料集(原始資料)
(x_train, t_train), (X_test, y_test) = imdb.load_data()
#查看 IMDb 資料集的內容
for x in x_train[0:6]:
    print(len(x))
#查看每筆資料內容
x_train[0:6]
```

```
word_index = imdb.get_word_index()
word_index = {k : (v+3) for k, v in word_index.items()}
word_index['START'] = 1
index_word = {v:k for k, v in word_index.items()}

text = ' '.join(index_word[id] for id in x_train[0])
print(text)
'''

#正式模型起點
#載入 IMDb 影評資料集，依據資料預處理參數
#預處理（一）：移除停用字（stop words）
#資料預處理參數
#n_words_to_skip = 50
max_review_length = 400
pad_type = 'pre' #字數太少前面補 0
trunc_type = 'pre' #字數太多刪除前面
n_unique_words = 5000
n_dim = 64
#訓練神經網路 DNN
n_dense = 256
dropout = 0.2
epochs = 1
batch_size = 128
drop_embed = 0.2
n_conv = 256
k_conv = 3

n_gru = 50
drop_gru = 0.2



#詞向量空間
(x_train, y_train), (x_test, y_test) = \
imdb.load_data(num_words = n_unique_words)
```

```
'''
print('案例數（訓練集）=', len(y_train))
print('案例數（測試集）=', len(y_test))
'''

#print(x_train[0:6])

#預處理（二）：統一影評長度
x_train = pad_sequences(x_train, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

x_test = pad_sequences(x_test, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

'''
for x in x_train[0:6]:
      print(len(x))
print(x_train[5])
'''
model = Sequential()
model.add(Embedding(n_unique_words, n_dim, input_length =
max_review_length))
model.add(SpatialDropout1D(drop_embed))
model.add(GRU(n_gru, dropout = drop_gru))
#model.add(Dense(n_dense, activation = 'relu'))
#model.add(Dropout(dropout))
model.add(Dense(1, activation = 'sigmoid'))

#輸出模型
print(model.summary())

#編譯模型
model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])
```

```
#設定每周期結束紀錄模型參數
output_dir = 'E:/學校/國立彰化師範大學/大三上/人工智慧
/20241209/weight'
modelcheckpoint = ModelCheckpoint(filepath = output_dir +
'/weightsGRU.{epoch:02d}.keras')

if not os.path.exists(output_dir):
      os.makedirs(output_dir)

#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨
機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型
的能力是相近的
model.fit(x_train, y_train,
              batch_size=batch_size, epochs=epochs, verbose=1,
              validation_data=(x_test,y_test),
              callbacks=[modelcheckpoint])
#模型評估(一):挑幾筆來觀察
model.load_weights(output_dir+ "/weightsGRU.01.keras")#請視以
上執行結果指定指定較佳的權重
y_hat = model.predict(x_test)

print('y_hat[0]',y_hat[0])
y_hat[0]

#模型評估(二):繪製直方圖
plt.hist(y_hat)
_ = plt.axvline(x = 0.5, color = 'orange')

#模型評估(三):ROC AUC  分數
#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨
機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型
的能力是相近的
pct_auc = roc_auc_score(y_test, y_hat)*100.0
"{:0.2f}".format(pct_auc)
print('ROC AUC  分數=', pct_auc)
```

```
##############################################
#新增程式碼 provided by 翁政雄老師##############
##############################################

pred = y_hat #以下用 pred 代替 y_hat

'''
#將預測結果(0~1)轉成 0/1，即 pred 只能是 0 或 1
#方法(1)
import numpy as np
pred = np.where(y_hat > 0.5, 1, 0)
#方法(2)
pred = y_hat.round(0)
print(pred)
'''

#Compute ROC curve and ROC area for each class
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, threshold = roc_curve(y_test, pred)
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
#plt.figure(figsize = (10, 10))
plt.plot(fpr, tpr, color = 'darkorange',
         lw = lw, label = 'ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color = 'navy', lw = lw, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('FPR')
plt.ylabel('TFR')
plt.title('ROC Curve')

#plt.legend(loc = 'lower right')
plt.legend()
```

```
plt.show()
print('ROC_auc area = %.4f' % (roc_auc))

from sklearn.metrics import precision_recall_curve
#lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_probs)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)

#lr_f1, lr_auc = f1_score(y_test, pred), auc(lr_recall, lr_precision)
lr_auc = auc(lr_recall, lr_precision)
#summarize scores
#print('MLP(ANN): f1 = %.3f PRC_auc = %.3f' % (lr_f1, lr_auc))
print('PRC_auc area = %.4f' % (lr_auc))

#plot the precision-recall curves
no_skill = len(y_test[y_test == 1]) / len(y_test)

#plt.figure(figsize=(10, 10))
#plt.plot([0, 1], [no_skill, no_skill], linestyle = '--', label = 'No
Skill')
plt.plot([0, 1], [1, 0], color = 'navy', lw = lw, linestyle = '--')
plt.plot(lr_recall, lr_precision, color = 'darkorange',
         lw = lw, label = 'PRC curve (area = %.4f)' % lr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

#axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
#show the legend
plt.legend()
#show the plot
plt.show()

#輸出混亂矩陣，顯示準確率
pred = y_hat.round(0)
```

```
from sklearn.metrics import confusion_matrix, classification_report
print('輸出混亂矩陣，顯示準確率：使用訓練資料')
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

from pandas import DataFrame
df = DataFrame({'Real': y_test, 'Pred': pred.flatten()})
filenameReport = output_dir + '/DNN_IMDb.csv'
print('輸出至檔案', filenameReport)
df.to_csv(filenameReport, index = True)
```

七、 MLP_IMDb_LSTM

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 10:21:36 2024

@author: ASUS
"""

from tensorflow.keras.datasets import imdb #new!
from tensorflow.keras.preprocessing.sequence import
pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout,
Conv1D, Embedding, SpatialDropout1D, GlobalMaxPooling1D,
LSTM
from tensorflow.keras.callbacks import ModelCheckpoint #new!
import os #news!
from sklearn.metrics import roc_auc_score, roc_curve
#import pandas as pd
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

'''
#載入 IMDb 影評資料集(原始資料)
(x_train, t_train), (X_test, y_test) = imdb.load_data()
```

```
#查看 IMDb 資料集的內容
for x in x_train[0:6]:
      print(len(x))
#查看每筆資料內容
x_train[0:6]

word_index = imdb.get_word_index()
word_index = {k : (v+3) for k, v in word_index.items()}
word_index['START'] = 1
index_word = {v:k for k, v in word_index.items()}

text = ' '.join(index_word[id] for id in x_train[0])
print(text)
'''

#正式模型起點
#載入 IMDb 影評資料集，依據資料預處理參數
#預處理（一）：移除停用字（stop words）
#資料預處理參數
#n_words_to_skip = 50
max_review_length = 400
pad_type = 'pre' #字數太少前面補 0
trunc_type = 'pre' #字數太多刪除前面
n_unique_words = 5000
n_dim = 64
#訓練神經網路 DNN
n_dense = 256
dropout = 0.2
epochs = 1
batch_size = 128
drop_embed = 0.2
n_conv = 256
k_conv = 3

n_lstm = 50
drop_lstm = 0.2
```

```
#詞向量空間
(x_train, y_train), (x_test, y_test) = \
imdb.load_data(num_words = n_unique_words)

'''
print('案例數（訓練集）=', len(y_train))
print('案例數（測試集）=', len(y_test))
'''

#print(x_train[0:6])

#預處理（二）：統一影評長度
x_train = pad_sequences(x_train, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

x_test = pad_sequences(x_test, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

'''
for x in x_train[0:6]:
      print(len(x))
print(x_train[5])
'''
model = Sequential()
model.add(Embedding(n_unique_words, n_dim, input_length =
max_review_length))
model.add(SpatialDropout1D(drop_embed))
model.add(LSTM(n_lstm, dropout = drop_lstm))
#model.add(Dense(n_dense, activation = 'relu'))
#model.add(Dropout(dropout))
model.add(Dense(1, activation = 'sigmoid'))

#輸出模型
```

```
print(model.summary())

#編譯模型
model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

#設定每周期結束紀錄模型參數
output_dir = 'E:/學校/國立彰化師範大學/大三上/人工智慧
/20241209/weight'
modelcheckpoint = ModelCheckpoint(filepath = output_dir +
'/weightsLSTM.{epoch:02d}.keras')

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨
機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型
的能力是相近的
model.fit(x_train, y_train,
          batch_size=batch_size, epochs=epochs, verbose=1,
          validation_data=(x_test,y_test),
          callbacks=[modelcheckpoint])
#模型評估(一):挑幾筆來觀察
model.load_weights(output_dir+ "/weightsLSTM.01.keras")#請視
以上執行結果指定指定較佳的權重
y_hat = model.predict(x_test)

print('y_hat[0]',y_hat[0])
y_hat[0]

#模型評估(二):繪製直方圖
plt.hist(y_hat)
_ = plt.axvline(x = 0.5, color = 'orange')

#模型評估(三):ROC AUC  分數
#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨
```

```
機性，
#因此底下(或您重跑一次)的結果不會與書中完全一样,但模型的能力是相近的
pct_auc = roc_auc_score(y_test, y_hat)*100.0
"{:0.2f}".format(pct_auc)
print('ROC AUC  分數=', pct_auc)

################################################
#新增程式碼 provided by 翁政雄老師##############
################################################

pred = y_hat #以下用 pred 代替 y_hat

'''
#將預測結果(0~1)轉成 0/1，即 pred 只能是 0 或 1
#方法(1)
import numpy as np
pred = np.where(y_hat > 0.5, 1, 0)
#方法(2)
pred = y_hat.round(0)
print(pred)
'''

#Compute ROC curve and ROC area for each class
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, threshold = roc_curve(y_test, pred)
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
#plt.figure(figsize = (10, 10))
plt.plot(fpr, tpr, color = 'darkorange',
         lw = lw, label = 'ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color = 'navy', lw = lw, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('FPR')
plt.ylabel('TFR')
plt.title('ROC Curve')

#plt.legend(loc = 'lower right')
plt.legend()
plt.show()
print('ROC_auc area = %.4f' % (roc_auc))

from sklearn.metrics import precision_recall_curve
#lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_probs)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)

#lr_f1, lr_auc = f1_score(y_test, pred), auc(lr_recall, lr_precision)
lr_auc = auc(lr_recall, lr_precision)
#summarize scores
#print('MLP(ANN): f1 = %.3f PRC_auc = %.3f' % (lr_f1, lr_auc))
print('PRC_auc area = %.4f' % (lr_auc))

#plot the precision-recall curves
no_skill = len(y_test[y_test == 1]) / len(y_test)

#plt.figure(figsize=(10, 10))
#plt.plot([0, 1], [no_skill, no_skill], linestyle = '--', label = 'No
Skill')
plt.plot([0, 1], [1, 0], color = 'navy', lw = lw, linestyle = '--')
plt.plot(lr_recall, lr_precision, color = 'darkorange',
         lw = lw, label = 'PRC curve (area = %.4f)' % lr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

#axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
#show the legend
plt.legend()
```

```
#show the plot
plt.show()

#輸出混亂矩陣，顯示準確率
pred = y_hat.round(0)

from sklearn.metrics import confusion_matrix, classification_report
print('輸出混亂矩陣，顯示準確率：使用訓練資料')
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

from pandas import DataFrame
df = DataFrame({'Real': y_test, 'Pred': pred.flatten()})
filenameReport = output_dir + '/DNN_IMDb.csv'
print('輸出至檔案', filenameReport)
df.to_csv(filenameReport, index = True)
```

八、 MLP_IMDb_LSTM_embed

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 10:21:36 2024

@author: ASUS
"""

from tensorflow.keras.datasets import imdb #new!
from tensorflow.keras.preprocessing.sequence import
pad_sequences
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Flatten, Dropout,
Embedding, Input, concatenate, SpatialDropout1D, Conv1D,
GlobalMaxPooling1D, LSTM
from tensorflow.keras.callbacks import ModelCheckpoint #new!
import os #news!
from sklearn.metrics import roc_auc_score, roc_curve
#import pandas as pd
import matplotlib.pyplot as plt
```

```
import warnings
warnings.filterwarnings('ignore')

'''
#載入 IMDb 影評資料集(原始資料)
(x_train, t_train), (X_test, y_test) = imdb.load_data()
#查看 IMDb 資料集的內容
for x in x_train[0:6]:
      print(len(x))
#查看每筆資料內容
x_train[0:6]

word_index = imdb.get_word_index()
word_index = {k : (v+3) for k, v in word_index.items()}
word_index['START'] = 1
index_word = {v:k for k, v in word_index.items()}

text = ' '.join(index_word[id] for id in x_train[0])
print(text)
'''

#正式模型起點
#載入 IMDb 影評資料集，依據資料預處理參數
#預處理（一）：移除停用字（stop words）
#資料預處理參數
#n_words_to_skip = 50
max_review_length = 400
pad_type = 'pre' #字數太少前面補 0
trunc_type = 'pre' #字數太多刪除前面
drop_embed = 0.2

n_unique_words = 5000
n_dim = 64

n_lstm_1 = n_lstm_2 = n_lstm_3 = 256 #濾鏡個數
#濾鏡大小
k_lstm_1 = 3
```

```
k_lstm_2 = 2
k_lstm_3 = 4

n_dense = 256
dropout = 0.2

epochs = 2
batch_size = 128

n_lstm = 50
drop_lstm = 0.2

#詞向量空間
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words =
n_unique_words)

#print('案例數（訓練集）=', len(y_train))
#print('案例數（測試集）=', len(y_test))

#print(x_train[0:6])

#預處理（二）：統一影評長度
x_train = pad_sequences(x_train, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

x_test = pad_sequences(x_test, maxlen = max_review_length,
                                padding = pad_type, truncating =
trunc_type, value = 0)

'''
for x in x_train[0:6]:
      print(len(x))
print(x_train[5])
'''

# 模型結構
```

```
input_layer = Input(shape=(max_review_length,), dtype='int16',
name='input')
embedding_layer = Embedding(n_unique_words, n_dim,
name='embedding')(input_layer)
drop_embed_layer = SpatialDropout1D(drop_embed,
name='drop_embed')(embedding_layer)

# 通道 1: LSTM
lstm_1 = LSTM(n_lstm_1, dropout=drop_lstm,
recurrent_dropout=drop_lstm, return_sequences=False,
name='lstm_1')(drop_embed_layer)

# 通道 2: LSTM
lstm_2 = LSTM(n_lstm_2, dropout=drop_lstm,
recurrent_dropout=drop_lstm, return_sequences=False,
name='lstm_2')(drop_embed_layer)

# 通道 3: LSTM
lstm_3 = LSTM(n_lstm_3, dropout=drop_lstm,
recurrent_dropout=drop_lstm, return_sequences=False,
name='lstm_3')(drop_embed_layer)

# 合併多通道輸出
concat = concatenate([lstm_1, lstm_2, lstm_3], name='concat')

# 全連接層
dense_layer = Dense(n_dense, activation='relu',
name='dense')(concat)
drop_dense_layer = Dropout(dropout,
name='drop_dense')(dense_layer)
dense_2 = Dense(int(n_dense / 4), activation='relu',
name='dense_2')(drop_dense_layer)
dropout_2 = Dropout(dropout, name='drop_dense_2')(dense_2)

# 輸出層
predictions = Dense(1, activation='sigmoid',
name='output')(dropout_2)
```

```
# 模型
model = Model(input_layer, predictions)

# 編譯模型
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# 檢視模型摘要
model.summary()

#編譯模型
model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

#設定每周期結束紀錄模型參數
output_dir = 'E:/學校/國立彰化師範大學/大三上/人工智慧
/20241216/weight'
modelcheckpoint = ModelCheckpoint(filepath = output_dir +
'/weightsLSTMembed.{epoch:02d}.keras')

if not os.path.exists(output_dir):
      os.makedirs(output_dir)

#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨
機性，
#因此底下(或您重跑一次)的結果不會與書中完全一样,但模型
的能力是相近的
model.fit(x_train, y_train,
                batch_size=batch_size, epochs=epochs, verbose=1,
                validation_data=(x_test,y_test),
                callbacks=[modelcheckpoint])
#模型評估(一):挑幾筆來觀察
model.load_weights(output_dir+
"/weightsLSTMembed.02.keras")#請視以上執行结果指定指定較
佳的權重
y_hat = model.predict(x_test)
```

```
print('y_hat[0]',y_hat[0])
y_hat[0]

#模型評估(二):繪製直方圖
plt.hist(y_hat)
_ = plt.axvline(x = 0.5, color = 'orange')

#模型評估(三):ROC AUC  分數
#註：由於神經網路的初始權重參數是隨機設定的，參雜了隨機性，
#因此底下(或您重跑一次)的結果不會與書中完全一樣,但模型的能力是相近的
pct_auc = roc_auc_score(y_test, y_hat)*100.0
"{:0.2f}".format(pct_auc)
print('ROC AUC  分數=', pct_auc)

################################################
#新增程式碼 provided by 翁政雄老師##############
################################################

pred = y_hat #以下用 pred 代替 y_hat

'''
#將預測結果(0~1)轉成 0/1，即 pred 只能是 0 或 1
#方法(1)
import numpy as np
pred = np.where(y_hat > 0.5, 1, 0)
#方法(2)
pred = y_hat.round(0)
print(pred)
'''

#Compute ROC curve and ROC area for each class
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

```
fpr, tpr, threshold = roc_curve(y_test, pred)
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
#plt.figure(figsize = (10, 10))
plt.plot(fpr, tpr, color = 'darkorange',
         lw = lw, label = 'ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color = 'navy', lw = lw, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('FPR')
plt.ylabel('TFR')
plt.title('ROC Curve')

#plt.legend(loc = 'lower right')
plt.legend()
plt.show()
print('ROC_auc area = %.4f' % (roc_auc))

from sklearn.metrics import precision_recall_curve
#lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_probs)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, pred)

#lr_f1, lr_auc = f1_score(y_test, pred), auc(lr_recall, lr_precision)
lr_auc = auc(lr_recall, lr_precision)
#summarize scores
#print('MLP(ANN): f1 = %.3f PRC_auc = %.3f' % (lr_f1, lr_auc))
print('PRC_auc area = %.4f' % (lr_auc))

#plot the precision-recall curves
no_skill = len(y_test[y_test == 1]) / len(y_test)

#plt.figure(figsize=(10, 10))
#plt.plot([0, 1], [no_skill, no_skill], linestyle = '--', label = 'No
Skill')
plt.plot([0, 1], [1, 0], color = 'navy', lw = lw, linestyle = '--')
plt.plot(lr_recall, lr_precision, color = 'darkorange',
```

```
            lw = lw, label = 'PRC curve (area = %.4f)' % lr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

#axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC Curve')
#show the legend
plt.legend()
#show the plot
plt.show()

#輸出混亂矩陣，顯示準確率
pred = y_hat.round(0)

from sklearn.metrics import confusion_matrix, classification_report
print('輸出混亂矩陣，顯示準確率：使用訓練資料')
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

from pandas import DataFrame
df = DataFrame({'Real': y_test, 'Pred': pred.flatten()})
filenameReport = output_dir + '/DNN_IMDb.csv'
print('輸出至檔案', filenameReport)
df.to_csv(filenameReport, index = True)
```

20241223. 針對不同地方的資料如何進行資料前處理

壹、 相關知識

一、 延伸性欄位可能助於預測更準測

二、 可以去找其他標籤值變化（string 轉 int）（one hotting…）

三、 皮爾森係數、機器學習、spearman corelation

四、 sensitivity vs. specificity

貳、 附錄 – 程式碼

一、 VBA – 資料前處理、轉換

```
Private Sub CommandButton1_Click()
Dim rno As Integer
    DataNo = 6571
    DayNo = (DataNo - 1) / 18
    rno = 2
    For i = 2 To (DayNo + 1)
        For j = 0 To 23
            rno = (i - 2) * 24 + j + 2
            工作表 2.Cells(rno, 1).Value =  工作表 1.Cells((i -
2) * 18 + 2, 1).Value
            工作表 2.Cells(rno, 2).Value = j

            工作表 2.Cells(rno, 3).Value =  工作表 1.Cells((i -
2) * 18 + 2, j + 3).Value
            工作表 2.Cells(rno, 4).Value =  工作表 1.Cells((i -
2) * 18 + 3, j + 3).Value
            工作表 2.Cells(rno, 5).Value =  工作表 1.Cells((i -
2) * 18 + 4, j + 3).Value
            工作表 2.Cells(rno, 6).Value =  工作表 1.Cells((i -
2) * 18 + 5, j + 3).Value
            工作表 2.Cells(rno, 7).Value =  工作表 1.Cells((i -
2) * 18 + 6, j + 3).Value
            工作表 2.Cells(rno, 8).Value =  工作表 1.Cells((i -
2) * 18 + 7, j + 3).Value
```

```
                工作表 2.Cells(rno, 9).Value =  工作表 1.Cells((i -
2) * 18 + 8, j + 3).Value
                工作表 2.Cells(rno, 10).Value =  工作表 1.Cells((i -
2) * 18 + 9, j + 3).Value
                工作表 2.Cells(rno, 11).Value =  工作表 1.Cells((i -
2) * 18 + 10, j + 3).Value
                工作表 2.Cells(rno, 12).Value =  工作表 1.Cells((i -
2) * 18 + 11, j + 3).Value
                工作表 2.Cells(rno, 13).Value =  工作表 1.Cells((i -
2) * 18 + 12, j + 3).Value
                工作表 2.Cells(rno, 14).Value =  工作表 1.Cells((i -
2) * 18 + 13, j + 3).Value
                工作表 2.Cells(rno, 15).Value =  工作表 1.Cells((i -
2) * 18 + 14, j + 3).Value
                工作表 2.Cells(rno, 16).Value =  工作表 1.Cells((i -
2) * 18 + 15, j + 3).Value
                工作表 2.Cells(rno, 17).Value =  工作表 1.Cells((i -
2) * 18 + 16, j + 3).Value
                工作表 2.Cells(rno, 18).Value =  工作表 1.Cells((i -
2) * 18 + 17, j + 3).Value
                工作表 2.Cells(rno, 19).Value =  工作表 1.Cells((i -
2) * 18 + 18, j + 3).Value
                工作表 2.Cells(rno, 20).Value =  工作表 1.Cells((i -
2) * 18 + 19, j + 3).Value

            Next j
        Next i
End Sub
```

每個時間序列模型的輸入維度比較

模型類型	模型名稱	輸入維度(單變量)	輸入維度(多變量)	備註
統計模型	ARIMA	[T]	[T,N]	T 為時間步長，N 為外生特徵數。
	SARIMA	[T]	[T,N]	類似 ARIMA，但處理季節性數據。
機器學習模型	Random Forest	[T,F]	[T,F+N]	F 是特徵數，需提取滯後特徵作為輸入。
	XGBoost	[T,F]	[T,F+N]	常用於回歸任務。
深度學習模型	LSTM	[T,F]	[T,F+N]	支援長期依賴處理。
	GRU	[T,F]	[T,F+N]	LSTM 的輕量級變體。
	Transformer	[T,F]	[T,F+N]	可處理多變量時間序列，對長序列效果好。
	TCN(時間卷積網路)	[T,F]	[T,F+N]	適合處理固定大小窗口的序列任務。
混合模型	HybridARIMA-LSTM	[T,F]	[T,F+N]	結合統計和深度學習的特點。

BERT (Bidirectional Encoder Representations from Transformers)

壹、 概念：

一、 BERT 是基於雙向 Transformer 的一個具體實現，專為 NLP 設計。它在大規模文本語料庫上進行預訓練，並結合特定的預訓練任務（例如 Masked Language Model 和 Next Sentence Prediction）

貳、 用途：

一、 BERT 主要用於 NLP 任務，例如文本分類、命名實體識別、問答系統等。它提供了一個預訓練的語言模型，可以用於遷移學習

參、 訓練方式：

一、 Masked Language Model (MLM)：隨機遮罩輸入文本中的部分單詞，模型需要預測這些遮罩的單詞

二、 Next Sentence Prediction (NSP)：判斷兩個句子是否連續出現

肆、 特點：

一、 是雙向 Transformer 的特化版本，專注於語言理解

二、 利用預訓練和微調來達到高效的遷移學習

三、 對於需要深層語意理解的任務表現極佳

伍、 比較

特性	雙向 Transformer	BERT
用途	通用	NLP 特化
訓練方式	任務特定	預訓練 + 微調
核心技術	雙向注意力	雙向注意力 + MLM + NSP
應用領域	序列任務 (文字、圖像等)	自然語言處理 (文本理解為主)

一、

二、 BERT 是雙向 Transformer 的一個專門實現，針對 NLP 任務進行了優化，並通過預訓練任務學習深層語意表示，適合用於需要強語言理解能力的應用

三、 Transformer 的雙向並不依靠 Bidirectional 實現，而是須將兩個方向的 Transformer 編碼器結合

其他

單獨使用時序性模型時，可能會導致以下幾種結果：

壹、 過於依賴歷史資料：時序性模型主要基於歷史數據來進行預測，若資料本身存在噪音或數據不足，模型可能無法準確捕捉未來趨勢或變化。

貳、 外部因素的忽視：僅使用時序性模型無法考量外部變量，如政策變動、氣候突變等對預測結果的影響，從而導致模型預測的準確度降低。

參、 長期預測不穩定：許多時序性模型在進行長期預測時，會出現誤差累積，導致結果隨時間變得不穩定或偏離實際情況。

肆、 複雜的非線性關係難以捕捉：單純的時序模型，如 ARIMA 或單層 LSTM，對於較複雜的非線性關係可能無法準確建模，從而影響預測表現。

時間序列預測

壹、 自迴歸(AR)模型

貳、 移動平均(MA)模型

參、 自迴歸移動平均(ARMA)模型

肆、 自迴歸綜合移動平均(ARIMA)模型

伍、 季節自迴歸綜合移動平均(SARIMA)模型

陸、 帶有外源迴歸量的季節自迴歸綜合移動平均(SARIMAX)模型

柒、 向量自迴歸(VAR)模型

捌、 向量誤差校正(VECM)模型

玖、 多層感知器(MLP)

壹拾、 迴圈神經網路(RNN)

壹拾壹、 長短期記憶網路(LSTM)

壹拾貳、 自迴歸 LSTMs

壹拾參、 卷積神經網路(CNN)