Data Mining 期末作業說明

Computer Vision classification

助教：李羚甄

CONTENTS

01 期末作業說明

期末作業說明_part1

- InAnalysis網址：http://140.112.26.241:8009/#/
- Deadline：2021/12/29 23:00

電腦視覺演算法模組實作

■ 以 Python keras 套件進行電腦視覺分析演算法實作。

1. 使用助教提供之資料集，或是自行尋找資料集，實作一種演算法。----- **60%**

   （提供之資料集於期末作業包中，詳細資訊下一頁會做說明）

2. 錄製期末報告影片（10min）： ----- **20%**

   A. 資料集與專案目的說明

   B. 演算法介紹與程式碼說明

   C. 使用InAnalysis進行Demo

   ➢ 報告投影片模板於期末作業包中（ Template）

3. 仿照期中專案形式解釋訓練與預測結果，撰寫成一個report.docx檔，

   並將報告影片連結附在該檔最後，內容架構可參考報告投影片模板。 ----- **20 %**

期末作業說明_part2

繳交內容及方式

- 總共需要繳交四樣東西
  1. algorithm 資料夾，其中包含 .py 及 .json 兩個檔案
  2. dataset 資料夾，放你使用的資料集
  3. 報告時製作的投影片檔（.pptx 或 .Key 等等都可以）
  4. 將撰寫好的report.docx檔**轉成 PDF 檔**，再次提醒請將報告影片連結附在該檔案最後，並確保觀看權限開啟。
- 將上述的檔案放置於同一個資料夾中，**壓縮為{學號}.zip**上傳至Cool作業。
- Deadline：2021/12/29 23:00

提供之資料集

資料集一：旅宿景點（trip_dataset）

- 提供28,915張景點圖片給同學自由發揮
- 圖片尚未標註label

資料集二：稻米（rice_dataset）

- 總共64個資料夾，每個資料夾中至少360張圖片，給同學自由發揮
- 影像大小：256*256
- 圖片尚未標註label，但已分類整理
  - 分類方式：
    1. 「年份 - 產期 - 產地」，如「107-1 秀水」為一類，共64類
    2. 「年份 - 產期」，如「107-1 」為一類，共6類

02 資料集格式設定

Data Format

Computer Vision 資料集格式 - ZIP

- 上傳的檔案為一個ZIP檔

Data Format

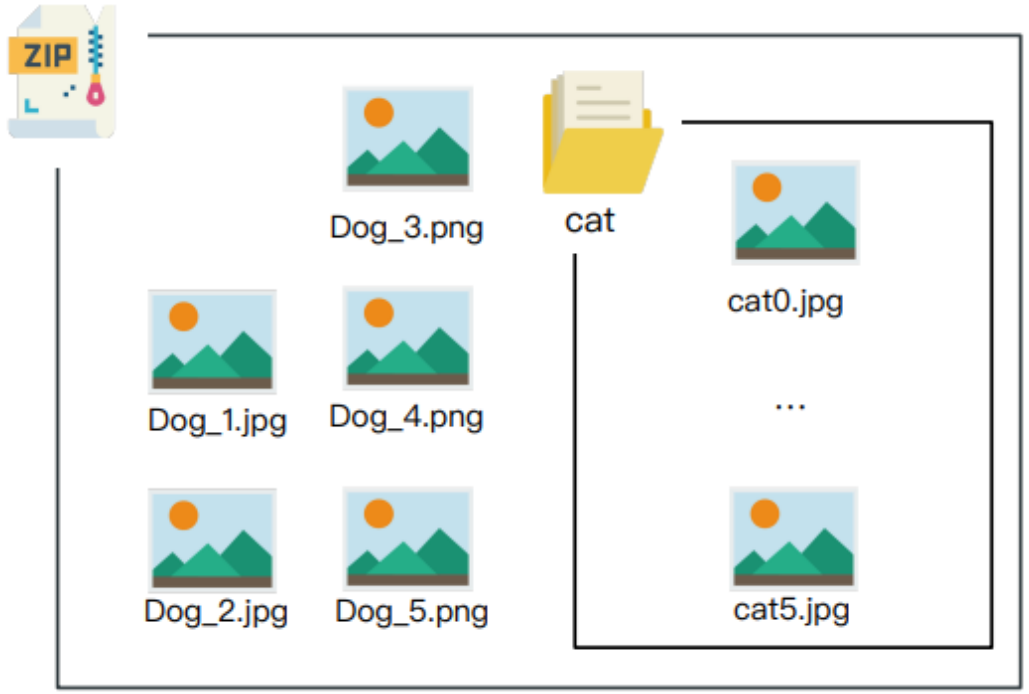
Computer Vision 資料集格式 - ZIP

- 裡面含有靜態影像

Data Format

Computer Vision 資料集格式 - ZIP

- 影像可以放在不同的資料夾

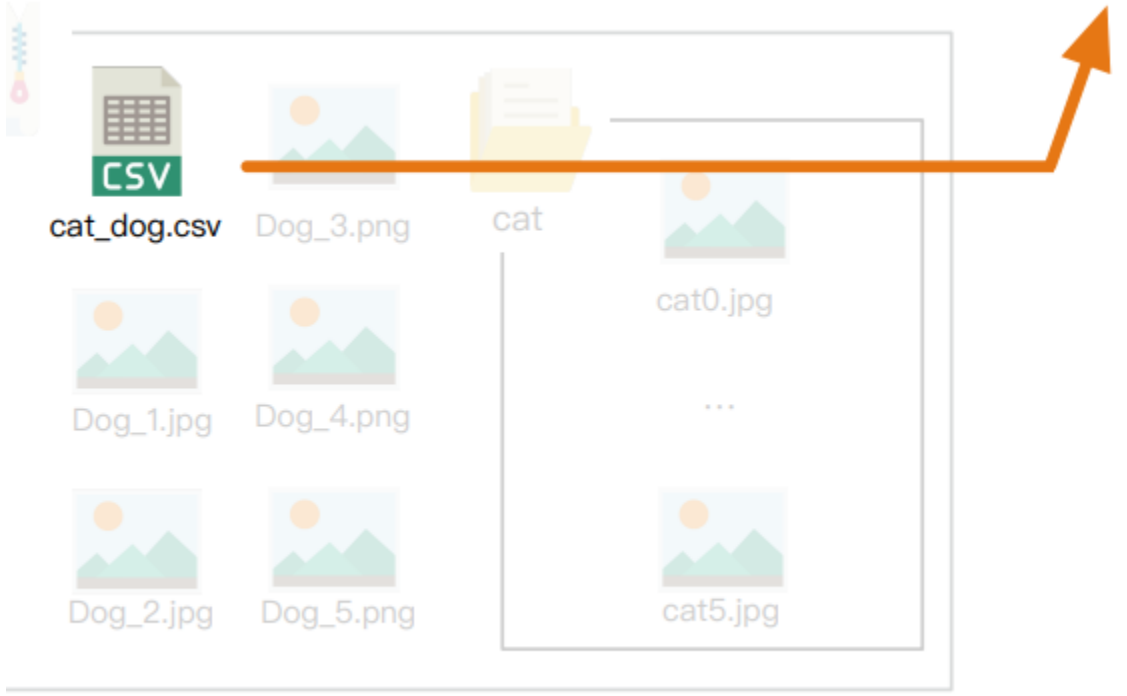


Data Format

Computer Vision 資料集格式 - ZIP

- 已相對於 zip 的相對路徑表示影像檔案
- 一筆資料（raw）可以有多個欄位（col）
- 必須有欄位提供該筆資料的類別或是數值

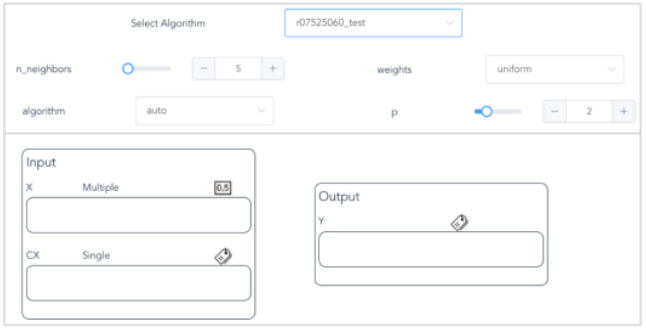
	A	B
1	image	label
2	Dog_1.jpg	dog
3	Dog_2.jpg	dog
4	cat/cat0.jpg	cat
5	cat/cat1.jpg	cat
6	cat/cat2.jpg	cat
7	Dog_3.png	dog
8	cat/cat3.jpg	cat
9	Dog_4.png	dog
10	cat/cat4.jpg	cat
11	cat/cat5.jpg	cat
12	Dog_5.png	dog



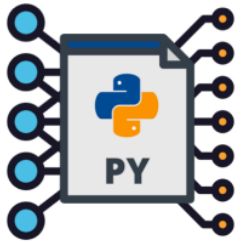
03 演算法實作介紹

Algorithm Implementation

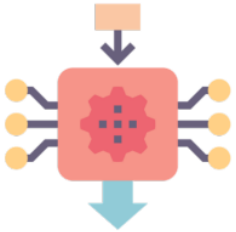
定義演算法的
- 使用者可調整的參數
- 演算法需要的輸入
- 演算法需要的輸出



前端生成欄位



收集使用者輸入的
- 參數
- 輸入
- 輸出



得到可重複使用的模型
- 預覽
- 測試
- 預測

Algorithm Implementation

需要實作的兩個檔案

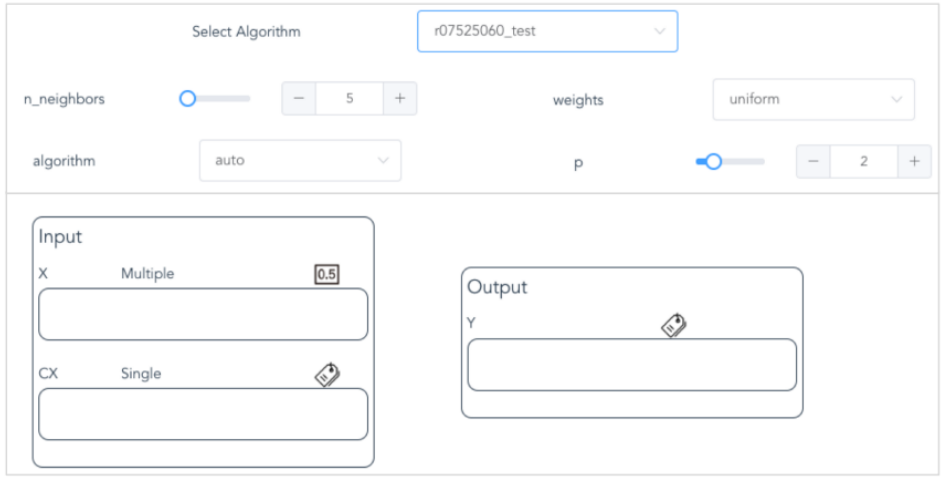
- 學號_演算法名稱.json
- 學號_演算法名稱.py

照著範例實作就可以，沒有想像中那麼困難!

JSON檔內容

定義演算法的

- 使用者可調整的參數
- 演算法需要的輸入
- 演算法需要的輸出



JSON檔內容

- **dataType**

  定義這個演算法要分析的資料類型（數值/電腦視覺/自然語言）

  `Enum("num","cv","nlp")`

- **projectType** `Enum("classification","regression","abnormal","clustering")`

  定義這個演算法要分析的目標類型（迴歸/分類/異常偵測/分群）

  每個 dataType 所支持的 projectType 如下

	regression	classification	abnormal	clustering
num	✔	✔	✔	✔
cv	✔	✔	✖	✖
nlp	✔	✔	✖	✖

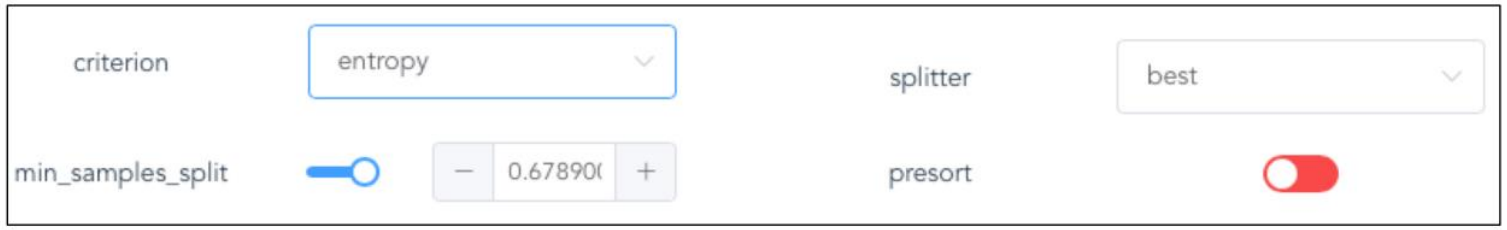
JSON檔內容

範例

- **algoName** 實作演算法的名稱 e.g. r09525000_oneLayerCNN。這個名稱需與檔名一致。
- **description** 實作演算法的描述。
- **lib** 所使用的函式庫，目前支援 sklearn 以及 keras。`Enum("sklearn","keras")`

```
{
    "dataType":"cv",
    "projectType":"classification",
    "algoName":"r09525000_oneLayerCNN",
    "description": "Single layer convolutional neural network.",
    "lib":"keras",
```

JSON檔內容 - Parameter Obj



- **Parameter oject**
  定義要讓使用者決定的參數。每個參數物件有以下這些屬性

JSON檔內容 - Parameter Obj

- **name**: "參數名稱"
- **description**: "此參數的描述"
- **type**: `Enum("int","float","bool","enum","string")`

int	float	bool	enum	string
參數為整數	參數為小數	0 代表 False 1 代表 True	參數為選項	參數為字串

- **default**: 此參數的預設值

  Example:

int	float	bool	enum	string
10	23.5	1	"op1"	"some string"

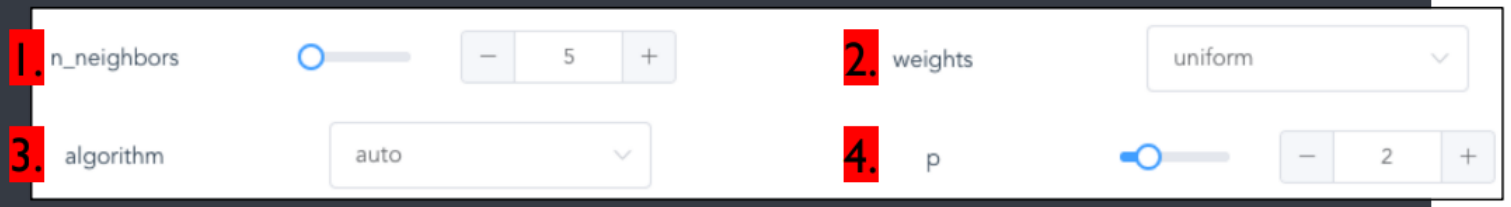
- **其他屬性**

屬性	意義	required by	example
upperBound	參數的上界	int, float	20
lowerBound	參數的下界	int, float	0
list	選項參數的候選選項	enum	["op1","op2"]

JSON檔內容 - Parameter Obj

```
"param":[
    {
        "name":"n_neighbors",
        "description":"Number of neighbors to use",
        "type":"int",
        "upperBound":50,
        "lowerBound":1,
        "default":5
    },
    {
        "name":"weights",
        "description":"weight function used in prediction.",
        "type":"enum",
        "list":["uniform","distance"],
        "default":"uniform"
    },
    {
        "name":"algorithm",
        "description":"Algorithm used to compute the nearest neighbors",
        "type":"enum",
        "list":["auto","ball_tree","kd_tree","brute"],
        "default":"auto"
    },
    {
        "name":"p",
        "description":"Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance
        "type":"int",
        "upperBound":5,
        "lowerBound":1,
        "default":2
    }
],
```

1. 2. 3. 4.



JSON檔內容 - Input Group Obj

此參數說明了此演算法應該要有那些輸入組。

一個算法可以有多個 input group

每個 input group 允許多個 feature

同個 input group 的型態應該要一致。

- **name**
- **description**
- **type**

  此 input 的型態

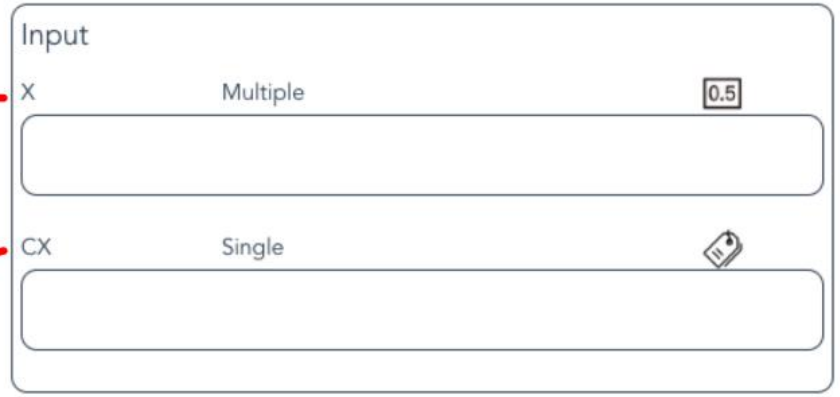
  `Enum("float","classifiable","string","path")`
- **amount**

  此輸入可否接受多個 feature

  `Enum("single", "multiple")`

JSON檔內容 - Input Group Obj

```
"input":[
    {
        "name": "X",
        "description": "input data",
        "type":"float",
        "amount":"multiple"
    },
    {
        "name": "CX",
        "description": "input data",
        "type":"classifiable",
        "amount":"single"
    }
],
```



JSON檔內容 - Output Obj

此參數說明了算法的輸出。
一個算法可以有多個output
每個 output 僅能有一個 feature。
對於沒有輸出的 Unsupervised learning，此物件為一個空list。

- **name**
- **description**
- **type**
  此 output 的型態
  `Enum("float","classifiable","string","path")`

JSON檔內容 - Output Obj

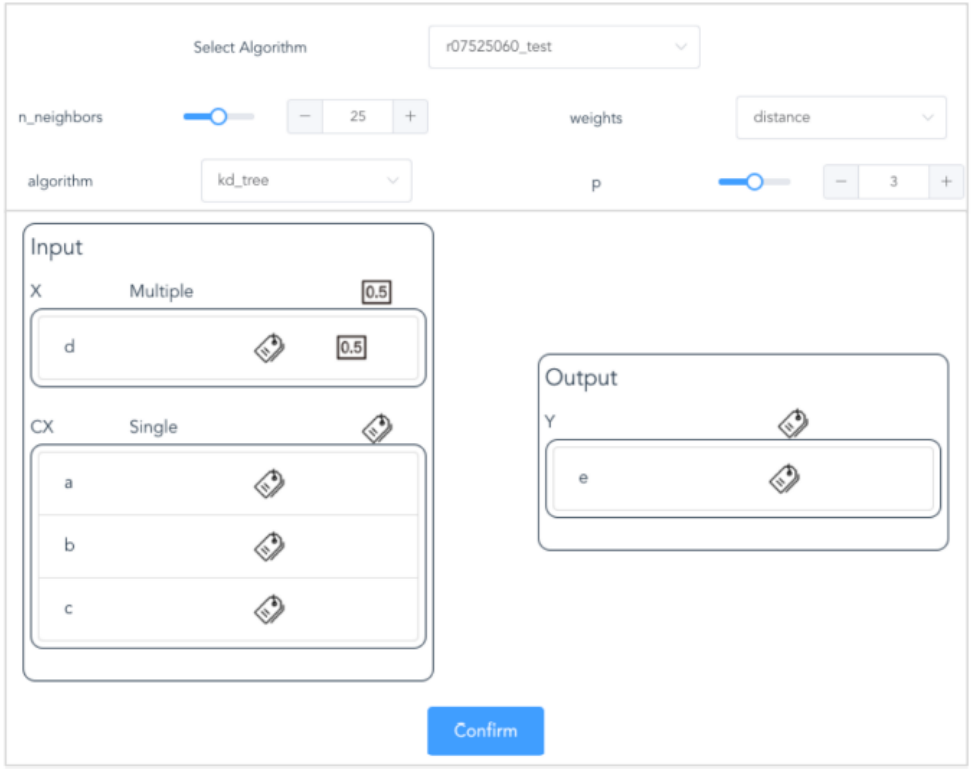
Supervised

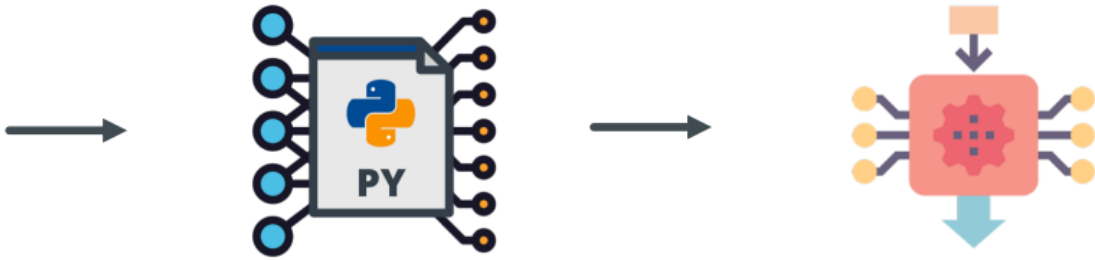
```
"output":[
    {
        "name": "Y",
        "description": "output data",
        "type":"classifiable"
    }
]
```

Unsupervised

```
"output":[
]
```

Python檔內容





演算法模組
將參數、輸入、輸出
經過演算法得到模型

Python檔內容 - 成員變數

使用者輸入的 參數、Input feature、Output feature
演算法產生的 最終模型、模型預測結果、其他附屬變數 (tokenizer 等)
皆以字典成員變數形式儲存

變數	內容
self.param	使用者指定參數
self.inputData	使用者指定 input feature
self.outputData	使用者指定 output feature
self.model	演算法產生之模型
self.result	模型預測出的結果
self.customObj	其他模型必要的變數

Python檔內容 - self.param

Parameter object

在程式中，使用者輸入的參數值皆以 dictionary 方式存在 `self.param` 中

例如，如果在參數定義中，有一個 parameter object 為

```
{
    "name": "param2",
    "description": "param2 Description",
    "type": "float",
    "upperBound": 30.5,
    "lowerBound": 0,
    "default":23.2
}
```

在程式中，此參數的輸入值會存於 `self.param['param2']`

Python檔內容 - self.inputData

Input group object

在程式中，使用者對於每個輸入組所指定的數據會以 dictionary 方式存在 `self.inputData` 中

每個輸入組的結構皆為

```
numpy.array(
    [
        [col1-row1,col2-row1,....,colN-row1],
        ....,
        [col1-rowM,col2-rowM,...,colN-rowM]
    ]
)
```

Python檔內容 - self.inputData

例如，一個如下的 csv

a	b	c
1	2	3
4	5	6
7	8	9
10	11	12

將 a, b, c 三個 column 以 [b c a] 的順序讀入一個名為 input1 的輸入組，則 `self.inputData['input1']` 將為

```
numpy.array(
    [
        [2,3,1],
        [5,6,2],
        [8,9,7],
        [11,12,10]
    ]
)
```

Python檔內容 - self.inputData

若該輸入組型態為 *classifiable*，則系統會自動將其轉換為 one-hot encoding 型式，並將對應關係存至 `self.c2d` 及 `self.d2c`

例如，一個如下的 csv

a	b	c
cat	apple	red
dog	banana	blue
dog	orange	black
cat	banana	green

將 a, b, c 三個 column 以 [c,a,b] 順序讀入一個名為 input2 的輸入組，系統會隨機為每個column產生類別對應如下面型式

Python檔內容 - self.inputData

原始資料

a	b	c
cat	apple	red
dog	banana	blue
dog	orange	black
cat	banana	green

類別–>數值對應

```
self.c2d = {
                "a":{
                    "cat":0,
                    "dog":1
                },
                "b":{
                    "apple": 0,
                    "banana": 1,
                    "orange":2
                },
                "c":{
                    "red":0,
                    "blue":1,
                    "black":2,
                    "green":3
                }
            }
```

數值–>類別對應

```
self.d2c = {
                "a":{
                    "0":"cat",
                    "1":"dog"
                },
                "b":{
                    "0": "apple",
                    "1": "banana",
                    "2":"orange"
                },
                "c":{
                    "0":"red",
                    "1":"blue",
                    "2":"black",
                    "3":"green"
                }
            }
```

Python檔內容 - self.inputData

原始資料

a	b	c
cat	apple	red
dog	banana	blue
dog	orange	black
cat	banana	green

類別–>數值對應

```
self.c2d = {
                "a":{
                    "cat":0,
                    "dog":1
                },
                "b":{
                    "apple": 0,
                    "banana": 1,
                    "orange":2
                },
                "c":{
                    "red":0,
                    "blue":1,
                    "black":2,
                    "green":3
                }
            }
```

數值–>類別對應

```
self.d2c = {
                "a":{
                    "0":"cat",
                    "1":"dog"
                },
                "b":{
                    "0": "apple",
                    "1": "banana",
                    "2":"orange"
                },
                "c":{
                    "0":"red",
                    "1":"blue",
                    "2":"black",
                    "3":"green"
                }
            }
```

根據輸入組以及以上對應關係，`self.inputData['input2']` 將為

```
numpy.array(
    [
        [ numpy.array([1,0,0,0]), numpy.array([1,0]), numpy.array([1,0,0]) ],
        [ numpy.array([0,1,0,0]), numpy.array([0,1]), numpy.array([0,1,0]) ],
        [ numpy.array([0,0,1,0]), numpy.array([0,1]), numpy.array([0,0,1]) ],
        [ numpy.array([0,0,0,1]), numpy.array([1,0]), numpy.array([0,1,0]) ]
    ]
)
```

Python檔內容 - self.inputData

原始資料

a	b	c
cat	apple	red
dog	banana	blue
dog	orange	black
cat	banana	green

類別–>數值對應

```
self.c2d = {
                "a":{
                    "cat":0,
                    "dog":1
                },
                "b":{
                    "apple": 0,
                    "banana": 1,
                    "orange":2
                },
                "c":{
                    "red":0,
                    "blue":1,
                    "black":2,
                    "green":3
                }
            }
```

數值–>類別對應

```
self.d2c = {
                "a":{
                    "0":"cat",
                    "1":"dog"
                },
                "b":{
                    "0": "apple",
                    "1": "banana",
                    "2":"orange"
                },
                "c":{
                    "0":"red",
                    "1":"blue",
                    "2":"black",
                    "3":"green"
                }
            }
```

根據輸入組以及以上對應關係，`self.inputData['input2']` 將為

```
numpy.array(
    [
        [ numpy.array([1,0,0,0]), numpy.array([1,0]), numpy.array([1,0,0]) ],
        [ numpy.array([0,1,0,0]), numpy.array([0,1]), numpy.array([0,1,0]) ],
        [ numpy.array([0,0,1,0]), numpy.array([0,1]), numpy.array([0,0,1]) ],
        [ numpy.array([0,0,0,1]), numpy.array([1,0]), numpy.array([0,1,0]) ]
    ]
)
```

cat–>0–>[1,0]

Python檔內容 - self.inputData

原始資料

a	b	c
cat	apple	red
dog	banana	blue
dog	orange	black
cat	banana	green

類別–>數值對應

```
self.c2d = {
            "a":{
                "cat":0,
                "dog":1
            },
            "b":{
                "apple": 0,
                "banana": 1
                "orange":2
            },
            "c":{
                "red":0,
                "blue":1,
                "black":2,
                "green":3
            }
        }
```

數值–>類別對應

```
self.d2c = {
            "a":{
                "0":"cat",
                "1":"dog"
            },
            "b":{
                "0": "apple",
                "1": "banana",
                "2":"orange"
            },
            "c":{
                "0":"red",
                "1":"blue",
                "2":"black",
                "3":"green"
            }
        }
```

根據輸入組以及以上對應關係，`self.inputData['input2']` 將為

```
numpy.array(
    [
        [ numpy.array([1,0,0,0]), numpy.array([1,0]), numpy.array([1,0,0]) ],
        [ numpy.array([0,1,0,0]), numpy.array([0,1]), numpy.array([0,1,0]) ],
        [ numpy.array([0,0,1,0]), numpy.array([0,1]), numpy.array([0,0,1]) ],
        [ numpy.array([0,0,0,1]), numpy.array([1,0]), numpy.array([0,1,0]) ]
    ]
)
```

banana–>1–>[0,1,0]

Python檔內容 - self.inputData

原始資料

a	b	c
cat	apple	red
dog	banana	blue
dog	orange	black
cat	banana	green

類別–>數值對應

```
self.c2d = {
                "a":{
                    "cat":0,
                    "dog":1
                },
                "b":{
                    "apple": 0,
                    "banana": 1,
                    "orange":2
                },
                "c":{
                    "red":0,
                    "blue":1,
                    "black":2,
                    "green":3
                }
            }
```

數值–>類別對應

```
self.d2c = {
                "a":{
                    "0":"cat",
                    "1":"dog"
                },
                "b":{
                    "0": "apple",
                    "1": "banana",
                    "2":"orange"
                },
                "c":{
                    "0":"red",
                    "1":"blue",
                    "2":"black",
                    "3":"green"
                }
            }
```

根據輸入組以及以上對應關係，`self.inputData['input2']` 將為

```
numpy.array(
    [
        [ numpy.array([1,0,0,0]), numpy.array([1,0]), numpy.array([1,0,0]) ],
        [ numpy.array([0,1,0,0]), numpy.array([0,1]), numpy.array([0,1,0]) ],
        [ numpy.array([0,0,1,0]), numpy.array([0,1]), numpy.array([0,0,1]) ],
        [ numpy.array([0,0,0,1]), numpy.array([1,0]), numpy.array([0,1,0]) ]
    ]
)
```

green–>3–>[0,0,0,1]

Python檔內容 - self.outputData

Output object

在程式中，使用者對於每個輸出所指定的數據會以 dictionary 方式存在 `self.outputData` 中

每個輸出的結構皆為

```
numpy.array(
    [colN-row1,colN-row2,....,colN-rowM]
)
```

若該輸出型態為 *classifiable*，則與輸入組一樣，系統會自動將其轉換為 one-hot encoding 型式，並將對應關係存至 `self.c2d` 及 `self.d2c`

此時該輸出會像

```
numpy.array(
    [[0,1,0],[0,0,1],....,[1,0,0]]
)
```

Python檔內容 - self. result

self.result 儲存了模型針對輸入計算後的預測值
他的結構應該要與 self.outputData 相同

範例：

假設你的演算法有兩個輸出 X 以及 CX，分別是數值以及 one-hot

```
self.outputData={
    "X": np.array([0.7,1,0.2]),
    "CX": np.array([[0,0,1],[0,1,0],[1,0,0]])
}
```

則預測後的結果就要以

```
self.result["X"]=數值預測結果
self.result["CX"]=類別預測結果
```

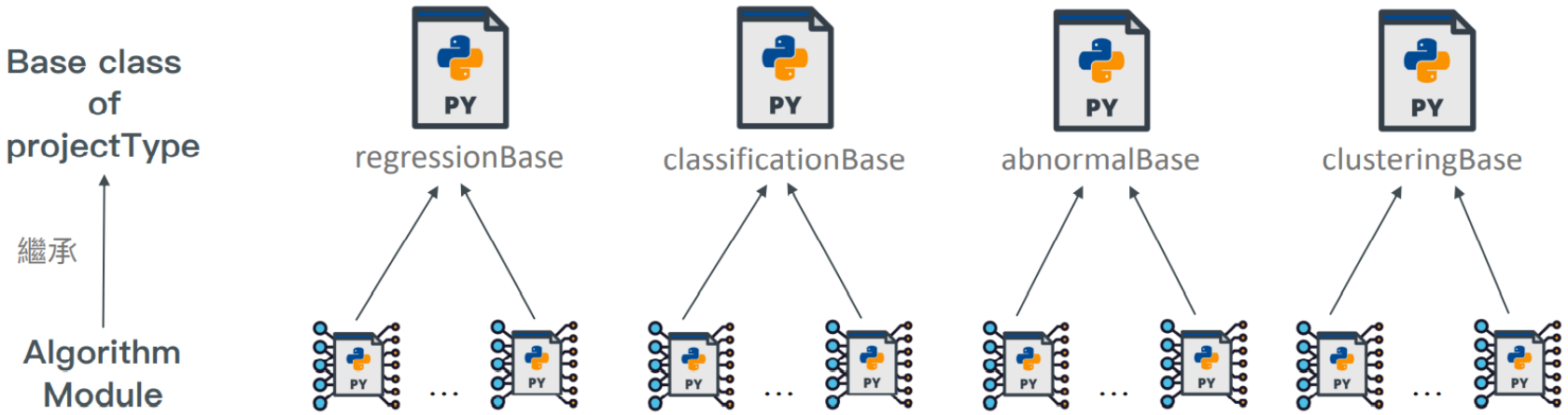
儲存如

```
self.result={
    "X": np.array([0.5,0,0.8]),
    "CX": np.array([[0.2,0.5,0.3],[0.7,0.1,0.2],[0.9,0.05,0.05]])
}
```

Python檔內容 - 類別圖

演算法模組為一個class，名稱就是演算法名稱

所有的演算法模組皆繼承自各自 project type 的 base class



Python檔內容 - 實作

必須實作兩個 function

trainAlgo

訓練時，請利用 `self.inputData`, `self.outputData` 以及 `self.param` 定義以及訓練模型，並將模型 instance 存至 `self.model`。

predictAlgo

預測時，請利用 `self.model` 以及 `self.inputData` ，將預測出的資料以與 `self.outputData` 相同的架構存放至 `self.result`。若想自行定義模型的結果表示 (如: loss, accuracy 等等)，請將結果字串存放至 `self.txtRes`。

Python檔內容 - CV分類演算法範例

系統本身提供簡易的
XYdataGenerator
以及
XdataGenerator
可參考所附原始碼
也可以自行實作

請使用
fit_generator
以及
predict_generator

```python
from service.analyticService.core.analyticCore.classificationBase import classification
from keras.models import Sequential
from keras.layers import Dense,Conv2D,Flatten
import tensorflow as tf
import keras.backend.tensorflow_backend as KTF
from service.analyticService.core.analyticCore.utils import XYdataGenerator,XdataGenerator
from math import ceil
class in_oneLayerCNN(classification):
    def trainAlgo(self):
        self.model=Sequential()
        self.model.add(Conv2D(
            self.param['hidden_neuron'],
            (self.param['hidden_kernel_size'],self.param['hidden_kernel_size']),
            input_shape=(32,32,3),data_format='channels_last',
            activation=self.param['hidden_activation']))
        self.model.add(Flatten())
        self.model.add(Dense(self.outputData['Y'].shape[1],activation='softmax'))
        self.model.compile(loss='categorical_crossentropy',optimizer=self.param['optimizer'])
        self.model.fit_generator(
            XYdataGenerator(self.inputData['X'],self.outputData['Y'],32,32,self.param['batch_size']),
            steps_per_epoch=int(ceil((len(self.inputData['X'])/self.param['batch_size']))),
            epochs=self.param['epochs']
        )
    def predictAlgo(self):

        r=self.model.predict_generator(
            XdataGenerator(self.inputData['X'],32,32,self.param['batch_size']),
            steps=int(ceil((len(self.inputData['X'])/self.param['batch_size'])))
        )
        self.result['Y']=r
```

inputData['X'] 的內容是
每筆資料的影像路徑

所附演算法範例

演算法範例

- CV
  - classification（./example/cv/classification）
  - generator 範例（./example/cv /utils.py）

演算法實作注意事項

注意事項

1. 請務必確認每個演算法 fit 的輸出輸入之shape，Sklearn 分類演算法的輸出**並非one-hot**，記得轉換。
2. 分類演算法預測時，請輸出**機率**。
3. 系統目前僅支援 **10個類別** 的分類。
4. 實作 CV 專案時，請務必使用 **fit_generator** 以及 **predict_generator**，請不要設定太大的batch_size。
5. 請及早開始時作、測試，避免死線前使用量過高而無法測試。

演算法實作注意事項

- json 檔名
- json algoName
- py 檔名
- py class 名稱

必須為 學號_演算法名稱

演算法實作注意事項

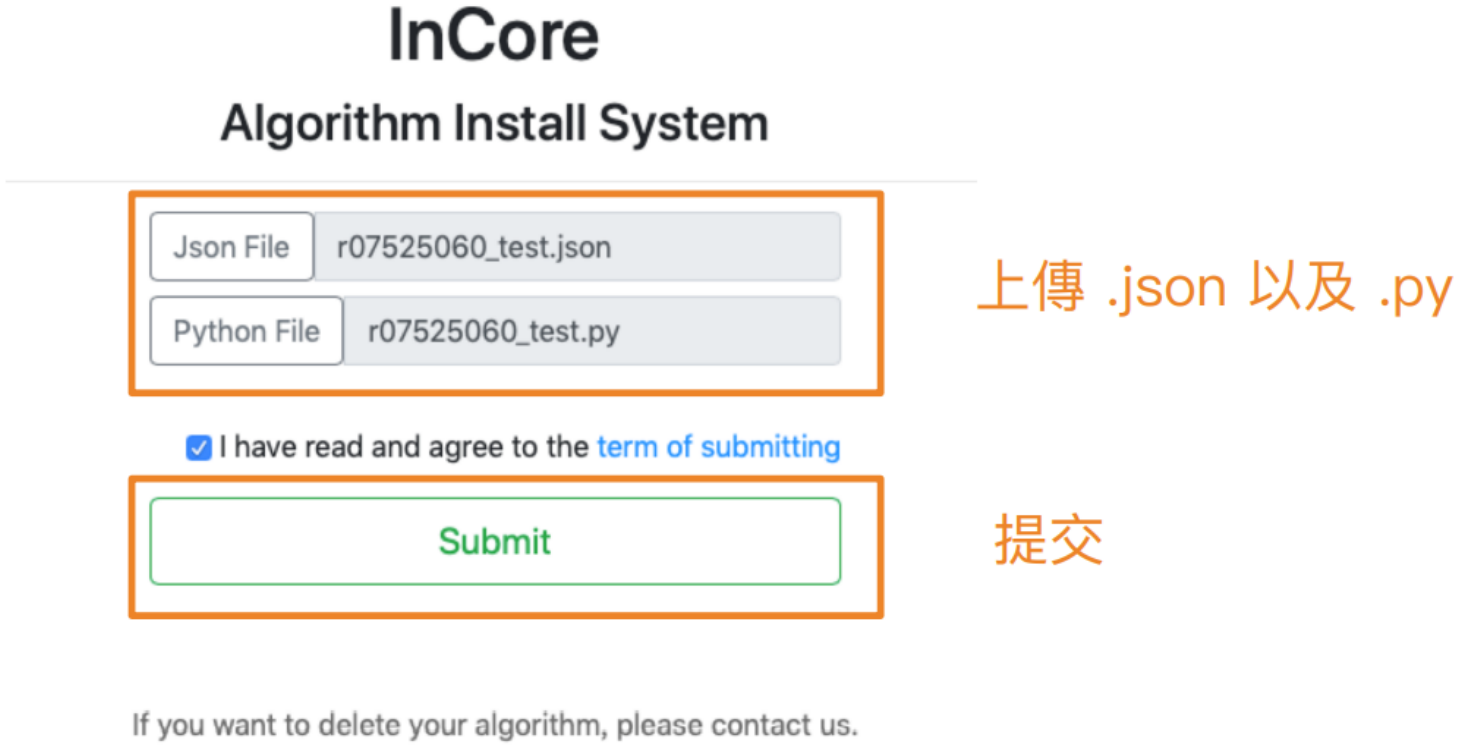
系統所使用版本

OS	Ubuntu 16.04.4 LTS
Python	3.6.8
numpy	1.17.1
scikit-learn	0.21.3
scipy	1.3.1
tensorflow	1.12.0
keras	2.2.4

演算法上傳教學

注意事項

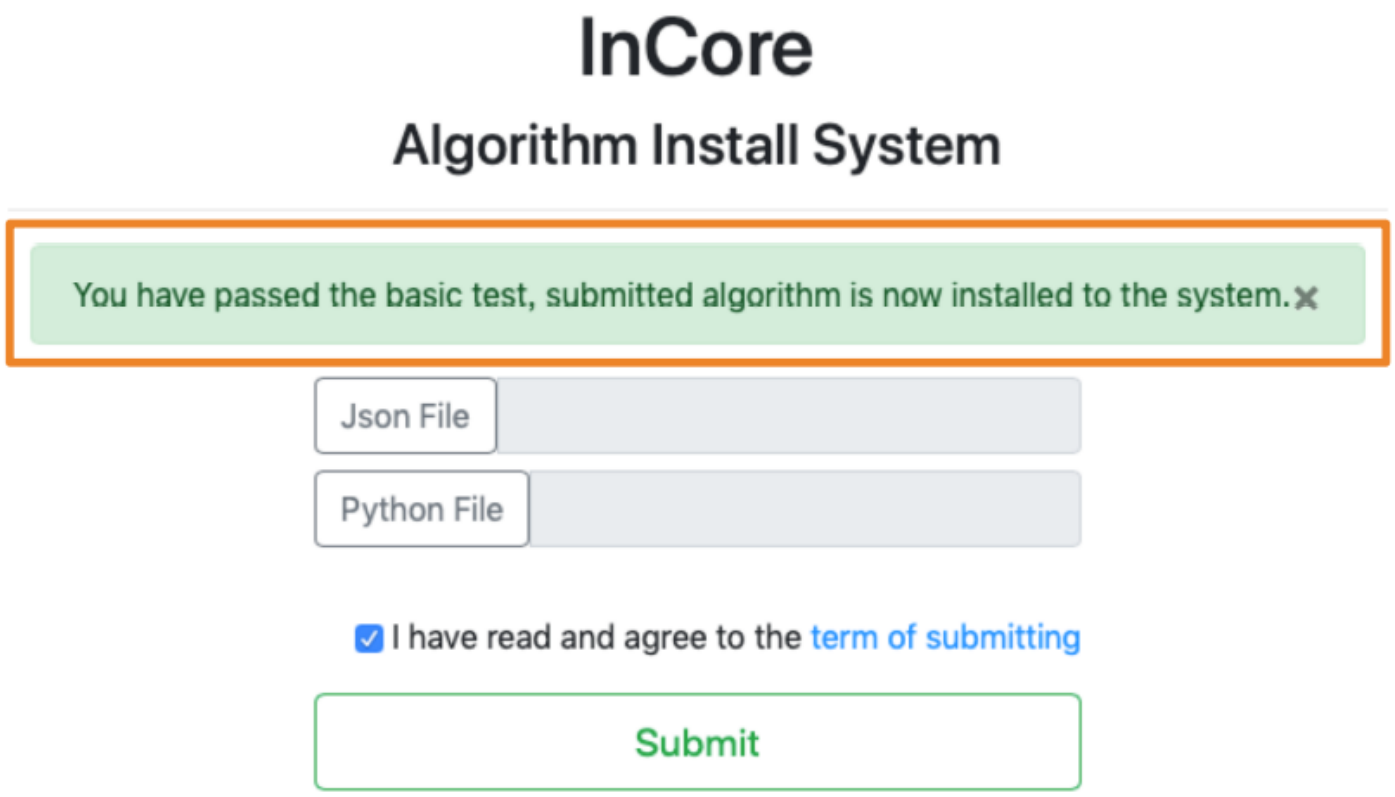
- 每個人上傳的網址不同，會寄信給各位。



演算法上傳教學

若初步檢查不通過
顯示錯誤訊息

演算法上傳教學



若初步檢查通過
顯示成功訊息

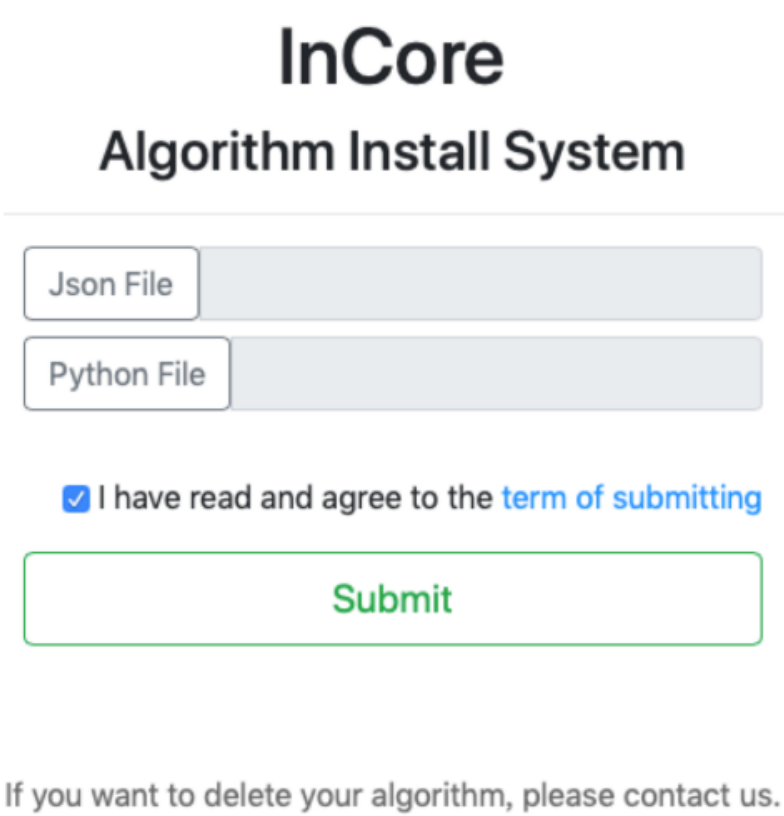
注意：這不代表演算法無誤
請至系統選擇你的演算法
並實際訓練、測試
能夠成功訓練模型才算完成

演算法上傳教學

若訓練不成功，可將滑鼠移到 fail 上
在錯誤訊息中找到自己的 py 檔
查看導致錯誤的行數
若不是程式問題，請向助教詢問

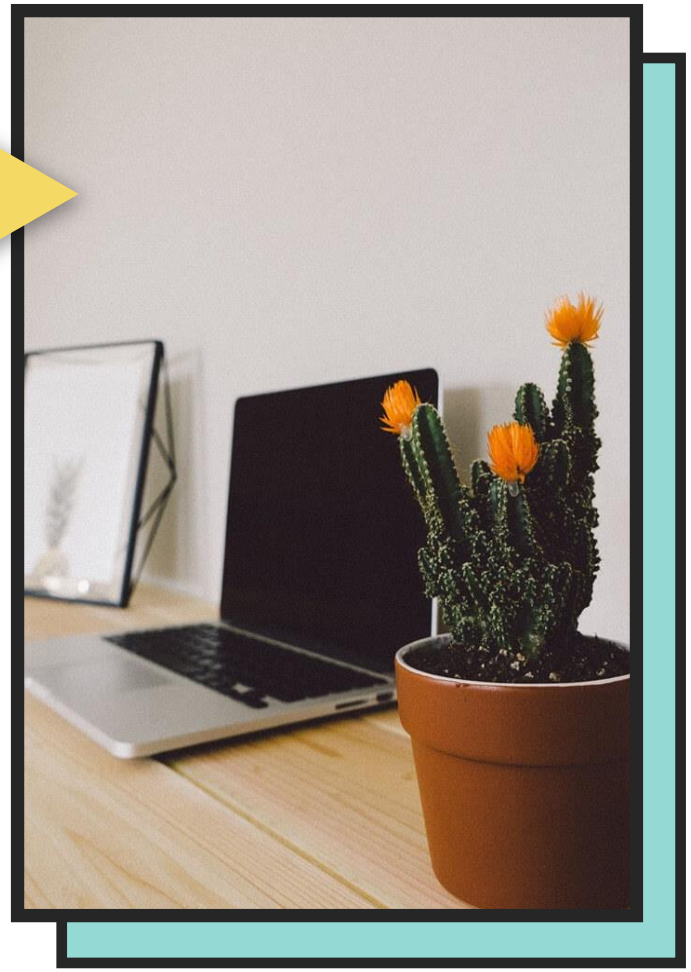
注意：助教不是 debug 機器
請先自行 **Google**
若懷疑是系統問題
或真的不知道怎麼解再提出

演算法上傳教學



若要修改已上傳的演算法
請以同樣的檔名再上傳一次即可

Q&A_TA information



Teacher assistant:

李羚甄、李丞彥

Mail:

r09525064@ntu.edu.tw

r10525067@ntu.edu.tw

TA Hour:

週三下午，工科125A

（請提前寄信預約，並說明問題）