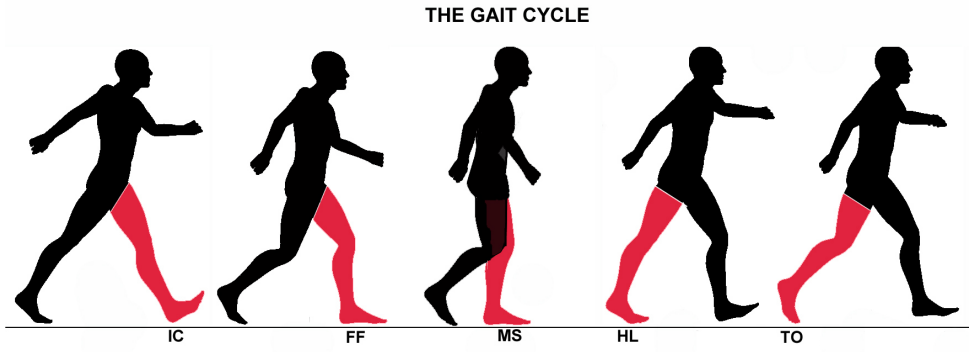
利用GaitPhase 資料進行步態速率分析



2021_1

109034570　林溥鈞

Outline

01 研究主題說明

02 資料前處理

03 預測模型設定

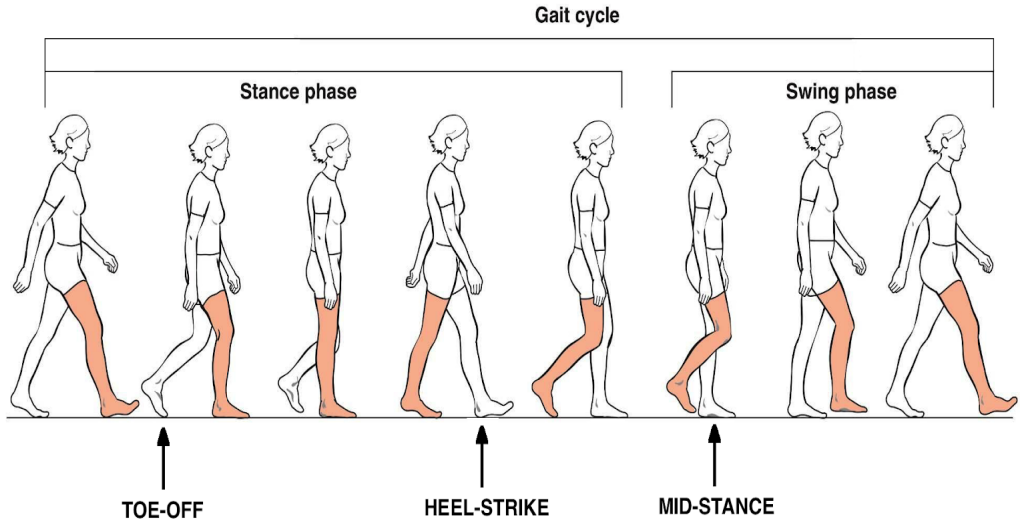
04 分析與模型效度驗證

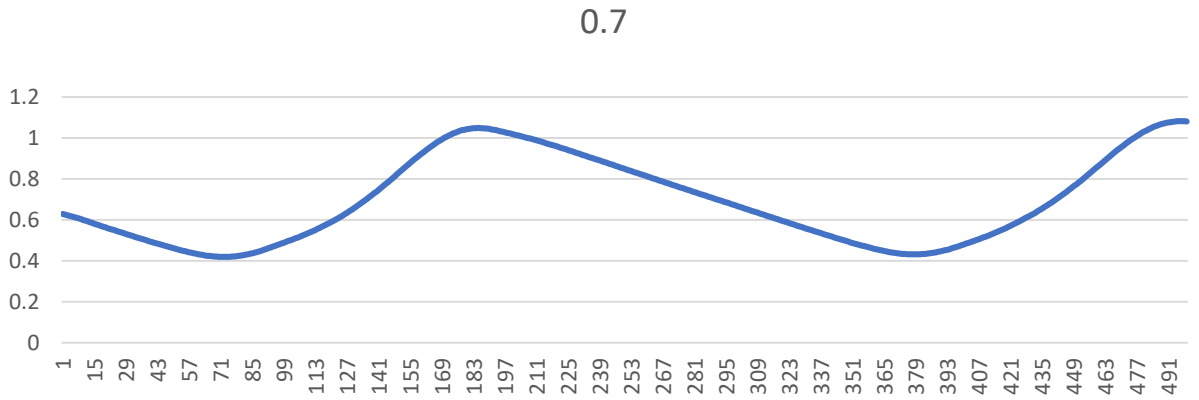
05 結論與未來展望

01 研究主題說明

研究動機

- Gait Cycle 每個人都有獨特的走路頻率
- 步態人體特徵
  - ➢ 週期性
  - ➢ 不變性 (外貌、形狀、顏色)
  - ➢ 獨特性
- 利用步態特徵進行辨識
  - ➢ 相同人--- 不同步態速率





5W1H

What 利用步態特徵的獨特性、不變化，週期性分辨不同步態速率

Why 有些細微變化是難以分辨的，又或是必須借助儀器才能分析，希望可以透過AI的方式輔助分析，找出準確率高的辨識模型

Who 可以用在運動員、復健病患、跌倒老人

When 運動員在訓練時，病患在復健時

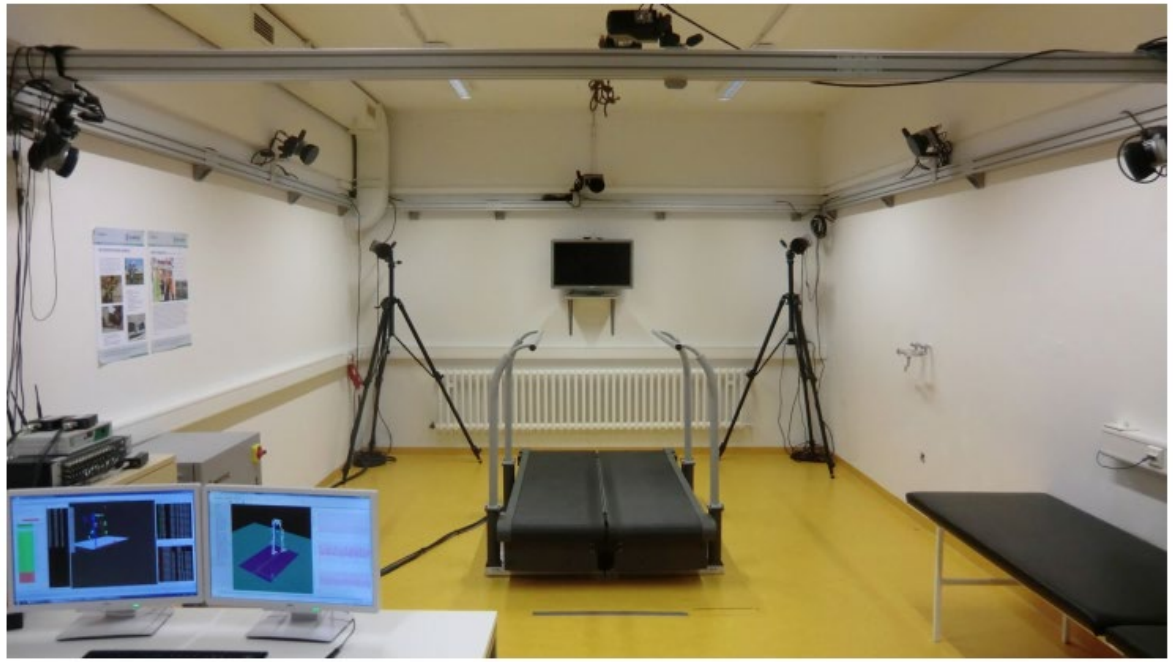
Where 實驗室、訓練室、醫院

How 利用分類模型、深度學習、資料分析

02 資料前處理

資料集介紹

受測者	21位
性別	10位男性、11位女性
年齡	23.8 yrs ±3.3 yrs
身高	172.8 cm ±9.4 cm,
體重	66.6 kg ±10.9 kg
速率	[0.6, 1.7] m/s at 0.1 m/s

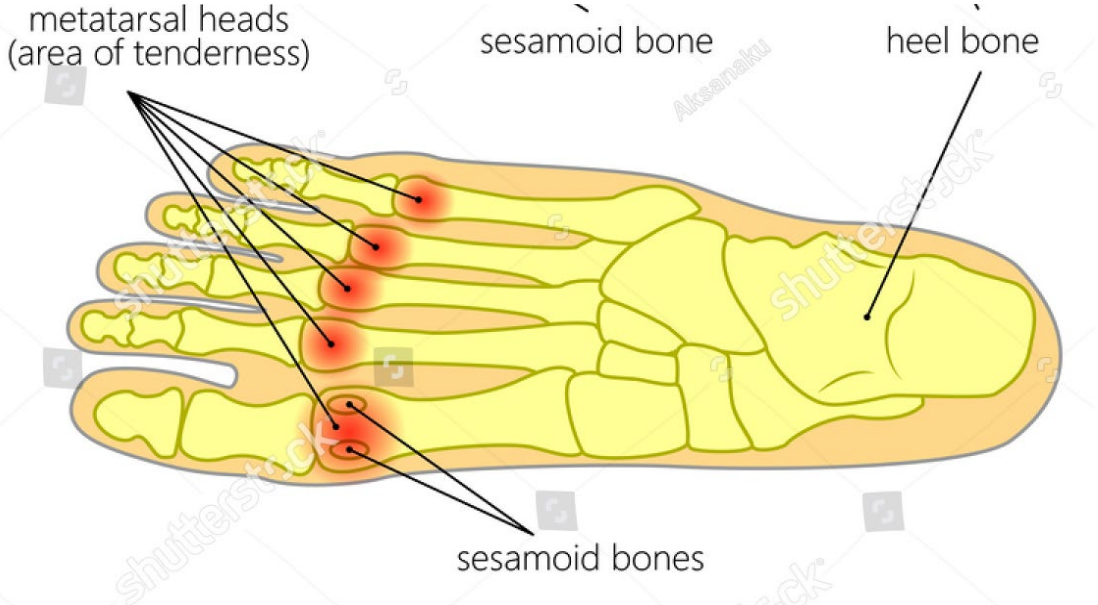


3D marker positions (200HZ)

Reference : Description of the GaitPhase Database

資料蒐集

蹠骨


metatarsal heads
(area of tenderness)
sesamoid bone
heel bone
sesamoid bones

資料蒐集

```
#0.7資料
datasix1  =  pd.read_csv("GP1_0.7_marker.csv")
datasix1.head()
```

	L_FCC_x	L_FM1_x	L_FM2_x	L_FM5_x	R_FCC_x
0	0.628926	0.836563	0.840168	0.823058	0.888503
1	0.626188	0.833782	0.837441	0.820210	0.894769
2	0.623270	0.830821	0.834532	0.817181	0.901406
3	0.620190	0.827697	0.831457	0.813991	0.908358
4	0.616968	0.824431	0.828239	0.810662	0.915562

特徵24個 每一個有1200筆資料	
雙腳	左腳、右腳
Marker	1、2、5、Achilles tendon
3軸	x: posterior-anterior direction (前後方向)
	y: right-left direction (左右方向)
	z: inferior superior (vertical) direction (上下垂直方向)

資料合併

每個速率資料都是獨立csv檔

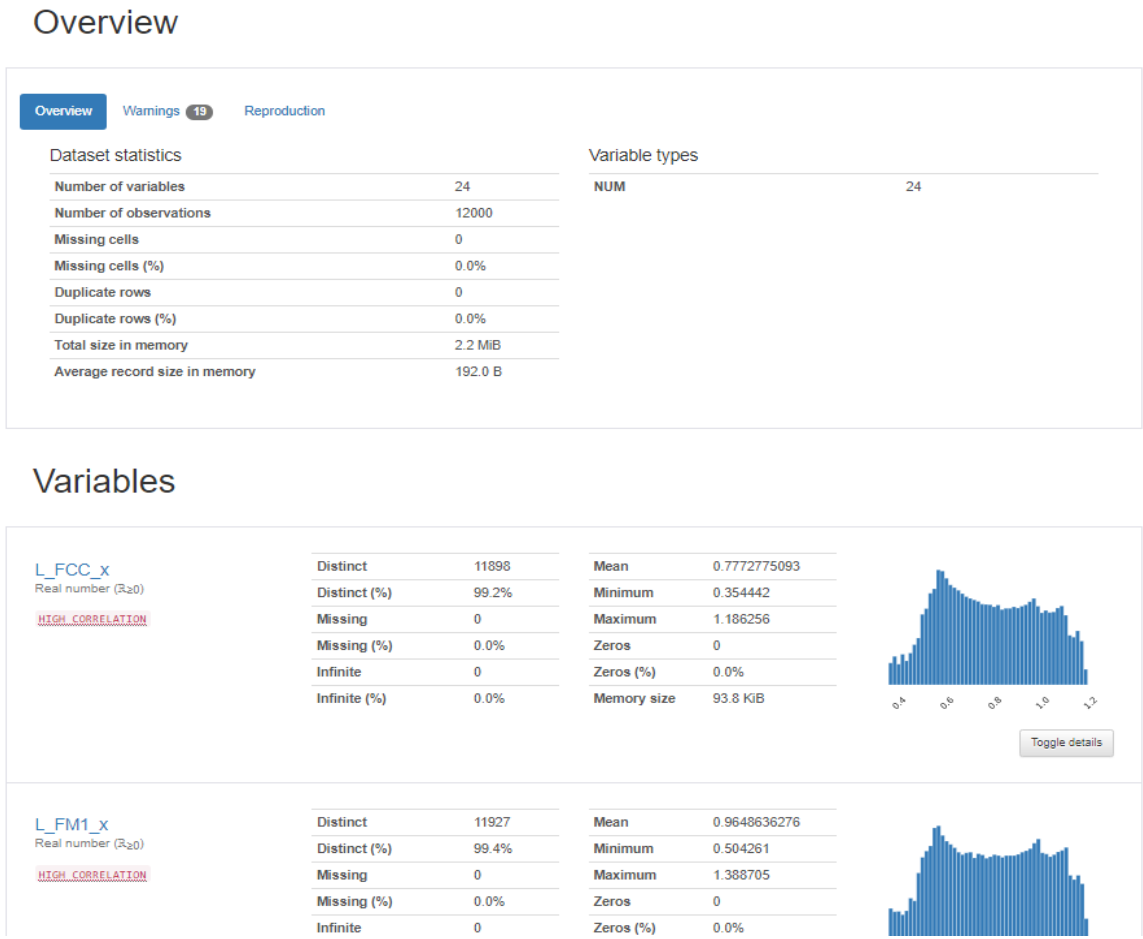
所以進行資料合併並分類命名 0、1

```
data1  =  pd.DataFrame({'secim':np.zeros(12000)}
data2  =  pd.DataFrame({'secim':np.ones(12000)})



data2  =  pd.concat([dataseven2,data2],axis=1)
data1  =  pd.concat([datasix1,data1],axis=1)

data  =  data1.append(data2,ignore_index=True)
data.head()
```

資料分析

```
pip install pandas.profiling # 安裝profiling套件
```

```
import pandas_profiling as pp

##0.7
pro = pp.ProfileReport(datasix1)
pro.to_file('gait0.7_output.html') # 以網頁形式輸出
```



資料分析

比較同一個marker點位資料

X軸 (前後) 資料必須呈現均勻

Z軸 (上下) 資料呈現偏態

近一步確認資料正確性

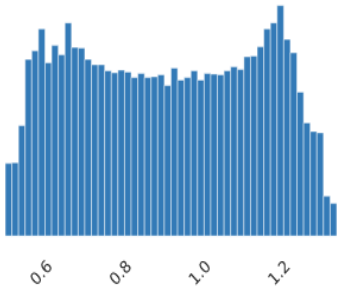
L_FM1_x

Real number (ℝ≥0)

HIGH CORRELATION

Distinct	11905
Distinct (%)	99.2%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	0.916007043
Minimum	0.507758
Maximum	1.343911
Zeros	0
Zeros (%)	0.0%
Memory size	93.8 KiB



Toggle details

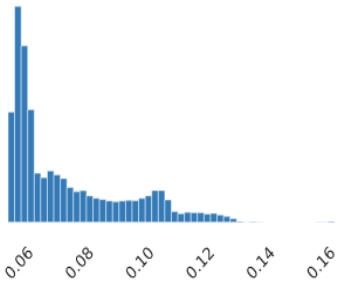
L_FM1_z

Real number (ℝ≥0)

HIGH CORRELATION

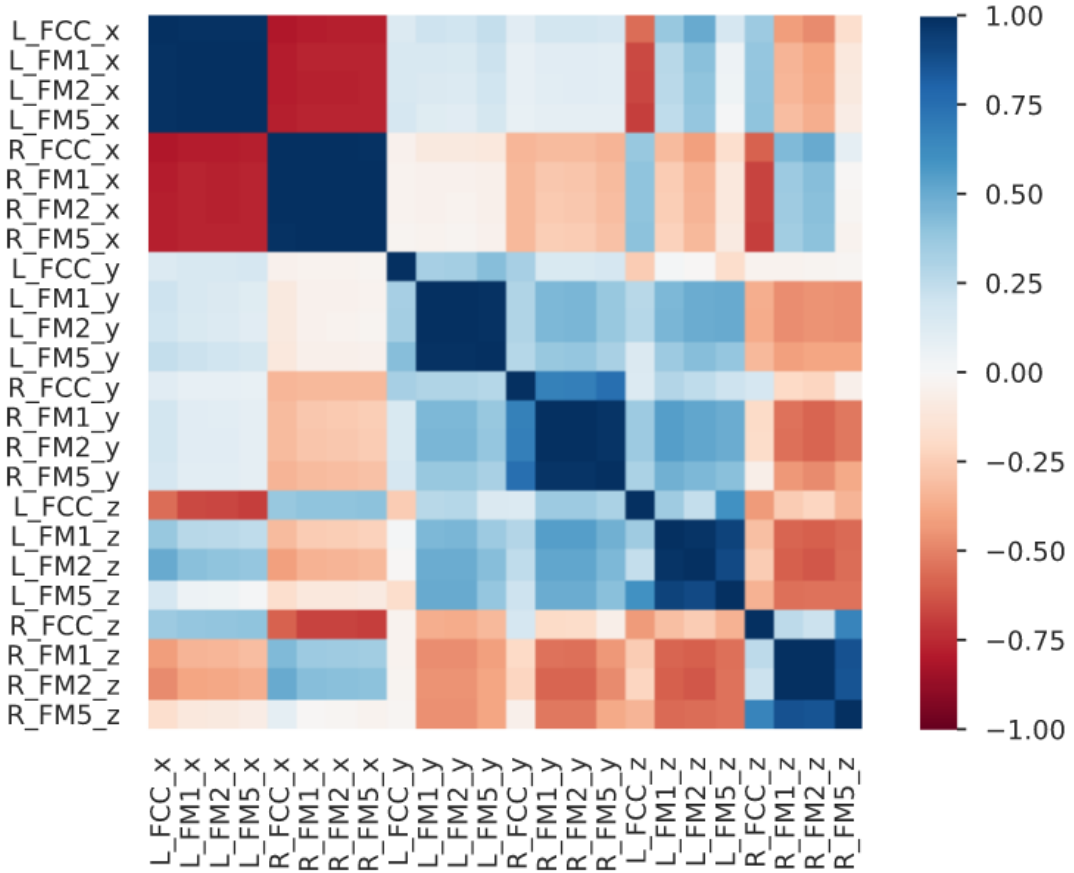
Distinct	10103
Distinct (%)	84.2%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	0.07608154675
Minimum	0.056193
Maximum	0.164298
Zeros	0
Zeros (%)	0.0%
Memory size	93.8 KiB



Toggle details

特徵選取



不顯著特徵
L_FCC_y
R_FCC_y
L_FCC_z
R_FCC_z
L_FM5_z

模型	準確率	決策
SVM 24特徵	0.915	V
SVM 19特徵	0.761	

資料區分

Training
70%
16800

+

Testing
30%
7200

```
from  sklearn.model_selection  import  train_test_split
x_train,  x_test,y_train,y_test  =  train_test_split(x,y,test_size=0.3,random_state=1)
```

03 預測模型設定

SVM
Logistic regression
XGBoost
DNN

模型實驗流程

相同的人

不同速率

選擇0.7 、0.8 m/s

超參數調整

SVM

Logistic

XGBoost

DNN

選出最好的模型後

進行泛化能力測試

SVM 超參數調整

超參數	結果	超參數	結果	超參數	結果	超參數	結果
C=1	0.728	C=1	0.765	C=100	0.914	C=1000	0.988
kernel='rbf'		kernel='rbf'		kernel='rbf'		kernel='rbf'	
gamma='auto'		gamma='scale'		gamma='scale'		gamma='scale'	

```
acc_train of svm is : 0.9866071428571429
acc_test of svm is : 0.9880555555555556
```

Logistic 超參數調整

超參數	結果
C=1	0.721
solver='liblinear'	
max_iter=10	

超參數	結果
C=1000	0.838
solver='liblinear'	
max_iter=10	

超參數	結果
C=1000	0.915
solver='liblinear'	
max_iter=100	

超參數	結果
solver='liblinear'	0.915
solver='sag'	0.871
lbfgs	0.896
newton-cg	0.914

```
train accuracy for Log Regressin is  0.9161309523809524
test accuracy for Log Regressin is  0.9155555555555556
```

XGBoost 超參數調整

超參數	結果	超參數	結果	超參數	結果	超參數	結果
learning_rate=0.1	0.936	learning_rate=0.2	0.975	learning_rate=0.2	0.997	learning_rate=0.2	0.999
max_depth=3		max_depth=3		max_depth=4		max_depth=5	

```
acc_train of XGB is : 1.0
acc_test of XGB is : 0.9994444444444445
```

DNN 架構

```
model = keras.Sequential([
        keras.layers.Dense(512,  activation='relu'),
        keras.layers.Dense(128,  activation='relu'),
        keras.layers.Dense(64,  activation='relu'),
        keras.layers.Dense(64,  activation='relu'),
        keras.layers.Dense(10,  activation='softmax')
])
```

```
#optimizer: descent algorithm
#loss: objective loss function
model.compile(optimizer='adam',
                        loss='sparse_categorical_crossentropy',
                        metrics=['accuracy'])
```

DNN 超參數調整

超參數	結果	超參數	結果	超參數	結果
epochs=100	0.867	epochs=500	0.983	epochs=500	0.999
batch_size=25		batch_size=25		batch_size=50	

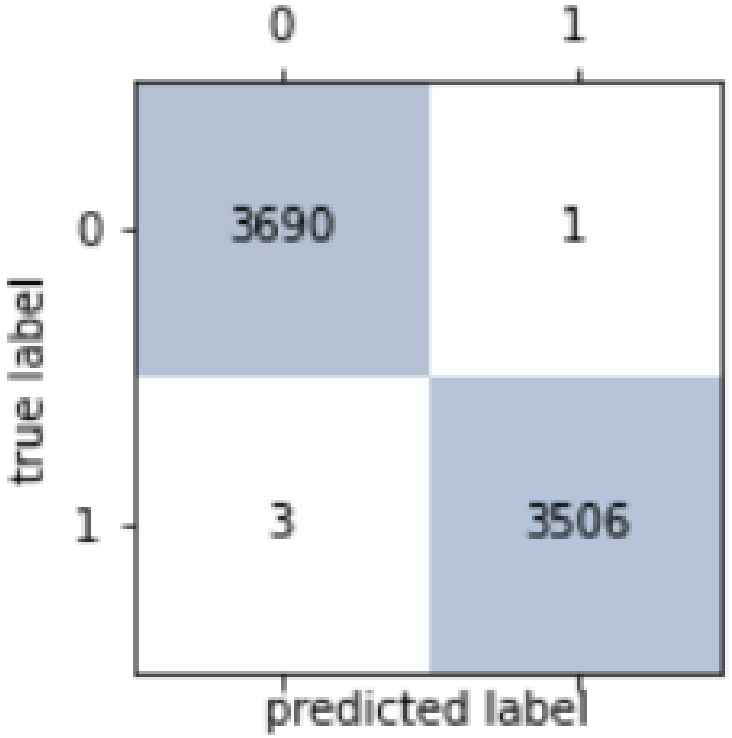
```
DNN Train accuracy: 0.9998809695243835

DNN Test accuracy: 0.9994444251060486
```

04 分析與模型效度驗證

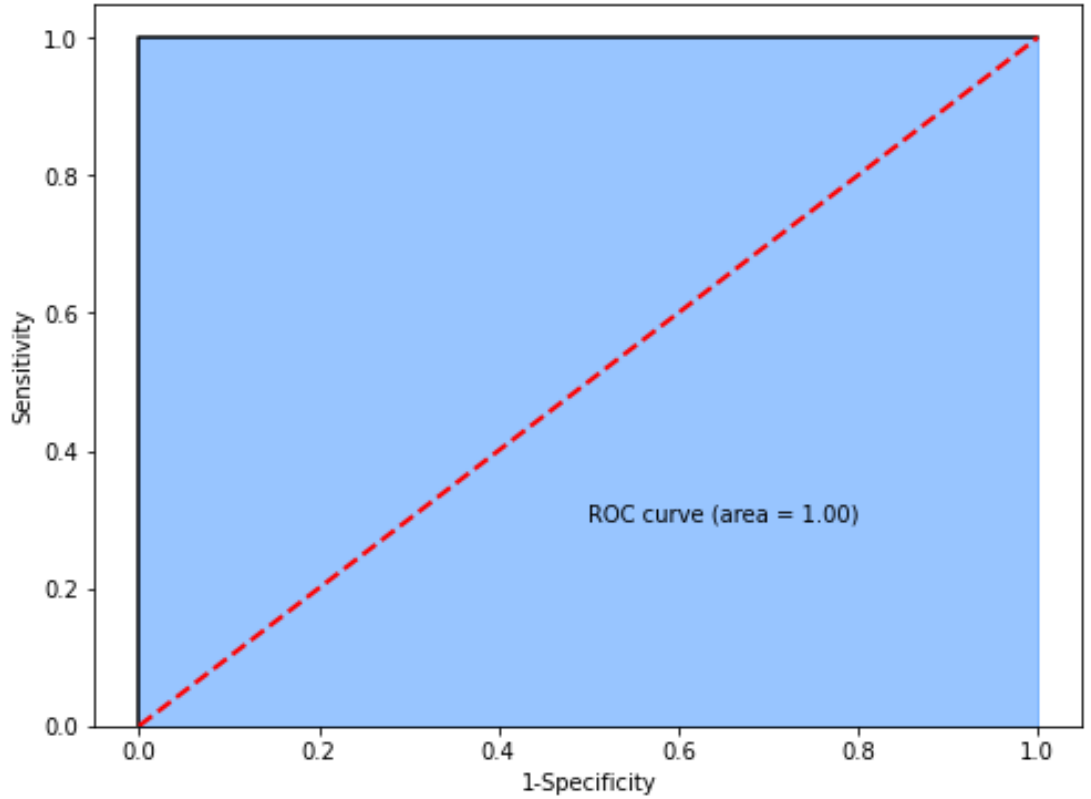
模型比較

模型	準確率	訓練速度	決策
SVM	0.988	1分30秒	
Logistic	0.915	1分內	
XGBoost	0.999	1分內	✔
DNN	0.999	6分	

模型評估



混淆矩陣



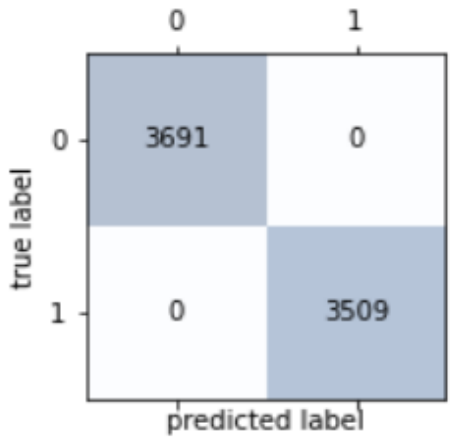
roc/auc計算

泛化能力評估

編號1受測者
1.2、1.3 m/s

train_accuracy: 1.0

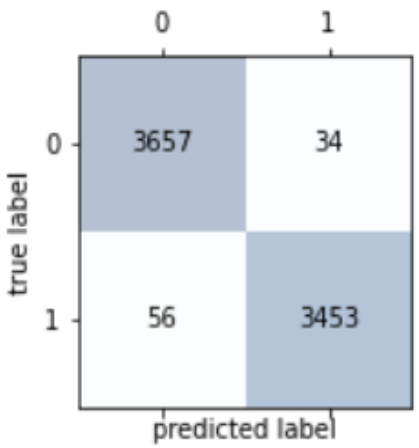
test_accuracy: 1.0



編號1受測者
1.4、1.6 m/s

train_accuracy: 0.9998214285714285

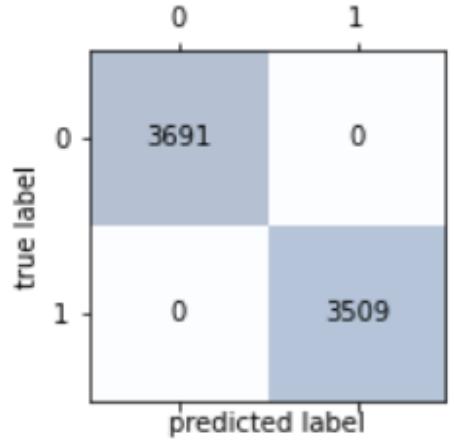
test_accuracy: 0.9875



編號2受測者
0.7、0.9 m/s

train_accuracy: 1.0

test_accuracy: 1.0

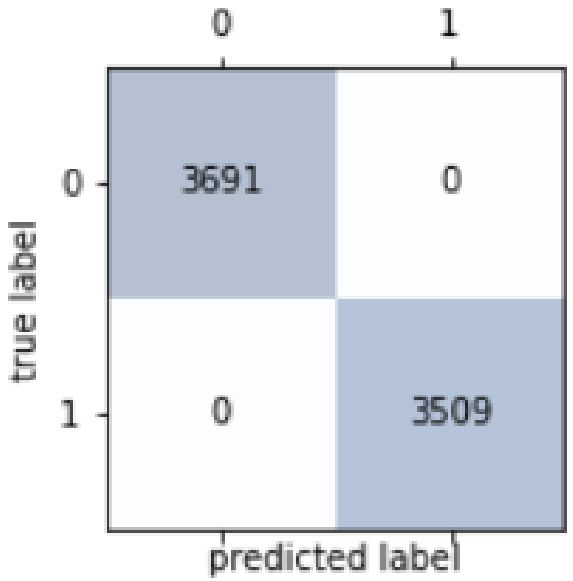


不同身份辨識

- 編號1 和 編號2 兩位不同的受測者
- 0.7 m/s
- 成功在相同速度下，辨識不同人

```
#第一位  0.7資料
datasix1  =  pd.read_csv("GP1_0.7_marker.csv")
datasix1.head()
#第二位  0.7資料
dataseven2  =  pd.read_csv("GP2_0.7_marker.csv")
dataseven2.head()
```

```
train_accuracy: 1.0

test_accuracy: 1.0
```



05 結論與未來展望

結論與未來展望

- 成功使用分類模型、深度學習分辨出速率差異
- 成功利用步態特徵的獨特性，在相同速率下辨識不同的人
- XGBoost 是個不錯的分類模型 (準確率高、速度快)
- 目前可以分辨速率，希望未來可以透過步態的週期性，結合時間序列型模型，去預測下一步的步態速率變化