Intelligent Integration of Enterprise

Project 2

登山攻頂成功機率預測模型之建構與驗證

Group 5

107034703 陳家安

108034537 黃子芸

108034540 吳欣蓉

108034545 黃唯真

指導教授：邱銘傳 教授

目錄

一、 研究主題說明

我們將本團隊定位為一個提供登山相關服務的企業，主要販售登山的套裝行程(如派遣嚮導出團)、登山裝備...等。在多種影響業績的因素中，天氣狀況是影響能否安全且成功登頂以及其他周邊商品販售情況的關鍵因素，因此我們希望能透過蒐集如氣溫、相對溼度、風速等相關天氣資料，搭配公開資料集針對這些因素下所記錄的登山成功率等歷史資料，來建立深度學習的預測模型並預測未來登山的成功機率，並以此為基礎，來訂定登山套裝行程的日期，有效降低行程因天氣因素取消的風險，提升顧客參與度。

針對此研究主題，5W1H 分析說明如下表：

表一、5W1H 分析

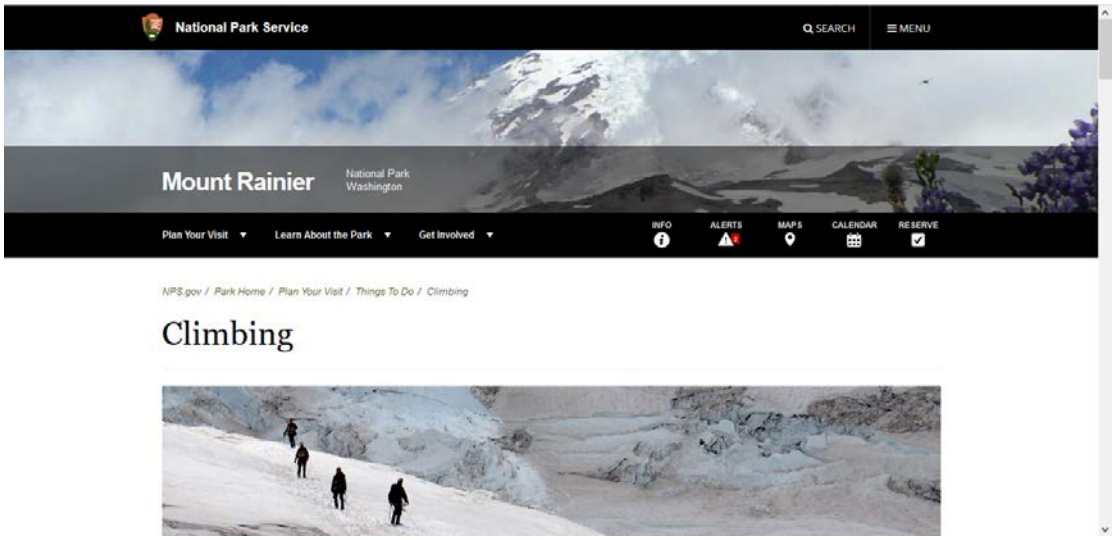
項目	內容
What	登山客無法自行判斷天氣狀況及路線選擇對於攻頂成功機率的影響
Where	Mount Rainier(瑞尼爾山)，分析山間天氣與各登山路線資料
Who	登山客、登山數據預測團隊
When	登山客預計登山日期
Why	透過深度學習分析登山路線與天氣各因素影響，提供登山客欲攻頂日的成功機率，為登山客考量是否應該改期登山，降低登山風險
How	登山路線與天氣狀況的資料前處理及 DNN 模型訓練

二、 資料分析與前處理

1. 資料來源與結構分析

我們由 Nation Park Service 官方網站(圖一)獲取 Mount Rainier(瑞尼爾山)共 21 條登山路線(如 Disappointment Cleaver、Little Tahoma、Kautz Glacier…..等)的登頂嘗試次數和成功率的歷史資料，共 4087 筆資料，資料格式如圖二所示。

此外為了結合氣象狀況進行分析，我們也蒐集了此山間的天氣歷史資料等公開資料，並依日期詳細記錄天氣狀況，包含平均氣溫、平均濕度、平均風速、平均太陽輻射等影響氣候的因素，以及身上攜帶設備所用電池的電壓，資料記錄格式如圖三所示。



圖一、Nation Park Service 官方網站

Date	Route	Attempted	Succeeded	Success Percentage
11/27/2015	Disappointment Cleaver	2	0	0
11/21/2015	Disappointment Cleaver	3	0	0
10/15/2015	Disappointment Cleaver	2	0	0
10/13/2015	Little Tahoma	8	0	0
10/9/2015	Disappointment Cleaver	2	0	0

圖二、登山成功機率歷史資料

Date	Battery Vo	Temperature AVG	Relative Humidity AVG	Wind Speed Daily AVG	Wind Direction AVG	Solare Radi
12/31/2015	13.845	19.06291667	21.87083333	21.97779167	62.32583333	84.91529
12/30/2015	13.82292	14.63120833	18.49383333	3.540541667	121.5054167	86.19283
12/29/2015	13.83458	6.614291667	34.07291667	0	130.2916667	85.10092
12/28/2015	13.71042	8.687041667	70.55791667	0	164.68375	86.24125
12/27/2015	13.3625	14.14041667	95.75416667	0	268.4791667	31.09071

圖三、Mount Rainier 山間氣象歷史資料

結合以上兩種資料來源，我們將透過資料前處理並應用 DNN 的深度學習方法來訓練模型，進行登山攻頂成功率的預測，並期望最後準確率有超過六成的表現。

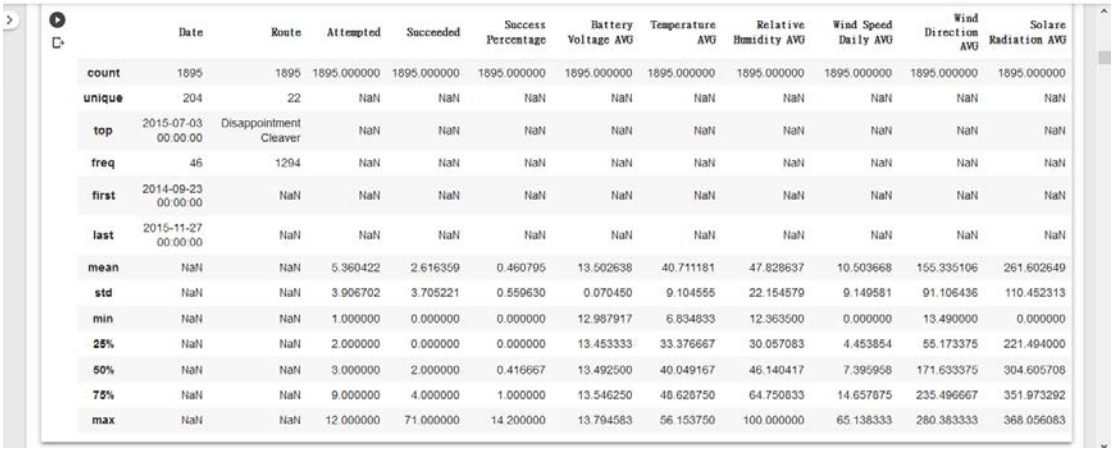
2. 資料前處理

(1) **Step1**：先將兩份資料檔(登山路線與天氣狀況)依據日期合併，列出合併結果

■ Code >

```
# Merge climbing and weather datasets on the Date columns
dfm = df.merge(wth, on='Date')
# Then, sort Date values ascending and reset index
dfm.sort_values('Date', inplace=True)
dfm.reset_index(drop=True, inplace=True)
dfm.head(3)
dfm.info()
dfm.describe(include='all')
```

■ Output >



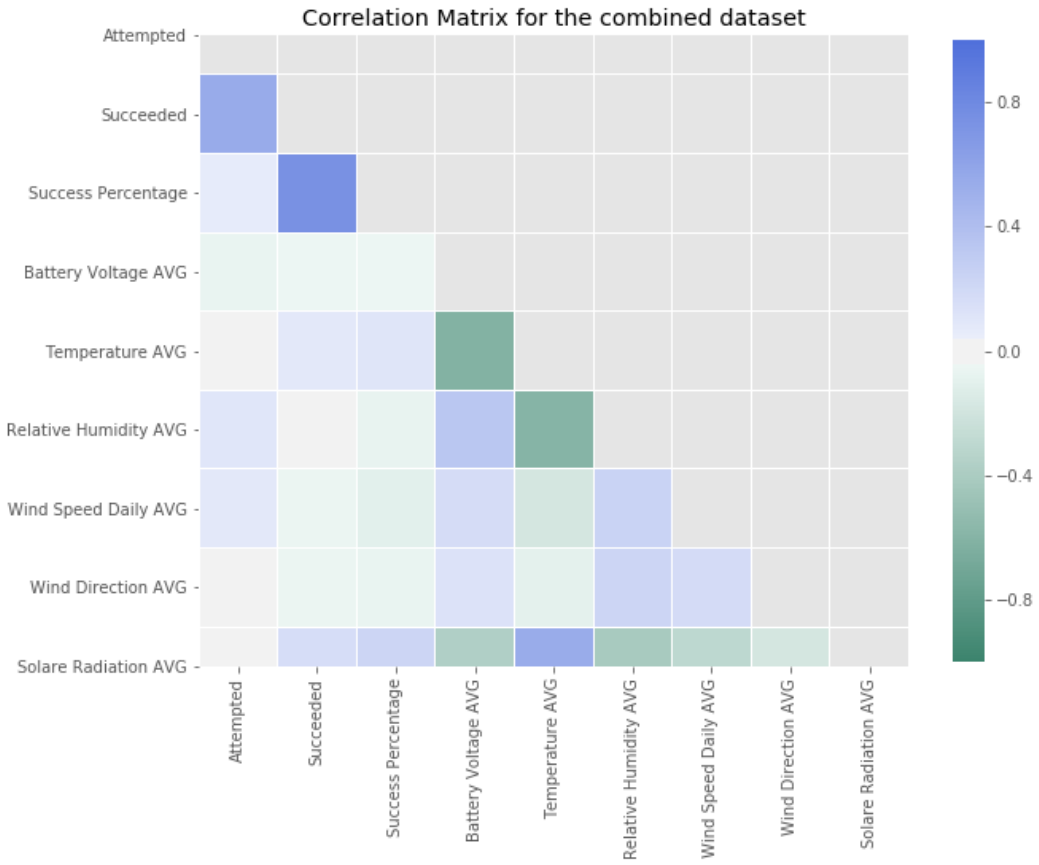
	Date	Route	Attempted	Succeeded	Success Percentage	Battery Voltage AVG	Temperature AVG	Relative Humidity AVG	Wind Speed Daily AVG	Wind Direction AVG	Solare Radiation AVG
count	1895	1895	1895.000000	1895.000000	1895.000000	1895.000000	1895.000000	1895.000000	1895.000000	1895.000000	1895.000000
unique	204	22	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	2015-07-03 00:00:00	Disappointment Cleaver	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	46	1294	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
first	2014-09-23 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
last	2015-11-27 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	5.360422	2.616359	0.460795	13.502638	40.711181	47.828637	10.503668	155.335106	261.602649
std	NaN	NaN	3.906702	3.705221	0.559630	0.070450	9.104555	22.154579	9.149581	91.106436	110.452313
min	NaN	NaN	1.000000	0.000000	0.000000	12.987917	6.834833	12.363500	0.000000	13.490000	0.000000
25%	NaN	NaN	2.000000	0.000000	0.000000	13.453333	33.376667	30.057083	4.453854	55.173375	221.494000
50%	NaN	NaN	3.000000	2.000000	0.416667	13.492500	40.049167	46.140417	7.395958	171.633375	304.605708
75%	NaN	NaN	9.000000	4.000000	1.000000	13.546250	48.628750	64.750833	14.657875	235.496667	351.973292
max	NaN	NaN	12.000000	71.000000	14.200000	13.794583	56.153750	100.000000	65.138333	280.383333	368.056083

(2) Step2：將資料中的參數進行參數間相關性的比較，做出相關矩陣圖，找出較高度的正負相關性，以便後續資料篩選處理，如下圖

■ Code >

```
[12] # Now plot a correlation matrix:

     corr = df_countable.corr()
     mask = np.zeros_like(corr, dtype=np.bool)
     mask[np.triu_indices_from(mask)] = True

     f, ax = plt.subplots(figsize=(10, 9))

     cmap = sns.diverging_palette(210, 5, as_cmap=True)

     sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
                 square=True, linewidths=.1, cbar_kws={"shrink": .8})
     plt.title('Correlation Matrix for the combined dataset')

     plt.show()
```

■ Output >



(3) Step3：數據修正與調整

1. 由原始資料中我們發現，「成功機率(Success Percentage)」有 14.2 的值，這顯然是一個異常值（機率應介於 0 到 1 之間），而為了保留觀察值，我們按以下規則進行數據修改：數值> 1 時，我們將其設為 1。
2. 將成功次數大於嘗試次數的數據，改為成功次數 = 嘗試次數
3. 「成功次數(Succeeded)」和「成功機率(Success Percentage)」之間（在邏輯上）有很強的相關性，因此我們將刪除「成功次數」那欄的數據。
4. 「Battery Voltage AVG」表示電池電壓，並且似乎與平均溫度負相關。 但我們選擇保留此列。

■ Code >

```
# Let's first rename the target feature column (for convenience purpose only):
dfm.rename(columns={'Success Percentage' : 'SuccPerc'}, inplace=True)

# Bring 'Succeeded' to be equal 'Attempted' where 'Succeeded' > 'Attempted'
dfm.loc[dfm['Succeeded'] > dfm['Attempted'], 'Succeeded'] = dfm['Attempted']

# Now, locate the outliers and bring them to 1.
dfm.loc[dfm['SuccPerc'] > 1, 'SuccPerc'] = 1
dfm['SuccPerc'] = dfm['SuccPerc'].round(2)
```

(4) Step4：將數據的日期設為索引，畫出日期與成功機率以及其他因子的分佈圖

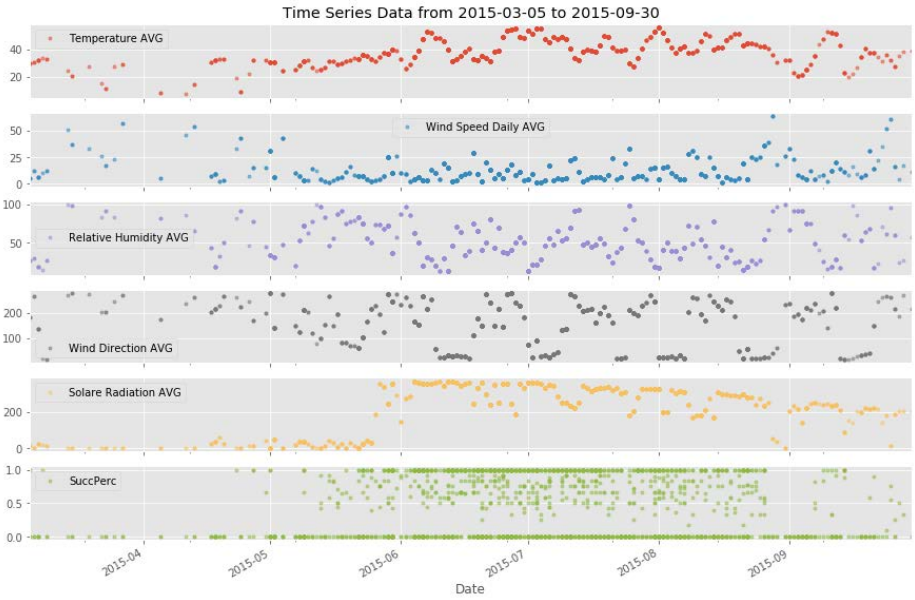
■ Code >

```
# Set Date as index
ts = dfm.set_index('Date')
# Sort it
ts.sort_index(inplace=True)

print('The records about climbing successfulness from %s to %s' % (dfm.Date.dt.date.min(), dfm.Date.dt.date.max()))
```

```
The records about climbing successfulness from 2014-09-23 to 2015-11-27
```

```
# Brief observation:
ts['SuccPerc'].plot(style='k.', figsize = (12,7))

plt.title('Climbing success rate distribution')
plt.ylabel('Success Percentage')
plt.yticks(np.linspace(0,1,21))
plt.show()

# Here we see that a climbing season starts roughly in March and ends in October. So, let's consider only this interval
# Set the range
ts_mo = ts['2015-03-01':'2015-10-01']
ts_mo.shape
```

■ Output >



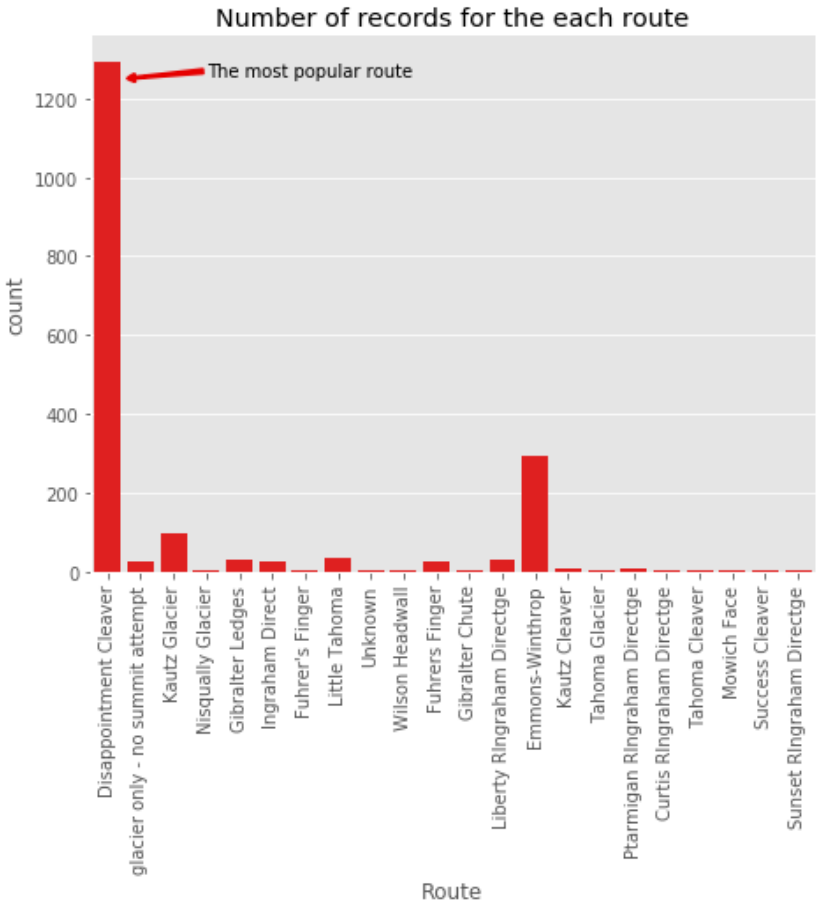
由觀察可以發現，登山的旺季約為每年 3~10 月，因此在後續模型的訓練，我們將只使用此時段的資料進行分析。

(5) Step5：統計各登山路線的登山次數，將重複的路線資訊合併，以利後續的資料計算，並將一些資料較少的登山路線剃除，後續不進行建模。

■ Code >

```
g = sns.catplot(x = 'Route', data=dfm, aspect=1.5, kind="count", color="r")
g.set_xticklabels(rotation=90)
plt.title('Number of records for the each route')
plt.annotate('The most popular route',
             xy=(3,1250),
             rotation=0,
             va='bottom',
             ha='left',
            )

plt.annotate('',
             xy=(0.5, 1250),
             xytext=(3, 1270),
             xycoords='data',
             arrowprops=dict(arrowstyle='fancy ,head_length=0.4,head_width=0.4,tail_width=0.2',
                             connectionstyle='arc3',
                             color='xkcd:red',
                             lw=2
                            ))
plt.show()
```

■ Output >



三、 預測模型架構

1. 模型架構與應用方法論

模型建構步驟如下：

從資料前處理的過程中可以發現到，每條登山路線其難易程度與天氣狀況皆不相同。因此我們分別對各條路線進行建模(若該路線資料少於 5 筆則不進行建模分析)，並設法提高模型預測準確率。

本模型使用 DNN (deep neural network，深度神經網路)演算法建構，此演算法可對數值型資料進型特徵學習。下段將對模型訓練及參數調整做進一步說明。

2. 參數設定與模型訓練過程

(1) 模型建立

以下將詳述模型建構過程。

Step 1：將各路線的資料拆分成獨立的 data frame

Step 2：將資料切分成訓練及測試集

```
train_test_split(np.asarray(temp.iloc[:,1:-2]), np.asarray(temp['SuccessPercentage']),
```

Step 3：建立 DNN 模型

```
model = Sequential()
model.add(Dense(12, activation='relu', input_shape=(6,)))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

input_shape = 6，表示取 6 個參數做為輸入變數 X，包括嘗試次數、平均電壓、平均溫度、平均濕度、平均風速、平均風向、平均輻射量

(2) 參數調整

以資料量最多的路線其模型為主，進行參數調整。

原始模型準確率：0.46

	loss	accuracy
df_disappointmentcleaver	0.987529	0.455598

I. 增加 model units

```
model = Sequential()
model.add(Dense(24, activation='relu', input_shape=(6,)))
model.add(Dense(12, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

準確率變化：0.46→0.43

II. 改變 activation function

```
model = Sequential()
model.add(Dense(12, activation='sigmoid', input_shape=(6,)))
model.add(Dense(8, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```

準確率變化：0.46→0.44

III. 增加一層 hidden layer

```
model = Sequential()
model.add(Dense(12, activation='relu', input_shape=(6,)))
model.add(Dense(8, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

準確率變化：0.46 → 0.45

IV. 將 model fitting 次數提高為 10 次

```
model.fit(x_tr, y_tr, epochs=10, batch_size=1, verbose=1)
```

準確率：0.46→0.43

綜合以上結果，參數調整後模型預測準確率不但沒有增加，反而還些微下降，因此維持原本模型的參數設定。

(3) 模型訓練

在訓練過程中，設定每一模型 fitting 均跑 5 個 epoch。

```
model.fit(x_tr, y_tr, epochs=5, batch_size=1, verbose=1)
```

四、 模型校度驗證

取得測試集準確率：

```
score = model.evaluate(x_tst, y_tst)
print('loss:',score[0] ,'accuracy', score[1])
```

各路線模型預測準確率，如下圖。整體平均預測準確率達 **63%**。

	loss	accuracy
df_disappointmentcleaver	0.987529	0.455598
df_emmons_winthrop	0.764610	0.440678
df_fuhrersfinger	1.971632	0.714286
df_gibralterledges	0.740012	0.833333
df_ingrahamdirect	1.140211	0.800000
df_kautzcleaver	41.850819	0.500000
df_kautzglacier	0.907372	0.300000
df_libertyringrahamdirectge	3.866420	0.500000
df_littletahoma	0.757317	0.625000
df_ptarmiganringrahamdirectge	10.142046	0.500000
df_glacieronly_nosummitattempt	0.154129	1.000000

五、 小結與未來展望

由參數調整的結果發現增加模型層數等方法都無法改善預測準確率，因此未來可以從資料前處理著手，對資料先進行正規化，或許可提高準確率。此外所蒐集的資料中部份路線的資料筆數較少(少於 50 筆)，可以進一步探討對於資料量少的預測問題是否適合使用類神經網路演算法，或許機器學習的方法就能有很好的預測能力。