Data Science HW1

a.

使用 jupyter notebook 執行程式，環境使用為 macOS Catalina。

b.

- **讀入資料集 Train 和 Test**

Read data files

```
Train = pd.read_csv('train.csv')
Test = pd.read_csv('test.csv')
Train
```

- **對資料集做 preprocessing。**

1. 我先將 Attribute1 的日期欄位只取月份的部分。

Take only month in Attribute 1

```
# train
Train['Attribute1']= pd.to_datetime(Train['Attribute1'])
Train['Attribute1'] = pd.DatetimeIndex(Train['Attribute1']).month

# test
Test['Attribute1']= pd.to_datetime(Test['Attribute1'])
Test['Attribute1'] = pd.DatetimeIndex(Test['Attribute1']).month
```

2. 把有 NAN 的 row 都捨棄掉。（發現捨棄比替換值準）

處理NAN

```
# 訓練data
Train = Train.dropna()
# 測試data
Test = Test.dropna()
```

3. 將 Yes No 轉為 0 1，風向欄位做 one hot encoding

處理字串欄位

字串轉數字

```
from sklearn.preprocessing import LabelEncoder
```

```
##.  Train Data ##
# 字串轉數字
gle = LabelEncoder()
Train['Attribute22'] = gle.fit_transform( Train['Attribute22'] )
Train['Attribute23'] = gle.fit_transform( Train['Attribute23'] )

## Test Data  ##
# 字串轉數字
Test['Attribute22'] = gle.fit_transform( Test['Attribute22'] )

# One-hot
Train = pd.get_dummies(Train)
Test = pd.get_dummies(Test)
Train
```

4. 因為資料集明天會不會下雨的 yes 和 no，數量相差太大，所以將資料做平衡

Balance Data

```
from sklearn.utils import resample

df_majority = Train[Train.Attribute23==0]
df_minority = Train[Train.Attribute23==1]

print(len(df_majority), len(df_minority))
print(Train['Attribute23'].value_counts())

# Downsample majority class
df_majority_downsampled = resample(df_majority,
                                   replace=False,    # sample without replacement
                                   n_samples=1225,     # to match minority class
                                   random_state=5659) # reproducible results

# Combine minority class with downsampled majority class
Train = pd.concat([df_majority_downsampled, df_minority])

# Display new class counts
print(Train['Attribute23'].value_counts())
```

5. 把資料打散，試圖獲得更好的效果

打散資料

```
from sklearn.utils import shuffle
Train = shuffle(Train)
```

6. 切割 Data 和 Target（訓練模型時所需）

切割 Data 與 Target

```
Target = pd.DataFrame(Train['Attribute23'])
Data = Train.drop(columns = ['Attribute23'],axis = 0)
```

做完 preprocessing 後的 Data 和 Target:

```
Data
```

	Attribute1	Attribute2	Attribute3	Attribute4	Attribute5	Attribute6	Attribute7	Attribute9	Attribute12	Attribute13	...	Attribute11_NNW	Attribute11_NW
13914	6	32	10.8	19.8	0.0	3.4	6.2	41.0	28.0	17.0	...	0	0
9157	10	44	12.1	26.2	0.0	6.6	5.3	69.0	31.0	37.0	...	0	0
16665	9	13	24.5	33.0	8.6	5.2	10.6	31.0	11.0	17.0	...	1	0
10906	9	39	20.4	29.1	0.6	6.6	2.8	56.0	22.0	15.0	...	0	0
9912	6	7	14.7	22.1	1.8	0.8	2.1	22.0	7.0	9.0	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
7569	10	19	9.1	12.5	0.0	4.4	1.2	41.0	19.0	22.0	...	0	0
15934	10	3	10.7	28.8	0.0	9.0	12.2	50.0	15.0	17.0	...	0	0
7642	7	19	12.2	18.6	0.0	4.8	7.3	89.0	44.0	46.0	...	0	1
14262	8	31	12.4	21.7	0.6	2.6	5.2	48.0	11.0	26.0	...	0	1
14046	8	32	11.6	20.7	0.0	4.6	3.7	46.0	22.0	20.0	...	0	0

2450 rows × 67 columns

```
Target
```

	Attribute23
13914	0
9157	1
16665	0
10906	0
9912	1
...	...
7569	1
15934	0
7642	1
14262	1
14046	1

2450 rows × 1 columns

- **製作模型**

先將訓練用的 Train data 以 5:5 切割成訓練與測試集，目的只是為了方便自己做準確率的測試。（訓練時不會用到這邊的 data_train 和 target_train）

Train Test split

```
from sklearn.model_selection import train_test_split

data_train,data_test,target_train,target_test = train_test_split(Data, Target,test_size = 0.5)
print('Train Data Shape: ',data_train.shape, '\nTrain target Shape: ',target_train.shape)
print('Test Data Shape: ',data_test.shape, '\nTest target Shape: ',target_test.shape)
```

不知道要用哪個模型效果會比較好，所以做了幾個，這邊只介紹最終採用的 Multi-layer Perceptron。

選擇模型我是參考下方連結：

https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

但最終方法是另外找到的。

1. Neural Network models （Multi-layer perceptron）

Neural Network models

```
from sklearn.neural_network import MLPClassifier
# nn = MLPClassifier(activation= 'logistic' ,solver= 'adam' ,alpha= 0.0001 ,learning_rate= 'adaptive' ,learning_rate
nn = MLPClassifier(solver='adam', activation='logistic',alpha= 0.0001,learning_rate= 'adaptive' , hidden_layer_sizes
nn.fit(Data, Target.values.ravel())
nn_pred = nn.predict(Test)
print('Test Accurancy:', nn.score(data_test,target_test))
```

演算法流程：

MLP 為一種監督式學習的演算法，藉由輸入特徵 X=x1,x2,.....,xm 和目標值 Y，此算法將可以使用非線性近似將資料分類或進行迴歸運算。MLP 可以在輸入層與輸出層中間插入許多非線性層。

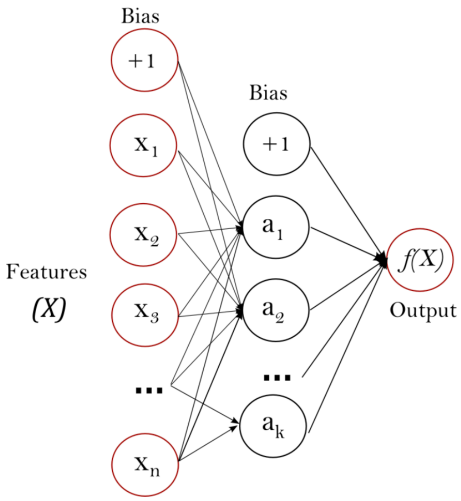

圖1:包含一層隱藏層的MLP

最左邊那層稱作**輸入層**，為一個神經元集合代表輸入的特徵。每個神經元在**隱藏層**會根據前一層的輸出的結果再乘上權重，做為此層的輸入 w1x1 + w2x2 +...+ wmxm 。再使用非線性的 activation function 做轉換（例如：tanh, sigmoid, logistic... ）。最終獲得**輸出層**的 output。

- 輸出 CSV

輸出csv

```
def output_csv(pred,filename):
    df = pd.DataFrame(pred)
    df.index = df.index.astype(float)
    df.to_csv(filename, header = ['ans'], index_label = 'id')a

output_csv(knn_pred, 'knn.csv')
output_csv(clf_pred, 'dictree.csv')
output_csv(gnb_pred, 'gnb.csv')
output_csv(nn_pred, 'nn.csv')
```