TITANIC與IRIS資料集實作

TITANIC

載入資料集

- 1.將資料集載入

```
#  載入資料集
df=  pd.read_csv("./titanic.csv")
```

- 2.查看資料集內容

  查看載入後的資料集前五筆確認有哪些內容以及pandas已讀取的資料集內容狀況

```
#  查看資料集內容
print(df.head())
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

調整資料集-了解資料集

- 透過老師提供的資料可以了解Titanic資料集的詳細各項內容

- 「train.csv」行資料說明

行名稱	中文	說明
PassengerId	乘客代號	共 891 筆
Survived	生還狀況	0：死亡，1：生還
Pclass	艙等	艙位等級，即一等艙、二等艙、三等艙，以 1,2,3 表示。1 最高，3 最低
Name	姓名	
Sex	性別	male：男性，female：女性
Age	年齡	
SibSp	手足及配偶數	手足、配偶也在船上的數量。 Sib (Sibling)：兄弟姐妹，Sp (Spouse)：配偶
Parch	雙親及子女數	雙親或子女也在船上的數量，若只跟保姆搭船則為 0
Ticket	船票編號	
Fare	票價	
Cabin	客艙編號	
Embarked	登船港口	C：Cherbourg，Q：Queenstown，S：Southampton

調整資料集-篩選資料集

- 刪除認為對辨識沒有幫助的欄位

使用drop函式對認為與無助於辨識結果的項目刪除。

```
#  篩選資料集
df  =  df.drop(["Name",  "Ticket",  "Cabin"],  axis=1)
#  查看資料集內容
print(df.head())
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S

請注意，進行刪除時需要指定座標軸，在pandas函式的各項操作中，axis= 0代表“橫軸”、axis= 1代表”縱軸”。

此code即是將Name, Ticket, Cabin欄位內縱軸的資料通通清除。

調整資料集-檢查資料集

- 進行資料集檢查

  檢查資料集的各個特徵。使用describe檢查是統計摘要看看各欄位的百分比分布狀況

  也可使用info來查詢是否有缺漏以及變數型態，右圖中我們可以發現Sex(性別)以及Embarked(登船口岸)欄位並非整數或浮點數型態，這樣的資料集是無法利用的。

```
#  檢查資料集的各項特徵
print(df.describe())  #檢查資料集統計摘要
print(df.info())  #檢查各欄位的變數值型態以及遺漏值狀況
```

```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Sex          891 non-null    object
 4   Age          714 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Fare         891 non-null    float64
 8   Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(2)
memory usage: 62.8+ KB
None
```

調整資料集-進行補值

- 將有發生資料遺失的欄位進行補值

```
#  處理遺失資料-填入平均值
df[["Age"]]  =  df[["Age"]].fillna(value=df[["Age"]].mean())
df[["Fare"]]  =  df[["Fare"]].fillna(value=df[["Fare"]].mean())
df[["Embarked"]]  =  df[["Embarked"]].fillna(value=df["Embarked"].
                                        value_counts().idxmax())
```

資料若缺失不多，可以用補值方式進行調整，若缺失太多則建議使用drop函式剃除該欄位。可自行判斷處理方式是否會對預測造成影響，也可透過實際測試準確度來決定要進行補值還是刪除資料。

本次選擇的方式是對Age、Fare進行補值，並尋找出佔比最大的登船口岸類別，將所有的遺失資料都填上該登船口岸。

調整資料集-進行補值

- 結果
  - 可以發現，所有欄位都已被賦予平均值。

```
#  處理遺失資料-填入平均值
df[["Age"]]  =  df[["Age"]].fillna(value=df[["Age"]].mean())
df[["Fare"]]  =  df[["Fare"]].fillna(value=df[["Fare"]].mean())
df[["Embarked"]]  =  df[["Embarked"]].fillna(value=df["Embarked"].
                                    value_counts().idxmax())
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Sex          891 non-null    object
 4   Age          891 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Fare         891 non-null    float64
 8   Embarked     891 non-null    object
dtypes: float64(2), int64(5), object(2)
memory usage: 62.8+ KB
None
```

調整資料集-轉換資料型態

- 將無法使用的非整數與浮點數的資料調整

  透過page4.投影片中顯示的資料，我們可以知道Sex性別應是二元類別資料，直接使用map函式進行布林值轉換處裡。
  而登船口岸欄位則是三元(含)以上的資料型別，可選擇拆分欄位進行One-hot編碼處裡，或者使用map函式轉換成整數表示。

調整資料集-轉換資料型態

- 處理性別欄-使用map函式轉換成布林值

```
#  轉換分類資料-把性別轉換成布林值
df["Sex"] = df["Sex"].map( {"female": 1, "male": 0} ).astype(int)
print(df.head())
```

```
   PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch     Fare Embarked
0            1         0       3    0  22.0      1      0   7.2500        S
1            2         1       1    1  38.0      1      0  71.2833        C
2            3         1       3    1  26.0      0      0   7.9250        S
3            4         1       1    1  35.0      1      0  53.1000        S
4            5         0       3    0  35.0      0      0   8.0500        S
```

調整資料集-轉換資料型態

- 處裡Embarked欄位

  這裡選擇拆分欄位的做法，將Embarked欄位的S、C、Q分別獨立出來，並用布林值表示。

```
#  Embarked欄位的One-hot編碼-將登船口岸的資料作One-hot編碼
embarked_one_hot  =  pd.get_dummies(df["Embarked"],  prefix="Embarked")
df  =  df.drop("Embarked",  axis=1)
df  =  df.join(embarked_one_hot)
print(df.head())
```

```
   PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch     Fare  \
0            1         0       3    0  22.0      1      0   7.2500
1            2         1       1    1  38.0      1      0  71.2833
2            3         1       3    1  26.0      0      0   7.9250
3            4         1       1    1  35.0      1      0  53.1000
4            5         0       3    0  35.0      0      0   8.0500

   Embarked_C  Embarked_Q  Embarked_S
0           0           0           1
1           1           0           0
2           0           0           1
3           0           0           1
4           0           0           1
```

調整資料集-移動欄位

- 移動預測結果的欄位

  由於生還狀況是想要預測的結果，
  為了方便可以將其移動到最後面。

```
# 將屬於預測結果的 survived 欄位移至最後
df_survived = df.pop("Survived")
df["Survived"] = df_survived
print(df.head())
```

```
   PassengerId  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked_C  \
0            1       3    0  22.0      1      0   7.2500           0
1            2       1    1  38.0      1      0  71.2833           1
2            3       3    1  26.0      0      0   7.9250           0
3            4       1    1  35.0      1      0  53.1000           0
4            5       3    0  35.0      0      0   8.0500           0

   Embarked_Q  Embarked_S  Survived
0           0           1         0
1           0           0         1
2           0           1         1
3           0           1         1
4           0           1         0
```

進行模型訓練-重新排序資料集

- 亂數排序資料集
  1.指定亂數種子
  2.重新排序

```
#  打亂資料集
np.random.seed(8)  #  指定亂數種子
dataset  =  df.values
np.random.shuffle(dataset)
```

進行模型訓練-預測目標標籤拆分

- 分割特徵資料與標籤資料

  將特徵資料跟標籤資料(結果)分開
  也就是將Survived欄位獨立出來與
  其他欄位分開

```
#分割特徵資料與標籤資料
X = dataset[:, 0:10]
Y = dataset[:, 10:]
```

進行模型訓練-標準化各項目

- 將特徵標籤內容標準化
- 將標籤資料做One-hot編碼

```
[11]  #  特徵標準化
      X  -=  X.mean(axis=0)
      X  /=  X.std(axis=0)
```

```
[ ]  # One-hot編碼
     Y = to_categorical(Y)
```

進行模型訓練-拆分訓練與驗證資料集

- 拆分訓練與測試的資料集

  將前712筆資料作為訓練用

  將後178筆資料作為驗證用

```
# 分割訓練和測試資料集
X_train, Y_train = X[:712], Y[:712]          # 訓練資料前712筆
X_test, Y_test = X[712:], Y[712:]            # 測試資料後178筆
```

進行模型訓練-定義模型

- 定義模型

  採用9層隱藏層設計，並設定神經元與啟動函數

```
# 定義模型
model = Sequential()
model.add(Dense(32, input_shape=(10,), activation="relu"))
model.add(Dense(32, activation="relu"))
model.add(Dense(32, activation="relu"))
model.add(Dense(32, activation="relu"))
model.add(Dense(32, activation="relu"))
model.add(Dense(32, activation="relu"))
model.add(Dense(32, activation="relu"))
model.add(Dense(32, activation="relu"))
model.add(Dense(2, activation="softmax"))
```

進行模型訓練-編譯模型

- 可使用summary顯示模型摘要資訊
- 編譯模型

  選擇損失函數、優化器以及評估函數

```
# 顯示模型摘要資訊
model.summary()
# 編譯模型
model.compile(loss="binary_crossentropy",  optimizer="adam",
                                metrics=["accuracy"])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                352
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_4 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_5 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_6 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_7 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 66
=================================================================
Total params: 7,810
Trainable params: 7,810
Non-trainable params: 0
_________________________________________________________________
```

進行模型訓練-訓練

- 指定以 epochs=20(20次訓練次數), batch_size=10(10個/每批次)的速度訓練，設定validation_split=0.2代表以80%訓練:20%驗證資料執行，verbose=0則是不輸出訓練過程的資料

```
# 訓練模型
print("Processing ...")
history = model.fit(X_train, Y_train, validation_split=0.2, epochs=20, batch_size=10, verbose=0)
```

```
Processing ...
```

進行模型訓練-檢驗結果

- 使用以下程式碼進行準確率評估

```
#  評估模型
loss,  accuracy  =  model.evaluate(X_train,  Y_train)
print("訓練資料集的準確度  =  {:.2f}".format(accuracy))
loss,  accuracy  =  model.evaluate(X_test,  Y_test)
print("測試資料集的準確度  =  {:.2f}".format(accuracy))
```

```
23/23 [==============================] - 0s 6ms/step - loss: 0.3078 - accuracy: 0.8834
訓練資料集的準確度 = 0.88
6/6 [==============================] - 0s 4ms/step - loss: 0.5166 - accuracy: 0.7765
測試資料集的準確度 = 0.78
```

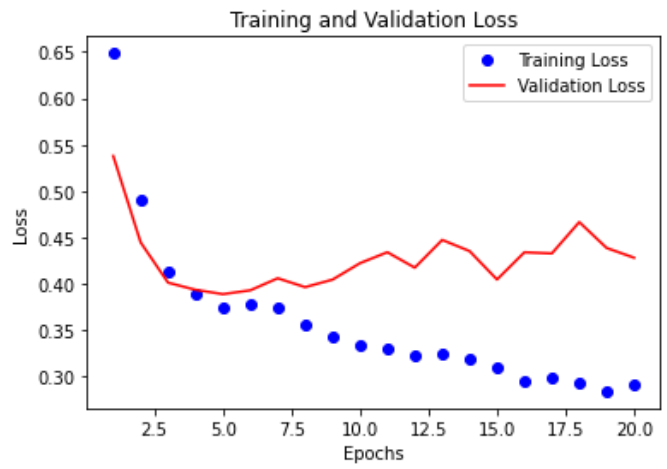
進行模型訓練-檢驗結果

- 使用以下程式碼進行結果輸出

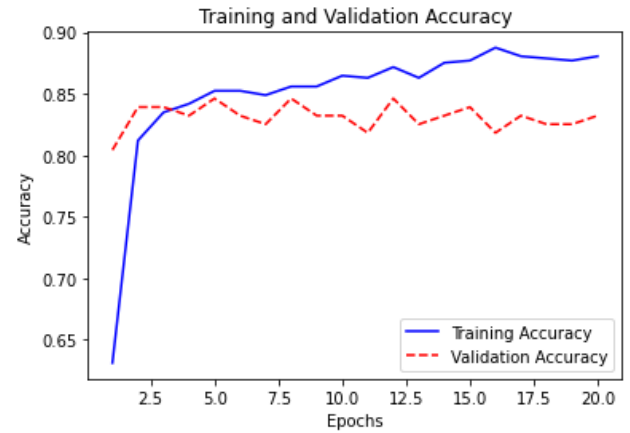
  可以用來檢查訓練過程的各項資訊

```python
# 顯示訓練和驗證損失圖表
import matplotlib.pyplot as plt

loss = history.history["loss"]
epochs = range(1, len(loss)+1)
val_loss = history.history["val_loss"]
plt.plot(epochs, loss, "bo", label="Training Loss")
plt.plot(epochs, val_loss, "r", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```python
# 顯示訓練和驗證準確度
acc = history.history["accuracy"]
epochs = range(1, len(acc)+1)
val_acc = history.history["val_accuracy"]
plt.plot(epochs, acc, "b-", label="Training Accuracy")
plt.plot(epochs, val_acc, "r—", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



IRIS

調整資料集

- 用與鐵達尼號的方式進行資料集分析

  發現target值並非整數或浮點數-進行調整

  沒有發現其他問題~

```
#  檢查資料集的各項特徵
print(df.describe())  #檢查資料集統計摘要
print(df.info())  #檢查各欄位的變數值型態以及遺漏值狀況
print(df.duplicated())  #檢查各欄位的重複情形
```

```
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   target        150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
0      False
1      False
2      False
3      False
4      False
       ...
145    False
146    False
147    False
148    False
149    False
Length: 150, dtype: bool
```

調整資料集

- 這次我們採用map函式來進行調整

```
[14] #將targer欄位的內容從文字轉換成整數形式
     target_mapping = {"setosa":0 ,
                       "versicolor":1 ,
                       "virginica":2 }
     df["target"] = df["target"].map(target_mapping)
     print(df.head())
```

	sepal_length	sepal_width	petal_length	petal_width	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

剩下的都一樣...除了 #定義模型

- 輸出層

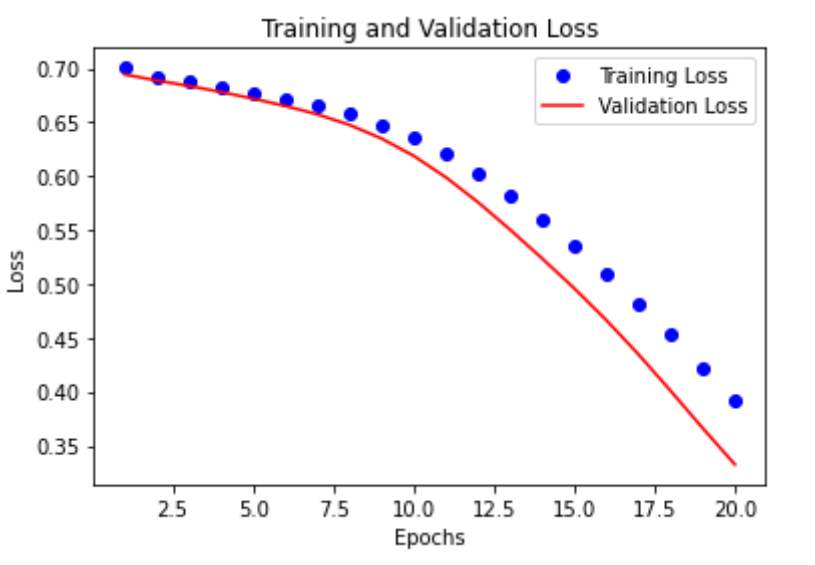
  結果有三種，與鐵達尼號只有兩種不同，
  要調整輸出層的神經元個數

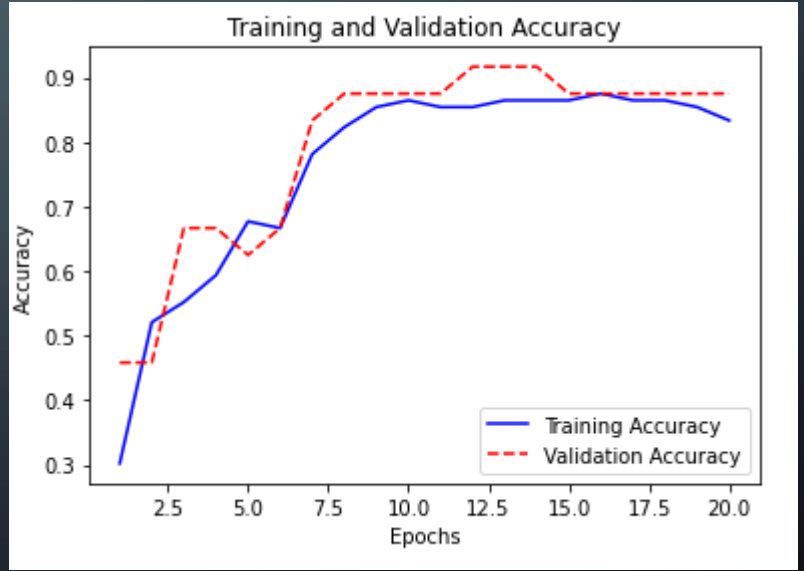
```
#  定義模型
model  =  Sequential()
model.add(Dense(9,  input_shape=(4,),  activation="relu"))
model.add(Dense(20,  activation="relu"))
model.add(Dense(20,  activation="relu"))
model.add(Dense(20,  activation="relu"))
model.add(Dense(20,  activation="relu"))
model.add(Dense(20,  activation="relu"))
model.add(Dense(20,  activation="relu"))
model.add(Dense(20,  activation="relu"))
model.add(Dense(3,  activation="softmax"))
```

結果

```
#  評估模型
loss,  accuracy  =  model.evaluate(X_train,  Y_train)
print("訓練資料集的準確度  =  {:.2f}".format(accuracy))
loss,  accuracy  =  model.evaluate(X_test,  Y_test)
print("測試資料集的準確度  =  {:.2f}".format(accuracy))
```

```
4/4 [==============================] - 0s 11ms/step - loss: 0.3633 - accuracy: 0.8333
訓練資料集的準確度 = 0.83
1/1 [==============================] - 0s 46ms/step - loss: 0.3499 - accuracy: 0.8000
測試資料集的準確度 = 0.80
```





小插曲....

- 如果漏做目標標籤的One-hot編碼會發生?

聊聊目標標籤ONE-HOT編碼有無的差異

- 猜猜看這兩者差多少

```
[34]  #  定義模型
      model  =  Sequential()
      model.add(Dense(9,  input_shape=(4,),  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(1,  activation="softmax"))
```

```
[9]  #  One-hot編碼
     Y  =  to_categorical(Y)
```

```
[24]  #  定義模型
      model  =  Sequential()
      model.add(Dense(9,  input_shape=(4,),  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(20,  activation="relu"))
      model.add(Dense(3,  activation="softmax"))
```

聊聊目標標籤ONE-HOT編碼有無的差異

- 差這麼多!!!!!
- 因為沒有進行One-hot編碼，模型無法計算損失分數，導致模型失效

```
# 評估模型
loss, accuracy = model.evaluate(X_train, Y_train)
print("訓練資料集的準確度 = {:.2f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, Y_test)
print("測試資料集的準確度 = {:.2f}".format(accuracy))
```

```
4/4 [==============================] - 0s 4ms/step - loss: -1.2260 - accuracy: 0.3583
訓練資料集的準確度 = 0.36
1/1 [==============================] - 0s 22ms/step - loss: -1.8825 - accuracy: 0.2333
測試資料集的準確度 = 0.23
```

```
# 評估模型
loss, accuracy = model.evaluate(X_train, Y_train)
print("訓練資料集的準確度 = {:.2f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, Y_test)
print("測試資料集的準確度 = {:.2f}".format(accuracy))
```

```
4/4 [==============================] - 0s 4ms/step - loss: 0.4563 - accuracy: 0.9167
訓練資料集的準確度 = 0.92
1/1 [==============================] - 0s 21ms/step - loss: 0.4646 - accuracy: 0.8667
測試資料集的準確度 = 0.87
```