movieRating - 電影分數預測

312706034 資管碩二 張棨翔

1. 資料前處理

- 使用pandas載入原始資料集。

```
import pandas as pd

df = pd.read_csv('movieRating.csv')
df.head()
```

	TrainDataID	UserID	MovieID	Rating
0	1	796	1193	5
1	2	796	661	3
2	3	796	914	3
3	4	796	3408	4
4	5	796	2355	5

- 因為TrainDataID這個欄位不會用到，所以把該欄位刪除。

```
# remove the unnecessary columns
df.drop(['TrainDataID'], axis=1, inplace=True)
df.head()
```

	UserID	MovieID	Rating
0	796	1193	5
1	796	661	3
2	796	914	3
3	796	3408	4
4	796	2355	5

- 觀察欄位屬性不重複的值的總數量，分別比對UserID的最小值和最大值，接著比對MovieID的最小值和最大值，經過觀察我們可以發現到UserID的值是連續的從1到6040並且和不重複值總數相同，而我們可以發現到MovieID的最小值是1和最大值是3952，這和不重複值的總數3688是對不上的，也就是說MovieID在1和3952之間有部分ID是沒有資料的。接著我們把正確的UserID和MovieID最大值儲存成兩個變數(之後會將變數用

在建構模型)。

```
# explore the data
# find count of unique values in each column
df.nunique()
```

```
UserID     6040
MovieID    3688
Rating        5
dtype: int64
```

```
#find min user id
df['UserID'].min()
```

```
1
```

```
#find max user id
df['UserID'].max()
```

```
6040
```

```
#find min movie id
df['MovieID'].min()
```

```
1
```

```
#find max movie id
df['MovieID'].max()
```

```
3952
```

```
user_count = df['UserID'].max()
movie_count = df['MovieID'].max()
```

- 原始資料是有照UserID排序，這邊我們用shuffle函數把資料重新打亂。

```
# data shuffling
from sklearn.utils import shuffle
df = shuffle(df)
df.head()
```

	UserID	MovieID	Rating
453930	5491	1954	4
29312	3181	3911	3
666399	1372	427	3
845252	1813	2973	5
45709	2775	836	2

- 準備模型訓練和測試的資料，這邊把要預測的電影分數從dataframe拿出來存入labels變數，然後把用來預測電影分數的feature從dataframe拿出來存入datas變數，接著透過sklearn套件的方法把訓練資料和測試資料以80%和20%的比例進行切分。

```
# split the data into train and test sets
from sklearn.model_selection import train_test_split
labels = df['Rating']
datas = df.drop(['Rating'], axis=1)
X_train, X_test, Y_train, Y_test = train_test_split(datas, labels, test_size=0.2, random_state=77)
```

```
X_train.head()
```

	UserID	MovieID
476959	3651	2015
546112	4658	2395
663582	609	2738
826011	4417	2456
351690	3913	2628

---

2. 模型建構

- import在模型建構過程中會使用到的套件並定義會用到的常數dimension_embedding和bias。

```
# create the model for the rating prediction
import numpy as np
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Dense, Concatenate, Dropout
from tensorflow.keras.layers import Multiply

# Define model architecture

dimension_embedding = 32  # Size of the embedding vector
bias = 1  # Size of the bias term
```

- 下面我們定義兩個輸入層user_input和movie_input，分別代表使用者和電影的ID。輸入的形狀則是 (1,)，表示每次輸入一個數值ID。

```
# Define model inputs
user_input = Input(shape=(1,), name='UserID')
movie_input = Input(shape=(1,), name='MovieID')
```

- 使用嵌入層(Embedding Layer)將離散的使用者ID和電影ID映射到維度為dimension_embedding=32的向量空間。這邊user_count+1和movie_count+1是為了處理從0開始的索引問題。

```
# Embedding layers
user_embedding = Embedding(user_count + 1, dimension_embedding, name='user_embedding')(user_input)
movie_embedding = Embedding(movie_count + 1, dimension_embedding, name='movie_embedding')(movie_input)
```

- 為每個使用者和電影分配一個偏差值，維度設為bias=1，幫助模型捕捉使用者ID和電影ID的baseline效果（例如，某些電影可能總是高分或低分）。

```
# Bias terms
user_bias = Embedding(user_count + 1, bias, name='user_bias')(user_input)
movie_bias = Embedding(movie_count + 1, bias, name='movie_bias')(movie_input)
```

- 將嵌入和偏差向量從2D轉為1D，以便後續模型的建構流程。

```
# Flatten embeddings and biases
user_vector = Flatten()(user_embedding)
movie_vector = Flatten()(movie_embedding)
user_bias_vector = Flatten()(user_bias)
movie_bias_vector = Flatten()(movie_bias)
```

- 進行元素逐項相乘，捕捉使用者ID和電影ID之間的交互影響。這類似於矩陣分解技術中的交互項建模。

```
# Multiply embeddings
interaction = Multiply()([user_vector, movie_vector])  # Element-wise multiplication
```

- 全連接層用於從交互向量中提取更高層次的特徵，使用ReLU激活函數。Dropout層（0.2的丟棄率）防止overfitting。神經網路的結構逐漸縮小，從128到16，形成金字塔形的結構。

```
# Pass the interaction through dense layers
dense1 = Dense(128, activation='relu')(interaction)
dense1 = Dropout(0.2)(dense1)
dense2 = Dense(64, activation='relu')(dense1)
dense2 = Dropout(0.2)(dense2)
dense3 = Dense(32, activation='relu')(dense2)
dense4 = Dense(16, activation='relu')(dense3)
```

- 將最終的特徵向量與使用者ID和電影ID的偏差向量進行串接，保留所有資訊。

```
# Concatenate bias terms with dense output
final_vector = Concatenate()([dense4, user_bias_vector, movie_bias_vector])
```

- 使用一個單輸出的全連接層，激活函數為ReLU，預測電影評分。輸出範圍被限制為非負數（ReLU的特性）。

```
# Output layer
output = Dense(1, activation='relu', name='rating_output')(final_vector)
```

- 損失函數設定為均方誤差（mse），適用於迴歸問題。優化器使用Adam，能夠動態調整學習率。評估指標則設定為平均絕對誤差（mean_absolute_error），可以直觀反映預測誤差大小。定義模型的部分，我們會輸入兩個資料和輸出一個預測值，接著我們將模型架構進行compile並定義loss值的指標，並且輸出的metrics設為MAE，最後印出整個模型的架構。

```
# Define the model
model = Model(inputs=[user_input, movie_input], outputs=output)

# Compile the model
model.compile(loss='mse', optimizer='adam', metrics=['mean_absolute_error'])

# Model summary
model.summary()
```

- 模型架構如下。

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 UserID (InputLayer)         [(None, 1)]               0         []

 MovieID (InputLayer)        [(None, 1)]               0         []

 user_embedding (Embedding)  (None, 1, 32)             193312    ['UserID[0][0]']

 movie_embedding (Embedding  (None, 1, 32)             126496    ['MovieID[0][0]']
 )

 flatten_8 (Flatten)         (None, 32)                0         ['user_embedding[0][0]']

 flatten_9 (Flatten)         (None, 32)                0         ['movie_embedding[0][0]']

 multiply_2 (Multiply)       (None, 32)                0         ['flatten_8[0][0]',
                                                                  'flatten_9[0][0]']

 dense_7 (Dense)             (None, 128)               4224      ['multiply_2[0][0]']

 dropout_4 (Dropout)         (None, 128)               0         ['dense_7[0][0]']

 dense_8 (Dense)             (None, 64)                8256      ['dropout_4[0][0]']

dropout_5 (Dropout)          (None, 64)                0         ['dense_8[0][0]']

dense_9 (Dense)              (None, 32)                2080      ['dropout_5[0][0]']

user_bias (Embedding)        (None, 1, 1)              6041      ['UserID[0][0]']

movie_bias (Embedding)       (None, 1, 1)              3953      ['MovieID[0][0]']

dense_10 (Dense)             (None, 16)                528       ['dense_9[0][0]']

flatten_10 (Flatten)         (None, 1)                 0         ['user_bias[0][0]']

flatten_11 (Flatten)         (None, 1)                 0         ['movie_bias[0][0]']

concatenate_2 (Concatenate   (None, 18)                0         ['dense_10[0][0]',
)                                                                 'flatten_10[0][0]',
                                                                  'flatten_11[0][0]']

rating_output (Dense)        (None, 1)                 19        ['concatenate_2[0][0]']

==================================================================================================
Total params: 344909 (1.32 MB)
Trainable params: 344909 (1.32 MB)
Non-trainable params: 0 (0.00 Byte)
```

---

3. 模型訓練/評估

- 下面透過fit函數開始訓練建構好的模型，epochs設為10(訓練10輪)，batch size設為128，表示一個batch會看128筆資料，圖片為訓練過程。

```python
# train the model
history = model.fit([X_train['UserID'], X_train['MovieID']], Y_train, epochs=10, batch_size=128, validation_split=0.05)
```

```
Epoch 1/10
5343/5343 [==============================] - 22s 4ms/step - loss: 1.0395 - mean_absolute_error: 0.7935 - val_loss: 0.8040 - v
al_mean_absolute_error: 0.7053
Epoch 2/10
5343/5343 [==============================] - 20s 4ms/step - loss: 0.7048 - mean_absolute_error: 0.6587 - val_loss: 0.7939 - v
al_mean_absolute_error: 0.6963
Epoch 3/10
5343/5343 [==============================] - 20s 4ms/step - loss: 0.6047 - mean_absolute_error: 0.6066 - val_loss: 0.8051 - v
al_mean_absolute_error: 0.7044
Epoch 4/10
5343/5343 [==============================] - 20s 4ms/step - loss: 0.5449 - mean_absolute_error: 0.5740 - val_loss: 0.8229 - v
al_mean_absolute_error: 0.7102
Epoch 5/10
5343/5343 [==============================] - 20s 4ms/step - loss: 0.5078 - mean_absolute_error: 0.5525 - val_loss: 0.8530 - v
al_mean_absolute_error: 0.7212
Epoch 6/10
5343/5343 [==============================] - 20s 4ms/step - loss: 0.4821 - mean_absolute_error: 0.5374 - val_loss: 0.8553 - v
al_mean_absolute_error: 0.7214
Epoch 7/10
5343/5343 [==============================] - 20s 4ms/step - loss: 0.4622 - mean_absolute_error: 0.5254 - val_loss: 0.8819 - v
al_mean_absolute_error: 0.7314
Epoch 8/10
5343/5343 [==============================] - 21s 4ms/step - loss: 0.4472 - mean_absolute_error: 0.5161 - val_loss: 0.8843 - v
al_mean_absolute_error: 0.7284
Epoch 9/10
5343/5343 [==============================] - 20s 4ms/step - loss: 0.4349 - mean_absolute_error: 0.5084 - val_loss: 0.9018 - v
al_mean_absolute_error: 0.7349
Epoch 10/10
5343/5343 [==============================] - 20s 4ms/step - loss: 0.4239 - mean_absolute_error: 0.5013 - val_loss: 0.9148 - v
al_mean_absolute_error: 0.7395
```

- 最後將測試資料集放入predict函數進行電影分數的預測，並將預測完的分數與實際分數透
  過mean_absolute_error函數計算MAE。在 5625 筆測試資料中預測出來的MAE結果為0.7324456572532654，圖片為測試資料集預測完電影分數並計算MAE的結果。

```python
# evaluate the model
test_loss, test_mae = model.evaluate([X_test['UserID'], X_test['MovieID']], Y_test)
print('Test loss (MSE) on test data:', test_loss)
print('Test MAE on test data:', test_mae)
```

```
5625/5625 [==============================] - 7s 1ms/step - loss: 0.9010 - mean_absolute_error: 0.7324
Test loss (MSE) on test data: 0.9010332822799683
Test MAE on test data: 0.7324456572532654
```