# HW5 movieRating

🔍 作業目標：

1. 將資料讀取進來 (可用pandas套件)
2. 亂數拆成訓練集 (80%) 與訓練集 (20%)
3. 建立矩陣分解模型：(1)產出預測結果 (2)計算MAE

## 將資料讀取進來

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
data = pd.read_csv('/content/drive/MyDrive/碩一上課堂/1131_dataMining/dataMining/HW5/movieRating.
data.head()
```

| | TrainDataID | UserID | MovieID | Rating |
|---|---|---|---|---|
| 335378 | 335379 | 2794 | 1643 | 4 |
| 743547 | 743548 | 4761 | 25 | 2 |
| 416126 | 416127 | 844 | 3673 | 1 |
| 570976 | 570977 | 3974 | 1683 | 3 |
| 635909 | 635910 | 2927 | 3478 | 3 |

## 亂數拆成訓練集 (80%) 與訓練集 (20%)

```
# 打亂資料集
data = shuffle(data, random_state=42)

# 分割訓練集與測試集 (80% 訓練集, 20% 測試集)
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)

# 獲取唯一使用者和電影數量
n_users = data['UserID'].nunique()
n_movies = data['MovieID'].nunique()

# 準備訓練和測試資料
train_user = train_data['UserID'].values
train_movie = train_data['MovieID'].values
train_rating = train_data['Rating'].values

test_user = test_data['UserID'].values
test_movie = test_data['MovieID'].values
test_rating = test_data['Rating'].values
```

- **打亂資料集**：隨機重新排列資料，避免因資料排序影響模型訓練，並確保樣本的分佈更均勻。

- **分割資料集**：將資料集分為訓練集（80%）與測試集（20%），用於模型訓練和驗證。
- **計算唯一值數量**：統計資料集中唯一的使用者數量（UserID）和電影數量（MovieID），以便初始化嵌入層的維度。
- **提取特徵與目標變數**：
  - 從訓練集和測試集中分別提取使用者 ID、電影 ID（feature），以及評分（target value），為後續的模型構建提供輸入數據。

# 建立矩陣分解模型：(1)產出預測結果 (2)計算MAE

## 模型訓練

```
# 建立矩陣分解模型
user_input = Input(shape=(1,))
movie_input = Input(shape=(1,))

user_embedding = Embedding(input_dim=n_users + 1, output_dim=50)(user_input)
movie_embedding = Embedding(input_dim=n_movies + 1, output_dim=50)(movie_input)

user_vec = Flatten()(user_embedding)
movie_vec = Flatten()(movie_embedding)

dot_product = Dot(axes=1)([user_vec, movie_vec])

user_bias = Embedding(input_dim=n_users + 1, output_dim=1)(user_input)
movie_bias = Embedding(input_dim=n_movies + 1, output_dim=1)(movie_input)

user_bias_vec = Flatten()(user_bias)
movie_bias_vec = Flatten()(movie_bias)

output = Add()([dot_product, user_bias_vec, movie_bias_vec])

# 編譯模型
model = Model(inputs=[user_input, movie_input], outputs=output)
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

# 訓練模型
model.fit(
    [train_user, train_movie],
    train_rating,
    validation_data=([test_user, test_movie], test_rating),
    epochs=5,
    batch_size=1024,
    verbose=1
)
```

```
Epoch 1/5
704/704 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 11.0304 - val_loss: 1.3557
Epoch 2/5
704/704 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 1.1435 - val_loss: 0.9245
Epoch 3/5
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 0.8915 - val_loss: 0.8676
Epoch 4/5
704/704 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.8507 - val_loss: 0.8464
Epoch 5/5
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.8154 - val_loss: 0.8317
<keras.src.callbacks.history.History at 0x7fe97aa1fdc0>
```

🔍

- **訓練 `loss` 持續下降**：
  - 由 `11.0304` 下降至 `0.8154`，表明模型在學習數據特徵，預測精度逐步提高。
- **驗證 `val_loss` )逐漸穩定**：
  - 由初始的 `1.3557` 降至 `0.8317`，顯示模型在測試集上的性能逐漸改善，且無過擬合跡象。

- **模型設計：**
  - **Input Layer**：
    - 定義兩個輸入：`user_input` 和 `movie_input`，分別代表使用者和電影的 ID。
  - **Embedding Layer**：
    - 將 `UserID` 和 `MovieID` 映射為低維vector，以捕捉使用者與電影的特徵表示。
    - 嵌入層的維度：
      - `input_dim`：使用者或電影的總數（加 1 是為了處理index從 0 開始）。
      - `output_dim`：嵌入向量的維度，這裡設為 50。
  - **Flatten**：
    - 將嵌入的高維矩陣壓縮成一維向量，便於後續計算。
  - **Dot**：
    - 計算使用者和電影嵌入向量的內積，表示使用者對電影的匹配度。
  - **Bias**：
    - 為使用者和電影分別增加一個偏差項，以捕捉特定使用者或電影的固有評分偏好。
  - **Output Layer**：
    - 將內積與偏差項相加，生成最終的預測評分。

# 模型預測、計算MAE

```
# 預測並計算 MAE
test_predictions = model.predict([test_user, test_movie])
mae = mean_absolute_error(test_rating, test_predictions)

print(f"Mean Absolute Error (MAE): {mae}")
```

```
5625/5625 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step
Mean Absolute Error (MAE): 0.7190045403204066
```

🔍 模型的 MAE 為 0.719，表示平均預測誤差接近 0.72

- 使用測試資料（`test_user` 和 `test_movie`）作為輸入，透過訓練好的矩陣分解模型進行評分預測。
- 輸出 `test_predictions` 為模型對測試集的預測結果（即對每部電影的預測評分）。
- 計算測試集的MAE，衡量模型預測值與真實評分之間的平均誤差。

# 增加embedding維度、增加epochs

```
# 增加 embedding 維度到 100
user_embedding = Embedding(input_dim=n_users + 1, output_dim=100)(user_input)
movie_embedding = Embedding(input_dim=n_movies + 1, output_dim=100)(movie_input)
```

```
user_vec = Flatten()(user_embedding)
movie_vec = Flatten()(movie_embedding)

dot_product = Dot(axes=1)([user_vec, movie_vec])

user_bias = Embedding(input_dim=n_users + 1, output_dim=1)(user_input)
movie_bias = Embedding(input_dim=n_movies + 1, output_dim=1)(movie_input)

user_bias_vec = Flatten()(user_bias)
movie_bias_vec = Flatten()(movie_bias)

output = Add()([dot_product, user_bias_vec, movie_bias_vec])

# 編譯模型
model = Model(inputs=[user_input, movie_input], outputs=output)
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

# 增加訓練輪數到 10
model.fit(
    [train_user, train_movie],
    train_rating,
    validation_data=([test_user, test_movie], test_rating),
    epochs=10,
    batch_size=1024,
    verbose=1
)

# 預測並計算 MAE
test_predictions = model.predict([test_user, test_movie])
mae = mean_absolute_error(test_rating, test_predictions)

print(f"Mean Absolute Error (MAE): {mae}")
```

```
Epoch 1/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 10.7721 - val_loss: 1.1109
Epoch 2/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.9929 - val_loss: 0.8806
Epoch 3/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.8567 - val_loss: 0.8461
Epoch 4/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 0.8061 - val_loss: 0.8138
Epoch 5/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7574 - val_loss: 0.7990
Epoch 6/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7202 - val_loss: 0.8086
Epoch 7/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 0.6898 - val_loss: 0.7929
Epoch 8/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 0.6461 - val_loss: 0.7898
Epoch 9/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 0.6033 - val_loss: 0.7995
Epoch 10/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.5639 - val_loss: 0.7975
5625/5625 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step
Mean Absolute Error (MAE): 0.697056373377975
```

🔍

- **模型收斂效果更好**：
  - 增加 embedding 維度和 epochs 後，訓練 loss 持續下降，模型更充分學習了數據特徵。
- **驗證 loss 改善有限**：
  - 雖然驗證 loss 降到更低值，但最終穩定在與先前相近的水準，可能存在輕微過擬合。
- **MAE 明顯改善**：
  - MAE 從 **0.719** 降到 **0.697**，表示模型對測試集的預測誤差進一步縮小，性能提升。

**Embedding 維度**：

- `output_dim=100`，嵌入向量的維度更高，有助於學習更豐富的特徵表示。

**Epochs**：

- 設定 `epochs=10`，訓練更充分，有助於提升模型性能。