

1. (1%)請比較有無normalize(rating)的差別。並說明如何normalize.

答：一開始取得全部 rating 的 mean 和 standard deviation, 接著運算

$$\text{rating} = (\text{rating} - \text{mean}) / (\text{std} + 1\text{e-}100)$$

將此處理過的 rating 拿去給 model fit。而在 test 的時候, 把預測出來的 predict 做以下運算

$$\text{predict} = \text{predict} * \text{std} + \text{mean}$$

處理過後的 predict 才是最終預測的 rating 結果。

而經過測試, 上傳到 kaggle 的分數如下表。發現這樣的 normalization 效果有些許進步。

| operation       | Public score |
|-----------------|--------------|
| 無 normalization | 0.87147      |
| 有 normalization | 0.86403      |

2. (1%)比較不同的latent dimension的結果。

答：先說明一下我測試時 collaborative filtering 的架構。optimizer 使用 adamax, 然後切 10% 測資作為 validation set 並使用其 rmse 作為收斂標準, batch size 是 256。然後調整不同的 embedding dimension 來看效果如何。發現 dimension 改變, 效果並沒有比較好。

| Dimension | Training loss | Training RMSE | Valid loss | Valid RMSE |
|-----------|---------------|---------------|------------|------------|
| 16        | 0.6344        | 0.7947        | 0.7537     | 0.8653     |
| 64        | 0.5672        | 0.7512        | 0.7412     | 0.8580     |
| 120       | 0.5465        | 0.7374        | 0.7477     | 0.8618     |
| 256       | 0.4762        | 0.6881        | 0.7356     | 0.8550     |
| 512       | 0.4275        | 0.6520        | 0.7395     | 0.8573     |

3. (1%)比較有無bias的結果。

答：沿用跟上題一樣的架構, 但是在 dot 後再加上 users bias 和 movies bias, 結果如下, 發現有 bias 效果會好一點。若是再加上 normalization 的話, 效果會再更好一點。

|        | Training loss | Training RMSE | Valid loss | Valid RMSE | Public score |
|--------|---------------|---------------|------------|------------|--------------|
| 無 bias | 0.5459        | 0.7370        | 0.7341     | 0.8542     | 0.86054      |
| 有 bias | 0.5472        | 0.7379        | 0.7305     | 0.8525     | 0.85983      |

4. (1%)請試著用DNN解決這個問題，並說明實作的方法(方法不限)。並比較MF和NN的結果，討論結果的差異。

答：我把 users 跟 movies 做 embedding 後，將他們兩個 vector concatenate 起來，接著丟進一個 dense layer 中，最後接上一個 output dense layer，輸出就是預測的 rating 值。詳細結構如下圖。

```
def build_deep_model(n_users, n_movies, dim, dropout=0.1):
    u_input = Input(shape=(1,))
    u = Embedding(n_users, dim)(u_input)
    u = Reshape((dim,))(u)

    m_input = Input(shape=(1,))
    m = Embedding(n_movies, dim)(m_input)
    m = Reshape((dim,))(m)

    out = concatenate([u, m])
    out = Dropout(dropout)(out)
    out = Dense(dim, activation='relu')(out)
    out = Dropout(dropout)(out)
    out = Dense(1, activation='relu')(out)

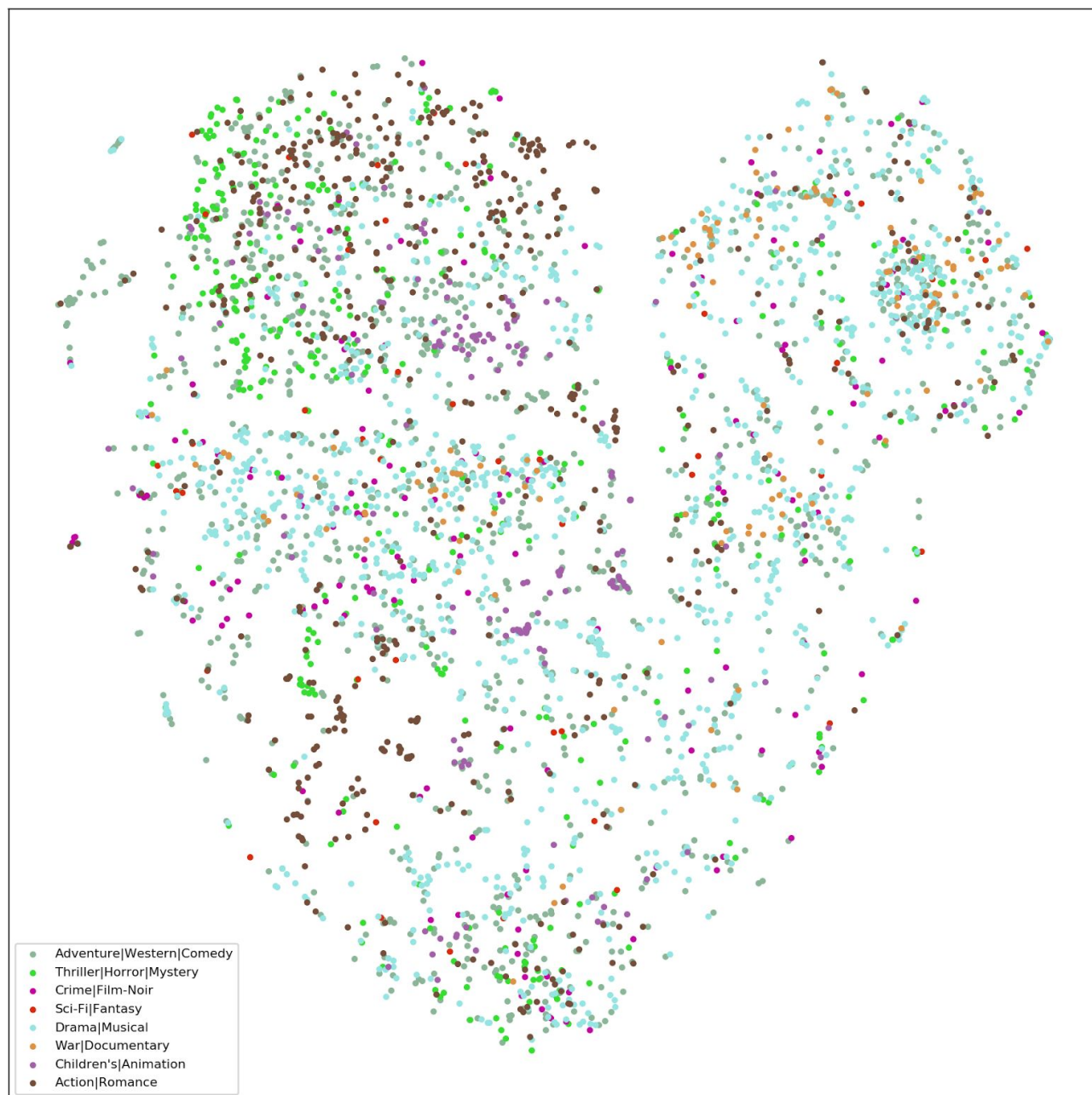
    model = Model(inputs=[u_input, m_input], outputs=out)
    return model
```

在經過測試後，在 public score 的呈現如下表，deep model 比純粹的 cf model 效果好。我覺得是因為 cf 的話是把兩個 embedding 相乘起來成一個數字，這樣的話，多出來的維度儲存的資訊就沒有發揮到太大的用處；而 deep model 是使用 concatenate 的方式，在訓練過程中從每個維度獲得的資訊就會比 cf 多出不少。

| operation  | Public score |
|------------|--------------|
| CF model   | 0.87147      |
| Deep model | 0.86175      |

5. 請試著將movie的embedding用tsne降維後，將movie category當作label來作圖。

答：下圖是我的分類。由圖中可以發現 Drama|Musical 散落在各處，比較沒有規律，但是 Action 集中在左上角、左下角跟右上角；然後 Horror|Thriller|Mystery 類的集中在左上角；War 類的在中間跟右上角。



6. (BONUS)(1%)試著使用除了rating以外的feature，並說明你的做法和結果，結果好壞不會影響評分。

答：我多使用 users.csv 和 movies.csv 兩份資料。在 users.csv 中，我抽出UserID, Gender, Age, Occupation 四個 features 並做成 categorical；在 movies.csv 中，我抽出 movieID,

Genres, Genres 的部分做成 one-hot encoding。所以我對於 user input 就變成 4 項 (有 user embedding, gender, age, occupation), movie input 的話則是 19 項 (movie embedding, 18 種 movie genre 的 one-hot encoding), 架構則是 deep model。

將這兩個 input 輸入後通過 keras 的 embedding layer, 然後 Flatten 後 concatenate 起來, 接著通過三層 dense layer 做預測, 其中穿插著 dropout, 詳細架構如下圖, 丟上去 kaggle 發現在 public score 的表現可以到 0.849 左右。

```
def build_deep_model(n_users, n_movies, dim, dropout=0.1):  
    u_input = Input(shape=(4,))  
    u = Embedding(n_users, dim)(u_input)  
    u = Flatten()(u)  
  
    m_input = Input(shape=(19,))  
    m = Embedding(n_movies, dim)(m_input)  
    m = Flatten()(m)  
  
    out = concatenate([u, m])  
    out = Dropout(dropout)(out)  
    out = Dense(256, activation='relu')(out)  
    out = Dropout(dropout)(out)  
    out = Dense(128, activation='relu')(out)  
    out = Dropout(dropout)(out)  
    out = Dense(64, activation='relu')(out)  
    out = Dropout(0.15)(out)  
    out = Dense(dim, activation='relu')(out)  
    out = Dropout(0.2)(out)  
    out = Dense(1, activation='relu')(out)  
  
    model = Model(inputs=[u_input, m_input], outputs=out)  
    return model
```