

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

(collaborator:)

我所使用的方法是減去 3(約為平均), 然後最後在 predict 時則逆運算, 加上 3, 最後出來的 validation loss 有 normalized 的會有比較好的結果, 不知道為什麼如果是直接減去平均再除上標準差下去 train 出來的結果會爆。

有 normalized: val\_loss: 0.7295 (kaggle:0.85761, 0.85836)

無 normalized: val\_loss: 0.8360 (kaggle:0.89600, 0.88291)

2. (1%)比較不同的 latent dimension 的結果。

(collaborator:)

我將 embedding dimension 分別設為 16 128 1024 然後比節出來結果在 validation loss 和 kaggle 上分數的不同, 最後發現 embedding dimension 的大小是適中就好, 推測結果為太大的話容易過於 sparse 不易學到東西, 太小的話容易遺失太多重要資訊, 也無法有較低的 loss

embedding dimension :1024 val\_loss: 0.7638 (kaggle:0.86004, 0.85913)

embedding dimension :128 val\_loss: 0.7283 (kaggle: 0.85761, 0.85836)

embedding dimension :16 val\_loss: 0.7527 (kaggle:0.86637, 0.86471)

3. (1%)比較有無 bias 的結果。

(collaborator:)

有 bias : val\_loss: 0.7295 (kaggle:0.85761,0.85836)

無 bias : val\_loss: 0.7370 (kaggle:0.85944,0.85868)

實驗結果發現兩者的差別其實並不大, 也許是因為 bias 中本身 feature 對準確度的預測本來就沒有很好的影響。

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

(collaborator:)

MF:

無 bias : val\_loss: 0.7370 (kaggle:0.85944,0.85868)

Layer (type)	Output Shape	Param #	Connected to
input_68 (InputLayer)	(None, 1)	0	
input_67 (InputLayer)	(None, 1)	0	
embedding_128 (Embedding)	(None, 1, 128)	773248	input_68[0][0]
embedding_127 (Embedding)	(None, 1, 128)	497152	input_67[0][0]
flatten_128 (Flatten)	(None, 128)	0	embedding_128[0][0]
flatten_127 (Flatten)	(None, 128)	0	embedding_127[0][0]
embedding_130 (Embedding)	(None, 1, 1)	6041	input_68[0][0]
embedding_129 (Embedding)	(None, 1, 1)	3884	input_67[0][0]
dot_34 (Dot)	(None, 1)	0	flatten_128[0][0] flatten_127[0][0]
flatten_130 (Flatten)	(None, 1)	0	embedding_130[0][0]
flatten_129 (Flatten)	(None, 1)	0	embedding_129[0][0]
add_31 (Add)	(None, 1)	0	dot_34[0][0] flatten_130[0][0] flatten_129[0][0]
Total params: 1,280,325			
Trainable params: 1,280,325			
Non-trainable params: 0			

NN: 我加入了三層 dense 和 dropout layer 來做，因為我其實沒有特別去條，所以做出來的結果除了收斂較慢，最後出來的結果也沒有比 MF 有更低的 loss  
val\_loss: 0.8268 (kaggle:0.85698, 0.85594)

```
Out = Dense(150,activation='relu')(Out)
Out = Dropout(0.3)(Out)
Out = Dense(50,activation='relu')(Out)
Out = Dropout(0.25)(Out)
Out = Dense(1)(Out)
Out = Dense(1)(Out)
```

Layer (type)	Output Shape	Param #	Connected to
input_66 (InputLayer)	(None, 1)	0	
input_65 (InputLayer)	(None, 1)	0	
embedding_124 (Embedding)	(None, 1, 128)	773248	input_66[0][0]
embedding_123 (Embedding)	(None, 1, 128)	497152	input_65[0][0]
flatten_124 (Flatten)	(None, 128)	0	embedding_124[0][0]
flatten_123 (Flatten)	(None, 128)	0	embedding_123[0][0]
embedding_126 (Embedding)	(None, 1, 1)	6041	input_66[0][0]
embedding_125 (Embedding)	(None, 1, 1)	3884	input_65[0][0]
dot_33 (Dot)	(None, 1)	0	flatten_124[0][0] flatten_123[0][0]
flatten_126 (Flatten)	(None, 1)	0	embedding_126[0][0]
flatten_125 (Flatten)	(None, 1)	0	embedding_125[0][0]
add_30 (Add)	(None, 1)	0	dot_33[0][0] flatten_126[0][0] flatten_125[0][0]
dense_10 (Dense)	(None, 150)	300	add_30[0][0]
dropout_10 (Dropout)	(None, 150)	0	dense_10[0][0]
dense_11 (Dense)	(None, 50)	7550	dropout_10[0][0]
dropout_11 (Dropout)	(None, 50)	0	dense_11[0][0]
dense_12 (Dense)	(None, 1)	51	dropout_11[0][0]
dense_13 (Dense)	(None, 1)	2	dense_12[0][0]
Total params: 1,288,228			
Trainable params: 1,288,228			
Non-trainable params: 0			

5. (1%)請試著將 movie 的 embedding 用 tsne 降維後，將 movie category 當作 label 來作圖。

(collaborator:)

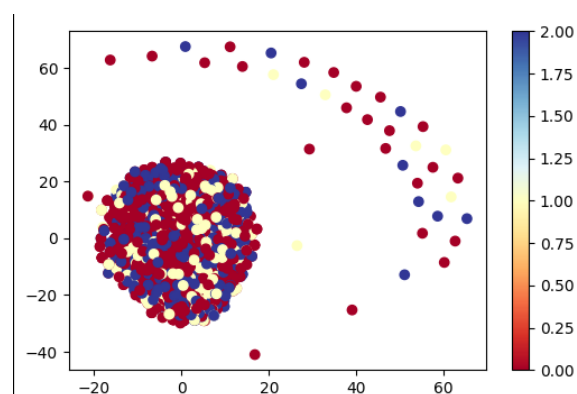
我主要分成三類，分類方法如下：

'Horror' or 'Thriller' or 'Crime': y = 1

'Drama' or 'Musical': y = 2

other: y = 0

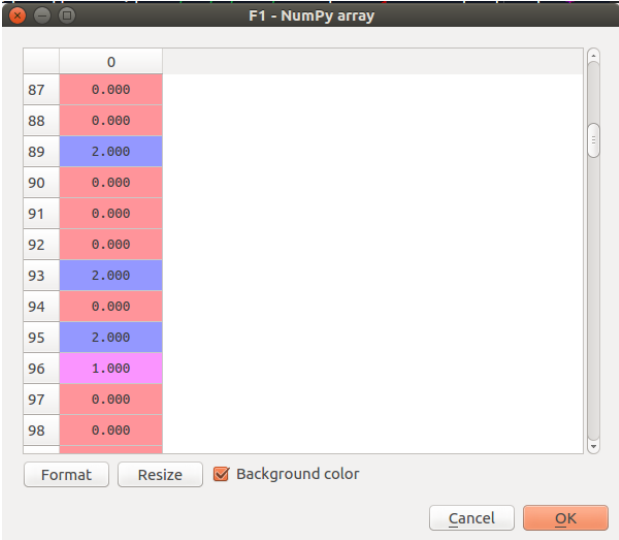
最後的結果會發現其實不太能分辨的很開，而且每次作圖時用 tsne 降維出來的結果都會不太相同，所以最後就沒有再試了。



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

Kaggle: 1.24179,1.24076

我利用上一題的分類將 feature 分成 0,1, 2 三種不同類型大概長這樣放進 bias 裡面, 下去 train, 實驗結果準確率並沒有很好, 推測原因是因為其實它本身在上一題圖形化的分佈中就沒有很明顯的分群了, 所以當作 bias 其實並不是一個很好的 feature。



	0
87	0.000
88	0.000
89	2.000
90	0.000
91	0.000
92	0.000
93	2.000
94	0.000
95	2.000
96	1.000
97	0.000
98	0.000