

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

在 `latent dimension` 為 120, 使用 35 個 `epoch`, `validation_split` 採用 0.1, `dropout_rate` 為 0.2 的情況之下：

無 `normalize` : Public \rightarrow 0.86507 Private \rightarrow 0.86750

使用 `(rating-mean) \div std` : Public \rightarrow 0.86216 Private \rightarrow 0.86983

使用 `rating \div 5` : Public \rightarrow 0.87354 Private \rightarrow 0.87590

感覺只對 `rating` 做 `normalize` 並無太大差別。

2. (1%)比較不同的 `latent dimension` 的結果。

使用 35 個 `epoch`, `validation_split` 採用 0.1, `dropout_rate` 為 0.2 的情況之下：

120 : Public \rightarrow 0.86507 Private \rightarrow 0.86750

240 : Public \rightarrow 0.86877 Private \rightarrow 0.87178

360 : Public \rightarrow 0.89038 Private \rightarrow 0.88971

可以發現到, `latent dimension` 落在 100 至 200 之間, 效果最佳。

3. (1%)比較有無 `bias` 的結果。

在 `latent dimension` 為 120, 使用 35 個 `epoch`, `validation_split` 採用 0.1, `dropout_rate` 為 0.2 的情況之下：

使用 `bias` : Public \rightarrow 0.86507 Private \rightarrow 0.86750

不用 `bias` : Public \rightarrow 0.91505 Private \rightarrow 0.91441

可以發現到, `bias` 所占的比重其實非常大, 如果沒有 `bias` 的存在, 容易導致訓練發生 `underfitting` 的情況。

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。

在 `latent dimension` 為 120, 使用 35 個 `epoch`, `validation_split` 採用 0.1, `dropout_rate` 為 0.2 的情況之下, 預測結果會比 `MF model` 好, 二者的原理不太相同, `MF` 是使用 `dot` 來變成純量, 而 `NN` 則是使用 `dense` 來歸化成純量, 不過, 如同老師上課所述, `NN` 的效果是可以贏過 `MF` 的。

	DNN model	MF model
Public	0.86171	0.86507
Private	0.86508	0.86750

DNN model

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 1)	0
input_2 (InputLayer)	(None, 1)	0
embedding_1 (Embedding)	(None, 1, 120)	724800
embedding_2 (Embedding)	(None, 1, 120)	474240
flatten_1 (Flatten)	(None, 120)	0
flatten_2 (Flatten)	(None, 120)	0
concatenate_1 (Concatenate)	(None, 240)	0
dropout_1 (Dropout)	(None, 240)	0
dense_1 (Dense)	(None, 120)	28920
dropout_2 (Dropout)	(None, 120)	0
dense_2 (Dense)	(None, 1)	121
Total params: 1,228,081		
Trainable params: 1,228,081		
Non-trainable params: 0		

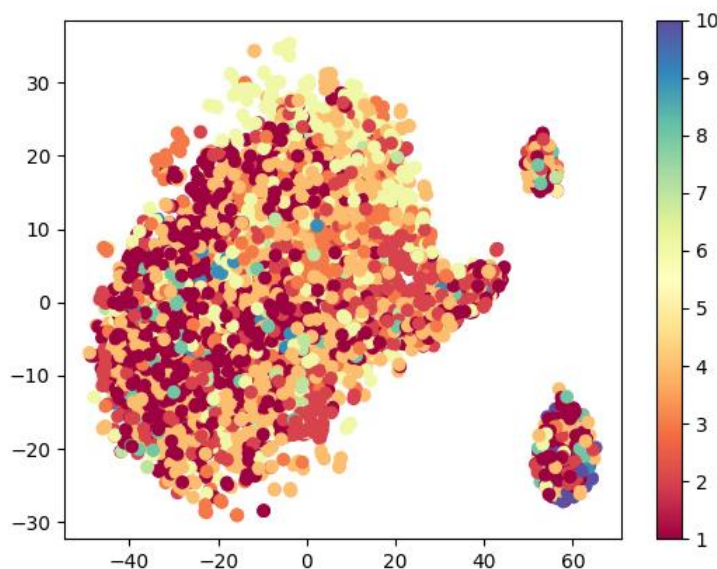
MF model

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 1)	0
input_2 (InputLayer)	(None, 1)	0
embedding_1 (Embedding)	(None, 1, 120)	724800
embedding_2 (Embedding)	(None, 1, 120)	474240
flatten_1 (Flatten)	(None, 120)	0
flatten_2 (Flatten)	(None, 120)	0
dot_1 (Dot)	(None, 1)	0
Total params: 1,199,040		
Trainable params: 1,199,040		
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。

使用下面的分類來做圖：

Drama,Musical	1
Thriller,Horror,Crime,Film-Noir	2
Adventure,Animation,Children	3
Comedy	4
Romance	5
Action,War	6
Mystery,Sci-Fi,Fantasy	7
Documentary	8
Western	9
No label	10



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。

在 **latent dimension** 為 180, 使用 35 個 **epoch**, **validation_split** 採用 0.1, **dropout_rate** 為 0.2 的情況, 並採用性別以及年齡, 並加上先前的使用者以及電影共 4 個 **embedding array**, 將他們全部 **concatenate** 之後, 丟到二層的神經網路之中訓練, 並分別使用 **relu** 以及 **linear** 來做 **activation**。

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 1)	0
input_2 (InputLayer)	(None, 1)	0
embedding_1 (Embedding)	(None, 1, 180)	1087200
embedding_2 (Embedding)	(None, 1, 180)	711360
input_3 (InputLayer)	(None, 1)	0
flatten_1 (Flatten)	(None, 180)	0
flatten_2 (Flatten)	(None, 180)	0
embedding_3 (Embedding)	(None, 1, 180)	360
input_4 (InputLayer)	(None, 1)	0
concatenate_1 (Concatenate)	(None, 360)	0
flatten_3 (Flatten)	(None, 180)	0
embedding_4 (Embedding)	(None, 1, 180)	2160
concatenate_2 (Concatenate)	(None, 540)	0
flatten_4 (Flatten)	(None, 180)	0
concatenate_3 (Concatenate)	(None, 720)	0
dropout_1 (Dropout)	(None, 720)	0
dense_1 (Dense)	(None, 180)	129780
dropout_2 (Dropout)	(None, 180)	0
dense_2 (Dense)	(None, 1)	181
Total params: 1,931,041		
Trainable params: 1,931,041		
Non-trainable params: 0		