

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

	有 normalize	無 normalize
Private	0.86175	0.85386
Public	0.86027	0.85223

我覺得 notmalize 出來的效果並沒有比無 normalize 出來的效果好，我這裡沒有 normalize 會先將 rating shift 到-2 ~ 2，之後 predict 的值在加回 3。我 normalize 的方法是先去計算 training data rating 的 mean 和 standard deviation，再將 training data rating 做 normalized 當作 training input 的 rating。

2. (1%)比較不同的 **latent dimension** 的結果。
(collaborator:無)

latent dimension	32	64	128	256
Private	0.86022	0.85722	0.85287	0.85350
Public	0.85991	0.85527	0.85234	0.85173

我發現在 public 的分數，當 **latent dimension** 越大，分數會越好，但是在 private 的分數就不會是 **latent dimension** 越大越好，反而是在 128 的時候是最好。我覺得可能是 128 維就能很好的代表 data，但是當過大的時候，可能就包含了過多比較沒有用的資訊，所以 **performance** 不一定比較好。

3. (1%)比較有無 **bias** 的結果。
(collaborator:無)

	bias	無 bias
private	0.85386	0.85345
public	0.85223	0.85280

我發現有加 bias 和沒加 bias 的 performance 看起來好像沒有差很多，可能 user 評分的傾向好像對這個 case 沒有太大的影響。

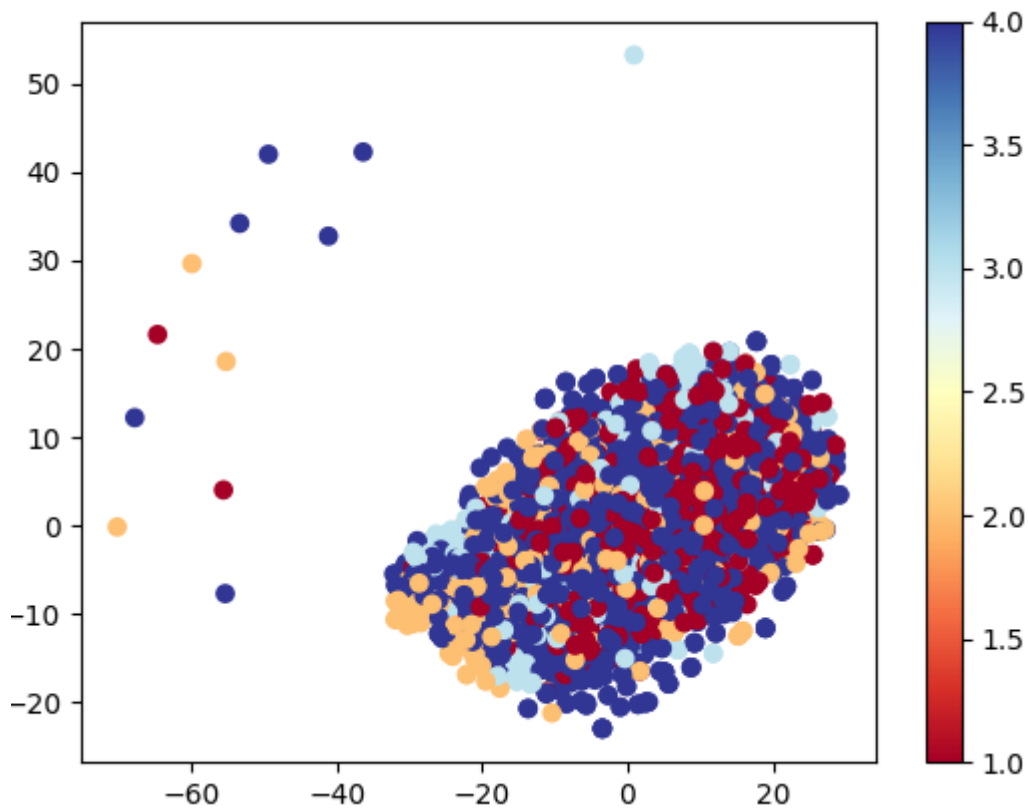
4. (1%)請試著用 **DNN** 來解決這個問題，並且說明實做的方法(方法不限)。並比較 **MF** 和 **NN** 的結果，討論結果的差異。
(collaborator:無)

我將 **user_input** 和 **movie_input** 分別接一層 **embedding layer**，再分別用 **flatten** 將他們攤平，之後將他們 **concatenate** 起來，在接三層 **Dense layers** 使他降到 1 維輸出。

DNN performance: private= 0.87403 public=0.87547			
Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 1)	0	
input_1 (InputLayer)	(None, 1)	0	
embedding_2 (Embedding)	(None, 1, 256)	1546496	input_2[0][0]
embedding_1 (Embedding)	(None, 1, 256)	1011968	input_1[0][0]
flatten_2 (Flatten)	(None, 256)	0	embedding_2[0]
flatten_1 (Flatten)	(None, 256)	0	embedding_1[0]
concatenate_1 (Concatenate)	(None, 512)	0	flatten_2[0][0] flatten_1[0][0]
dense_1 (Dense)	(None, 150)	76950	concatenate_1[0]
dense_2 (Dense)	(None, 50)	7550	dense_1[0][0]
dense_3 (Dense)	(None, 1)	51	dense_2[0][0]
Total params: 2,643,015			
Trainable params: 2,643,015			
Non-trainable params: 0			
..			
MF performance: private= 0.85386 public=0.85223			
Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 1)	0	
input_1 (InputLayer)	(None, 1)	0	
embedding_2 (Embedding)	(None, 1, 256)	1536256	input_2[0][0]
embedding_1 (Embedding)	(None, 1, 256)	1536256	input_1[0][0]
flatten_2 (Flatten)	(None, 256)	0	embedding_2[0][0]
flatten_1 (Flatten)	(None, 256)	0	embedding_1[0][0]
embedding_4 (Embedding)	(None, 1, 1)	6001	input_2[0][0]
embedding_3 (Embedding)	(None, 1, 1)	6001	input_1[0][0]
dot_1 (Dot)	(None, 1)	0	flatten_2[0][0] flatten_1[0][0]
flatten_4 (Flatten)	(None, 1)	0	embedding_4[0][0]
flatten_3 (Flatten)	(None, 1)	0	embedding_3[0][0]
add_1 (Add)	(None, 1)	0	dot_1[0][0] flatten_4[0][0] flatten_3[0][0]
Total params: 3,084,514			
Trainable params: 3,084,514			
Non-trainable params: 0			

我的 dnn 的 performance 比 MF 的 performance 還要不好，我覺得可能是我 Dense layer 的參數設的不夠好，可能還要再多加嘗試。

5. (1%)請試著將 movie 的 embedding 用 tsne 降維後，將 movie category 當作 label 來作圖。
(collaborator:無)
1: 'Drama'、'Musical'
2: 'Thriller'、'Horror'、'Crime'
3: 'Adventure'、'Animation'、"Children's"
4: Other



我覺得這個分類看起來沒有把 **data** 分得很開，我覺得有可能這樣的分類沒辦法很明顯的區分，或是他在高維上可以區分，但降到 2 維之後，可能在這樣的平面上無法將它區分出來。

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。

(collaborator: 無)

我將 **movie.csv** 中的每一行先用 **::** 區隔開來，再將最後一個個資料(ex: "Animation|Children's|Comedy") 用 **|** 來隔開取第一個資料(ex: "Animation")，我將這個部分稱為 **movie** 的類別。把每個 **train data** 的 **movie** 類別當作 **movie_bias** 的 **train input**。但做出來的效果沒有很好，可能是因為加入的 **feature** 對這個 **case** 沒有甚麼影響。

	無加入額外的 features	有加入額外的 feature
Private	0.85386	0.85916
public	0.85223	0.85742