1. (1%)請比較有無 normalize(rating)的差別。並說明如何 normalize.
(collaborator: b04901022 黃家翰)

a. 有無 normalize 的差別:

我在有 bias 跟無 bias 的模型各自嘗試了下面有無 normalization 的實驗，結果如下表所示:

CFModel(no bias): dim = 24 epoch=20	
With no normalization	With normalization
Kaggle public score: 0.86606	Kaggle public score: 0.86539
MFModel(with bias) dim = 24 epoch = 20	
With no normalization	With normalization
Kaggle public score: 0.87129	Kaggle public score: 0.85979

由上表可以知道，有做 normalize 可以降低 testing loss，讓預測更精確。推測其原因，應該是因為產生的模型比較可以適應 testing data 的資料分布(假定 training data 的分布不會與 testing 相差太多)，可以減少誤差。而如果直接丟了 training data rating 下去訓練的話可能會因為機器不是按照分布去預測而是按照數值去預測而有比較大的分布誤差。

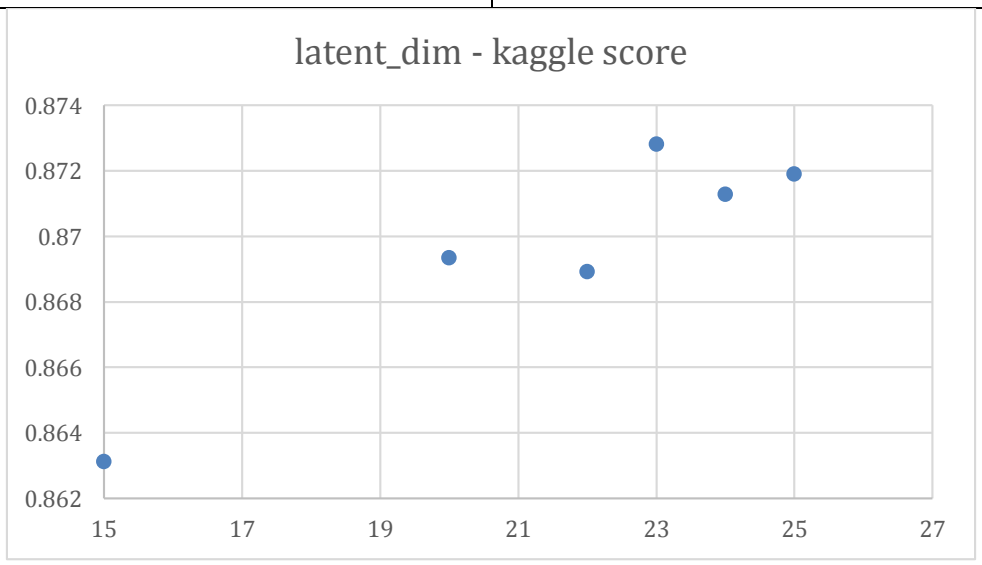
b. 如何 normalize:

我直接對所有 rating 做 standard normalization，即減去平均除以標準差。我試過用 clip 到[0,1]的方式(/=5)但是效果沒有比直接做 standard normalization 好。推測原因可能是以標準化的資料來訓練模型模型比較能適應新的資料，受新資料的其他外布雜訊影響較低。

2. (1%)比較不同的 latent dimension 的結果。
(collaborator: b04901022 黃家翰)

以下所有模型都是相同的架構(用 Dot, 有 bias, normalize 到[0,1])

Latent dimenson	Kaggle public score
25	0.87190
24	0.87129
23	0.87281
22	0.86893
20	0.86934
15	0.86313



跟據實驗的結果，得到 latent dimension 在 15 時會有最好的表現。而我自己的 best model 也是直接用相同架構的 model(用 dim=24, 由選最佳 valid_loss 而決定) 只是將 rating 做了 standard normalization (減去平均除以標準差)，在 kaggle public score 得了 0.858979 的成績。(最佳成績 0.84735 是來自三個結果 ensemble)

3. (1%)比較有無 bias 的結果。

(collaborator: b04901022 黃家翰)

在我的模型中，寫好兩種 class 方便自己呼叫 model，其中有兩種就分別是為了做有 bias 的 Matrix Factorization (名為 MFModel, 用 Dot)和無 bias 的 Matrix Factorization (名為 CFModel, 亦為用 Dot)。以下對兩者的結果進行比較:

Model name	CFModel(no bias)	MFModel(with bias)
Model structure	Merge with Dot Standard normalization Epoch = 20 Latent dimension = 24	Merge with Dot Standard normalization Epoch = 20 Latent dimension = 24
Kaggle public Score	0.86539	0.85979

由結果可見，MFModel 在相同的模型架構之下，有著比較好的成績。自己推論原因，可能是因為模型中多了一個 bias 可以加入一些無關乎個別使用者與電影的因素所造成的影響(如: 人們本來就喜歡看電影的程度、電影評分的傾向…)，讓 rating 預測可以隨著 training data 做調整。另外，如果將 normalization 的方式換成 clip to [0,1](全部 rating /= 5)，則結果會如下表:

Model name	CFModel(no bias)	MFModel(with bias)
Model structure	Merge with Dot Normalize to [0,1] Epoch = 20 Latent dimension = 23	Merge with Dot Normalization to [0,1] Epoch = 20 Latent dimension = 23
Kaggle public Score	0.86902	0.87281

在大多數的情況下，無 bias 的反而會取得較好的成績。推測原因為在做 clip 下，加入 bias 反而會是一種不當的行為，因為會將原本的評分區間[0,1]shift 到[bias, 1+bias]，導致結果偏移，RMSE 變大。因此在選擇是否要有 bias 前，應考慮 normalization 的方式。

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:)

a. 實做方法:

我的模型 class 中(CFModel.py)中，其中一個模型 DeepModel 即是用來實現 DNN 解決 movie recommendation 的問題的模型。我使用一層 concatenate 將 Embbeding 後的 user, movie 丟入，在將結果 input 進 DNN model 進行 training。我的模型架構如下圖所示:

```
Layer (type)                 Output Shape              Param #     Connected to
====================================================================================================
merge_1 (Merge)              (None, 48)                0           reshape_1[0][0]
                                                                   reshape_2[0][0]
----------------------------------------------------------------------------------------------------
dropout_1 (Dropout)          (None, 48)                0           merge_1[0][0]
----------------------------------------------------------------------------------------------------
dense_1 (Dense)              (None, 150)               7350        dropout_1[0][0]
----------------------------------------------------------------------------------------------------
dropout_2 (Dropout)          (None, 150)               0           dense_1[0][0]
----------------------------------------------------------------------------------------------------
dense_2 (Dense)              (None, 50)                7550        dropout_2[0][0]
----------------------------------------------------------------------------------------------------
dense_3 (Dense)              (None, 1)                 51          dense_2[0][0]
====================================================================================================
Total params: 253,151
Trainable params: 253,151
Non-trainable params: 0
```

在這個架構之下，validation loss (0.0297) 降到比之前訓練 MF model 的 validation loss (0.0310) 還低，在 kaggle 上的成績為: 0.86485，也比很多之前訓練的 MF model 的成績還要好。

為使兩邊公平比較，我將 DNN 架構改為一層 24 個 unit 德 Dense 使參數量接近，下表為相近參數量下的 MFModel 與 DeepModel 在 kaggle 上的成績:

Model name	MFModel	DeepModel
Model structure	Merge with Dot Normalization to [0,1] Epoch = 15 Latent dimension = 24 Parameters: 238,202	Merge with Concatenate Normalization to [0,1] Epoch = 15 Latent dimension = 24 Parameters: 239,401
Kaggle public score	0.87129	0.87539

在相近的參數量之下，以有 bias 的單純 merge with dot 的模型有較好的成效。推究原因有二: 一是因為 DNN 的參數可能調整得還不夠，加上我將二者的參數盡量調成相近限制了 DNN 的表現；其二是在較少的參數限制之下，因為 Matrix Factorization 所造成的預測已經滿接近實際預測值，因而若是直接將參數相接進行 DNN 可能也至多學到跟 dot 起來相去不遠的結果。

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

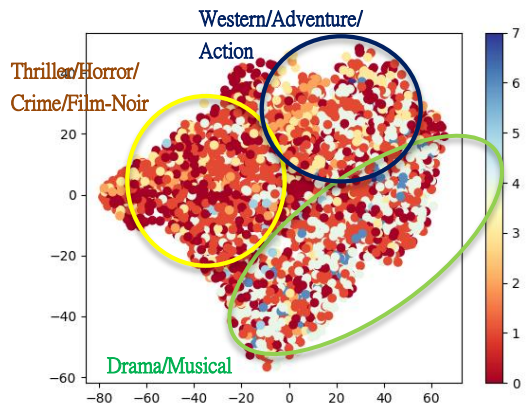
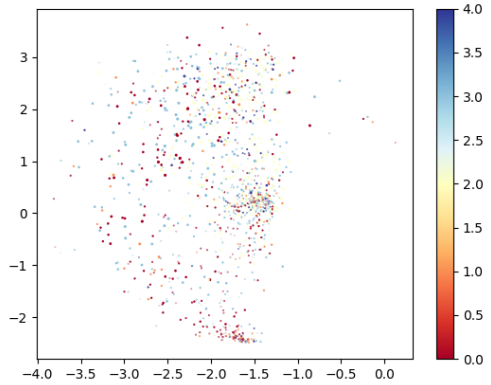
(collaborator:)

我使用 sklearn 的 TSNE 做為降維的工具，並且手動將所有電影切分至 7 個類別，分別為

0) Thriller/Horror/Crime/Film-Noir 1) Comedy/Romance 2) Childrens/Animation

3) Western/Adventure/Action 4)Drama/Musical 5)Fantasy/Sci-Fi 6)Documentary 7)War

以下是做了 TSNE 的結果(二維與三維)

TSNE(n_components=2)	TSNE(n_components=3)
	
二維降維圖中，儘管 perplexity 已經調整到非常高，但是分離效果仍然不明顯。大概可以看出比較明顯得三個分群: 白點的 Drama/Musical(綠圈)、橙點的 Thriller/Horror/Crime/Film-Noir (黃圈)跟米黃點的 Western/Adventure/Action(藍圈)。	在三維降維圖中，點與點的分離比較明顯。可以看得出來點的分布呈現類似圓錐狀，但是相同 tag 的點的聚落卻不能明顯得看出來。

綜上兩張圖所示，區分並不是很明確。推測原因除了可能模型中的 embedding 沒有做到很好的與 genres 相關之外，另外的可能是我 tag 取的不好，導致也許模型的分布是明顯得但是我的 tag 使他們看似又混在一起了。(但考量到幾乎所有點都集中一塊，可能 embedding 還是沒有做

的很好)。

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:)

我將 movie 分類後的 genre 一起當做模型參數進行 training，分類後的 genres 是指類似 tsne 那題的分類方式(即指將所有電影分成 5 類:見下表)。

Class 1	Class 2	Class 3	Class 4	Class 5
Thriller/Horror/ Crime/Film-Noir	Comedy/Romance	Western/ Action/ Adventure/War	Drama/Musical	Else

將 input 的 movie ID 除了做之前的 embedding 外，另外再將其 embed 至 one-hot 之後的這五類的 vector，全部(user_id embed/ movie_id embed/ one-hot genre/ bias)進行 concatenate 再丟入 DNN 進行 training，最終得到預測的 rating 值。詳細模型架構如下圖:

Model summary

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1)	0	
input_2 (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 24)	144984	input_1[0][0]
embedding_2 (Embedding)	(None, 1, 24)	93216	input_2[0][0]
embedding_3 (Embedding)	(None, 1, 5)	19420	input_2[0][0]
flatten_1 (Flatten)	(None, 24)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 24)	0	embedding_2[0][0]
flatten_3 (Flatten)	(None, 5)	0	embedding_3[0][0]
concatenate_1 (Concatenate)	(None, 48)	0	flatten_1[0][0] flatten_2[0][0]
dense_1 (Dense)	(None, 30)	180	flatten_3[0][0]
concatenate_2 (Concatenate)	(None, 78)	0	concatenate_1[0][0] dense_1[0][0]
dense_3 (Dense)	(None, 500)	39500	concatenate_2[0][0]
dropout_1 (Dropout)	(None, 500)	0	dense_3[0][0]
dense_4 (Dense)	(None, 250)	125250	dropout_1[0][0]
dropout_2 (Dropout)	(None, 250)	0	dense_4[0][0]
dense_5 (Dense)	(None, 50)	12550	dropout_2[0][0]
input_3 (InputLayer)	(None, 1)	0	
dense_6 (Dense)	(None, 1)	51	dense_5[0][0]
dense_2 (Dense)	(None, 1)	2	input_3[0][0]
add_1 (Add)	(None, 1)	0	dense_6[0][0] dense_2[0][0]

Total params: 435,153
Trainable params: 415,733
Non-trainable params: 19,420

最終果不盡理想，val_loss 一直無法降到 0.039 以下，相比於 MF model 的結果是表現比較差的。最終在 kaggle 上得 0.89872。推究可能的原因有以下幾點:

a. 選擇的 feature 與 movie_id 的 embed 結果之中有所衝突(如第五題所說，因為 movie_id 的 embedding 可能本來就有 genres 的因素在裡面，如果跟我自己人工分的 genre 不匹配可能使結果變差)。

b. 模型的架構本身有改進空間。參數調整不當、模型架構不符合問題直覺…這些都是可能造成結果不準確的因素。