

Student: Bahram Khodabandehlouee

Cs 614 - 900

Assignment: 2 (Recommender Systems)

## Contents

Pitch .....	2
Data Source .....	2
Model and data justification .....	2
Commented examples .....	3
Testing.....	5
Code and instructions to run it .....	6

## Pitch

Recommending preferred contents to TV enthusiasts for media companies such as Netflix, YouTube TV or Amazon Prime is crucial. we have implemented Collaborative filtering that uses the preferences of a group of users to make recommendations for those users. The goal therefore is to train a neural network to predict ratings on unwatched movies by users, then, media companies can recommend those movies that get the highest predicted ratings to their users.

## Data Source

To build this recommender system, we train our model on the ratings data in the ml-20m dataset. The ml-20m dataset contains over 20 million ratings of movies by users, as well as additional information such as movie titles and user demographics. To get the dataset on grouplens.org click [here](https://grouplens.org/datasets/movielens/). To get the data (only ratings.csv) from my personal google drive, click [here](#).

GroupLense.org: <https://grouplens.org/datasets/movielens/>

## Model and data justification

IN this model, I have two Inputs, one for userID and the other one for movieID. So, we pass them as a list in our model. To do this, one approach is to go deep, with many layers, but less units per layer. The other approach is to go wide, with less layers with more units. So, I decided to go with a single layer with 1024 units and in last layer I will have just 1 unit since we are doing regression and not classification.

```
x = Dense(1024, activation='relu')(x)
# x = Dense(400, activation = 'relu')(x)
# x = Dense(400, activation = 'relu')(x)
x = Dense(1)(x) # last layer has just 1 unit since we are doing regression
```

At the time to build my model and compiling it I used 'mse' measure to get loss and used SGD as my optimizer. You can see them in bellow.


### Building the Model and Compiling it

```
[16] from scipy.optimize import optimize
# This model does not have 1 input and insted it has 2 inputs. we pass them as a list
model = Model(inputs=[u,m], outputs=x)
model.compile(loss='mse', optimizer=SGD(learning_rate=0.08, momentum=0.9),)
# Since we are doing the regression, the loss is mse and we choose the SGD optimizer.
```

## Commented examples

I build my NN as follow. I have 2 inputs, u for userID and m for movieID. I only used one layer and it was my best model. I have a Dense layer with 1024 units and my last layer is a Dense layer with 1 unit as it is a regression.

## Building our Neural Network

```
 # User input
# Since the input is just a simple integer, the shape is 1
u = Input(shape=(1,))

# Movie input
m = Input(shape=(1,))

# User embedding
u_emb = Embedding(N,K)(u) # output is (num_samples, 1 , K)

# Movie embedding
m_emb = Embedding(M,K)(m) # output is (num_samples, 1 , K)

# At this point we flatten both embeddings
u_emb = Flatten()(u_emb) # now it is (num_samples, K)
m_emb = Flatten()(m_emb) # now it is (num_samples, K)

# Now we create our feature vector by concatenating u_emb and m_emb
x = Concatenate()([u_emb, m_emb]) # now it is (num_samples, 2K)

# Now we have feature vector that we can pass it through the NN,
# One approach is to go deep, with many layers, but less units per layer.
# Other approach is to go wide, with less layers with more units.
# So, we go with a single layer with 1024 units
# we need a regular ANN
x = Dense(1024, activation='relu')(x)
# x = Dense(400, activation = 'relu')(x)
# x = Dense(400, activation = 'relu')(x)
x = Dense(1)(x) # last layer has just 1 unit since we are doing regression
```

Here we have pass the list of inputs as a list into our model

```
[16] from scipy.optimize import optimize
# This model does not have 1 input and insted it has 2 inputs. we pass them as a list
model = Model(inputs=[u,m], outputs=x)
model.compile(loss='mse', optimizer=SGD(learning_rate=0.08, momentum=0.9),)
# Since we are doing the regression, the loss is mse and we choose the SGD optimizer.
```

Finally we ran on 25 epoch. This took a while to run and I used colab GPU and still around 30 minutes for each run.

```
user_ids, movie_ids, ratings = shuffle(user_ids, movie_ids, ratings)
Ntrain = int(0.8 * len(ratings))

train_user = user_ids[:Ntrain]
train_movie = movie_ids[:Ntrain]
train_ratings = ratings[:Ntrain]

test_user = user_ids[Ntrain:]
test_movie = movie_ids[Ntrain:]
test_ratings = ratings[Ntrain:]

# Center the ratings
avg_rating = train_ratings.mean()
train_ratings = train_ratings - avg_rating
test_ratings = test_ratings - avg_rating

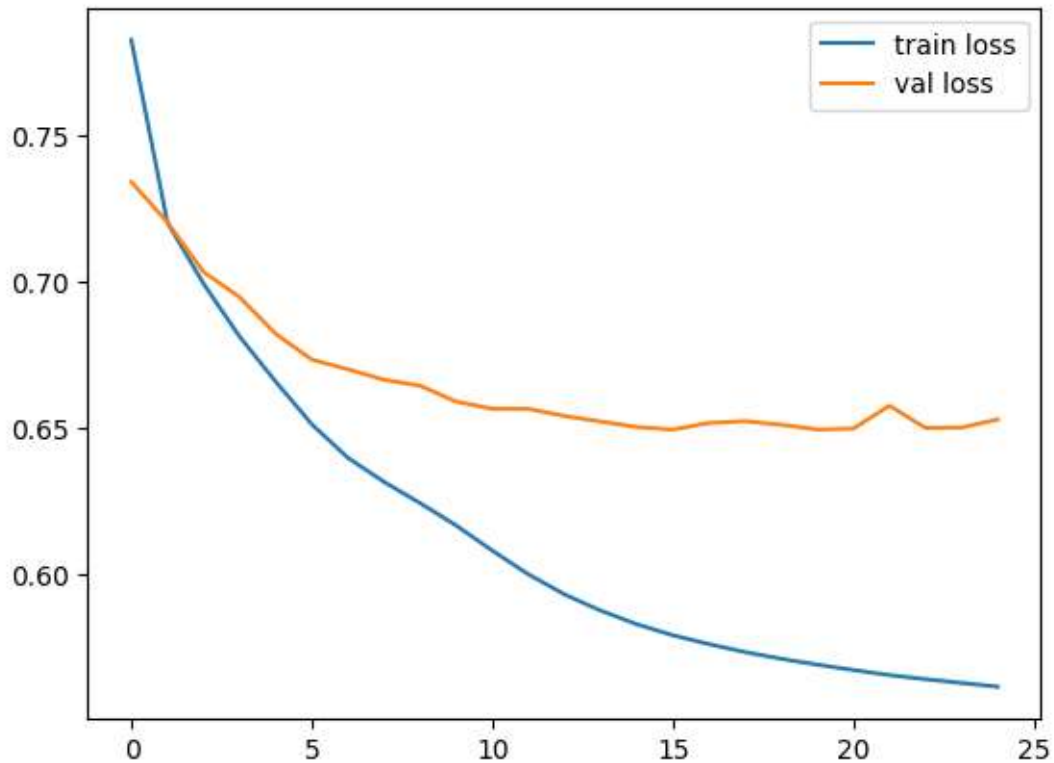
r = model.fit(
    x=[train_user, train_movie],
    y = train_ratings,
    epochs = 25,
    batch_size = 1024,
    validation_data = ([test_user, test_movie], test_ratings),
)
```

```
Epoch 1/25
15626/15626 [=====] - 97s 6ms/step - loss: 0.7825 - val_loss: 0.7825
Epoch 2/25
15626/15626 [=====] - 78s 5ms/step - loss: 0.7201 - val_loss: 0.7201
Epoch 3/25
15626/15626 [=====] - 78s 5ms/step - loss: 0.6992 - val loss: 0.6992
```

Here you see the loss value in last epoch

```
Epoch 25/25
15626/15626 [=====] - 78s 5ms/step - loss: 0.5613 - val_loss: 0.6521
```

Here we see the train\_loss vs test\_loss



Here is the loss on test set

## loss on test sets



```
# Evaluate the model on the test set
test_loss = model.evaluate([test_user, test_movie], test_ratings)

# Print the test accuracy
print(f"Test accuracy: {test_loss:.2%}")
```



```
125002/125002 [=====] - 284s 2ms/step - loss: 0.6527
Test accuracy: 65.27%
```

## Testing

At first, when I tried to get my confusion matrix it came out like a 1x1 array, as follow:

Confusion matrix:  
[[4000053]]

This indicates that I only have one class, in other words, all of the ratings in my test set are either positive or negative. That happened because I was subtracting the `y_pred` and `test_ratings` from 3.5. Later I realized that the mean value for ratings is much less than that, so I decided to use `test_ratings.mean()` value and got the following result in a 2x2 confusion matrix.

```
[▶] from sklearn.metrics import confusion_matrix

# Get the predicted ratings for the test set
y_pred = model.predict([test_user, test_movie])

# Convert the ratings to binary values (0 or 1)
y_pred_binary = (y_pred >= test_ratings.mean()).astype(int)
y_true_binary = (test_ratings >= test_ratings.mean()).astype(int)

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_true_binary, y_pred_binary)

print("Confusion matrix:")
print(conf_matrix)
```

```
↳ 125002/125002 [=====] - 222s 2ms/step
Confusion matrix:
[[1419907  581060]
 [ 466740 1532346]]
```

Based on this result,

**true positives**= 1419907  
**false positives** = 581060  
**false negatives** = 466740  
**true negatives** = 1532346

## Code and instructions to run it

You can find the code for my project in my **google drive** in the link bellow:

Link to google Drive: <https://drive.google.com/drive/folders/1fFB08lAXDs-Wv4kknBmksgOtPkNgXbdq?usp=sharing>

To **run the code**, you should run it in **google colab** (copy only the code in your own drive or run it in mine) as it will load the dataset in colab setting. Otherwise, you have to download the ratings.csv file from my google drive (link to my drive is provided) or from [grouplense.org](https://grouplense.org) and then load that dataset in your system and run the rest of the code.

I agree to share my code with other students