User-book Ratings Prediction Modeling

All-things-data
- Loading
- Processing
- Cleaning

Model Design and Architecture
- Training and Validation Set
- Loss Function and Error Metrics

References

Predictive modeling is one of the many recommendation systems class of problems that this project aims to solve.

Specially, given historical data, one could learn the preferences of the users towards ratings certain genre books. The classical approach called collaborative filtering is employed in this project. Our technique will be based on the following observations

- Users who rate books similarly, have one or more hidden preferences.
- Users with shared preferences are likely to give ratings in the same way to the same books.

There are a few different approaches to solving this problem.

- **User-based Collaborative Filtering**: Similarity metrics such as cosine similarity could be used to leverage underlying patterns in user behavior.
- **Item-based Collaborative Filtering**: With similarity shifting focus on the books, we can predict what users who've rated other similar book may give for the candidate in question.
- **Autoencoders**: With smaller hidden layer than input layer. This effect forces the model to create a compressed representation of data in the hidden layer by learning correlations in the data. Additional hidden layers enable the encoders to learn more mathematically complex underlying patterns in the data.
- **Matrix Factorization**: Learning low dimensional representation of user and books. We then combine the embeddings to estimate the ratings on unread books.Embeddings provide a mapping from discrete objects - such as words or ids of books in our case, to a vector of continuous values. This finds similarities between discrete objects that wouldn't be apparent to the model if it didn't use embedding layers.

- **Neural Network with embedding layers:** This drives accuracy up of the previous Matrix Factorization Embedding representation even further, with concatenated embeddings feeding the fully connected layers at the output instead of the Dot Product.

For the purposes of this project, the last approach - Neural Network architecture was chosen to find underlying patterns in preferences for rating prediction.

All-things-data

The dataset as per the requirement, was picked from the Goodreads archive. For purposes of this exercise, goodreads_interactions_fantasy_paranormal.json.gz was chosen as per request.
https://github.com/MengtingWan/goodreads

As per the problem statement, the mandatory columns in the dataset considered were
user_id
book_id
rating
read_at
date_updated

Loading

This was probably the most challenging part in figuring out how to peacemeal the data to be loaded in a form my lightweight Asus laptop could handle with 16GB memory.

Reading in batches definitely helped. So was leveraging Pandas' dataframes rather than loading chunks into memory and then fusing them.

The number of JSON objects/records in the given set amounts to 55,397,550.
I decided to load them with sanity checks, although I could've loaded them all and then sanitized them after.

Upto 100K records were loaded in 2 mins.
Next 100K - 5M records in, 10 mins.
Next   5M - 20M records in, 30 mins - (569,752 valid records).
Next 20M - 40M records in, 25 mins - (938,547 valid records).
Next 40M - 55M records in, 16.67 mins - (462,251 valid records).

Processing

Reading data of massive nature such as this, required efficient loading and organization. JSONs are unstructured dictionary formats that make storing large contents in memory very resource intensive.

Pandas however provides an efficient SQL-like data schema structure in so-called dataframes that can be manipulated chunks at a time without loading the entire dataset in memory.

Loading was also efficiently done in batches, using Python’s itertools as iterators with slicing to read rows of the file into dataframes.

Cleaning

Since we’re interested in finding correlations between user preferences of read books, discarding data that didn’t have **is_read** field set (or set to False) was done. Also, with some preliminary inspection of data, noticing the **read_at** field is not necessarily indicative of the status/review. Thus for the purposes of timestamp observation (the year 2016 was used to filter further for historical data of interest, as per problem statement), the **date_updated field was chosen.**

Outside of the above logical cleaning steps, sanity checks were performed for validity of the mandatory fields **- user_id**, **book_id**, **ratings**

After loading the data into several dataframes and merging them together, and dropping the **date_updated** column, we’re ready to use the dataset for training.

Model Design and Architecture

While Matrix Factorization (with Embedding Model) can give reasonably close approximations to learning the underlying pattern, a Neural Network would be preferable in reaching higher accuracies.

Neural Network with merged embeddings with fully-connected layers and ReLu activation helps learning more complex non-linear relationships. For the purposes of the current scope of the problem, 5 latent (hidden) embedding layers for user and book features were used. The embeddings are the weights that are learned during the training.

I use Keras with TensorFlow backend framework for Network modeling and training of the Embedding architecture. I trained this network **without a GPU** on my Asus Zenbook lightweight laptop, and thus went for choosing a simplistic yet accurate enough model architecture and number of layers. Fully-Connected layers in the end helped increase accuracy beyond the Embedding layers.

Training and Validation Set

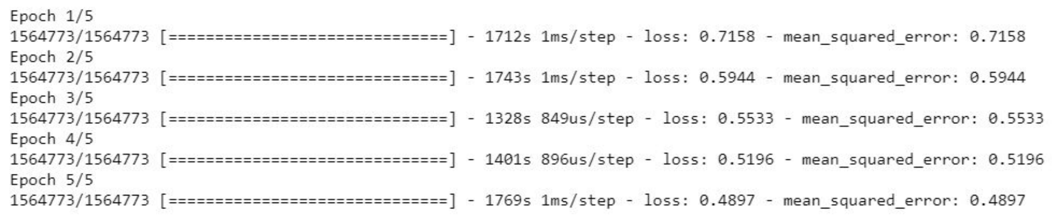
From the given dataset for 2016, the training and validation sets were obtained by filtering all values devoid of December 2016 from the timestamp column (**date_updated**).

Using a 80:20 split for Training and Validation sets, the model was trained on the training set and evaluated on the validation set.

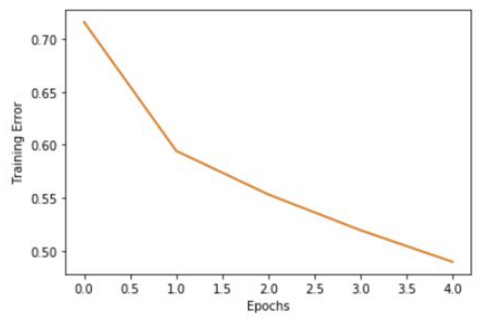
Loss Function and Error Metrics

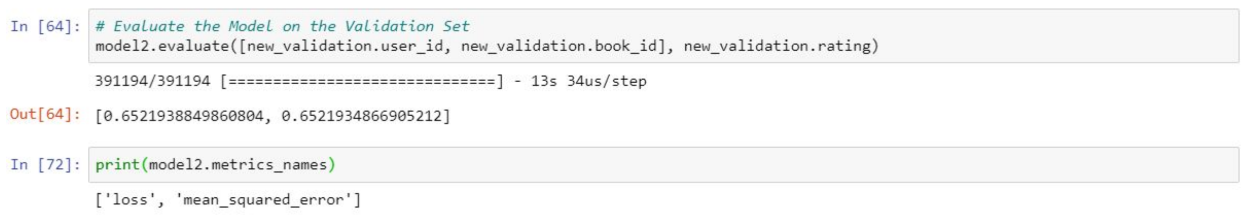
I used the Adam Optimizer and MSE as both - loss function and error metric.
Trained for **5 epochs** as the errors decreased fast enough to provide a reasonably close accuracy. Training for more epochs could've helped further, but as you can see in the Jupyter Notebook attached, on my windows laptop it was taking quite some time for each epoch (~1750 secs). A GPU would have helped speed it up and I could've trained on more epochs.

As shown in the IPython notebook attached, the training loss over the epochs shows a downward trend



```
Epoch 1/5
1564773/1564773 [==============================] - 1712s 1ms/step - loss: 0.7158 - mean_squared_error: 0.7158
Epoch 2/5
1564773/1564773 [==============================] - 1743s 1ms/step - loss: 0.5944 - mean_squared_error: 0.5944
Epoch 3/5
1564773/1564773 [==============================] - 1328s 849us/step - loss: 0.5533 - mean_squared_error: 0.5533
Epoch 4/5
1564773/1564773 [==============================] - 1401s 896us/step - loss: 0.5196 - mean_squared_error: 0.5196
Epoch 5/5
1564773/1564773 [==============================] - 1769s 1ms/step - loss: 0.4897 - mean_squared_error: 0.4897
```





```
In [64]: # Evaluate the Model on the Validation Set
         model2.evaluate([new_validation.user_id, new_validation.book_id], new_validation.rating)

         391194/391194 [==============================] - 13s 34us/step

Out[64]: [0.6521938849860804, 0.6521934866905212]

In [72]: print(model2.metrics_names)

         ['loss', 'mean_squared_error']
```

While the noted MSE and Loss is ~0.65 on the ratings, we're concerned with positive/negative recommendations for this problem. With future scope for tuning our models, we could use this to evaluate our test dataset and predict positive or negative vote for (user, book) combinations.

Predicting Outcomes as Real Values instead of Binary Classification

I didn't resort to Sigmoid in the last layer with minimizing Binary Cross-Entropy loss treating this as a binary classification problem, as reducing the MSE of absolute value predictions could get us more accurate results in the ranking values themselves.

If preferred, one could have a sigmoid layer and one-hot encoded the ground-truths to train for BCE after the 128-to-32 neurons ReLU conversion layers in the end.

References

My Journey to building Book Recommendation System.
https://towardsdatascience.com/my-journey-to-building-book-recommendation-system-5ec959c41847

Building A Collaborative Filtering Recommender System with TensorFlow
https://towardsdatascience.com/building-a-collaborative-filtering-recommender-system-with-tensorflow-82e63d27b420

Deep Autoencoders for Collaborative Filtering
https://towardsdatascience.com/deep-autoencoders-for-collaborative-filtering-6cf8d25bbf1d