CSE 258 Assignment 2: Neural CF for Rating Prediction

Patrick Youssef - psyoussef@ucsd.edu

1 Introduction

As part of this course we were tasked with proceeding with an open-ended recommender system project. Here I will discuss my dataset selection, predictive task, model, literature, and results regarding this project. I completed this project alone so no partners to list here.

2 Dataset

I chose to use a beer reviews dataset from kaggle [1] that the homework dataset was sampled from. As opposed to the 50k samples we have in the homework, the original dataset gives me access to 1.5m reviews. Practically I was not able to train my models, especially for tuning, with any where near the full samples and instead used a subset of around 100k. To try and mitigate issues regarding selection bias, I shuffled the data before sampling the 100k samples I used moving forward.

2.1 Exploration

While the dataset contains many different pieces of information regarding the beer reviews such as sub ratings for different aspects and review text, I opted to analyze only the information being fed into the model (user id, beer id, rating).

2.1.1 User Review Counts

One metric I wanted to explore was how many reviews were present for each user in the dataset. To do so I iterated through the list of users and saw how many occurrences of the user were present across the reviews. I sorted the data to show the progression of the minimum number of reviews up to the maximum and a general idea of the distribution. In Figure 3 we can see that the majority of users have 100 or less reviews which is more than expected! At the tail end we have some super reviewers with a max of around 350 reviews for a single user.

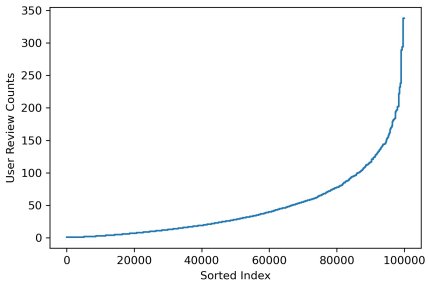

Figure 1: User Review Counts

2.1.2 Beer Review Counts

Another metric I wanted to look at was the count of beer reviews. The result of this check was less exciting as compared to the user review counts as the number of beer reviews is quite low with a few super popular beers having significantly more reviews (note this is plotted on a log scale). Nearly half the beers only have a single review. This is going to make the task of predicting for those beers more difficult as we can't even present a distribution of ratings. This could be improved by using more of the dataset but with my hardware it was difficult to use more than 100k samples. Issues like this are where simple filtering methods fall apart and more complicated methods can begin to really shine by extracting additional features.

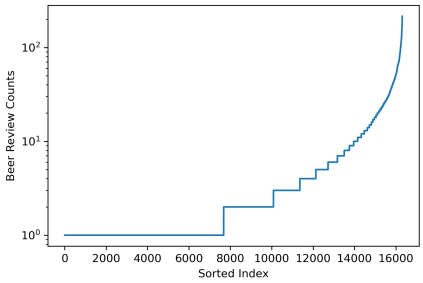

Figure 2: Beer Review Counts

2.1.3 Rating Distribution

One really important aspect to consider is how is the output variable distributed. This can greatly affect the performance of the model. As an example, if all ratings were very similar it can be quite trivial to predict a value near the average and not actually characterize the distinctions. Or if the model distribution is odd, it may be difficult to characterize through our model.

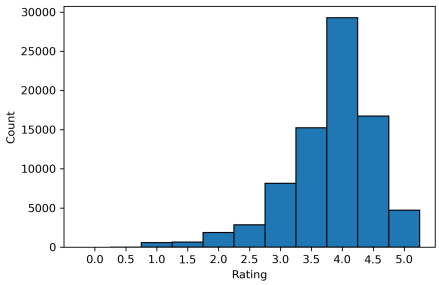

Figure 3: Rating Distribution

3 Predictive Task

Inspired by some of the challenges and interest that I had during rating prediction in Homework 3 and Assignment 1, I opted to proceed with solving the same problem on this dataset. Given some of the simpler models that we used prior (average only and bias-only latent vector) I wanted to step into something better, more on this later.

3.1 Problem Statement

The goal of rating prediction is to determine the real-valued rating that a user would give to a new item (in this case beer). The goal of our model is to act as a function such that the following is achieved:

$$f(\text{User}, \text{Item}) \rightarrow \text{Real-Valued Rating}$$

3.2 Evaluation

As with many real-valued regression type problems, it's fairly standard to compute the Mean Squared Error (MSE) as an evalutaion metric of bulk performance. The objective of MSE is to quadratically penalize differences between the true labels and the predictions over the entire test set.

3.3 Validity

To help ensure the validity of our results I am following a classic Train/Validation/Test split. This way all model training and tuning will be on data that is not present in a final evaluation or test set.

3.4 Baseline

There are two simple baselines to consider as means of comparison.

3.4.1 Random Selection

Although the predictive task is real-valued, the original values are discrete with ratings ranging from $[0, 5]$ in steps of 0.5. The simplest, and one of the worst, baselines we can use is random selection. Essentially for all new samples, we randomly sample from this set of possible ratings and compute the MSE or accuracy (accuracy can only be used here as the output space is also discrete).

This baseline performed really poorly which is no surprise. Over a uniformly distributed we would expect around 10% accuracy as there are 10 discrete classes of ratings but we only achieved roughly half of that with an accuracy of 4.98%. As the ratings are clearly not uniformally distributed the random samples are even less likely to be correct for sample outside of the popular ratings. As expected as well the MSE was also terrible at 11.963.

3.4.2 Average Prediction

Another simple baseline that I used which was inspired from the homeworks is an average only baseline. Here we compute the average rating across the training set and predict the average for all new queries. I expect this to perform much better as it's performance is directly related to how distributed the ratings are and as we saw they are quite tightly distributed near the higher ratings. With an MSE of 0.516 this performed significantly better than the random samples. I'm hoping with further models we can improve this value further.

3.5 Features

The features selection is quite simple, I will only give a query to the model as a user,item pair of ids. The model will maintain embeddings about the users and items and will predict from there. This feature set is quite restrictive but I think it would be fun to see how far we can go with just simple collaborative filtering features.

4 Model

I opted to implement a modified version of the NeuMF model detailed in the paper Neural Collaborative Filtering [2]. Given my experience with the framework and it's popularity, I opted to implement this model using PyTorch.

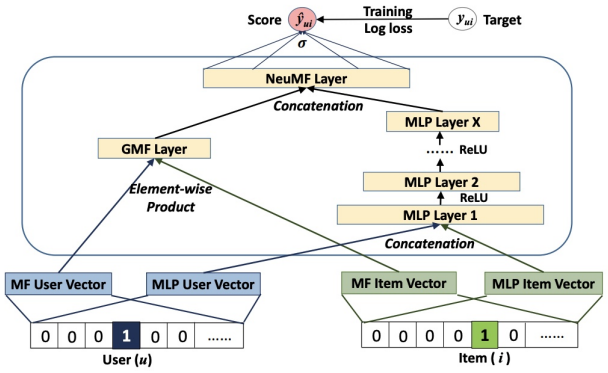

Figure 4: NeuMF Model

4.1 Why This Model?

In my time keeping up with deep learning research, there seems to be highlight papers that breakthrough current paradigms with novel solutions. In terms of using deep learning for collaborative filtering, this paper was clearly one of those novel "need-to-read" papers. In particular the strengths of this model are it employs deep learning, not as a glorified feature extractor, but as the tool to characterize the interactions between user and item embeddings past simple matrix factorization and inner product. In particular the model imploys two instances of their more general neural collaborative filtering (NCF) model: a generalized matrix factorization (GMF) that applies a linear kernel

to the user and item latent embdeddings and a multi-layer perceptron network to learn non-linear interactions between the user and item interactions.

4.1.1 Generalized Matrix Factorization

The authors show in the paper how matrix factorization can be looked at as a special case of their more general neural collaborative filtering model. In practice, the GMF section of the model is flexible enough to recover many factorization models through training. As many of these models were used in prior SOTA, it's no surprise to see that it's included here.

4.1.2 Multi Later Perceptron

One of the limitations of prior matrix factorization models is that the interactions between the user and item were limited to a simple inner product. The authors opted to implement a multi-layer perceptron network to break this limitation and use the network as a universal function approximator for the more complicated interactions.

4.1.3 Fusing

One interesting choice that the authors made in fusing the GMF and MLP components is to produce separate embedding layers for each model. This relieves some constraints on how the models fuse such as enforcing identical latent space dimensions. This provides additional flexibility when tuning the model as you can optimize the embeddings for each individually.
Now that the results from both models can be produced, we concatenate them to create one large feature vector used in the final prediction layer. This is where my biggest change to the model was done, as opposed to the single sigmoid output typically used for more modern recommendation tasks, I opted to remove the final activation after linear layer to produce a real-valued output that will be my rating prediction.

4.2 Optimizing

To optimize the model, I treated the GMF and MLP components as separate to train and tested a variation of parameters for each of the components and then chose the best of both for a final model.

4.2.1 Optimizer

I opted to use the Adam optimizer [3] with a learning rate of 0.002 and a batch size of 512.

4.2.2 GMF Tuning

For the GMF, the only hyperparameter I optimized was the output latent space dimension. I fixed the MLP layers to a simple baseline of 3 layers with reducing dimensionality with each layer, in particular I used $[16, 8, 4]$. As we can see in Figure 5, the lowest validation MSE of 0.387 was obtained when the output dimensions was just $N = 1$. Many of the losses were quite similar to each other and with more time I would explore further dimensions past $N = 20$ but for now I proceeded with using $N = 1$.

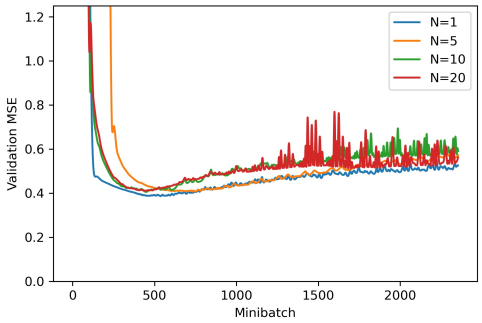

Figure 5: GMF Tuning Validation MSE

4.2.3 MLP Tuning

Tuning the MLP component is a little more challenging as a pure grid search is quite intractable as the number of layers is an additional factor in creating the space of all possible inputs. I opted to fix the layer count to three from prior deep learning experience and maintained a consistent pattern of dimensionality reduction with each layer as follow: $[N, N/2, N/4]$. As we can see in Figure 6, the lowest validation MSE of 0.390 was achieved with $N = 128$. This is a little concerning as we did not expose any inflection point so it's likely a better parameter lies past $N = 128$, but for now given how long this took to train I will be using $N = 128$.

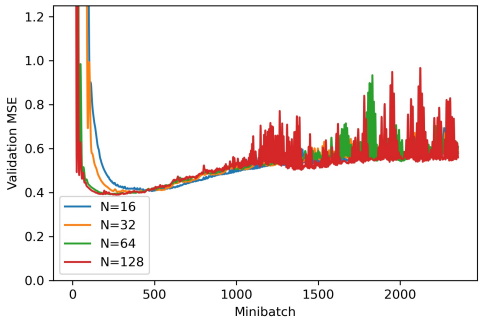

Figure 6: MLP Tuning Validation MSE

4.3 Overfitting

An interesting issue I came across is that model began overfitting very quickly. My prior experience with tuning deep learning models is in the computer vision space where models require many more epochs to train correctly. I applied some light regularization and while it pushed the overfitting to a later epoch, the lowest validation loss was not lower than what I initially got.

4.4 Cold Start

One important aspect to note about this model is that because of the discrete structure, there is no ability to cold start for prior unobserved users or beers. This is currently one issues with deep learning based approaches but within a select corpus of users and items these models can perform quite well. There have been attempts at solving this [4], but I did not implement anything like this and instead filtered out my validation and test set to not deal with this.

4.5 Other Models

I opted to compare this model to the a simple matrix factorization model, much like the GMF componenet of the NeuMF model. All of the comparison is done in the results section as I felt it was more fitting there.

5 Literature

Here I will discuss some of the conversations and similarities to the work I am doing in modern literature.

5.1 Dataset

The dataset I am using is not prevalent in common literature but I was able to find some older papers that used it. It would seem to like like the relevance of that dataset was possibly before many of the modern papers starting using other datasets. The paper "Learning Attitudes and Attributes from Multi-Aspect Reviews" in 2012 [5] leverages many of the aspects that I neglected to use in producing my results which I found very interesting. Among many rating prediction or rating ranking models that I see in common literature such as AutoRec [6] and UserReg [9], it's common to see the MovieLens dataset [7] and Yelp dataset [8]. The two aforementioned datasets are really ubiquitous across many recommender system tasks but they're very common in rating based tasks as the amount of implicit feedback present in the sets is very dense compared to many others.

5.2 Other Models

In terms of keeping up with state of the art methods, I really enjoy going through PapersWithCode.com where different tasks will list the rankings of different papers on that task/dataset. When looking at the MovieLens 1M dataset, we can see quite a variety of different types of models that would be considered SOTA based on the rankings such as autoencoders and graph neural nets. Although many of the models follow a similar path and mentality to my approach, I found that the performance and relevance of pure matrix factorization seemed more popular in research than what I was getting in my results.

6 Results

I opted to present my final model results here as opposed to the model section, seemed more fitting.

6.1 Final NeuMF Model

My final model used a latent space size of 1 for the GMF and $[128, 64, 32, 16]$ for the MLP layer sizes. I opted to include a fourth layer as it was easier to test for the layer variation on a single model and it did help a little bit.

6.1.1 Training Results

Below in Figure 7, you can see the shortened training (based on prior epoch counts) with only 2 epochs achieves the lowest validation loss yet of 0.381 without overfitting. As a reminder our baseline to beat, the average only model, has an MSE of 0.516 so we are doing much better!

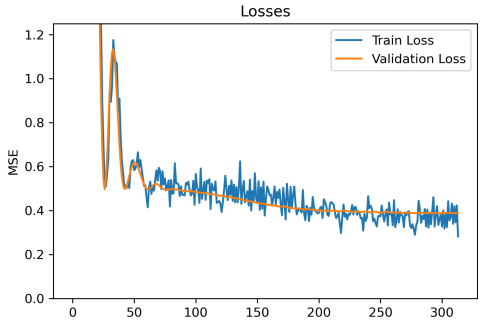

Figure 7: Final Model Training/Validation Loss

It's interesting to see the oscillatory nature of the losses up to about the 50th minibatch. It stabilizes after that but I'm not sure what is causing that (possibly learning rate is too high).

6.2 Tracking Down Error

One small experiment I wanted to do was to see if my prior concerns about beer and user review counts would manifest in more or less error. With very few user and beer reviews I was worried that a query relating to a user or item with very little data would have a difficult time predicting as all learned parameters would be based on very few samples.

6.2.1 User Review Counts

In Figure 8, you can see that with more reviews from a user, queries for that user are typically more accurate. This indicates that, as expected, much of our error is from the limited data that we have to characterize users and items.

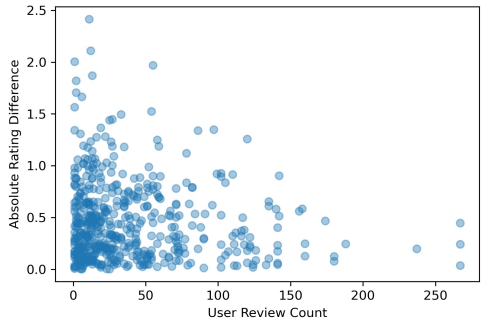

Figure 8: User Review Count Error

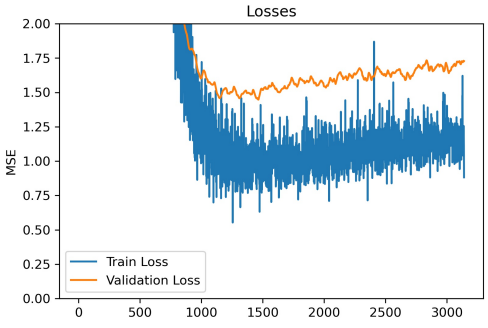

Figure 10: Matrix Factorization Losses

6.2.2 Beer Review Counts

In Figure 9, you can see there is a very clear reduction in the magnitude of error when plotted against the number of prior reviews that query beer had.

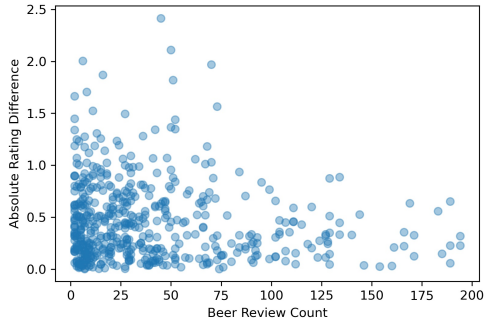

Figure 9: Beer Review Count Error

6.3 Compared against Matrix Factorization

With a simple matrix factorization model I could not get anywhere near the same MSE as what I am getting with NeuMF. I created the matrix factorization model in PyTorch as a subset of the code from the whole NeuMF model where I only had one embedding layer for the user and item and a dot product between the latent space output. In Figure 10l, you can see that for the matrix factorization model I could not achieve a validation loss lower than 1.50 which is much worse than our simple average only model. It seems that the additional non-linear interaction learning happening in the MLP is a massive improvement over the base MF models.

6.4 Conclusion

To conclude, the NeuMF model performed extremely well in the rating prediction task, especially compared to our baseline and prior popular matrix factorization models. The inclusion of the non-linear function approximation using an MLP seems to introduce a massive improvement to the performance and I believe that it's the real meat of the function of this model.
Thanks for a great quarter and a very fulfilling set of homeworks and assignments, I really enjoyed this course!

References

[1] Datadoume. *Beer Reviews*. Oct. 2018. URL: `https://www.kaggle.com/rdoume/beerreviews`.

[2] Xiangnan He et al. *Neural Collaborative Filtering*. 2017. arXiv: `1708.05031 [cs.IR]`.

[3] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: `1412.6980 [cs.LG]`.

[4] Ivan Maksimov, Rodrigo Rivera-Castro, and Evgeny Burnaev. "Addressing Cold Start in Recommender Systems with Hierarchical Graph Neural Networks". In: *2020 IEEE International Conference on Big Data (Big Data)* (2020). DOI: `10.1109/bigdata50022.2020.9378302`.

[5] Julian McAuley, Jure Leskovec, and Dan Jurafsky. *Learning Attitudes and Attributes from Multi-Aspect Reviews*. 2012. arXiv: `1210.3926 [cs.CL]`.

[6] Suvash Sedhain et al. "AutoRec". In: *Proceedings of the 24th International Conference on World Wide Web* (2015). DOI: `10.1145/2740908.2742726`.

[7] Anne-Marie Tousch. *How robust is MovieLens? A dataset analysis for recommender systems*. 2019. arXiv: `1909.12799 [cs.IR]`.

[8] *Yelp Open Dataset*. URL: `https://www.yelp.com/dataset`.

[9] Haiyang Zhang et al. *UserReg: A Simple but Strong Model for Rating Prediction*. 2021. arXiv: `2102.07601 [cs.IR]`.