The Art of Coding

Michael Reigelman & Zhiyi Ren

MSDS 630 Kaggle Team Competition
Spring 2022

Competition Details

Goal

- Predict the probability that a user would make a purchase.

Data Supplied:

- Training Dataset 1: user_id, item_id, context_feature_id, rating
- Training Dataset 2: item_id, item_feature_id
- Test Dataset: user_id, item_id, context_feature_id

Rules:

- Use various Machine Learning techniques (Pytorch)

Data Preparation Process

Negative Sampling Process:

- Create a separate data frame that randomly groups a user_id and item_id
- One negative sample is created for every user_id in the train data range
- Remainder are random combinations until a 1:1 positive to negative ratio

Train-Validation Split Process:

- 80%-20% split
- Take 80% of positive results, and 80% of negative samples to create train set
    - Ensures 1:1 positive to negative ratio in training dataset
- Setup as a function and called in the training loop, so the data can be reshuffled every few epochs if desired (this is a hyperparameter)

Models Used

Matrix Factorization Hyperparameters (trial & error and grid search)

- Embedding size, learning rates, # of epochs, dynamic learning rate, weight decay

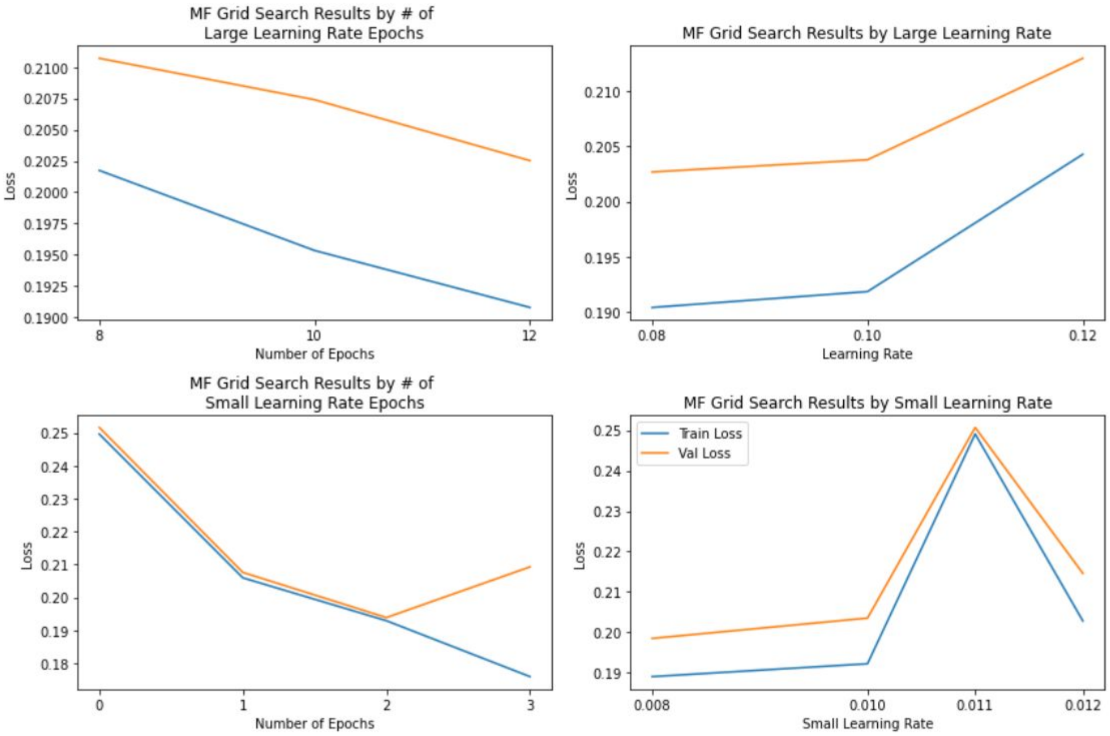
Neural Net Hyperparameters (trial & error and grid search)

- Embedding size, learning rates, weight decay, # of epochs, dynamic learning rate, dropout, hidden layer count

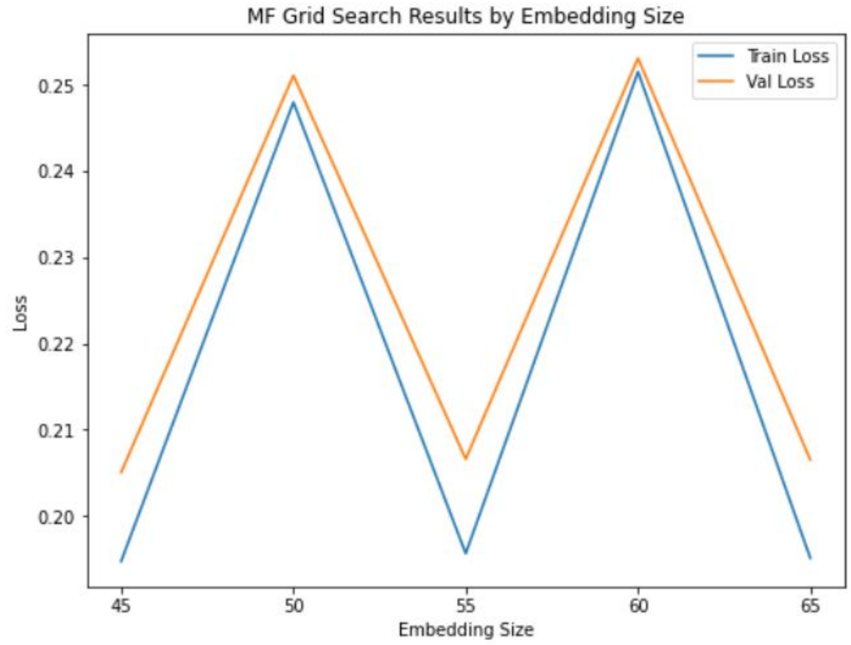
Results Ensemble Hyperparameters (trial & error)

- Mean, median , top 5 results, top 10 results

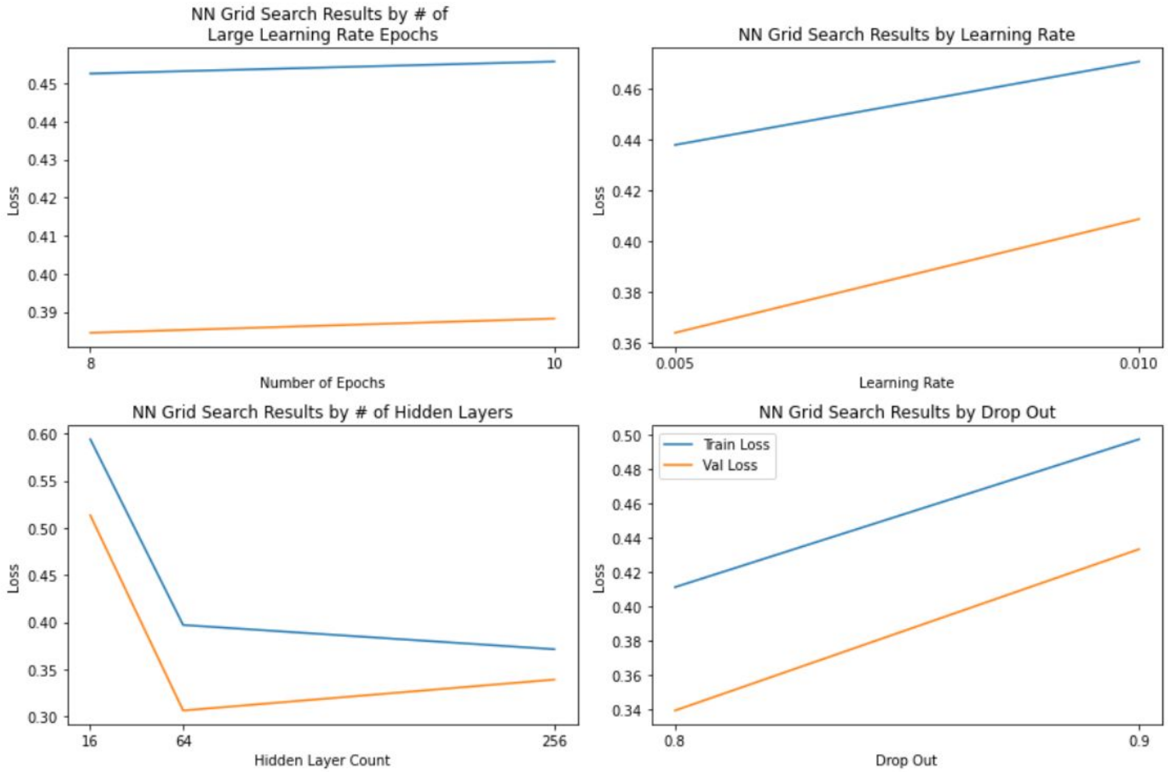
Matrix Factorization Grid Search Results



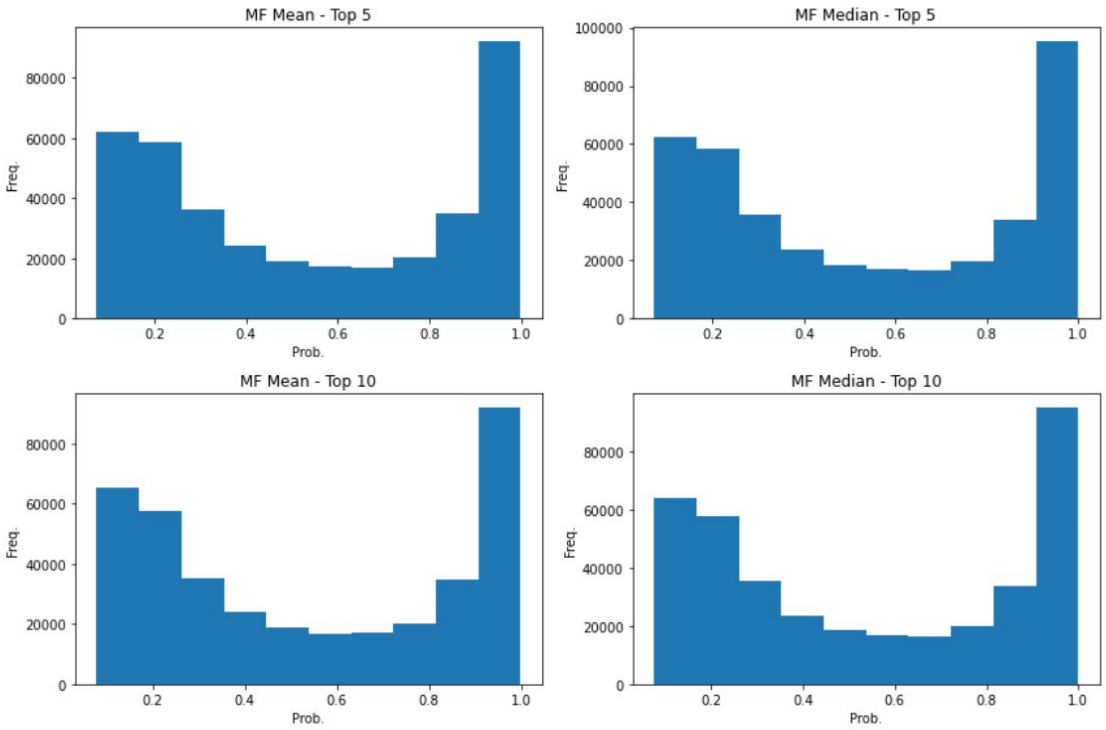
Embedding Size Results



Neural Net Grid Search Results



Ensemble Results



Best Model Selection

Best Single Model — Train loss: 0.185 Valid loss: 0.187:

- Matrix Factorization using user_id and item_id
- Dynamic Learning Rate:
  - 10 Epochs at learning rate = .1
  - 2 Epochs at learning rate = .01
- Embedding Size = 55
- Weight Decay = 1×10^{-6}

Best Overall Model:

- Mean/Average ensemble of top 5 matrix factorization model predictions

Lessons Learned

- Use visualizations and test (other than just validation accuracy) earlier to get a better idea of what makes a good model, and fine tune around those hyperparameters
- A more complex model won’t always perform better than a simpler model.
- Optimizing a neural network and performing hyperparameter tuning to obtain a high-performing model is a tough topic and we should pay attention to balance the train and validation loss to avoid model overfitting.