H&M Recommendation System

CONNOR MIGNONE, TY PAINTER, and LOGAN KING

Fig. 1. H&M product recommendation system.

ACM Reference Format:

Connor Mignone, Ty Painter, and Logan King. 2022. H&M Recommendation System. 1, 1 (May 2022), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 ABSTRACT

Our project was an attempt to compete in the HM Recommendation System challenge. This project came with many challenges: big data scaling, model complexity, and issues with the original plan technologies. We used two recommendation systems. A graph neural network and a more basic customer-based collaborative filter. We simplified the competition rules and made twelve predictions for the top 12 most likely articles of clothing bought by a customer. Our best model produced a mean average precision of 4.24

2 INTRODUCTION

Consumers have preferences for which articles of clothing that they will buy. It is possible to use consumer buying habits to recommend which items that consumers are most likely to purchase. The development of a recommendation system will improve the shopping experience of customers and also improve the bottom line of HM as customers will

Authors' address: Connor Mignone; Ty Painter; Logan King.

Manuscript submitted to ACM

be buying more items since they are recommended items which they are most likely to buy. The approaches that we used to develop our recommendation system included Graph Neural Networks and Collaborative Filtering.

This project was possible due to the HM Personalized Fashion Recommendation challenge on Kaggle. HM provided over 34GB in data made up of datasets on customer information, item information, and transactions. The goal of the challenge was to predict the 12 most relevant items that a customer is likely to purchase. Submissions are evaluated with the mean average precision of predictions, with no penalization for incorrect predictions.

During the project lifecycle, several challenges were faced which mainly centered around data preprocessing and deciding upon a recommendation system. Initially, the TorchRec library was to be used to develop the recommendation system. However, due to the framework's complexities and difficulty of working with embeddings, Graph Neural Networks and Collaborative Filtering were chosen as preferred methods.

Formatting the data to develop a graph was difficult as there were possibilities of making use of user based and item based graphs. Based on the provided data, our group chose to focus on the transaction data, which formed a heterogeneous bipartite graph, as both customers and articles of clothing represented nodes with edges represented by transactions (whether a customer purchased an article). This graph construction worked well for our chosen recommendation system frameworks.

The remainder of the report details the project framework in-depth, discussing the subsetting and preprocessing of data, the model development process, and finally the results of modeling and recommendations.

3 PREPROCESSING

The first step of data processing was subsetting observations and columns of interests. The original dataset had over 30 million rows, but due to time and computing constraints, only the first 20,000 rows were analyzed. The two columns that were selected in creating nodes and edges were the customer ID and the article ID. A standard train test split was made with 70% of the data being used for training and 30% of the data reserved for testing. The 14,000 rows of training data resulted in 5,099 unique customers and 6,134 unique articles of clothing. Next, the subsetted data needed to be structured for modeling. Both the customer and article IDs were label encoded to create unique, numeric IDs when combined together. This was done to eliminate the alphabetic characters contained in the customer IDs. After creating uniform ID labels, a union had to be made between the train and test sets to ensure that the IDs in the test set intersected with the train set. This was done to prevent the cold start problem where the test set included IDs with no previous information, which limits model learning and performance when transferred from training and applied to testing. The next step was to reset indexes in order to form a heterogeneous, bipartite graph. A heterogeneous graph includes nodes of different types, in this case both customers and articles. The combination of two types of nodes requires the indexes to be reset so that both types of nodes are unique. A bipartite graph (Fig.2) is a graph that can be separated into two distinct sets of nodes with every edge connecting nodes in separate, independent sets. A data loader function was deployed to create the graph. The function iterated through all possible combinations of customer and article interactions to evaluate if there was a purchase that would connect the two nodes.This formed an undirected graph, with edges formed between customers and articles if an article was bought by a customer.

4 MODELING METHODOLOGY

There were two modeling techniques used in this project with the first being graph neural networks (GNN) with collaborative filtering. This type of recommender system model starts with node embeddings that are representative of

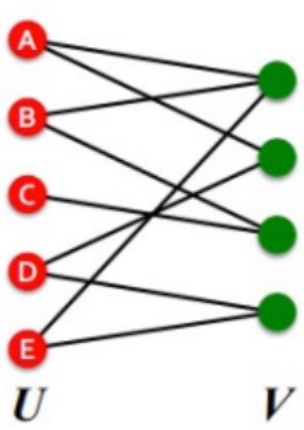

Fig. 2. Bipartite graph with distinct sets U and V.

the latent features created from the training set. The embeddings describe the similarity between neighboring nodes along with characteristics of the overall structure of the graphical network.

4.1 GNN Recommendation System

The GNN layer(s) follow the input embedding layer. There are three phases within each GNN layer (Fig3): message, aggregation, and update. The message portion transfers messages between neighboring nodes through the connecting edges. When multiple messages from various edges reach a node they are aggregated together. The node will then update its features based on the aggregation of all of the messages and propagate the resulting embeddings through the remaining layers to produce output node embeddings. The collaborative filtering aspect of the model makes the assumption that nodes with interacting features of similar interests will translate to purchasing related products.

4.2 FastAI Collaborative Filter

FastAI Collaborative Filter: After issues with the GNN recommendation system we decided to strategize and look for a simpler, 'premade' model that maybe would refocus our group. We found an open source GitHub from Microsoft that provided extremely clear documentation on classic recommendation systems. After reviewing closely, reading the documentation, and inspecting the use cases it was decided to use a FastAI Collaborative Filter.

The model use case was movie ratings. Movie ratings is a classic machine learning recommendation system example, which was similar enough to our own. We used our own negative sampling method to get the negative interactions which were described in the data preprocessing steps. Due to the speed of FastAI Collaborative Filter we used more data, reading in 20,000 rows. This expanded after the negative sampling into 327,712 rows. This model runs on PyTorch under the hood but has high level API calls for easy use. The model works by creating embeddings using a torch dot product on the customers and articles, then adds the customer and article biases.

These embeddings get passed to a learner. The learner is essentially a Python class for training and testing. The learner uses an Adam optimizer with a learning rate of .001. We introduced a small amount of weight decay, .001. We did not yet introduce more. We decided this was not a necessary next step as our training loss appears to be jumping

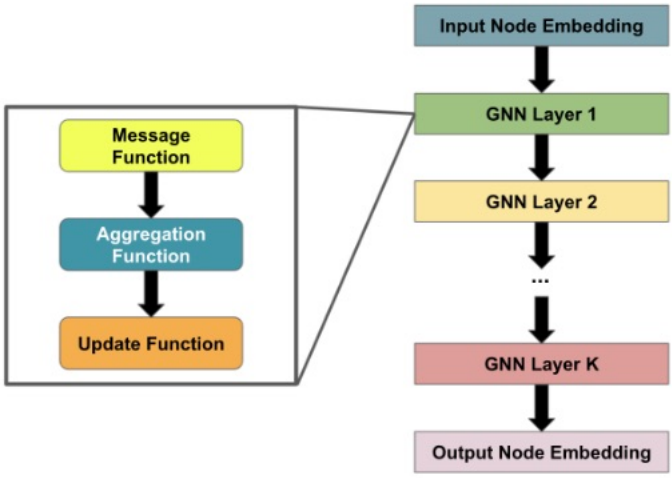

Fig. 3. Graph Neural Network (GNN) model layers.

Table 1. GNN model parameters

Parameter	Value
Weight Decay	0.001
Learning Rate	0.01
Dropout Rate	0.01
Epochs	50

over a global or local minimum. We think decreasing the learning rate would be a better parameter to tune. We specified a 'y_range' which lets the learner assert the output to the closest integer within the range. Our range was [-.5,1.5]. We initially had a range of [0,1.5], but experienced significantly more training loss, this indicates that the sigmoid function did not pass values to 0 and needed a negative integer in order to do so.

'One_cycle' is a high level API call which essentially runs the model for one training iteration until all batches have been used. We then call 'one_cycle' in a for loop for the set number of epochs.

The model took 486.4585 seconds to train 10 epochs, and 0.0179 seconds to make 6148 predictions

5 RESULTS

5.1 GNN Recommendation System

The GNN model was difficult to train because of its extensive computation time. The GNN model consisted of two linear hidden layers with a 10% percent drop out rate and a softmax activation function. The model included 64 latent features to predict the 12 most related articles in comparison to a customer's purchase history that were evaluated with a softplus loss function. During training, the parameters that were tuned were the number of epochs, batch size, decay rate, and learning rate. The parameters that performed the best are displayed in Table 1.

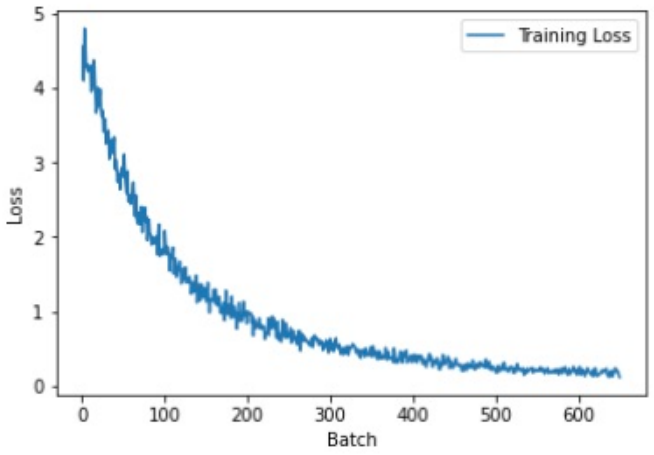

Fig. 4. GNN model loss between batches.

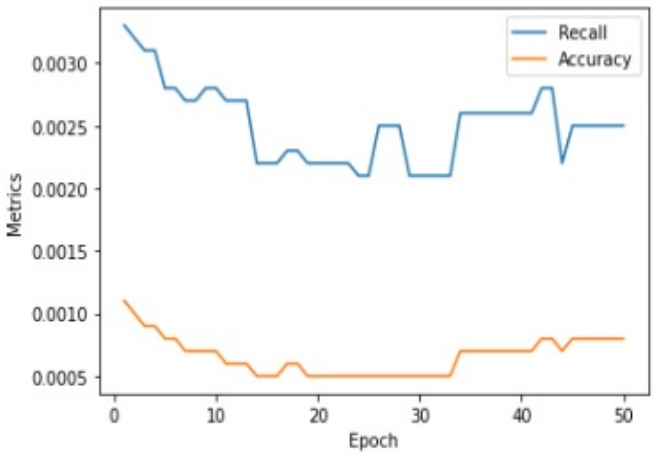

Fig. 5. GNN recall vs. accuracy between epochs.

The softplus loss function also had a beta value of 1.5 which is used to emphasize the loss value. Additionally parameter tuning was performed but no results could be produced due to a bug in Colab when importing torch packages. The results of the model were somewhat underwhelming, but when considering the number of unique articles, the performance was on par with the rest of the class. In Figures 4 and 5, the loss function looks great between batches as it is decreasing which would imply there is learning taking place. However, the accuracy and recall seem to flatline which could suggest that the model is performing randomly and is unstable. The results are better than a random guess but a more consistent model should be pursued which led to the exploration of FastAI Collaborative Filtering.

Table 2. FastAI Collaborative Filter Training Results

Epoch	Training Loss	Time
1	0.001873	00:53
2	0.002515	00:47
3	0.002932	00:47
4	0.002236	00:48
5	0.002231	00:48
6	0.001521	00:48
7	0.001803	00:48
8	0.001766	00:47
9	0.001651	00:48
10	0.002101	00:48

Table 3. FastAI Collaborative Filter Training Results Contin.

Metric	Value
Mean Average Precision	4.24%
Precision @ K	100%
Recall @ K	4.24%
Mean Absolute Error	0.005146
Root Mean Square Error	0.069825

5.2 FastAI Collaborative Filter

The FastAI Collaborative Filter exceeded most expectations. Although it was using higher API calls, source code documentation was some of the simplest and understandable we saw. As mentioned in the modeling section we are concerned about skipping minimums. It seems our filter, with a reasonably small learning rate, can easily jump out of minimums. It appears from the first to second epoch, we successfully jumped out of a local minimum. This also appeared to happen between epochs 2-3 and 6-7. But, we suspect between epoch 9-10 we may have stepped away from a global minimum. Our initial thoughts are to decrease the learning rate to .0005 and keep the weight decay the same.

Our FastAI Collaborative Filter performed much better than expected. Precision @ K simply means out of what relevant items could have been predicted, what percentage of guesses were relevant? 100 percent is unlikely, but possible with the data and sampling techniques we are working with. It is entirely possible all recommended articles of clothing had some sort of relevance to the customer. This is especially true as the model makes 12 predictions and only needs to make 1 relevant prediction (if the customer only had one relevant item in the test set) to get 100 percent.

Mean average precision is what HM is basing the competition around. HM does not release the MAP scores themselves but rather model errors so we do not know how a MAP of 4.24 percent compares to the competition leaders. We are pleased with the MAP and Recall @ K as this model implementation performed better than any GNN we tuned or other implementations of collaborative filters on smaller samples. When comparing our model to the class results for the same competition, we can see it is performing extremely well. At the time of presentations, most group recall and MAP were between 2 percent and 3 percent.

6 CONCLUSION / NEXT STEPS

One of our original project goals was to use TorchRec, a new high level API for creating embeddings. This proved to be much more difficult than we expected. Due to the lack of tutorials and documentation we simply were not able to implement it. We would like to return to TorchRec now that we have an even firmer understanding of embeddings and PyTorch. Doing so might simplify the embedding process for our GNN recommendation system.

One opportunity to improve the GNN learning is to modify the loss function. We used a softplus loss function. This simply takes the negative embedding scores minus the positive embedding scores. Meaning, if the positive embedding scores is larger than the negative, the loss will go below zero. The softplus acts similar to a ReLU function where anything below zero just becomes zero. That was not our issue, we had marginal loss that allowed for training but not for increased performance. The softplus loss function has a beta parameter which acts as a scalar to increase loss. The default value for beta is one, we set ours to 1.5. We believe this slightly increases performance, but next steps include increasing beta even higher to increase loss, allowing the backpropagation to react even more to the error.

Since class presentation we have attempted to scale the FastAI Collaborative Filter. As mentioned in the modeling section we doubled our initial sample. We are still experiencing issues with negative sampling data expansion. The sample grows exponentially the more rows we initially read-in. Although we see an increase in positive results with scaling, as more data helps the model train even more effectively, we suspect this is still not the best course of action. There are less customers than articles of clothing. The articles also include features describing the articles, such as color and type of garment. A logical next course of action would be doing an item-based collaborative filter. In addition to dealing with less data, the features will allow for improved relevancy scores. In our own shopping experiences we see personalized recommendations but it seems much more along the lines of, 'others who have purchased this, have also purchased this'. This is most likely some kind of item-based collaborative filter that effectively handles big data. Content-Based Personalization with LightGBM would be the next evolution of our model.

To conclude, we are not overly happy with the methodology or process of our project. We spent too much time debugging TorchRec with no results to show for it. Then in a crunch for time settling for well-known existing methods that are not truly creative or innovative. We are pleased that we successfully were able to adapt existing methods to our data to get reasonable precision and recall.

7 REFERENCES AND BIBLIOGRAPHY

We used many medium-Towards Data Science articles to improve our understanding, get ideas for the project, and reference for code. Instead of listing what TDS referenced themselves we just listed the links. We were not sure what citation formation website articles and links need to be in.

7.1

TEXRecommenders, author = Microsoft, title = Recommenders, year = 2022, publisher = GitHub, journal = GitHub repository, howpublished =https://github.com/microsoft/recommenders, commit = d4181cf1d1df6e71f7e6b202b0875bb3bd54150c

FastAI. 2022. FastAI: Making Neural Nets Uncool Again. Retrieved from https://www.fast.ai/.

FastAI. 2022. Learner, Metrics, and Basic. Callbackshttps://docs.fast.ai/learner.htmlLearner

FastAI. 2022. Tabular Learner. Retrieved from https://docs.fast.ai/tabular.learner.html.

Brideau, Ryan. Precision@k: The Overlooked Metric for Fraud and Lead Scoring Models. Retrived from https://towardsdatascience.com/prec
k-the-overlooked-metric-for-fraud-and-lead-scoring-models-fabad2893c01

X. He et al, Neural Collaborative Filtering, WWW 2017.

Y. Hu et al, Collaborative filtering for implicit feedback datasets, ICDM 2008.

Simple Algorithm for Recommendation (SAR). See notebook sar_deep_dive.ipynb.

Y. Koren and J. Sill, OrdRec: an ordinal model for predicting personalized item rating distributions, RecSys 2011.

Kung-Hsiang, Huang. Hands-on Graph Neural Networks with PyTorch PyTorch Geometric. Retrived from https://towardsdatascience.com/
on-graph-neural-networks-with-pytorch-pytorch-geometric-359487e221a8.

Li, Derrick. Recommender Systems with GNNs in PyG. Retrived from https://medium.com/stanford-cs224w/recommender-
systems-with-gnns-in-pyg-d8301178e377.

Rajamanickam, Santhosh. Graph Neural Network (GNN) Architectures for Recommendation Systems. https://towardsdatascience.com/grap
neural-network-gnn-architectures-for-recommendation-systems-7b9dd0de0856