Adversity in Online Recommendation Systems

Matthew Kluska
Boston University
kluska@bu.edu

Christopher Trinh
Boston University
ctrinh@bu.edu

Ning Wan
Boston University
ningwan@bu.edu

Shiping Zhao
Boston University
zshiping@bu.edu

Abstract

The paper proposes a theoretical analysis of recommendation systems in an online setting, where items are sequentially recommended to users over time. Specifically, the paper focuses on the Netflix streaming platform where users are constantly recommended TV shows and movies based on a variety of factors. In this sense, the goal is to effectively create a network where users are recommended content that the system believes they would not like – thus, instead of minimizing the loss function as recommendation systems typically try to do, the inverse of the loss function is minimized instead. After running our results through a degree of post-processing and normalization, our analysis reveals that our model, while not accurate enough for general use, proves to be a significant study into how recommendation systems can be improved and the underlying structure.

1 Introduction

Recommendation systems over the last few decades have garnered more prominence in our daily lives, especially considering the substantial rise of web streaming and e-commerce services like Netflix, Youtube, Amazon, etc. Generally speaking, recommendation systems are algorithms which aim to suggest relevant items and content to users, whether they are products or movies for users to purchase and consume, and these algorithms have been improved upon throughout the past few years to varying degrees of success. Typically, these algorithms are all based on the idea of *collaborative filtering*, or the idea that (1) similar items should yield similar user responses, and (2) similar users have similar probabilities of liking or disliking a given item. Using this as an advantage, efficient recommendation systems can learn this structure which connects users to items together.

While most recommendation systems try to recommend content that users will like, the goal this time around is to try to find the inverse where users will hopefully be recommended content that they will dislike. To this aim, we study the adverse of online recommendation systems.

2 Dataset

This paper will primarily use the Netflix Prize dataset from 2019, which originated from Netflix's competition to improve their recommendation algorithm, where entrants competed to create the best algorithm to predict user ratings for films. The dataset itself is about two gigabytes in size, with the first part of the data being a testing dataset consisting of movie ids, customer ids, and rating dates. The ratings are withheld in the original dataset. The goal of this competition was to predict all the ratings the customers gave the movies based on the information in the training dataset. A probe dataset file was also provided as training data for the entrants' models, which includes the rating

unlike the qualifying dataset. This is where the bulk of the dataset came from, as the training data originally came in 17,000 files which was then split into four text files by the dataset maintainer.

3 Models and Preliminaries

3.1 Dataset selection

We consider the dataset of over 12,000 movies. It is worth noting that the top fifty movies with the most ratings (not to be confused with the top fifty movies with the highest best ratings) has about 7,000,000 ratings in total already (which is just about ten percent of all the ratings in the dataset) so we choose to consider only the top 50 movies in our data analysis.

Likewise, we can consider the users in the dataset in a similar manner. There are around 480,000 users, and the top 100 users with the most ratings (again, not to be confused with the top fifty users with the highest best ratings) has about 2,000,000 in total. However, we choose *not* to consider only the top 100 users in this scenario, as we hope to avoid bias through the possibility of these top 100 users having very similar interests or through these users watching the movies due to their popularity, as we seek to gather a wide range of interests and reduce bias as much as possible.

3.2 Dataset cleaning

We made sure to go through the entire dataset to clean invalid entries or NaN entries, but thankfully, the maintainer of the dataset made sure to do that beforehand. We also made sure to delete all duplicated data, which also did not occur. It should be noted that we opted to ignore the timestamp column of the movie ratings data, as it did not pertain to our research and was outside the scope of this paper.

3.3 Parquet conversion

A parquet is a open source storage file format available to any project working within the Hadoop ecosystem. Parquets are perfect for our needs, as they are designed for efficient and performant flat columnar storage format of data, especially when compared to row based files like CSV format, as parquets can only read the necessary columns thus greatly minimizing the IO needed to perform data analysis. Creating a parquet will allow us to significantly cut down the file size (upwards of a 75 percent reduction!) and also allow streaming of data, which allowed us to access the data without storing it locally. We then again select the top 50 movies ranked by the number of ratings, which we believed was important to do as we believed the noise of those movies are smaller.

3.4 Matrix factorization

For the majority of our models, we referenced Fast Forward Lab's Pytorch tutorial. Three layers are built: users, factors, and items. These layers allow us to generate the relationship between users and items through the matrix building process and multiplication process. In other words, this serves as a matrix-building system that works to connect the relationship between both the "users and factors" and the "factors and items." These factors (embedding vectors) are essential in order for us to analyze the relationship built.

3.5 DenseNet

DenseNet, otherwise known as the Dense Convolutional Network, is a network architecture where each layer is connected directly to every other layer in what is called a feed-forward fashion (within each *dense block*, hence the name). The input used is "users, items" combined; this combination method is similar to matrix factorization with factors embedded. However, we forego to use of direct multiplication and instead opt to use a neural network with linear layers created.

3.6 Loss function

The loss function we chose to use for our analysis was mean of squared error (MSE). We picked MSE as it works best on models that have no huge outlier errors in their predictions. Furthermore, MSE

allows us to compare the error of different models. We planned to test different hyperparameters (both DenseNet and matrix factorization using mean of squared error (MSE) and stochastic gradient descent (SGD), aiming to test our model performance with a factor number in the range (20, 50).

The orignal idea was to use the negative MSE so that we can make a adverse prediction. However we found that was not viable as the combination of MSE and Stochastic gradient descent will only push the model's prediction further from the truth label. So we decided to use regular MSE and output predictions just as normal, but we will select the user/item pair as the outcome of the whole algorithm.

3.7 Optimizer

We decide to optimize using stochastic gradient descent (SGD).

4 Algorithms

4.1 Model training

To train our model, we first take a single rating record, and then feed the user id and movie id into the model to get a prediction from the model. We then take this prediction and calculate the MSE using the generated prediction and the label. Once done, we update the model using SGD.

4.2 Model testing

For us to effectively test our model, we first take a rating record, and then feed the user id and movie id into the model to get a prediction from the model. We then take this prediction and use sklearn's *MinMaxScalar* (which transform features by scaling each feature to a given range) to transform the resulting prediction into the range (0, 5). We then compare the prediction and the label. If the prediction is close enough to the label that it is within plus or minus 1.0 of the truth label, we consider the prediction to be accurate.

4.3 Model usage

For proper usage of the model discussed in this paper, we take the user of interest and put the corresponding user id and all the movie ids into the model. We then transform and rank the result. Once done, we take the lowest 100 predictions and select those as our output, which in other words, will be the top 100 most disliked movies from this particular user.

5 Results

Our models have better accuracy with matrix factorization. Specifically, our model has highest accuracy when *nfactors* = 30, which means our model achieves a 0.49 accuracy rate.

However, we learned that the training set is not large enough for the DenseNet to generate better results; using DenseNet results in only around 0.25 accuracy rate. A larger training set and more layers added on the neural network implementation could be done to attempt to improve the algorithm.

6 Further Approaches

If we had more time, we could seek to improve our model even more in a couple of different ways.

6.1 Different hyperparameters

If we had more time, we could have considered *n_factors* as one of our hyperparameters.

6.2 Different error function

MSE worked well enough for our general purposes, but during our analysis, we recognized that MSE might have trouble working with polarized movie results, like split ratings of 80 percent 5-star and 20 percent 1-star ratings.

6.3 Different method for calculating loss

Given more time, we would have liked to explore the idea of maybe calculating the loss using transformed predictions, and then using that transformed loss for SGD which is *MinMaxScaler* (MSE).

6.4 Alternate matrix factorization

We also thought of perhaps exploring the usage of the biased matrix factorization model instead. This would allow us to experiment a little more, as in a factorization model, the model typically tries to incorporate both bias and interaction terms. Bias terms describe the effect of one dimension on the output, which work well in our Netflix dataset, as the bias of a movie would describe how well this movie is rated compared to the average rating, spread across all movies.

6.5 Hybrid approach

We also considered using the label of users and/or movies to support the model's predictions. We can calculate the movie labels that a user like, and then when making predictions, combine the predictions outputted by our model with the prediction made by the movie labels.

7 Further exploration

We also hoped to create a ranking system apart from the recommendation system. Since each user may have reviews for multiple movies, we can generate many battles based on those ratings (pair-up each user's movie rating). Those battles can be transformed into a matrix (row: battles, columns: movies). Then, we can create y datasets based on the ratings in each battle with definition and regard the "movies" as features for X data. Those battles can be filled in our matrix. We can use linear regression, pageRank, etc. on this matrix to help achieve the ranking system.

8 Conclusion

This paper proposes and analyzes several models for online recommendation systems. These models are a broad application, and is still a work in progress. We provide different algorithms and ideas to approach these limits for all the models, but many interesting and challenging questions remain open. There are many avenues of possibility for us to extend our analysis as the potential to revolutionize is significant.

References

[1] Fast Forward Labs. "PyTorch for Recommenders 101." *Cloudera Fast Forward Blog*, 10 Apr. 2018, blog.fastforwardlabs.com/2018/04/10/pytorch-for-recommenders-101.html.