Recommendation System

Badrinath Reddy Gujjula bg2310@nyu.edu
Mateo Castro mc7212@nyu.edu
Yuqi Liu yl8406@nyu.edu

Abstract

Recommendation Systems exist in many forms, from Content Based to Latent Factor models. In this paper we built a hybrid Recommendation System using Content Based, Collaborative Filtering, and Latent Factor components for different use cases branching into old users and new users. Our recommendations are based on available 2018 Amazon Magazine Subscription Ratings data which consists of over 80,000 ratings.

1 Github

For full code and report please visit.

https://github.com/badri449/MMDS

2 Problem Statement

Given a set of m users, n items, and set of ratings from user for some items, try to recommend the top items for a given user (either new or old) under different scenarios. These scenarios include when the user is browsing items or when they have just purchased an item.

3 Data Set

The dataset used in this project is the Magazine Subscriptions set in Amazon review data, which is downloaded from [1]. The Magazine Subscriptions set contains a ratings set and a meta data set. The ratings dataset has 3 columns and 89689 rows. The rating set has three columns: Product ID, Customer ID, and Rating, and each row corresponds to a rating record. we have 72,098 unique users and 2,428 unique products. The meta data has 19 columns and 2320 unique rows. Each row contains the information of one item, and each column corresponds to one feature. The features we mainly used in meta data set are description, category, and brand.

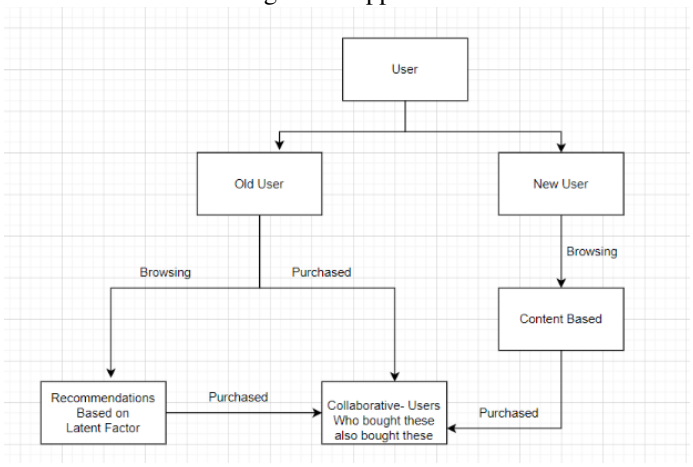
4 Approach

Our approach is to build a recommendation system using Content Based, Collaborative Filtering and Latent Factor to give recommendations under 3 different scenarios.

- New user browsing - Content Based Recommendation
- Old User Browsing - Latent Factor
- Old User purchased an item - Collaborative Filtering

The flow diagram can be seen here

Figure 1: Approach



5 Content Based

Content Based model recommends items to users by finding items that have similar features as the item the user in searching for. This model does not have cold start problem, which means it can make recommendations to users with no purchase history, and can recommend items without ratings.

5.1 Implementation

To find similar items with Content Based model, we first computed the TF-IDF (Term Frequency-Inverse Document Frequency) scores of those items based on their brand, description, and category. Then we compute the cosine similarity score from the TF-IDF score, and return the N items with the highest similarity score.

6 Collaborative Filtering

Collaborative Filtering models take into account the ratings of other users that are similar to the given user and recommend items based on their ratings. In the case of item-item it takes flips the user and items roles, with the item now finding similar items to it based on user ratings. Thus the recommendations given to a single item/user are a collaboration of similar user/items.

6.1 Implementation

For our recommendation system we used a KNN Baseline model, this is a hybrid of Global Baseline and Collaborative Filtering. We wanted a KNN model in particular so that we could get the nearest neighbors when recommending similar items to a user. Using the Surprise Library [2], we did a Grid Search 5 Fold Cross Validation in order to find the optimal parameters for the model. In particular we had 3 sets of parameters to test: K, Baseline Options, Sim Options. K being the number of clusters to be created for KNN, which was tested for K = 10, 50, 100, 150, 200. The Baseline Options are the parameters for the Global Baseline component of the model. The method used for the Baseline was always ALS (Alternating Least Squares), the hyper-parameter that was tested was:

- Number of Epochs

The sim options are used for the Collaborative Filtering component, since we were creating an Item-Item model, this hyper parameter stayed constant. The other hyper parameters tested were:

- Name of Similarity Method
- Minimum Supported Common Items for Consideration

Using the Grid Search we found that the optimal parameters for the model were:

- Baseline Method: ALS
- Number of Epochs: 10
- K: 50
- Similarity Method: Pearson Baseline
- Minimum Supported Items: 3
- User Based: False

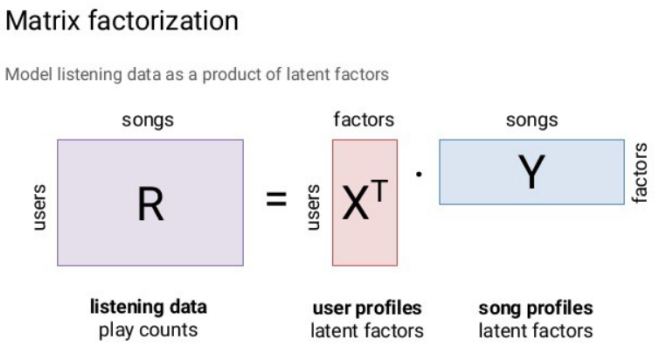
The RMSE(Root Mean Square Error) for the optimal parameters came out to be: 1.3118879603008418
(All combinations of parameters and their results are shown in the Python Notebook located in the GitHub)

Finally, the recommendation algorithm was created. With parameters of item ID, and number of recommendations N. It finds the N nearest neighbors of the item, and returns them to the user after they have purchased the item.

7 Latent Factor

Latent Factor models, also known as Matrix Factorization models, are feature extractor models used to capture meaningful latent connections between users and items inferred from rating patterns. Latent-factor based approaches relate users and items in a low-dimensional latent feature space.
A user might have purchased hundreds of food products, but those products might belong to only 20 distinct categories. Therefore, 20 categories are enough to describe this users tastes. similarly an item could be possibly describe with 20 features. Those interactions between users and items explain the patterns of the ratings intuitively i.e if some certain characteristics of an item match a user's interests on those item characteristics, the user tends to rate this item highly.
We'll initialize two matrices from a Gaussian Distribution(alternatively, randomly initialize them). The first one will be a m x k matrix P while the second will be a k x n matrix Q. When these two matrices multiply with each other, they result in an m x n matrix, which is the size of our Rating matrix in which we are trying

Figure 2: latent Factor



to predict. The dimension k is the amount of latent factors and one of our hyper-parameters,
With our Matrices P, Q, we'll optimize their values by using **Stochastic Gradient Descent**. Therefore, we will have two more hyper-parameters to optimize, learning rate and epochs along with Number of latent Factors.

7.1 Implementation

We inherited the surprise library and modified the fit and estimate to do Latent Factoring. In Fit method uses Stochastic Gradient Descent to be able to estimate the best probabilistic matrix factors. The different combinations hyper parameters we tried are as follows:

- Learning Rate - [0.005,0.05,0.1,0.5]
- Epochs - [10,15,20,25,30,40,45,50,60]
- Number of Latent Factors - [2,3,4,5,6,7,8,9,10,15,20,25,30,35,40,50]

In Google colab it took more than **4 hours** to get the Training and Test RMSE for all these combinations. The results have been stored in GitHub Repo. Some combinations which have high number of latent factors has very less Training error but High Test Error(**over fitting**) so we choosed the model which have less Test Error. After sorting the results based on Test Error it looks like 3
The best possible hyper parameters are as follows:

- Learning Rate - 0.050
- Epochs - 30
- Number of Latent Factors - 2

Tested these optimal model with **cross validation** which had similar results. Finally developed the model with entire ratings data set with above parameters.

8 Why Latent Factor

Instead of Latent Factor we could have possibly used a neighbourhood model such as Collaborative Filtering based on customers. We tried to develop this model but since there are around 72000 unique

Figure 3: Training Results

	LearningRate	Epochs	LatentFactors	TrainRMSE	TestRMSE
208	0.050	30.0	2.0	1.392116	1.412927
400	0.100	50.0	2.0	1.399411	1.412991
160	0.050	15.0	2.0	1.398663	1.413018
575	0.500	60.0	50.0	1.416786	1.413019
479	0.500	20.0	50.0	1.416786	1.413019
...	...	...	...	...	...
8	0.005	10.0	10.0	2.496395	2.036969
11	0.005	10.0	25.0	2.319462	2.043628
1	0.005	10.0	3.0	2.716431	2.044642
3	0.005	10.0	5.0	2.649409	2.054577
0	0.005	10.0	2.0	2.832111	2.055095

576 rows × 5 columns

users it quickly crashed that's wehere we can see the advantages of latent factors which greatly helps in dimensionality reduction and therefore computational costs.The latent (hidden) features captures more of the relationships between users and items than neighborhood models.Lot of flexibility in training to optimize the two matrices i.e we can use Adam,SGD,..etc and try different hyperparameters.

9 Final Results

Developed a Function called get_recommendation which takes user_name, N(No:of recommendations) and item_id(If relevant) and gives recommendation based on one of the following situation

- Old user purchased an item - Collaborative Recommendation
- Old user just browsing - Latent Factor Recommendation
- New user browsing - Content Based Recommendation.

The code snippet is 9

10 Business Applications

The way the recommendation system is built it is suitable in all those situations where we have some users,items, ratings and users can browse or buy items. The most popular one is the E-commerce sites, Video Streaming platforms such as Amazon Prime Video, Netflix[Where instead of buying users view videos]. Music Platforms such as Spotify, Amazon Music[instead of buying users listen to music]

```python
# return a list of N top recommendations
def get_recomendation(user_name,N=5,item_id=None):
  if user_name in unique_users:
    if item_id is not None:
      # Old user purchased an item
      return collaborative_recommendation(item_id,N)
    else:
      # Old user just browsing
      return latent_recommendation(user_name,N)
    # New user browsing
  return content_based_recomendation(item_id,N)
```

Figure 4: Function

```python
# New User browsing
get_recomendation('xx',5,'B00005N7O6')
```

```
['B00007AWME', 'B00YQH9874', 'B015PRLQSW', 'B000ILZ6YQ', 'B000H4W7VO']
```

```python
# Old user Browsing
get_recomendation("AQJDVLLWELFJ1",5)
```

```
['B00005N7P0', 'B00005N7OJ', 'B00005N7Q1', 'B00005N7OD', 'B00005N7PS']
```

```python
# Old user Purchased an item
get_recomendation("AQJDVLLWELFJ1",3,"B00005N7O6")
```

```
['B00007AWME', 'B000UEI4JU', 'B00005N7P0']
```

Figure 5: Function Calling

11 Conclusion

The entire recommendation works but one thing to note is that in the meta data we do not have a proper name for the items. As next steps we can try to improve RMSE for Latent Factor and Collaborative models. Compare our results to any benchmark tests if available. Implement more features by giving recommendations based on search history, recently visited items and items added to cart. We used Magazines data set to develop the system we tried to extend it to other data sets such as Sports and Electronics but these data-sets are huge to compute and it quickly crashed after eating up the entire ram. One way to solve this problem is to give priority to recent ratings and remove old ratings based on time stamps.

12 Live system

The problems in Live system is where to store the new ratings, when to update the model with these new ratings and what to do if the data grows exponentially. When ever new ratings occurs we can store it in some data structures such as lists and retrain the model by adding new data to existing data after a day or week or month of collecting new ratings or if the new ratings go beyond a pre-defined limit.
If the data set becomes huge we remove old ratings or give some kind of preference to some ratings like verified users.

References

[1] DataSets

[2] Surprise Library

[3] Latent Factors Intro

[4] Latent Factors Reading

[5] Content Based Intro