Personalization : Final Project

MOVIE LENS RECOMMENDER SYSTEM

Karan Sindwani (ks3631)

Raj Biswas (rb3370)

Table of Contents

1.Introduction

1.1 Background

Most of today's top online services use recommendation and personalization in some form to attract users and improve their market share. In this project, we are focusing on a platform that provides movie recommendations to users based on their liking and preferences. Online streaming services such as Netflix, Amazon Prime Video, YouTube, etc. use these techniques and their advanced variants extensively to power their websites. For such online streaming services, providing users with extremely good matches is essential. The better the recommendations, the more likely is it for the user to consume more content which is directly linked to the revenue of such services. Accurate recommendations is also important for these services to retain users and prevent them from switching to their competitors. Thus, having a good recommender system is key to their survival.

We are proceeding with the same data set as Homework-2 : Movie lens . Apart from having over a million user-item ratings, the movie lens data also has additional features such as timestamp and movie genre, which can be used as features to augment the recommender system. The more data collected, the

wider range of algorithm can be applied and the output would be more personalized

1.2 Objective

The projects aims to create a recommender system which improves traditional recommender systems . A traditional Collaborative based recommender system works solely on User -Item matrix to predict the ratings of items and thereby recommending items to users. But these traditional systems face challenges such as cold start, rich getting richer etc.

The new algorithm is designed with 2 approaches :

- Factorization Machines (FMs) :
    - They are a state of the art solution to problems including multiple entity types. They are similar to other collective approaches in that they allow for multiple relationships between multiple entities
    - FM's using just user item data
    - FM's using user-item and side information

- Content Based Recommender
    - build models to understand the item content
    - recommend based on items that are perceived as similar to things one has consumed

2. Data

2.1. Overview

The dataset has more than 100,000 ratings , each rating ranging from 1 to 5 (1 being the lowest and 5 being the highest). There are more than 1000 unique users and 17000 unique movies in dataset. For each movie we have a list of associated genres which is used as a side information

2.2 Data Description

Ratings Data : It contains UserId (ID corresponding to a user) , MovieId (Id corresponding to a movie) , rating and timestamp

	userId	movieId	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

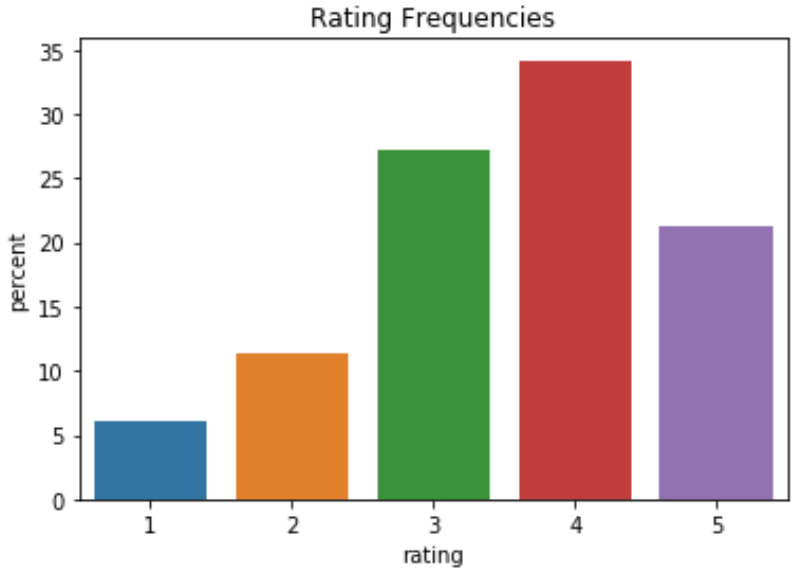
User_meta_data : It contains UserId, age of the user, gender of the user, occupation of the user, zip_code

	user_id	age	gender	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

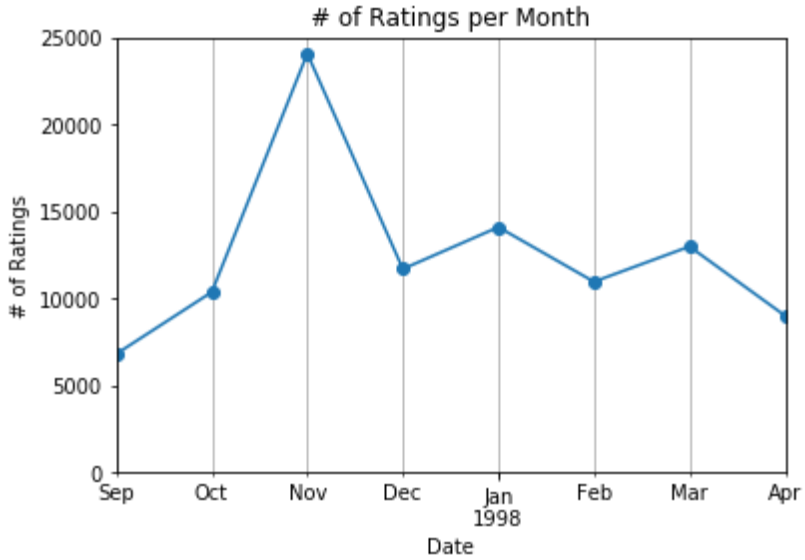
Item_meta_data : It contains MovieId, release date and all genres as dummy variables

	movie_id	release_date	unknown	action	adventure	animation	children	comedy	crime	documentary	drama
0	1	01-Jan-1995	0	0	0	1	1	1	0	0	0
1	2	01-Jan-1995	0	1	1	0	0	0	0	0	0
2	3	01-Jan-1995	0	0	0	0	0	0	0	0	0
3	4	01-Jan-1995	0	1	0	0	0	1	0	0	1
4	5	01-Jan-1995	0	0	0	0	0	0	1	0	1

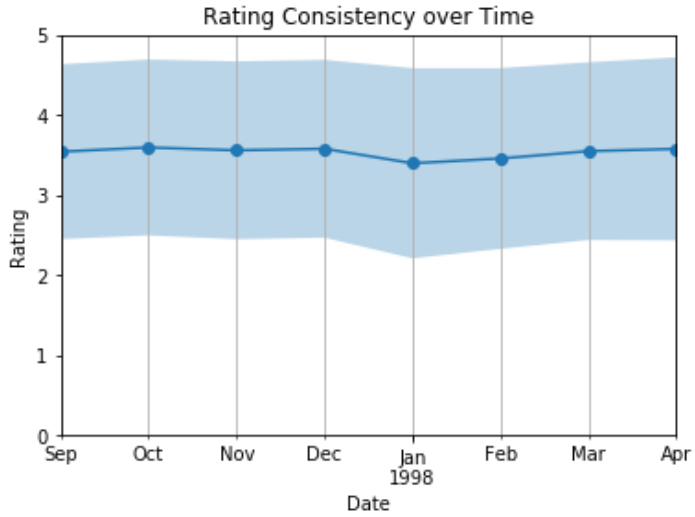
2.3 Exploration of Data



From the above graph we can see that most of the ratings in our data range in the middle , i.e. between 3 and 4. There are fewer ratings with extreme values



While exploring side information, we explored month of the year as feature and we infer that most of data is posted in November and in all the months the remaining data is uniformly distributed.



Also, we wanted explore if the user rating behaviour is based on month of the year, but we can infer from the graph that it is almost constant across months.

3. Model

3.1 Factorization Machines

3.1.1 Description

Factorizations Machines were traditionally introduced as supervised Machine Learning Technique. They get their name from their ability to reduce problem dimensionality**.**

They algorithms aims to predict ratings of movies through FM. The reason for choosing FM is because:

- The user item matrix is extremely sparse
- The complexity of FM is linear in terms of input size of factorization.

They are much more computationally efficient on large sparse datasets. This property is why FM are widely used for recommendation.

For implementation of this algorithm Lightfm python package was used along with scipy.
This can be separated into 3 sub parts

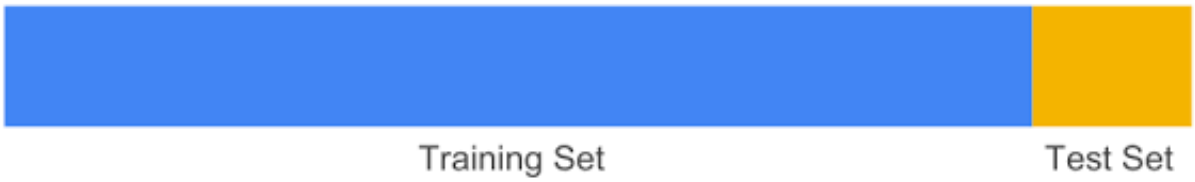
- Pre-processing
- Model fitting
- Post processing

The model is implemented on two levels :

- Using only user-item matrix
- Using user-item matrix and meta data

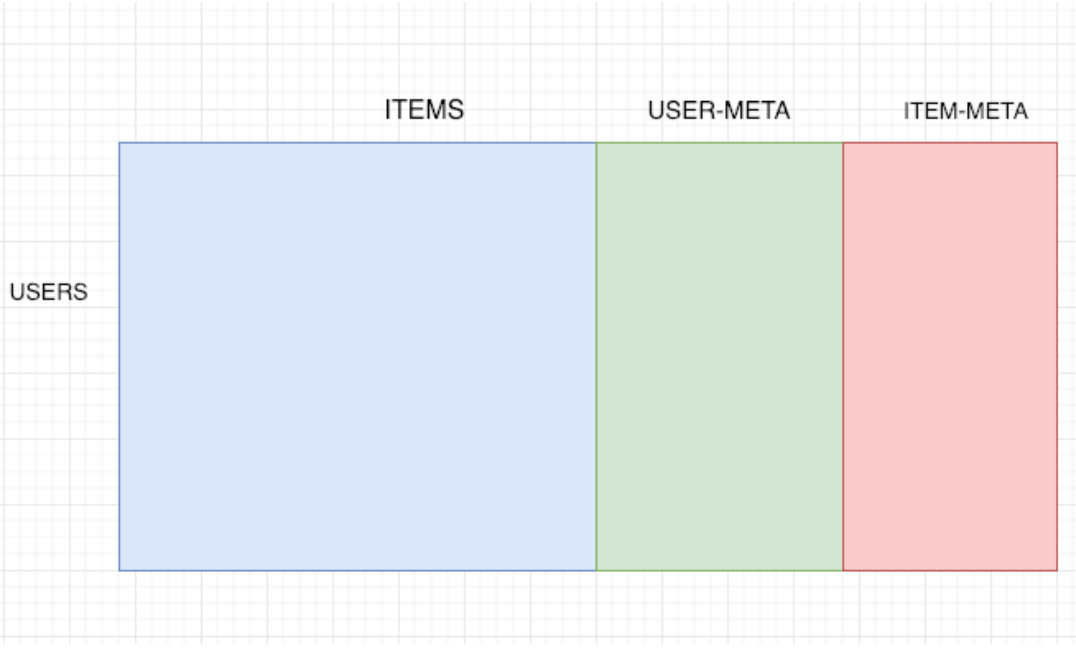
3.1.2 Pre-processing

Firstly the data in its raw form is split into 2 parts of 80 and 20 percent respectively. This split is currently done randomly but a better way of dealing it could through stratified sampling keeping distribution of user rating same across training and test sets.



The ratings data is currently in tabular form. But we needed a sparse matrix representation of user-item data for input to lightfm. So both train and test data frames were converted to sparse user-item matrices

To augment user-item matrix with side information. The available user and item meta data are converted to dummy variables as one hot vectors and appended to the user -item matrix.



3.1.2 Model fitting

Before fitting the model, we defined a baseline model which defines the most simplest algorithm to recommend movies to users. The algorithm recommends the most popular items to users. Through a baseline we are defining an upper bound for erros(RMSE and MAE), i.e. the errors should not increase this threshold.

```
def baseline(train, test):
    test_matrix = test.rating.values
    prediction  = np.zeros(shape=(test_matrix.shape))
    prediction.fill(5)
    rmse = (sum((prediction - test_matrix)**2) / prediction.shape[0]) **0.5
    mae = sum(prediction - test_matrix) / prediction.shape[0]

    print("the baseline rmse is ", rmse)
    print("\nthe baseline mae is ", mae)
```

```
baseline(train, test)
```

```
the baseline rmse is  1.843936007566423

the baseline mae is  1.4617
```

LightFm makes it possible to incorporate both item and user metadata into the traditional matrix factorization algorithms. It represents each user and item as the sum of the latent representations of their features, thus allowing recommendations to generalise to new items (via item features) and to new users (via user features).

We'll use two metrics of accuracy: precision@k and ROC AUC. Both are ranking metrics: to compute them, we'll be constructing recommendation lists for all of our users, and checking the ranking of known positive movies. For precision at k we'll be looking at whether they are within the first k results on the list; for AUC, we'll be calculating the probability that any known positive is higher on the list than a random negative example.

```
]: def predict_fm_user_item(train, test, learn_rate, latent_dimension):
    model = LightFM(learning_rate = learn_rate, no_components = latent_dimension)
    model.fit(train, epochs=50)

    train_precision = precision_at_k(model, train, k=10).mean()
    test_precision = precision_at_k(model, test, k=10).mean()

    train_recall = recall_at_k(model, train, k=10).mean()
    test_recall = recall_at_k(model, test, k=10).mean()

    train_auc = auc_score(model, train).mean()
    test_auc = auc_score(model, test).mean()

    print('Precision: train %.2f, test %.2f.' % (train_precision, test_precision))
    print('Recall: train %.2f, test %.2f.' % (train_recall, test_recall))
    print('AUC: train %.2f, test %.2f.' % (train_auc, test_auc))
```

3.1.3 Evaluation

There are two algorithms which were developed :

- Using only user -item matrix
- Using additional information and user -item matrix

DATA USED	Evaluation Metric	Data	Value
USER-ITEM	Precision	Train	0.64
USER-ITEM	Recall	Train	0.14
USER-ITEM	AUC	Train	0.95
USER-ITEM	Precision	Test	0.31
USER-ITEM	Recall	Test	0.13
USER-ITEM	AUC	Test	0.94

While using just User -item data we get a good AUC value in both train and test but the recall is both low in train and test. The reason might be that user is consuming movies apart from the recommendations. This problem can be addressed by increasing the number of recommendations shown to the user and also by overall making a better recommender system. We also see a relatively low precision value. This means that the items that are recommended to the user are not consumed by him/ her.

By using the side information along with user item matrix, we are trying to push new items to the recommendation by comparing their item vector.

DATA USED	Evaluation Metric	Data	Value
USER-ITEM-META	Precision	Train	0.03
USER-ITEM-META	Recall	Train	0.02
USER-ITEM-META	AUC	Train	0.54
USER-ITEM-META	Precision	Test	0.02
USER-ITEM-META	Recall	Test	0.01
USER-ITEM-META	AUC	Test	0.54

After including meta data as feature vectors in the user -item matrix, un-conventionally we see a drop in all evaluation metrics. This could possibly be due to increase in number of features and not all features contributing positively to the model.
The way to deal with such issues is explained in the future prospects section.

3.1.3 Hyper parameter tuning

For Factorization machine lightfm allows us to tune many parameters but we choose two important parameters:

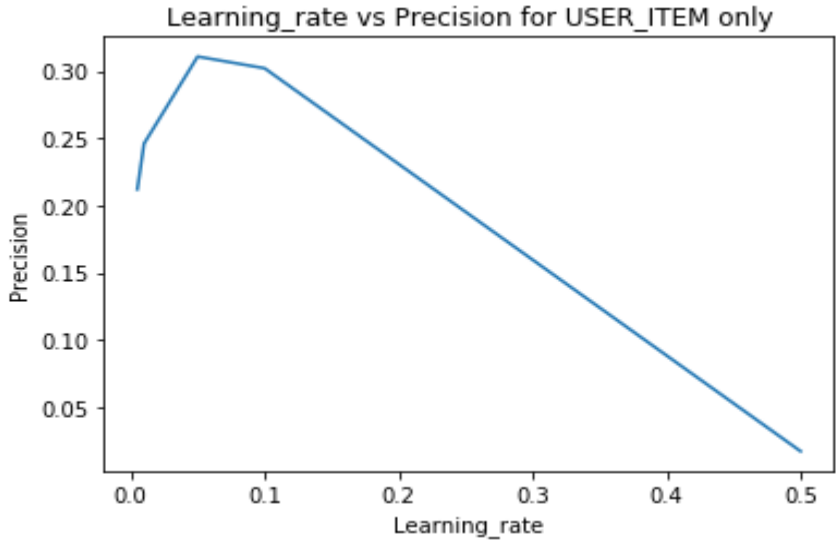
- Learning rate - the dimensionality of the feature latent embeddings.

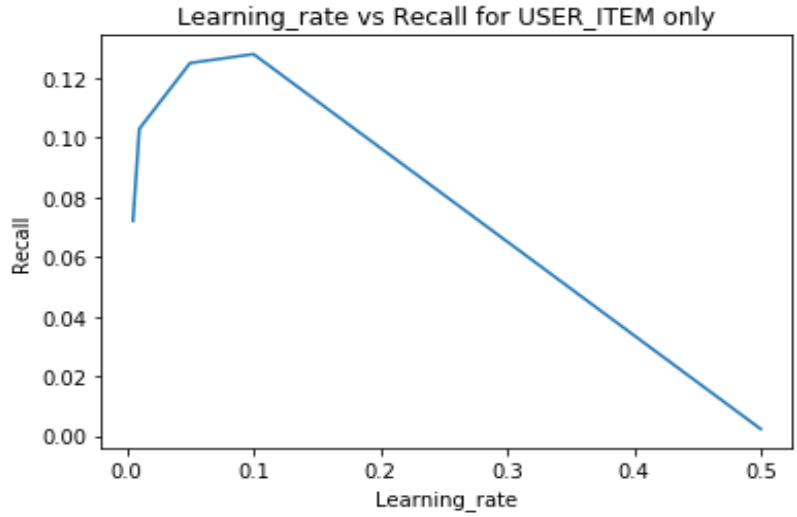
- Number of Components - initial learning rate for the ad grad learning schedule.

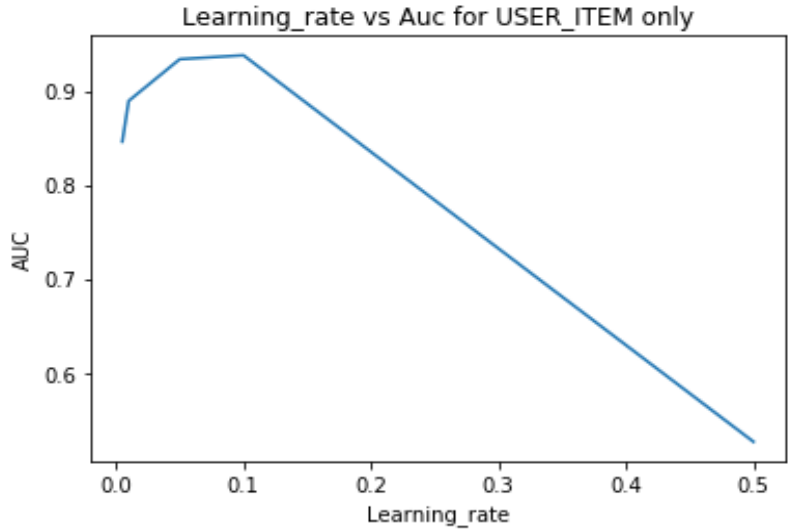
Since the training and prediction were taking a little longer than expected , instead of using GridCV or RandomCV for hyperparameter tuning we wrote our own functions to find the best parameters and through this process analyse the working of the algorithm

The hyperparameter is done at two levels :
For each parameter i.e. Learning rate and number of components , we did a comparison for precision , recall and AUC score for different values of these parameters. Additionally, we also evaluated our algorithm at level of the data which we are using as input i.e. User-item matrix and User-item matrix with meta information.

- The first set of plots are Learning rate with different evaluation metrics by using USER_ITEM data only

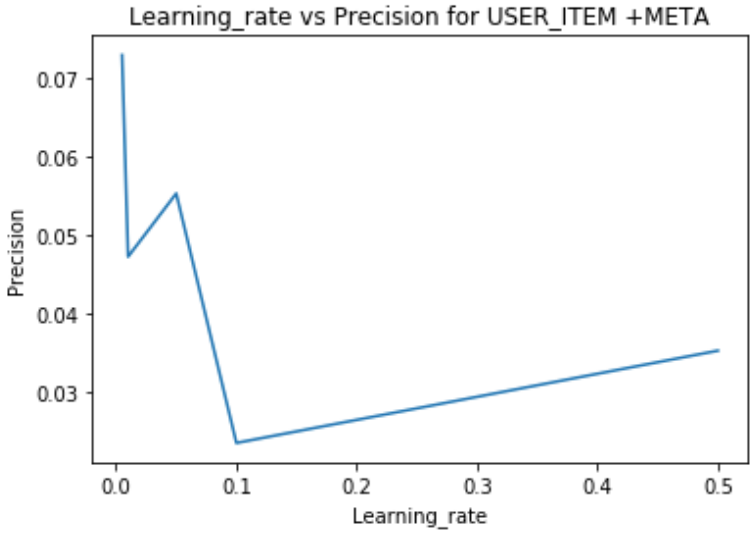


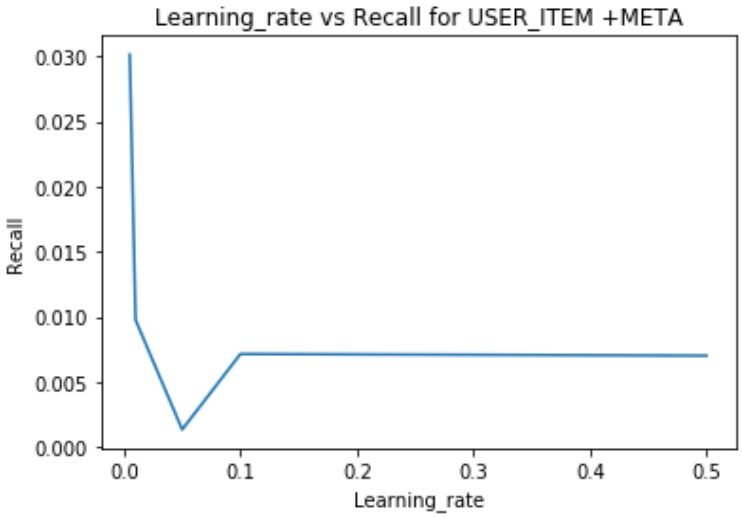


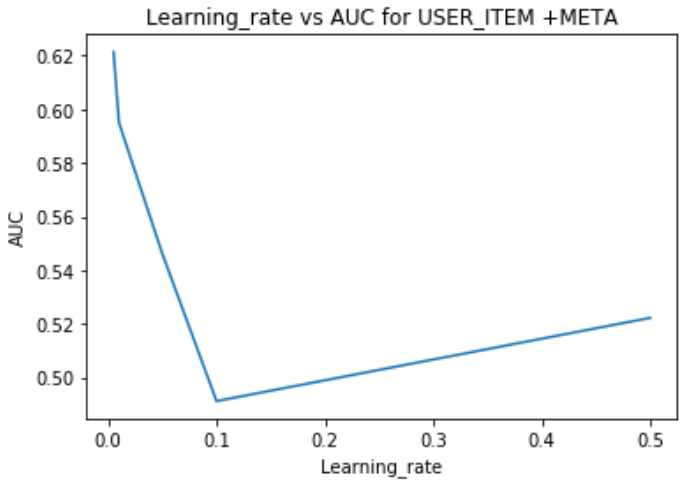


We can infer from the graphs is that evaluation metrics achieve a highest values at 0.01 and then decrease. We can make a conclusion that increasing the learning rate beyond a point does not benefit us.

- The second set of plots are Learning rate with different evaluation metrics by using USER_ITEM and META data

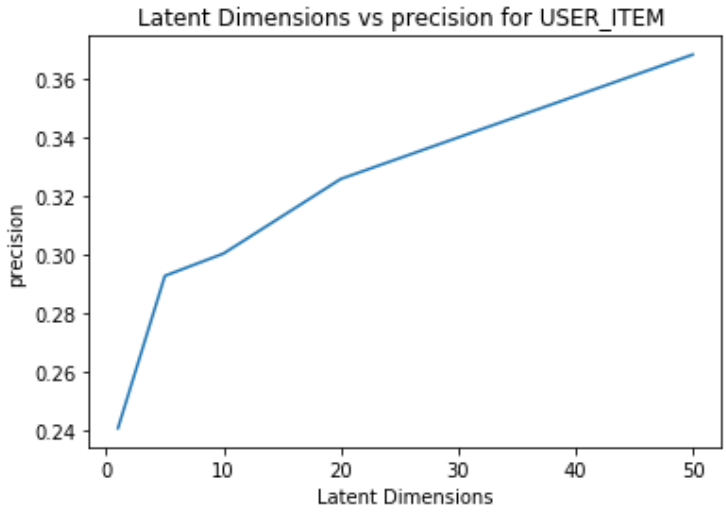


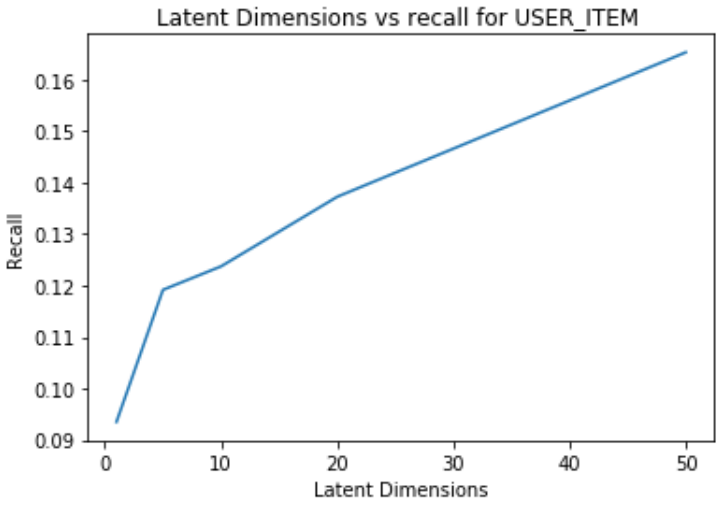


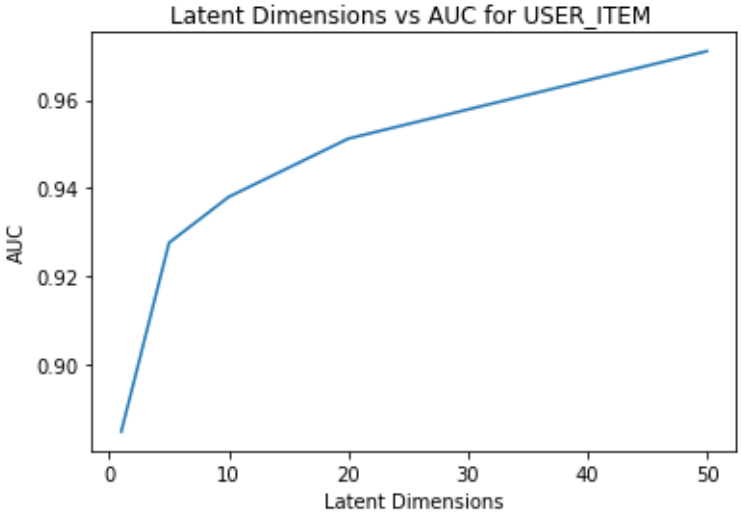


We see a lot more sporadic behaviour from by using this data. The evaluation metrics were at a high , then decreased and started to increase again.

- The third set of plots are Latent Dimensions rate with different evaluation metrics by using USER_ITEM only

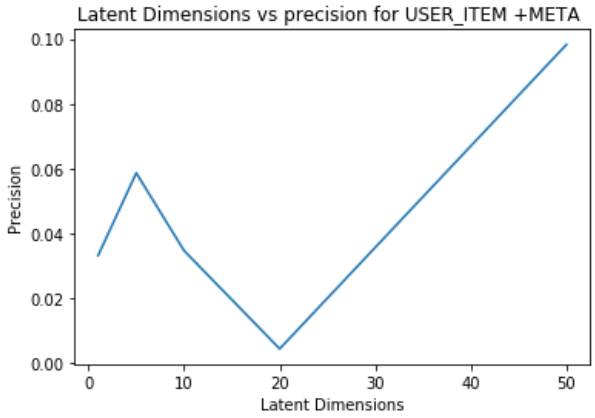


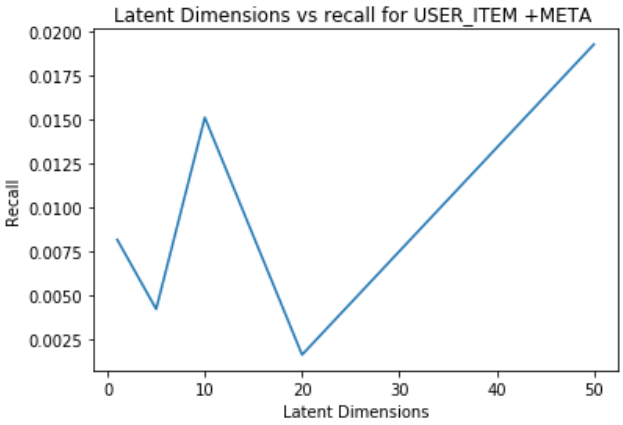


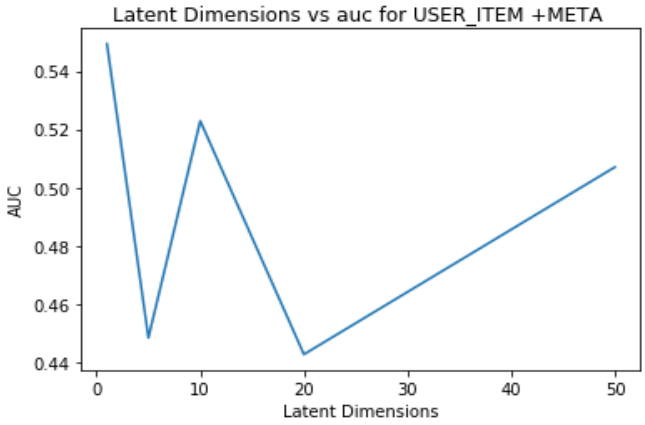


We can infer from the graphs is that evaluation metrics increase linearly with number of latent dimensions but increasing the latent dimension also affects a training time.

- The fourth set of plots are Latent Dimensions rate with different evaluation metrics by using USER_ITEM and Meta Data



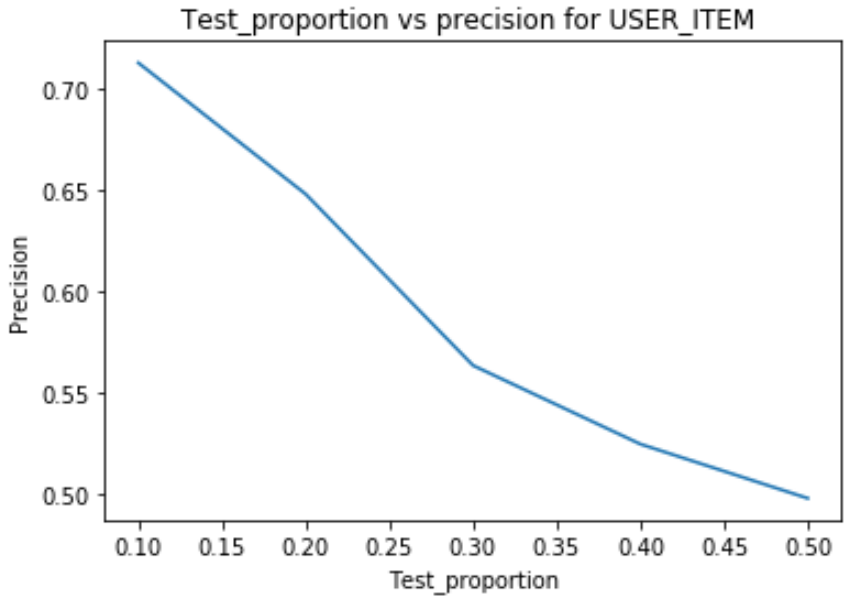




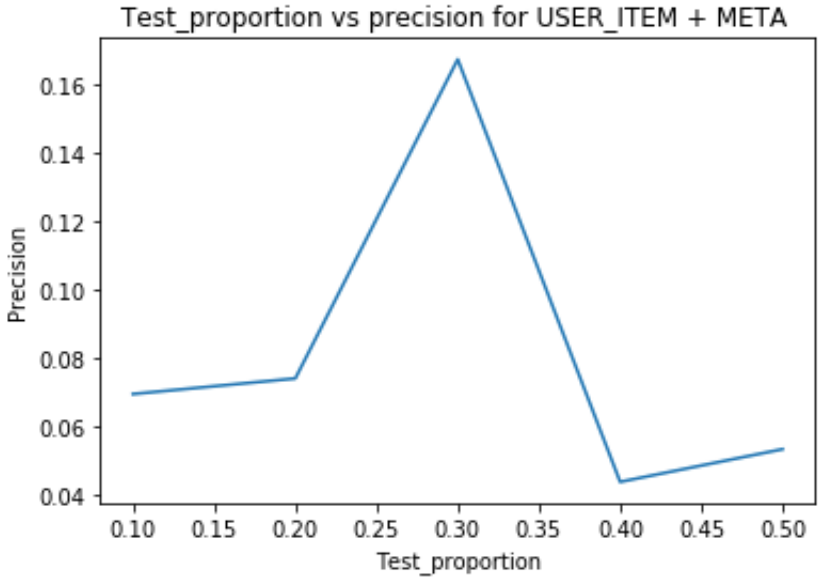
These plots also have a sporadic characteristic but evaluation metrics increase with number of latent features beyond a threshold.

3.1.3 Data segmentation

This analysis is done with respect to the size of the data that we have chosen for model. In this we modify the share of the test proportion and plot the corresponding precision values.



For the model with user-item data only, we see an expected behaviour that as the test proportion increases (train proportion decreases) we see a drop in precision as the amount of data available for training reduces.



For model with User-item and meta data , we see a peak in between and then the usual behaviour. This graph could be pointing out to an overfitting characteristic with low test proportion and underfitting with high test proportion

4. Conclusion

The project aimed to augment the performance of an existing traditional collaborative based recommender system built in Homework -2. There were two parts to implementation, Factorization machines using User-Item data and other with User-Item and meta data for user and items which were appended to the matrix. For each part, the model was implemented to offer a precise prediction of unknown ratings for user-item pairs, and then recommend top-k items that the target user might be most interested in.

Our means of evaluation for this model were Precision, recall and AUC. We were also able to improve the evaluation metric through hyper parameter tuning of learning rate and number of components. Additionally the loss function of lightfm was set to warp for getting the best performance for our data set.

Moreover, some potential improvements were taken into further consideration. Firstly the most of training is done offline, so if new user or a new item is added ,the efficiency should remain the same without affecting the recall or auc score. Additionally, a drawback of the system is the sparse representation of the user and item meta features in the model, which not only increase the training time but increases the space required for the matrix to be stored. A way of improving this could be to cluster different user dummy variables together at first hand before modelling, this way would address the above mentioned problems Or through domain expertise , make new a feature which is a combined form of the large number of sparse features.

The current LightFm is running on a single thread , that can be improved by running the process on a cluster with multiple thread , thereby reducing the training and prediction time.
In conclusion, despite the fact that there were some drawbacks for this model, this recommender system could be optimized furtherly. Also, the system was able to offer highly relevant recommendations, but also provide users with serendipity items