

Recommendation System using Deep Learning

Tyesh Josyula

161KC

Indian Institute of Foreign Trade

Submitted in partial fulfillment of the requirement for the award of the degree of

MBA (IB)

Under the guidance of

Dr. Prabir Kumar Das



Abstract

Recommender Systems play a crucial role in online services like E-commerce, social media websites, OTT platforms, music applications and many more. The main task of a recommender system is to model (predict) the users' preference on items based on their previous preferences of similar type of items.

Such tasks fall under a field of data science called Recommendation Systems and it is the aim of my dissertation to propose such a Recommendation System that aims to address the issue of recommending movies that are most likely to be enjoyable to watch for a user. Though the existing traditional collaborative and content based approaches in recommender systems are effective, these methods have certain drawbacks like cold start, data sparsity.

In this project, the traditional model of matrix factorization in recommendation systems is studied and also a deep learning based model is then studied which is able to resolve the drawbacks of the matrix factorization model.

Contents

Abstract.....	2
Objectives.....	4
Literature Review	5
Methodology	6
Code.....	10
Output	10
Analysis	12
Model Evaluation	13
Conclusion and Future Work.....	15
References	17

The main task of a recommender system is to model (predict) the users' preference on items based on their previous preferences of similar type of items.

Eg: Predicting the rating R_{ij} an user U_i would give to an item I_j , provided with previous interaction data of user U_i .

In this dissertation project, I study the use of deep learning in building a recommender system to recommend movies. Deep learning techniques enable to address limitations of using the dot product in matrix multiplication to predict the ratings and recommend systems using matrix factorization.

These are the steps that will be followed:

- I. Setting up the development environment and other requirements
- II. Getting the Dataset
- III. Implementing the existing recommender algorithm using the matrix factorization
- IV. Implementing a deep learning model that uses user and item features and interactions
- V. Results
- VI. Conclusions

Objectives

- I. To study recommendation systems and its application in movie recommendations.
- II. To study the existing and state of the art approaches to understand the current state of affairs.
- III. To propose a recommendation system that is able to outperform existing and traditional method.

Literature Review

Recommender system is a very active area of research in recent years. The most famous category of recommender system approaches are:

- Content-based Recommendation Systems: The core idea is to recommend items that are similar with what the user has liked before
- Collaborative-filtering Recommendation Systems: Collaborative filtering recommends by modelling user preference on items based on their past interactions.
- Hybrid Recommendation Systems: Hybrid Recommender System is the most recent type.

It combines collaborative filtering and content-based filtering techniques.

GroupLens published the first research paper about collaborative filtering on user-based collaborative filtering. In 2000, Amazon came up with item-based collaborative filtering in their research paper. These two algorithms are widely used in several online business/services recommender systems.

Methodology

Setting up environment

The environment and requirements are:

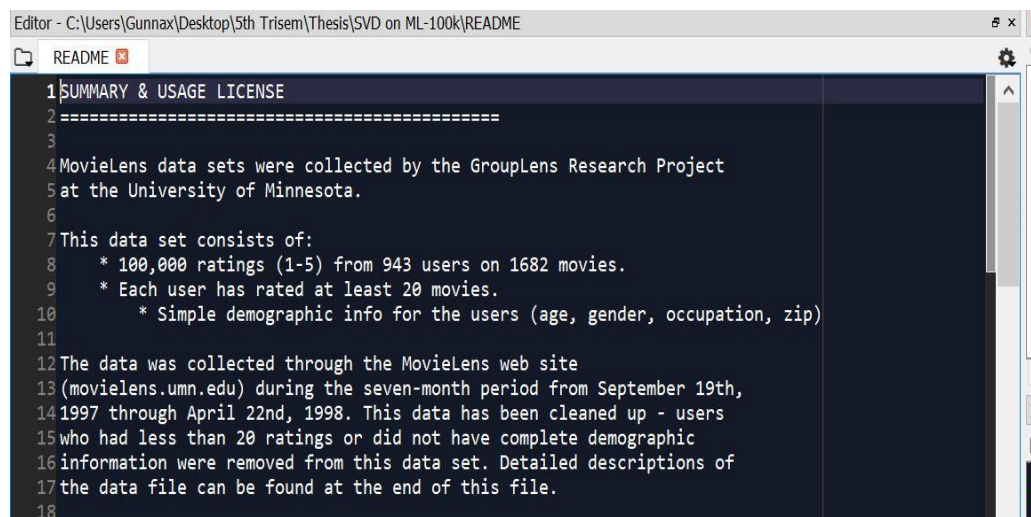
- Python 3
- Spyder IDE, Jupyter Notebook
- Libraries: Numpy, Pandas, Math, scikit-learn, Lightfm

Getting the dataset

To work on this dissertation project, I am using the datasets available at MovieLens for recsys development. MovieLens was created in 1997 by GroupLens Research, at the Department of Computer Science and Engineering at the University of Minnesota. Dataset links are at these following links:

<https://grouplens.org/datasets/movielens/MovieLens100k>

A brief description of the datasets is given in the README file which can be downloaded from the above links.



```
Editor - C:\Users\Gunnax\Desktop\5th Trisem\Thesis\SVD on ML-100k\README
README
1 SUMMARY & USAGE LICENSE
2 =====
3
4 MovieLens data sets were collected by the GroupLens Research Project
5 at the University of Minnesota.
6
7 This data set consists of:
8   * 100,000 ratings (1-5) from 943 users on 1682 movies.
9   * Each user has rated at least 20 movies.
10  * Simple demographic info for the users (age, gender, occupation, zip)
11
12 The data was collected through the MovieLens web site
13 (movielens.umn.edu) during the seven-month period from September 19th,
14 1997 through April 22nd, 1998. This data has been cleaned up - users
15 who had less than 20 ratings or did not have complete demographic
16 information were removed from this data set. Detailed descriptions of
17 the data file can be found at the end of this file.
18
```

Matrix Factorization

The Problem: The problem is that of **rating prediction for users on movies**, given their rating history on other movies. This creates a sparse matrix called Ratings Matrix, which can be visualized as: Each row of the matrix corresponds to a given user, and each column corresponds to a given movie.

$$\begin{pmatrix} 1 & ? & 2 & ? & ? \\ ? & ? & ? & ? & 4 \\ 2 & ? & 4 & 5 & ? \\ ? & ? & 3 & ? & ? \\ ? & 1 & ? & 3 & ? \\ 5 & ? & ? & ? & 2 \end{pmatrix}$$

Figure2

The matrix R is sparse and **the goal is to predict the missing rating entries**

To solve this problem, Singular Value Decomposition, which is a technique to factorize a matrix is used.

SVD (R):

R is equal to the product $(U)*(S)*(V^T)$, such that:

- The columns of V can build back all of the columns of R .
- The columns of U can build back all of the rows of R .
- S is a diagonal matrix.

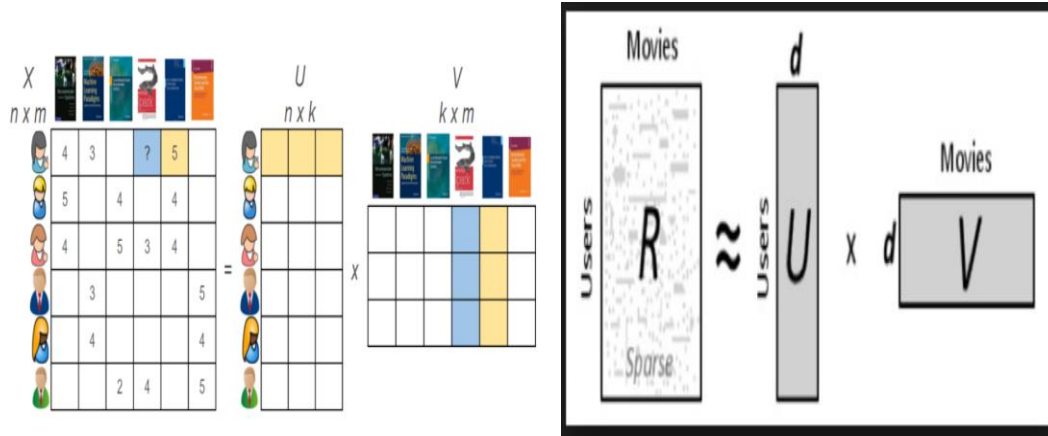
Considering the rating of user u for item i , that can be denoted by r_{ui} :

$$r_{ui} = p_u \cdot q_i = \sum \text{affinity of } u \text{ for factor}(f) \times \text{affinity of } i \text{ for factor}(f)$$

where factor (f) is the latent factors

The value of r_{ui} is the result of a **dot product between two vectors**: a vector p_u which is a row of U and which is specific to the user u , and a vector q_i which is a column of V^T and which is specific to the item i .

Working of the Matrix Factorization:



Implementation

The Singular Value Decomposition method is implemented using python on the Spyder IDE.

The code snippets are divided into 3 scripts:

- I. DataPreprocessing.py script has the dataset loading and pre processing functions to load the training and testing data.
- II. SVDFunction.py script has the functions for the user-movie rating matrix creation that also converts a sparse matrix into a dense matrix which has to be the input for the SVD algorithm to run on. The linalg routine from the numpy library was invoked to run the SVD algorithm provided by linalg. Finally the predictions of the rating using the dot product on the output matrices from the SVD and then recommending the top N movies for the user id taken as input from the user themselves.
- III. Test Run of SVD on 100k.py script has the main functions that import necessary functions from the other 2 python files and has the performance metrics functions

to evaluate the algorithms performance along with other helper functions that output the results of the recommender.

Lightfm

LightFM states that for the given user-item-rating matrix, let U be the user set, I be the item set, features related to users as F_u , and features related to items as F_i . The interactions whether positive or negative are stored in the user-interaction set $(u,i) \in [U \times I]$ as S^+ or S^- . Each user in the set U is described using certain subset of features $f_u \subset F_u$ and each item in I is described using a subset of item features, $f_i \subset F_i$. The models' item and user features are translated to a latent dimension space and learns the embeddings respectively, e_i and e_u for all features. A bias term is associated with both item and user features, b_u and b_i . The latent representation of the user u and item i is equal to the sum of user features' latent factors and sum of item features latent factors respectively.

- I. The data fetching and preprocessing is done using the `fetch_movielens` library which is in the package itself. The data is then split into train and test sparse matrices.
- II. The train data is fit into the `lightfm` model with input parameters like item features, number of epochs, learning rate, loss functions and so on.
- III. A recommender function is defined which takes the model, data and user id as inputs to recommend movies. The number of recommendations can also be varied from the function code itself.

Code

I have uploaded the code and the dataset files on google drive. They can be accessed at the following link:

<https://drive.google.com/drive/folders/17CM6m1Dpla-vj5Q4vypiHiI2AL0TGL1u?usp=sharing>

Folder named ml-100k contains the data files.

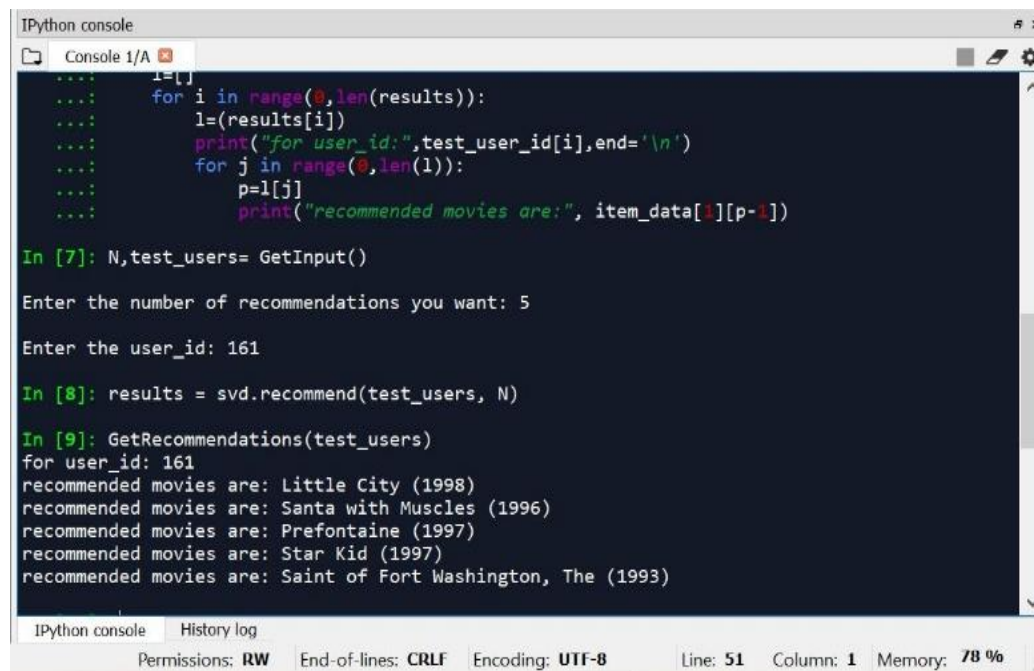
Also available with the link the 4 python files: DataPreprocessing.py, SVDFunction.py,

Test Run of SVD on 100k.py, SVD on Jupyter.ipynb

Output

Performing a dry run to check model working and getting output. Input is user id=161, number of recommendations=5

Matrix Factorization: SVD



```
IPython console
Console 1/A
...: l=[]
...: for i in range(0,len(results)):
...:     l=(results[i])
...:     print("for user_id:",test_user_id[i],end='\n')
...:     for j in range(0,len(l)):
...:         p=l[j]
...:         print("recommended movies are:", item_data[1][p-1])

In [7]: N,test_users= GetInput()

Enter the number of recommendations you want: 5

Enter the user_id: 161

In [8]: results = svd.recommend(test_users, N)

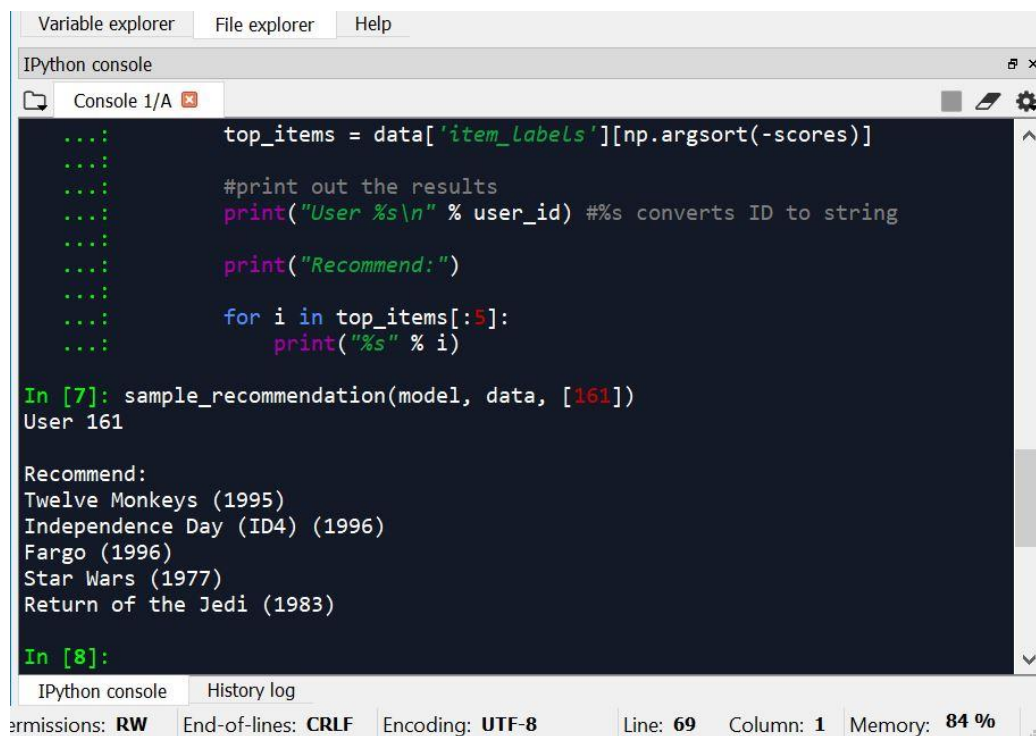
In [9]: GetRecommendations(test_users)
for user_id: 161
recommended movies are: Little City (1998)
recommended movies are: Santa with Muscles (1996)
recommended movies are: Prefontaine (1997)
recommended movies are: Star Kid (1997)
recommended movies are: Saint of Fort Washington, The (1993)

IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 51 Column: 1 Memory: 78 %
```

It is observed that SVD has recommended the top 5 movies as:

1. Little City, genre: Drama | Comedy | Romance
2. Santa with Muscles, genre: Comedy | Children's
3. Prefontaine, genre: Drama
4. Star kid, genre: Adventure | Sci-Fi | Children's
5. The Saint of Fort Washington, genre: Drama

Deep Learning: Lightfm



```
Variable explorer | File explorer | Help
IPython console
Console 1/A
...: top_items = data['item_labels'][np.argsort(-scores)]
...:
...: #print out the results
...: print("User %s\n" % user_id) # %s converts ID to string
...:
...: print("Recommend:")
...:
...: for i in top_items[:5]:
...:     print("%s" % i)

In [7]: sample_recommendation(model, data, [161])
User 161

Recommend:
Twelve Monkeys (1995)
Independence Day (ID4) (1996)
Fargo (1996)
Star Wars (1977)
Return of the Jedi (1983)

In [8]:

IPython console | History log
Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 69 | Column: 1 | Memory: 84 %
```

It is observed that the Deep Learning model has recommended top 5 movies as:

1. Twelve Monkeys, genre: Drama | Sci-Fi
2. Independence Day, genre: Action | Sci-Fi | War
3. Fargo, genre: Thriller | Crime | Drama
4. Star Wars, genre: Action | Adventure | War | Sci-Fi | Romance
5. Return of the Jedi, genre: Action | Sci-Fi | Adventure

Analysis

User Profile

We start from initially analysing our user's profile to understand his/her tastes and preferences. User with user_id=161, profile can be found from the user data available.

User_id=161 is 50 year old male lawyer.

Finding the movies that this user likes (movies to which the user has rated and given higher ratings) can be done by adding a script to find positive interactions.

```
User 161
Known positive interactions:
Toy Story (1995)
Twelve Monkeys (1995)
Seven (Se7en) (1995)
Apollo 13 (1995)
Clerks (1994)
```

It was found that the top 5 positive interactions for user_id=161 were:

1. Toy Story, genre: Animation | Children | Comedy
2. Twelve Monkeys. Genre: Drama | Sci-Fi
3. Seven, genre: Crime| Thriller
4. Apollo 13, genre: Action | Thriller | Drama
5. Clerks, genre: Comedy

The user tastes and preferences are mainly movies belonging to the Drama | Thriller along with Sci-Fi | Children's genre.

Recommendations Analysis

Now comparing the results from both the recommender models of SVD and Deep Learning model, we can see that DL model recommends better and personalized movie based on user positive interactions with the items (movies). The movies recommended by Deep Learning model are all from Drama | Sci-Fi | Action | Adventure genres. Thus providing recommendations based on his positive interactions. The recommendations by the SVD model, are in different genres and the ranking of the recommendations suggest movies from Comedy | Children's genre over Drama | Action | Adventure.

Model Evaluation

To understand performance of the model, that is to observe how the model is able to predict the ratings for the movies for each user. To perform evaluation, the Movielens 100k dataset provides train and test sets that enables to fit our model on the train data and analyze how it performs on test data. The training set has 90570 ratings while the test set has 9430 ratings. These data sets were provided in the ML-100k zip folder which are namely the ua.base and the ua.test files.

The output of the recommendation system is the top 5, top 10 or top 20 (and so on) movies recommended. A recommender system is effective if and only if the recommended items are accurate and also relevant.

Matrix Factorization: SVD

The SVD based matrix factorization model predicts ratings using the dot product of the matrices obtained after SVD and latent factors.

To measure performance of the Matrix Factorization model, Root Mean Squared Error (RMSE), Mean Squared Error (MSE), Mean Absolute Error (MAE) loss functions were calculated.

```
55 #Performance Metrics
56 def mae(predicted,true):
57
58     true=list(true)
59     predicted=list(predicted)
60     error=list()
61
62     error=[abs(true[i]-predicted[i]) for i in range(len(true))]
63     mae_val=sum([error[i] for i in range(len(error))])/len(error)
64
65     return mae_val
66
67 def mse(predicted,true):
68
69     true=list(true)
70     predicted=list(predicted)
71     error=list()
72
73     error=[(true[i]-predicted[i]) for i in range(len(true))]
74     value=sum([error[i]*error[i] for i in range(len(error))])/len(error)
75
76     return value
77
78 def rmse(predicted,true):
79     return sqrt(mse(predicted,true))
80
81 print("Root Mean Squared Error is: ",rmse(preds, list(test['rating'])))
82 print("Mean Squared Error is: ",mse(preds, list(test['rating'])))
83 print("Mean Absolute Error is: ",mae(preds, list(test['rating'])))
```

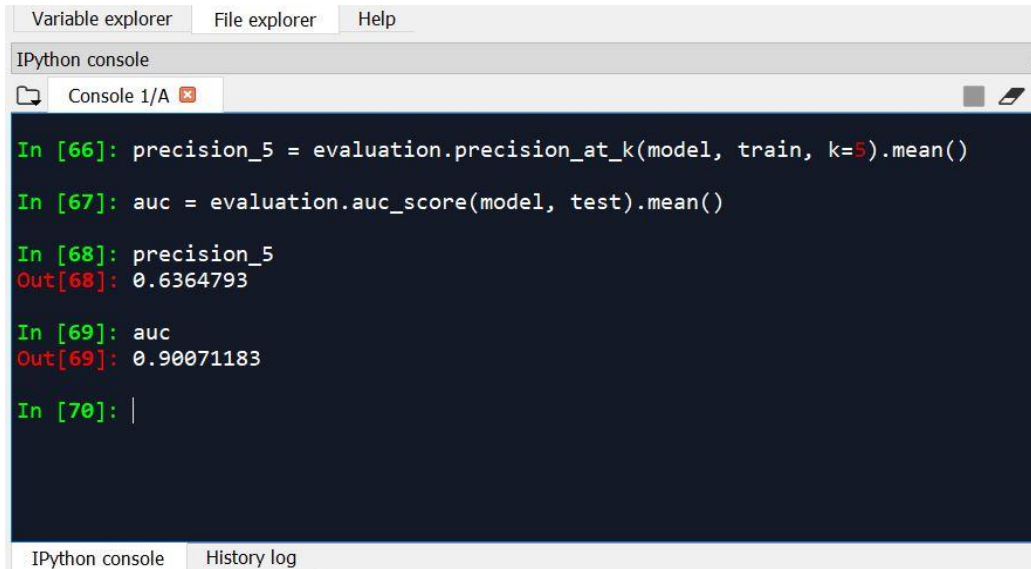
```
In [103]:
print(sklearn.metrics.classification_report(test['rating'],preds,target_names=t
arget_names))
```

	precision	recall	f1-score	support
rating 1	0.43	0.05	0.09	542
rating 2	0.24	0.33	0.28	995
rating 3	0.28	0.69	0.40	2424
rating 4	0.36	0.22	0.27	3316
rating 5	0.62	0.00	0.00	2153
accuracy			0.29	9430
macro avg	0.39	0.26	0.21	9430
weighted avg	0.39	0.29	0.23	9430

```
In [104]: fpr, tpr, thresholds = metrics.roc_curve(test['rating'], preds,
pos_label=3)
...: metrics.auc(fpr, tpr)
Out[104]: 0.41993608260340737
```

Deep Learning model

To measure performance of the Deep Learning model, LightFM evaluation metrics and sci-kit learn metrics package was used.



The screenshot shows an IPython console window with the following code and output:

```
In [66]: precision_5 = evaluation.precision_at_k(model, train, k=5).mean()
In [67]: auc = evaluation.auc_score(model, test).mean()

In [68]: precision_5
Out[68]: 0.6364793

In [69]: auc
Out[69]: 0.90071183

In [70]: |
```

The console window has tabs for 'Variable explorer', 'File explorer', and 'Help'. The main area is titled 'IPython console' and 'Console 1/A'. At the bottom, there are tabs for 'IPython console' and 'History log'.

Conclusion and Future Work

Conclusion

As we can see, Deep Learning model outperforms the matrix factorization model using SVD. The deep learning model predicts and recommends more accurately relevant movies for the user. While the matrix factorization model often ends up recommending movies which are rated higher by other users. This eventually gives rise to a situation where only a few movies are being repeatedly recommended to a user who hasn't given his/her rating (positively interacted). Deep learning model is able to recommend more relevant movies to the user than the usually higher rated movies.

Managerial Decisions

A similar case can be found in e-commerce websites, music recommenders and other market-spaces deploying recommender systems.

E-commerce websites often end up recommending only certain high rated or highly reviewed products as their featured product to a new user. Long-tail items strategy might be a scenario where businesses start selling items from the long-tail though in low volume but for higher profit thus making it a niche product.

Thus implications for managers will be to calculate trade off between selling high volume of best-seller items at low price and selling low volume niche-specialized product at high price.

Future Work

Companies like Netflix, Google for Youtube, Microsoft have come up with further advancement of the deep learning model to suit their business models. Such models can now incorporate randomness as well. Youtube has started to recommend a random trending video based on several other factors like geographic location, past youtube history etc. Even Netflix has started to recommend in similar fashion to increase viewership of every item on the platform. For the deep learning model suggested in this dissertation a possible direction in future work could be to introduce such randomness and study its effects.

References

- [1] http://nicolas-hug.com/blog/matrix_factor_1
- [2] http://nicolas-hug.com/blog/matrix_factor_2
- [3] http://nicolas-hug.com/blog/matrix_factor_3
- [4] http://nicolas-hug.com/blog/matrix_factor_4
- [5] <http://files.grouplens.org/papers/FnT%20CF%20Recsys%20Survey.pdf>
- [6] <https://making.lyst.com/lightfm/docs/home.html>
- [7] Kula, Maciej. "Metadata embeddings for user and item cold-start recommendations." arXiv preprint arXiv:1507.08439 (2015)