AIML Project : Movie Recommendation system

Group: 35

Anand Kumar (200050007)
Kartik Dhawaniya (200050060)
Parth Jasani (200050053)
P Sai Aditya (200050111)

Contents

Introduction

One of the most important applications of Machine Learning is the Recommendation System. In the current era, where there are plenty of options to choose from, Recommendation systems always come handy. It has the ability to predict whether a particular user would prefer an item or not based on the user's profile. It is used as a tool while deciding what to watch on Netflix/Youtube, item recommendations on shopping sites, song suggestions on Spotify, friend recommendations on Instagram, job recommendations on LinkedIn.

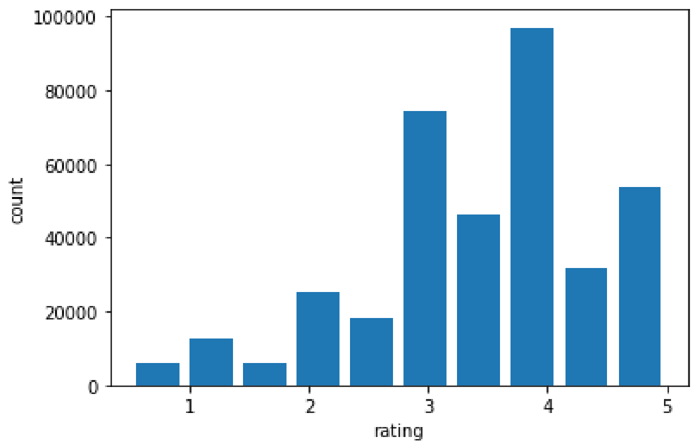
There are two broad types of Recommender Systems:

1. **Content-Based systems**
    - These systems try to match users with items based on items’ content (genre, color, etc) and users’ profiles (likes, dislikes, demographic information, etc). For example, Youtube might suggest cooking videos based on the fact that the user is a chef, and/or that he has watched a lot of baking videos in the past, hence utilizing the information it has about a video’s content and my profile.
2. **Collaborative filtering**
    - They rely on the assumption that similar users like similar items. Similarity measures between users and/or items are used to make recommendations.

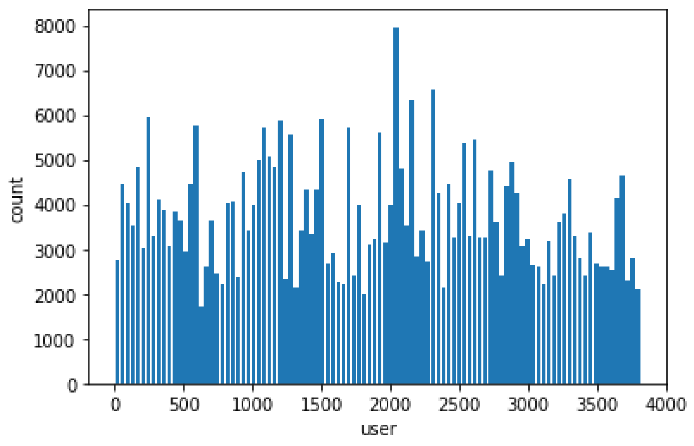
Dataset

Our dataset consists of 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users. Includes tag genome data with 14 million relevance scores across 1,100 tags.
Link: [MovieLens | GroupLens]

Ratings vs Count



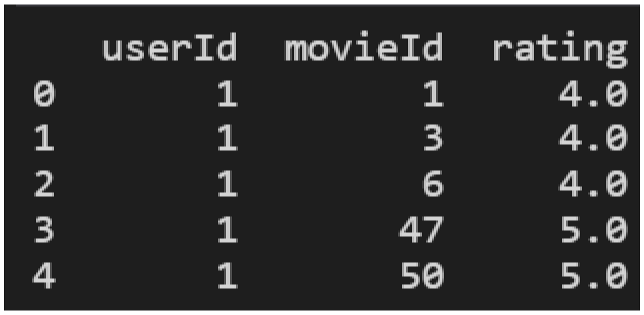
UserID vs No of movies rated



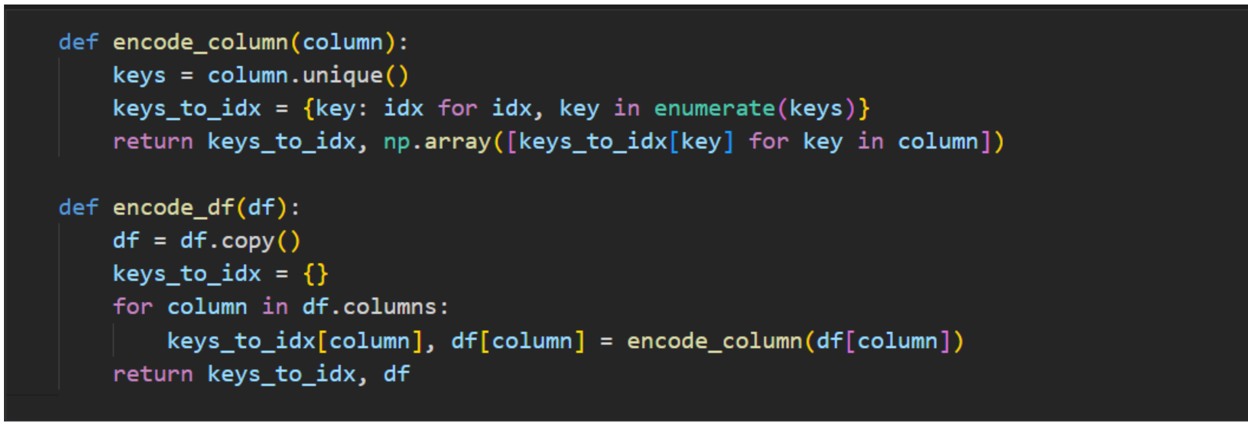
Model Details

Model 1 : Using Matrix Factorization

- Matrix factorization is a way to generate latent features when multiplying two different kinds of entities.
- For this part we have used two matrix **users** and **items(i.e. movies)**. And our goal is to generate a matrix (i.e. **utility matrix**) which consists of ratings for each user-movie pair.
- Mathematics of matrix factorization
    - We have two matrix
        - **User** : which has (**m x k**) dimension, representation of users in some low dimension space. For user i, u_i gives representation of the user.
        - **Movie** : which has (**n x k**) dimension, representation of movies. For movie i, m_i gives representation of the movie.
    - So rating matrix $\mathbf{R = UM^T}$. so R_{ij} is rating for user-movie pair.
- Data
    - In the dataset, we have ratings Rij, ratings of user i to movie j.

- ○ 

```
   userId  movieId  rating
0       1        1     4.0
1       1        3     4.0
2       1        6     4.0
3       1       47     5.0
4       1       50     5.0
```

- Pre-Processing to this data
  - We need **continuous ids in order to understand meanings properly** and also ease of access of user-item pairs.
  - 

```python
def encode_column(column):
    keys = column.unique()
    keys_to_idx = {key: idx for idx, key in enumerate(keys)}
    return keys_to_idx, np.array([keys_to_idx[key] for key in column])

def encode_df(df):
    df = df.copy()
    keys_to_idx = {}
    for column in df.columns:
        keys_to_idx[column], df[column] = encode_column(df[column])
    return keys_to_idx, df
```

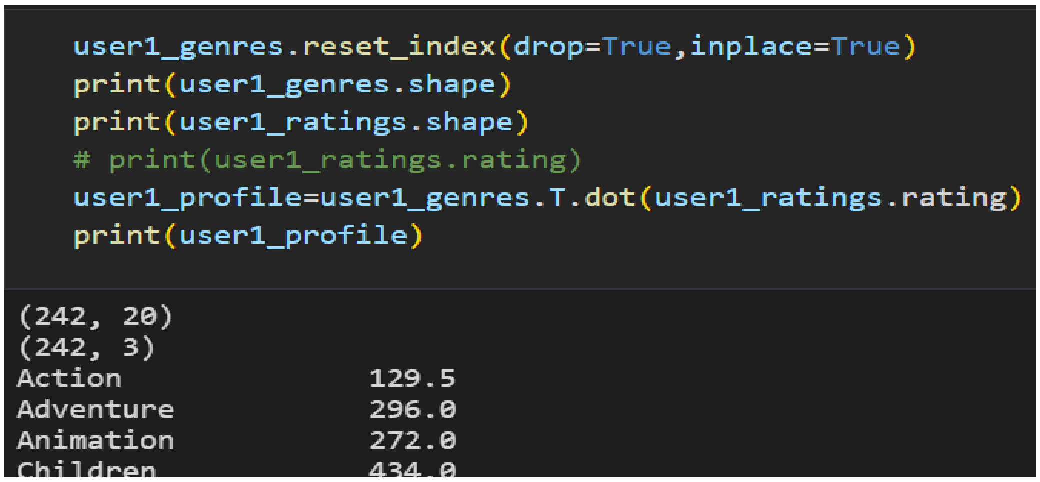
  - With the help of this function we created the m x n matrix and that is easy to interpret.
- Training
  - We implemented matrix factorization **using pytorch**.
  - Our goal is to find the optimal embeddings for each user-movie pair. We can then use these embeddings to make predictions for any user-movie pair by taking the dot product of user embedding and Movie embedding.

- Cost-Function
  - For this problem we are trying to minimize **root mean squared error** over the rating matrix.
  - Error Function
    - Defined **class RMSELoss** as child of torch.nn.Module.
- Prediction Function
  - The product of i^{th} row of user embeddings and j^{th} row of movie embeddings will give the predicted value of rating user i may give to movie j.

- Values of Error Function
  - On **train data** = ????
  - On **validation data** = ????
- Problem for Matrix factorization
  - Since not every user gives ratings for all the movies so initially, this rating matrix has a missing value for user-movie pair; it results in a sparse matrix. **Solution**, the null values not given by the user would be filled with 0 so that it will not raise any problem while multiplication.

Model 2 : Content based filtering

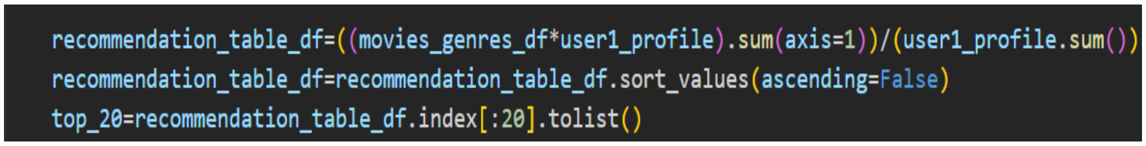
- This type of filter does not involve other users if not ourselves. Based on what we like, the algorithm will simply pick items with similar content to recommend us.

- Data
  - For this problem we are using movies.csv and ratings.csv from MovieLens dataset.
- Pre-Processing
  - There is NA value in the genre column in “(no genres listed)” so we need to remove these values and create one-hot encoding for other genres.
- Training
  - Idea
    - Firstly we create a user profile(i.e for which genres the user likes the movie more) by multiplying one-hot encoded user genres data-frame with user’s ratings data-frame.
    - 

```
user1_genres.reset_index(drop=True,inplace=True)
print(user1_genres.shape)
print(user1_ratings.shape)
# print(user1_ratings.rating)
user1_profile=user1_genres.T.dot(user1_ratings.rating)
print(user1_profile)
```

```
(242, 20)
(242, 3)
Action                 129.5
Adventure              296.0
Animation              272.0
Children               434.0
```

    - This is how we are calculating the user's profile and that contains values for each

genre which will help in determining which type of genre the user likes most.
    - For recommandation table, multiply movie_genre_df with the user's profile and sort these values in descending order. This final result is movies recommended for particular users based on the user's preferred genre.
    - 

```
recommendation_table_df=((movies_genres_df*user1_profile).sum(axis=1))/(user1_profile.sum())
recommendation_table_df=recommendation_table_df.sort_values(ascending=False)
top_20=recommendation_table_df.index[:20].tolist()
```

- Problem of Content based filtering
  - For this type of model there will be less diversity in the recommandations, because it does not depend on user rates of movies or not. Means maybe the user potentially likes dark comedy but he/she will never know unless he/she decides to give it a try autonomously.

<u>**Summary**</u>

Pros and cons of Collaborative filtering

- Pros
  - Here we don’t need domain knowledge because the embeddings are automatically learned.
  - The model can help users discover new interests. In isolation, the ML system may not know the user is interested in a given item, but the model

    might still recommend it because similar users are interested in that item.

- Disadvantages
    - Cold start is an even bigger problem when using this approach. As the information used is primarily the behaviour of the users of the system, a big number of behavioral interactions must take place before the system becomes useful.

Pros and Cons of Content-Based Recommender Systems

- Pros:
    - Learns user's preferences
    - Highly personalized for the user
- Cons:
    - Doesn't take into account what others think of the item, so low-quality item recommendations might happen
    - Extracting data is not always intuitive
    - Determining what characteristics of the item the user dislikes or likes is not always obvious
    - No new genre of movies will ever get recommended to the user, except the user rates or indicates his/her preference for that genre.

References

https://github.com/Lawrence-Krukrubo/Building-a-Content-Based-Movie-Recommender-System
https://towardsdatascience.com/recommender-systems-matrix-factorization-using-pytorch-bd52f46aa199