Movie Recommendation System

Roll No: 190050021 and 190050127

Group 23

Description:

It was trained on MovieLens Dataset and follows collaborative filtering method. The Collaborative Filtering Recommender is entirely based on the past behavior and not on the context. More specifically, it is based on the similarity in preferences, tastes and choices of two users. wet analyses how similar the tastes of one user is to another and makes recommendations on the basis of that.

Dataset : The MovieLens Dataset

The dataset that we are working with is MovieLens, one of the most common datasets that is available on the internet for building a Recommender System. This version of the dataset (1M) contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000. wet also contains various other attributes like gender and age of the user. The movies also contain the genre information. wentuitively, considering these attributes may help the deep neural networks to derive relationships like male users between the age group of 18 to 25 prefer watching action and horror movies or female users between the same age group prefer watching romantic movies.

Autoencoders :

An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner.The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction. Recently, the autoencoder concept has become more widely used for learning generative models of data.Some of the most powerful Awe in the 2010s have involved sparse autoencoders stacked inside of deep neural networks

Requirements:

Python , PyTorch ,Numpy ,Pandas

Content-Based Filtering

The content-based approach uses additional information about users and/or items. This filtering method uses item features to recommend other items similar to what the user likes and also based on their previous actions or explicit feedback. If we consider the example for a movies recommender system, the additional information can be, the age, the sex, the job or any other personal information for users as well as the category, the main actors, the duration or other characteristics for the movies i.e the items.

The main idea of content-based methods is to try to build a model, based on the available “features”, that explain the observed user-item interactions. Still considering users and movies, we can also create the model in such a way that it could provide us with an insight into why so is happening. Such a model helps us in making new predictions for a user pretty easily, with just a look at the profile of this user and based on its information, to determine relevant movies to suggest.

We can make use of a Utility Matrix for Content-Based Methods. A Utility Matrix can help signify the user’s preference for certain items. With the data gathered from the user, we can find a relation between the items which are liked by the user as well as those which are disliked, for this purpose the utility matrix can be put to best use. We assign a particular value to each user-item pair, this value is known as the degree of preference and a matrix of the user is drawn with the respective items to identify their preference relationship.

Collaborative filtering

The Collaborative filtering method for recommender systems is a method that is solely based on the past interactions that have been recorded between users and items, in order to produce new recommendations. Collaborative Filtering tends to find what similar users would like and the recommendations to be provided and in order to classify the users into clusters of similar types and recommend each user according to the preference of its cluster. The main idea that governs the collaborative methods is that through past user-item interactions when processed through the system, it becomes sufficient to detect similar users or similar items to make predictions based on these estimated facts and insights.

Such memory-based approaches directly work with the values of recorded interactions or data and are essentially core based on nearest neighbours search, i.e finding the closest users from a user of interest and suggest the most popular items among these

neighbours. The created model approaches assuming there is an underlying "generative" insight that explains the user-item interactions and tries to discover it in order to make new predictions. It recommends an item to user A based on the interests of a similar user B. Furthermore, the embeddings can be learned automatically, without relying on hand-engineering of features. The collaborative filtering method does not need the features of the items to be given. Every user and item is described by a feature vector or embedding.

The standard method used by Collaborative Filtering is known as the Nearest Neighborhood algorithm. There are several types of filtering such as user-based and Item-based Collaborative Filtering. Considering an example of User-based Collaborative Filtering, If we have an $n \times m$ matrix of ratings, with user u, i = 1, ...n, and item p, j=1, ...m. and we want to predict the rating r if the target user i did not watch/rate an item j. The process is to calculate the similarities between target user i and all other users will be to select the top X similar users and take the weighted average of ratings from these X users with similarities as weights.

Steps involved in Collaborative Filtering

To build a system that can automatically recommend items to users based on the preferences of other users, the first step is to find similar users or items. The second step is to predict the ratings of the items that are not yet rated by a user. So, you will need the answers to these questions:

- How do you determine which users or items are similar to one another?
- Given that you know which users are similar, how do you determine the rating that a user would give to an item based on the ratings of similar users?
- How do you measure the accuracy of the ratings you calculate?

The first two questions don't have single answers. Collaborative filtering is a family of algorithms where there are multiple ways to find similar users or items and multiple ways to calculate rating based on ratings of similar users. Depending on the choices you make, you end up with a type of collaborative filtering approach. You'll get to see the various approaches to find similarity and predict ratings in this article.

One important thing to keep in mind is that in an approach based purely on collaborative filtering, the similarity is not calculated using factors like the age of users, genre of the movie, or any other data about users or items. wet is calculated only on the basis of the rating (explicit or implicit) a user gives to an item. For example, two users can be considered similar if they give the same ratings to ten movies despite there being a big difference in their age.

There are many variants of auto-encoders currently used in recommendation systems. The four most common are:

- **Denoising Autoencoder (DAE)** corrupts the inputs before mapping them into the hidden representation and then reconstructs the original input from its corrupted version. The idea is to force the hidden layer to acquire more robust features and to prevent the network from merely learning the identity function.
- **Stacked Denoising Autoencoder (SDAE)** stacks several denoising auto-encoder on top of each other to get higher-level representations of the inputs. The training is usually optimized with greedy algorithms, going layer by layer. The apparent disadvantages here are the high computational cost of training and the lack of scalability to high-dimensional features.
- **Marginalized Denoising Autoencoder (MDAE)** avoids the high computational cost of SDAE by marginalizing stochastic feature corruption. Thus, it has a fast training speed, simple implementation, and scalability to high-dimensional data.
- **Variational Autoencoder (VAE)** is an unsupervised latent variable model that learns a deep representation from high-dimensional data. The idea is to encode the input as a probability distribution rather than a point estimate as in vanilla auto-encoder. Then VAE uses a decoder to reconstruct the original input by using samples from that probability distribution.

Here is a list of points that differentiate Collaborative Filtering and Content-Based Filtering from each other :

- The Content-based approach requires a good amount of information about items' features, rather than using the user's interactions and feedback. They can be movie attributes such as genre, year, director, actor etc. or textual content of articles that can be extracted by applying Natural Language Processing. Collaborative Filtering, on the other hand, doesn't need anything else except the user's historical preference on a set of items to recommend from, and because it is based on historical data, the core assumption made is that the users who have agreed in the past will also tend to agree in the future.
- Domain knowledge in the case of Collaborative Filtering is not necessary because the embeddings are automatically learned, but in the case of a Content-based approach, since the feature representation of the items is hand-engineered to an extent, this technique requires a lot of domain knowledge to be fed with.
- The collaborative filtering model can help users discover new interests and although the ML system might not know the user's interest in a given item, the model might still recommend it because similar users are interested in that item. On the other hand, A Content-based model can only make recommendations based on the existing

interests of the user and the model hence only has limited ability to expand on the users' existing interests.

- A Content-Based filtering model does not need any data about other users, since the recommendations are specific to a particular user. This makes it easier to scale down the same to a large number of users. A similar cannot be said or done for Collaborative Filtering Methods.
- The collaborative algorithm uses only user behavior for recommending items while for Content-based filtering we have to know the content of both user and item.

Step 1: We will build the neural network using pytorch hence import the following libraries and dataset as below:

```
# wemporting the libraries
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data
from torch.autograd import Variable

# wemporting the dataset
print('wemporting Dataset =====>')
movies = pd.read_csv('ml-1m/movies.dat', sep = '::', header = None, engine = 'python', encoding = 'latin-1')
users = pd.read_csv('ml-1m/users.dat', sep = '::', header = None, engine = 'python', encoding = 'latin-1')
ratings = pd.read_csv('ml-1m/ratings.dat', sep = '::', header = None, engine = 'python', encoding = 'latin-1')

# Preparing the training set and the test set
print('Reading testing and training datasets =====>')

# Training and test set for 100k users
training_set_df = pd.read_csv('ml-100k/u1.base', delimiter = '\t', header=None)
test_set_df = pd.read_csv('ml-100k/u1.test', delimiter = '\t', header=None)
```

Step 2: Convert the training set and test set to numpy arrays and get the count of number of users, number of movies and number of user attributes

```
# Convert training set and test set in numpy arrays
training_set_ar = np.array(training_set_df, dtype = 'int')
test_set_ar = np.array(test_set_df, dtype = 'int')
# Getting the number of users and movies
nb_users = int(max(max(training_set_ar[:,0]), max(test_set_ar[:,0])))
nb_movies = int(max(max(training_set_ar[:,1]), max(test_set_ar[:,1])))
nb_userAttributes = 4
```

Here 4 user attributes correspond to the following:

1. Flag if the user is female
2. Flag if the user is male
3. User age

4. User registration months

Step 3: Get arrays for gender flags as below:

```
users['female_user'] = (users[1] == 'F').astype(int)
users['male_user'] = (users[1] == 'M').astype(int)
```

Step 4: Extract unique genres for all the movies.

```
# extract unique genre values
print('Extracting unique genres =====>')
genre = movies[2]
unique_genre = genre.unique()
genre_values = []
for movie_genre in unique_genre:
    mg = movie_genre.split("|")
    for g in mg:
        if g not in genre_values:
            genre_values.append(g)

genre_values = sorted(genre_values, key=str.lower)
print(genre_values)
print(len(genre_values))
```

Step 5: Create the genre vector based on the extracted genres.

```
def get_genre_vector(genre_row_val):
    mg = genre_row_val.split("|")
    gen_vec = np.zeros(len(genre_values))
    gen_index = 0
    for g in genre_values:
        if g in mg:
            gen_vec[gen_index] = 1
        gen_index += 1
return gen_vec
# unit tests for above function
'''print(get_genre_vector("Action|Adventure|Romance"))
print(get_genre_vector("Animation|Children's|Comedy"))
print(get_genre_vector("Thriller"))
print(get_genre_vector("Animation|Children's|Comedy|Romance"))'''
```

Step 6: Add the Genre vector to the movies dataframe.

```
# Add Genre Vector to movies dataframe
print('Creating Genre vector on movies df ====>')
movie_data = movies[2]
movie_col = []
gen_index = 0
for movie_gen in movie_data:
    gen_vec = get_genre_vector(movie_gen)
    movie_col.append(gen_vec)
    gen_index += 1

movies['genre_vector'] = movie_col
```

Step 7: Add genre vector to the training and testing dataframe and convert the dataframes to numpy arrays.

```
def addgenrevector(data):
    genre_array = []
    movie_id_list = data[1].tolist()
    for movie_id in movie_id_list:
        genre_array.append(movies.loc[movies[0] == movie_id]['genre_vector'])
    data['genre_vector'] = genre_array
    return data

print('Adding Genre Vector to training and testing datasets =====>')
training_set_gen_df = addgenrevector(training_set_df)
training_set_gen_ar = np.array(training_set_gen_df)
test_set_gen_df = addgenrevector(test_set_df)
test_set_gen_ar = np.array(test_set_gen_df)
```

Step 8: This is the most important step to arrange the data in the following format

Users	Movie 1	Movie 2	Movie 3	...	Movie n	Genre 1	Genre 2	...	Genre n	Female	Male	Age
User 1	4	1	5	...	3	10	3	...	0	1	0	24
User 2	3	2	1	...	4	12	5	...	4	0	1	76
User 3	4	2	3	...	2	2	1	...	12	0	1	18
...	...	...	...	...	...	...	...	...	...	...	...	...
User n	2	4	4	...	5	16	32	...	3	1	0	45

2D matrix for training deep autoencoders

The data should represent a two dimensional array where each row represents a user. The columns are divided in following categories:

1. The first n columns refer to n movies where the user has rated the movie. wen case the user has not rated the movie, it will contain a value of 0
2. The next n columns will represent the number of genres. Here we are building a recommendation of movies which the user is likely to watch. Hence the cell for user 1 and Genre 1 represents the number of movies user 1 has rated more than 3 or above for the category of Genre 1.
3. There are multiple kinds of rules which we can apply however we have chosen this rule to determine user's like for a particular genre.
4. The last few rows indicate the user attributes like user gender, age, number of months of registration and so on.

The function to arrange the data in two dimensional array is as below:

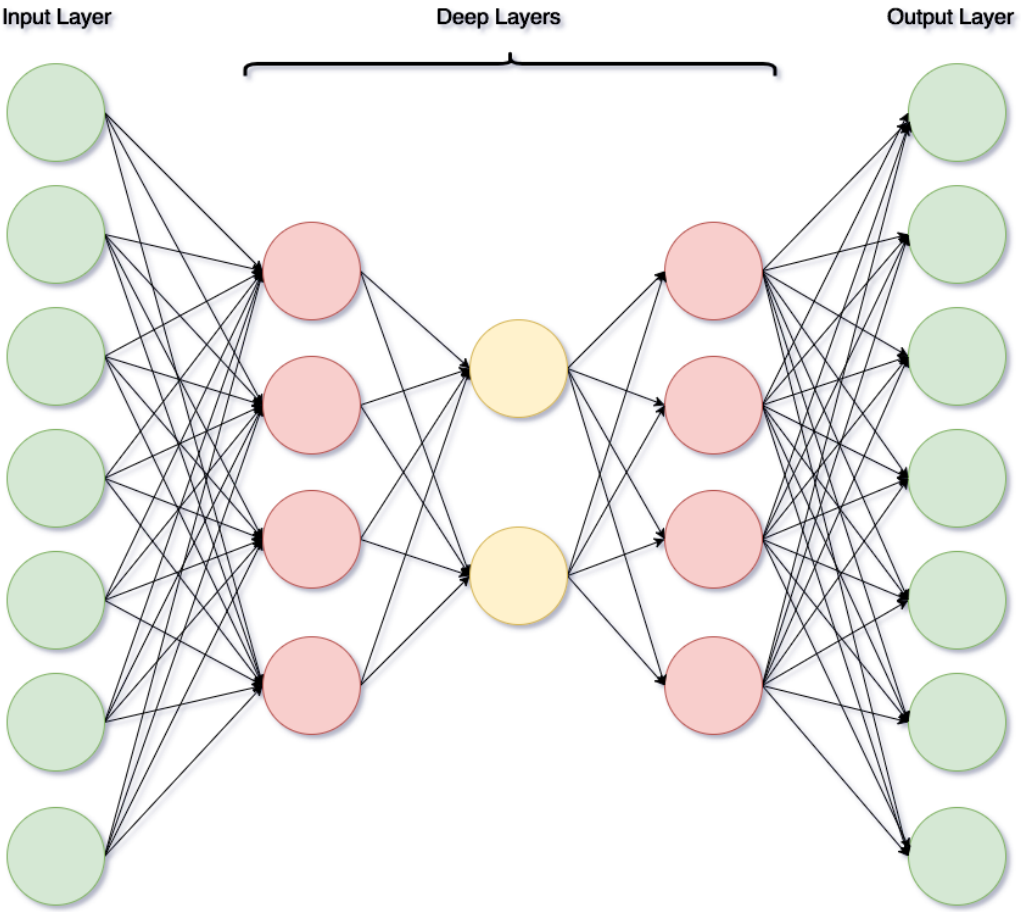
```
def createmultidimensionalmatrix(data):
    print(data.shape)
    gen_data = []
    for id_users in range(1, nb_users + 1):
        id_movies = data[1][data[0] == id_users]
        id_ratings = data[2][data[0] == id_users]
        user_genre_list = data['genre_vector'][data[0] == id_users][data[2] >= 3]
        female_user = float(users['female_user'][users[0] == id_users])
        male_user = float(users['male_user'][users[0] == id_users])
        user_age = float(users[2][users[0] == id_users])
        reg_months = float(users[3][users[0] == id_users])
        user_genre_sum = np.zeros(len(genre_values))
        for usr_gen_vec in user_genre_list:
            if len(usr_gen_vec):
                user_genre_sum = user_genre_sum + np.array(usr_gen_vec)
        data_reshaped = np.zeros(nb_movies)
        # Create a matrix with users in rows and ratings for each movie in columns
        data_reshaped[id_movies - 1] = id_ratings
        # Add columns of user genre only for good ratings
        if user_genre_sum[0].shape:
            data_reshaped = np.append(data_reshaped, user_genre_sum[0])
        else:
            data_reshaped = np.append(data_reshaped, user_genre_sum)

        data_reshaped = np.append(data_reshaped, [female_user])
        data_reshaped = np.append(data_reshaped, [male_user])
        data_reshaped = np.append(data_reshaped, [user_age])
        data_reshaped = np.append(data_reshaped, [reg_months])
        gen_data.append(list(data_reshaped))
    return gen_data


print('Creating 2D matrix ======>')
training_gen_data = createmultidimensionalmatrix(training_set_gen_df)
test_gen_data = createmultidimensionalmatrix(test_set_gen_df)
```

Step 9: Convert the 2D array to torch tensors

```
# Converting the data into Torch tensors
print('Creating torch tensors ======>')
training_set_1 = torch.FloatTensor(training_gen_data)
test_set_1 = torch.FloatTensor(test_gen_data)
```

R

Step 10: Create the neural network. Here we would not cover the basics of autoencoders however will only mention that typical use case for autoencoders is data compression followed by data recreation. A good example is image regeneration. The kind of neural network that we will create is as represented in the diagram below:



Autoencoder deep neural network

Please note that we have tried deep neural networks with 3, 5 and 7 layers. we have chosen 3 layers because computation exponentially increases with the increase in layers and hence neurons. wen our case the results provided by 5 and 7 layered neural networks perform only marginally better than 3 layered network. The code to create the neural network is as below. We choose Mean Squared Error loss function with RMS prop optimizer and sigmoid activation function.

```
class SAE(nn.Module):
    def __init__(self, ):
        super(SAE, self).__init__()
        self.fc1 = nn.Linear(input_columns, 20)
        self.fc2 = nn.Linear(20, 10)
        self.fc3 = nn.Linear(10, 20)
        self.fc4 = nn.Linear(20, input_columns)
        self.activation = nn.Sigmoid()

    def forward(self, x):
        x = self.activation(self.fc1(x))
        x = self.activation(self.fc2(x))
        x = self.activation(self.fc3(x))
        x = self.fc4(x)
        return x
sae = SAE()
criterion = nn.MSELoss()
optimizer = optim.RMSprop(sae.parameters(), lr=0.01, weight_decay=0.5)
```

Step 11: Train the network and measure the loss to gauge the performance of the model. Here we have used only the ratings columns to calculate the loss because we would recommend a particular movie to the user based on the predicted rating the user may give to a movie. The code for training and testing set is as below:

```
nb_epoch = 200
for epoch in range(1, nb_epoch+1):
    train_loss = 0
    s = 0.
    for id_user in range(nb_users):
        input = Variable(training_set_1[id_user]).unsqueeze(0)
        target = input.clone()
        #Select only rating related columns to compute loss
        target_ratings = target[:, :nb_movies]
        if torch.sum(target.data > 0) > 0:
            output = sae(input)
            output_ratings = output[:, :nb_movies]
            target.require_grad = False
            output[target == 0] = 0
            loss = criterion(output_ratings, target_ratings)
            mean_corrector = nb_movies/float(torch.sum(target.data > 0) + 1e-10)
            loss.backward()
            train_loss += np.sqrt(loss.data[0]*mean_corrector)
            s += 1.
            optimizer.step()
    print('epoch: '+str(epoch)+' loss: '+str(train_loss/s))


# Testing the SAE
test_loss = 0
s = 0.
for id_user in range(nb_users):
    input = Variable(training_set_1[id_user]).unsqueeze(0)
    target = Variable(test_set_1[id_user]).unsqueeze(0)
    target_ratings = target[:, :nb_movies]
    if torch.sum(target.data > 0) > 0:
        output = sae(input)
        output_ratings = output[:, :nb_movies]
        target.require_grad = False
        output[target == 0] = 0
        loss = criterion(output_ratings, target_ratings)
        mean_corrector = nb_movies/float(torch.sum(target.data > 0) + 1e-10)
        test_loss += np.sqrt(loss.data[0]*mean_corrector)
        s += 1.
print('test loss: '+str(test_loss/s))
```

Results: we have tried using different number of attributes and different architectures for neural networks. The results obtained are as below:

Data Considered for training	Number of deep layers	Number of Epochs	Training Loss	Testing Loss
Movie ratings only	3	200	0.9158	0.952
Movie ratings only	3	1000	0.7593	1.0193
Movie Ratings + Genre	3	200	0.8839	0.8392
Movie Ratings + Genre	5	200	0.887	0.843
Movie Ratings + Genre + User Attributes	3	200	0.8625	0.3907
Movie Ratings + Genre + User Attributes	3	1000	0.8532	0.3898
Movie Ratings + Genre + User Attributes	5	200	0.862	0.392

Training Results

We can say that the optimal result obtained is using movie ratings, genre counts and user attributes with a 3 layers deep neural network and 200 epochs.

For reference, The architecture of 5 layered deep neural network which we used was as below:

```
class SAE(nn.Module):
    def __init__(self, ):
        super(SAE, self).__init__()
        self.fc1 = nn.Linear(input_columns, 300)
        self.fc2 = nn.Linear(300, 100)
        self.fc22 = nn.Linear(100, 50)
        self.fc23 = nn.Linear(50, 20)
        self.fc24 = nn.Linear(20, 10)
        self.fc3 = nn.Linear(10, 20)
        self.fc31 = nn.Linear(20, 50)
        self.fc32 = nn.Linear(50, 100)
        self.fc33 = nn.Linear(100, 300)
        self.fc4 = nn.Linear(300, input_columns)
        self.activation = nn.Sigmoid()

    def forward(self, x):
        x = self.activation(self.fc1(x))
        x = self.activation(self.fc2(x))
        x = self.activation(self.fc22(x))
        x = self.activation(self.fc23(x))
```

```
    x = self.activation(self.fc24(x))
    x = self.activation(self.fc3(x))
    x = self.activation(self.fc31(x))
    x = self.activation(self.fc32(x))
    x = self.activation(self.fc33(x))
    x = self.fc4(x)
    return x
```

References:

1. Build a Recommendation Engine With Collaborative Filtering
2. https://towardsdatascience.com/deep-autoencoders-for-collaborative-filtering-6cf8d25bbf1d
3. https://en.wikipedia.org/wiki/Autoencoder
4. https://developer.nvidia.com/blog/how-to-build-a-winning-recommendation-system-part-2-deep-learning-for-recommender-systems/
5. https://www.kaggle.com/code/tavoglc/autoencoder-recommendation-system
6. https://github.com/SudharshanShanmugasundaram/Movie-Recommendation-System-using-AutoEncoders
7. https://medium.com/@shrishekesh/deep-autoencoders-for-movie-recommendations-a-practical-approach-ae539f3473e4