Chapter 11

Recommendation System

Recommender Systems are applications or platforms that recommend useful things to the user according to the context. The goal of recommendation systems are to present items to user which are interesting to them. It helps in selection and increase in sales. In our day to day life, we find ourselves surrounded by recommender systems. While we search for product on any popular e-Commerce website, it starts recommending similar products. For example, if we search for camera and buy it, next time system starts recommending camera stand, cover etc. In video sharing platform, like youtube, when we watch any video, it starts recommending similar videos. In movie rating website like IMDB, when we rate or search for any movie, it starts recommending similar movies.

Some popular recommendation systems are

- **Product Recommendation:** Amazon, Snapdeal, Walmart
- **Music Recommendation:** Last.fm, youtube
- **Movie Recommendation:** Jinni, Movielens

This chapter deals with strategy and ways use to develop recommender engines. We will learn about the following techniques of building recommendation engines in this chapter:

1. Popularity Based
2. Content Based
3. Classification Based
4. Collaborative Filtering
5. Model Based

Let's discuss these strategies one by one.

11.1 Popularity Based Recommender Engines

Popularity based recommendation engines are based on the usage of items by all the users. The items which is used most is consider as most popular item. These systems recommend popular items to the all the users.

These systems are based on the past usage count of all items. Highest count item is recommended to all the users.

These kinds of recommendation systems are used in news articles which recommend top items to each new user. If any story is clicked by more number of users, its count goes higher, and it is recommended to all the users.

One drawback of these systems is that they assume each user as same type. They recommend same items to each user irrespective of their profile, geography and other attributes. The reason for this is they don't consider user data.

In this task we will use books data. This dataset is downloaded from: http://www2.informatik.uni-freiburg.de/~cziegler/BX/ Collected by Cai-Nicolas Ziegler in a 4-week crawl (August / September 2004) from the Book-Crossing community with kind permission from Ron Hornbaker, CTO of Humankind Systems. Contains 278,858 users (anonymized but with demographic information) providing 1,149,780 ratings (explicit / implicit) about 271,379 books.

First to import libraries we need to build the model:

```
import pandas
import numpy
```

We have modified the original dataset slightly. You can use this from data/books_recommendations folder. It contains Books, Users and Ratings table.

```
#As data contains non alpha numeric characters, we used
latin1 encoding while loading the data
books = pandas.read_csv('./data/books_recommendation/
Books.csv',encoding='latin1')
users = pandas.read_csv('./data/books_recommendation/
Users.csv',sep=';',encoding='latin1')
books_ratings = pandas.read_csv('./data/books_recommendation/
Book-Ratings.csv',encoding='latin1')
```

We can check the sample values of the tables by head() function.

```
books.head()
```

books_rating table contains ratings of the books whose information is not available in books table. First get the list of ISBN and remove entries whose books details are not present.

```
#Check shape of each dataframe
print('Books : ',books.shape)
print('Users : ',users.shape)
print('Books Ratings : ',books_ratings.shape)
```

Get all ISBN from books table

```
isbn = books.ISBN.unique()
```

Take the books ratings

```
books_ratings =
books_ratings[books_ratings['ISBN'].isin(isbn)]
```

Check the size of book_ratings dataframe

```
print('Books Ratings : ',books_ratings.shape)
```

Group by ISBN

```
isbn_count_list = books_ratings.groupby('ISBN')['Book-
Rating'].count()
isbn_count_frame = pandas.DataFrame(isbn_count_list)
```

Get the list of top-15 books

```
top_15_isbn = top_15_isbn.index.values
```

Create a dataframe from top isbn

```
top_books = pandas.DataFrame(top_15_isbn,columns=['ISBN'])
```

Fill the other books details

```
top_books = pandas.merge(top_books,books,on='ISBN')
```

Get details of top books

```
top_books = top_books[['ISBN','Book-Title','Book-Author',
'Publisher']]
```

Show top books

```
top_books
```

	ISBN	Book-Title	Book-Author	Publisher
0	0971880107	Wild Animus	Rich Shapero	Too Far
1	0316666343	The Lovely Bones: A Novel	Alice Sebold	Little Brown
2	0385504209	The Da Vinci Code	Dan Brown	Doubleday
3	0060928336	Divine Secrets of the Ya-Ya Sisterhood: A Novel	Rebecca Wells	Perennial
4	0312195516	The Red Tent (Bestselling Backlist)	Anita Diamant	Picador USA
5	044023722X	A Painted House	John Grisham	Dell Publishing Company
6	0142001740	The Secret Life of Bees	Sue Monk Kidd	Penguin Books
7	067976402X	Snow Falling on Cedars	David Guterson	Vintage Books USA
8	0671027360	Angels & Demons	Dan Brown	Pocket Star
9	0446672211	Where the Heart Is (Oprah's Book Club (Paperba...	Billie Letts	Warner Books
10	059035342X	Harry Potter and the Sorcerer's Stone (Harry P...	J. K. Rowling	Arthur A. Levine Books
11	0316601950	The Pilot's Wife : A Novel	Anita Shreve	Back Bay Books
12	0375727345	House of Sand and Fog	Andre Dubus III	Vintage Books
13	044021145X	The Firm	John Grisham	Bantam Dell Publishing Group
14	0452282152	Girl with a Pearl Earring	Tracy Chevalier	Plume Books

Check total number of authors

```
books['Book-Author'].describe()
```

We can see that total 102,042 unique authors are present. In top 15, John Grisham and Dan Brown appeared twice.

11.2 Content Based Recommendation Engine

These recommendation engines are based on features of the items. Items are compared or recommended based on their feature values. Item having similar properties gets higher weights.

We will show an example by using nearest neighbor algorithm. Nearest neighbor algorithm is an unsupervised learning algorithm. It is known as **memory-based system** because it memorizes the items and for any new instances or item recommend the similar item from its memory. So, if the training data is large and cannot be fitted in memory, then these systems are not preferred.

Let's take an example of online mobile shop. It contains different types of mobile with different features. Consider this mobile database

Mobile-ID	Battery Life (In Hrs.)	Camera (In MP)	ROM (In GB)
ACT-23S	48	12	16
ACT-56P	42	20	64
NTK9	60	8	32
LK0	72	4	16
LK3	48	16	32

Now a user started searching for different product. Website first show him top popular mobiles chosen by other users. Now, user decided to put a filter and see the mobile which matches the criteria most.

User has the following requirement

Battery Life (In Hrs.)	Camera (In MP)	ROM (In GB)
55	10	32

Based on user's requirement most suitable mobile is NTK9. Let's understand this concept by using python code.

We will use house dataset. This dataset is created from user responses. This is available in /data folder. This dataset contains following fields

- **ID:** Unique ID of each property
- **Size of House:** Small, Medium, Large, Big
- **Number Of Bathrooms:** Number of Bathrooms available in the house
- **Age of building:** Less than a year, 1-5 Years, 5-10 Years, 10+ Years

- **Number of Fireplace:** 0,1
- **Furnished Status:** Unfurnished, Semi Furnished, Fully Furnished
- **Car Parking:** Yes/No
- **Air Conditioning:** Yes/No
- **Private Garden:** Yes/No
- **Life Available:** Yes/No
- **Area Status:** High, Medium, Low
- **Rent:** Integer

Import libraries that is needed for this recommendation engine.

```
import pandas
import numpy
import sklearn
from sklearn.neighbors import NearestNeighbors
```

Read Survey data

```
house_data = pandas.read_csv('./data/House_Survey.csv')
```

Check initial data

```
house_data.head()
```

To apply any algorithm first we will convert categorical variable to numeric numbers. We can do this with predefined functions in python. But we want to assign weights according to their categories.

```
#Convert Size
house_data.loc[house_data['Size of House'] == 'Small', 'Size
of House'] = 1
house_data.loc[house_data['Size of House'] == 'Medium', 'Size
of House'] = 2
house_data.loc[house_data['Size of House'] == 'Large', 'Size
of House'] = 3
house_data.loc[house_data['Size of House'] == 'Big', 'Size
of House'] = 4
```

We do similar for all the other attributes.

Now, we start by creating nearest neighbor object. We use this algorithm to get the data point which is nearest to the requirement (neighbour =1).

```
nn1 = NearestNeighbors(n_neighbors=1)
```

Fit model

```
nn1.fit(house_data.loc[:,'Size of House':'Area Status'])
```

Now define our requirement in the same format (not taking ID and rent values).

```
requirement = [2,2,3,1,0,2,1,0,0,0,1]
```

Check the most suitable house for this requirement

```
print(nn1.kneighbors(requirement))
```

Output:

```
(array([[ 1.73205081]]), array([[5]], dtype=int64))
```

Checking the value in house_data at index = 5.

```
house_data.loc[[5]]
```

If we want to get top-3 list

```
top3 =
nn1.kneighbors(requirement,n_neighbors=3)[1].tolist()[0]
```

Check top3 house of the test house

```
house_data.loc[top3]
```

11.3 Classification Based Recommendation Engine

In this technique we can use classification methods to recommend items. We have seen different classification techniques (Logistic regression, decision tree, random forest, naïve bayes) in previous chapters. We will build our model by using logistic regression in this section.

The advantage of these recommendation is personalization. They can take data of users, items, user's past purchase history and some other contextual data if present. Based on value of different parameters, model can decide that user will purchase this product or not.

We can enhance the power of these recommendations with rich contextual data. For example, on an online mobile store, contextual data can be hour of the day, season, festival season or not, day of the month, cookies of the user's browser etc.

Consider another example of car website. The website provides free test drive at home to selected users. To participate in this offer, user must fill a form and provides correct details. Suppose these are the fields which users need to answer.

House Location
Employment Status
Previous Car
Number of Active loans
Number of Credit cards
Loan defaulter status
Single or Married
Gender
Family Size

Now, we can build a classifier that classify each user in two categories, which are: Potential buyer or not. If the customer looks like a potential buyer, information should be forwarded to marketing team and they can proceed with the offer. If the customer does not seem to be a potential buyer, there is no point to give him an offer of free home test ride. So, a machine learning classifier seems to be most appropriate in this case. It can use past customer data to build classifier.

To understand classification-based recommendation we will use bank marketing data. This dataset is available publicly and can be downloaded from: http://archive.ics.uci.edu/ml/datasets/Bank+Marketing. This dataset is based on marketing calls of Portuguese banking institution.

Import additional libraries

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import SMOTE
```

Read the bank data in pandas dataframe

```
bank_data = pandas.read_csv('.\\data\\bank_data.csv',sep=';')
```

Show all columns

```
pandas.set_option('display.max_columns', None)
```

Check the head of the dataframe

```
bank_data.head()
```

In this task we will use only few attribute to build our first classification model.

```
bank_data =
bank_data[['age','job','marital', 'education', 'default',
'housing', 'loan','contact','y']]
```

Convert categories to 0 and 1

```
bank_data.y.replace(('yes', 'no'), (1, 0), inplace=True)
```

Check the shape of the data

```
bank_data.shape
```

Extract features form the dataset

```
X = bank_data.iloc[:, :7]
X = pandas.get_dummies(bank_data.iloc[:, :7]).values
```

Extract class

```
Y = bank_data.iloc[:, 8:].values
```

Split the data in training and test set

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.10)
```

Create object of logistic regression

```
logistic_regression = LogisticRegression()
```

Over sampling using SMOTE algorithm

```
smote_object = SMOTE(random_state=4)
X_res, y_res = smote_object.fit_sample(X_train, Y_train)
```

Fit the model

```
logistic_regression.fit(X_res,y_res)
```

Predict using logistic regression model

```
Y_pred = logistic_regression.predict(X_test)
```

Calculate Accuracy

```
accuracy_score(Y_test,Y_pred)
```

Accuracy of this model is less. But this was only the demonstration to use classification in recommenders. We can use other improvement like more features, feature engineering, better algorithms, parameter tuning to get better results.

11.4 Collaborative Filtering

Collaborative filtering is based on the idea of similar interest. Take an example of two users Ankita and Neha. Ankita likes product 1,3,4 and Neha likes products 3,4,7. As both are agreed on two products, it is highly that Ankita will like product

7 and Neha will like product 1. This technique is completely based on past behavio of users. This is most widely used technique in recommendation engine building

There are two types of collaborative filtering exists:

- **User-User collaborative filtering:** based on similar users, based on their item purchase.
- **Item-Item collaborative filtering:** based on similar items, based on purchased by users.

In this section we will focus on item-item collaborative filtering. We will use Pearson correlation coefficient to recommend item to the user based on his past item selection. Correlation between items is calculated to understand their similarity based on user reviews.

Pearson correlation coefficient is the measure of correlation between two variables It is represented by r.

r = 1: Strong positive correlation

r = –1: Strong negative correlation

r = 0: No correlation

To demonstrate collaborative filtering we will use MovieLens dataset provided by grouplens for research. This dataset can be downloaded from: https://grouplens.org/datasets/movielens/.

```
movies = pandas.read_csv('./data/grouplens_movies.csv')
movies_ratings = pandas.read_csv('./data/
grouplens_ratings.csv')
```

Check the head of the dataframe.

```
movies.head()
movies_ratings.head()
```

First, we check the mean rating given to each movie.

```
movies_ratings_Data =
pandas.DataFrame(movies_ratings.groupby('movieId')['rating'].mean())
movies_ratings_Data.head()
```

Let's see how popular each movie is among the users. We can check this by adding one more column count to our movie_rating_Data DataFrame.

```
movies_ratings_Data['Count'] =
pandas.DataFrame(movies_ratings.groupby('movieId')['rating'].count())
movies_ratings_Data.head()
```

Now, we have to build user-item utility matrix. We can do this by using pivot_table() function of pandas.

```
movies_crosstab = pandas.pivot_table(data=movies_ratings,
values='rating', index='userId', columns='movieId')
movies_crosstab.head()
```

Above crosstab is full of null values. This is because each user rated very less movies. Hence, the above matrix is sparse.

```
#Let's take an example of movie with movieid=260
movies.loc[movies['movieId'] == 260]
```

This movie is Star Wars IV. Create a variable and remove null values.

```
star_wars = movies_crosstab[260]
star_wars = star_wars[star_wars>=0]
star_wars
```

Now our objective is to find similar movies. We will understand this by finding movies similar to Star Wars.

```
similar = movies_crosstab.corrwith(star_wars)
similar_dframe =
pandas.DataFrame(similar,columns=['PearsonR'])
```

Drop NA values

```
similar_dframe.dropna(inplace=True)
similar_dframe.head()
```

Above code shows each movie id and their correlation with star_wars. This is not very useful at this moment.

We also want to check for popular movies only which are rated by atleast 50 users. To do this let's join our frame with rating.

```
similar_dframe_summary =
similar_dframe.join(movies_ratings_Data['Count'])
similar_dframe_summary =
similar_dframe_summary[similar_dframe_summary['Count']>=50].
sort_values('PearsonR', ascending=False).head(5)
```

Get top 5 similar movies

```
top_5_similar =
similar_dframe_summary.head(5).index.values.tolist()
```

Print top 10 similar movies to Star Wars

```
movies.loc[movies['movieId'].isin(top_5_similar)]
```

Output:

	movieId	title	genres
232	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi
953	1196	Star Wars: Episode V - The Empire Strikes Back...	Action|Adventure|Sci-Fi
966	1210	Star Wars: Episode VI - Return of the Jedi (1983)	Action|Adventure|Sci-Fi
6944	59315	Iron Man (2008)	Action|Adventure|Sci-Fi
7218	68358	Star Trek (2009)	Action|Adventure|Sci-Fi|IMAX

Above result is very interesting. First record is the movie itself. But second and third are the next part of similar movies. So, if a user likes Star Wars IV, we can recommend him another two movies in the same series. These results are self-explainable. This is also a reason why collaborative filtering method are very popular in recommendation engine techniques.