Project Report:

Restaurant Recommendation System

T.Sai Seetharamaiah

Abstract

A Collaborative Filtering was successfully implemented using the computer programming language python to produce recommendations between users such that only similar users were considered. Both SVD and general matrix Factorization Techniques are implemented.

Contents

1. Introduction:

Since the arrival of the Internet, the amount of information that users are confronted with on a regular basis has expanded dramatically. Making it increasingly more difficult for a user to differentiate between relevant and irrelevant data. This phenomena is generally referred to as information overload. To help overcome this problem, different information filtering mechanisms have been devised. One such mechanism is Collaborative Filtering (CF). CF is a technique used to alleviate information overload by identifying which items a user will find worthwhile [1].

The rational on which collaborative filtering is based is the idea that people often get the best recommendations from someone with similar tastes to themselves. As such, CF explores techniques for matching people with similar interests and making recommendations on this basis. This project incorporates CF to implement a recommender system in the Restaurants domain. Using the provided system framework, a selection of core methods required to build a collaborative filtering system were implemented.

The remainder of this report discusses the theory and experiments carried out in order to build a functions of collaborative filtering system. Section 2 describes what Collaborative Filtering is. Section 3 details the theory behind the Gradient Descent Optimization. Section 4, 5 and 6 shows the implementation details. Finally, Section 7 concludes this report with an overview of the main points and findings of the project.

3. Collaborative Filtering Schemes:

Standard collaborative filtering methods use the feedback provided by users, to a subset of products/services on offer, to make relevant predictions on the remaining; unlike content-based methods, that rely on domain knowledge and explicit characterization of users and items. The feedback is usually captured in the form of a rating matrix – a matrix of partially filled values wherein each available entry (rating value) is a measure of the affinity of a user (along rows) for an item (along columns). Feedback or the ratings can be of two types - the first one is implicit, i.e. the prediction needs to be made based on implicitly gathered data such as browsing history or buying pattern of users; the user does not explicitly specify if he/she likes the item or not. The second type of rating is explicit, for example, movie ratings on a 5-point scale, or the 'like' option in YouTube. The following Figure gives examples of the explicit rating matrix. Implicit ratings are easier to obtain but are not very informative; especially for negative feedbacks. For example, if a person does not buy a product one does not know if he/she does not like it or does not know about it. On the other hand, explicit ratings are more dependable and thus, more widely used in recommender system (RS) design. However, as they involve the active participation of the user, they are hard to capture and thus, extremely limited. Despite the associated (data) scarcity, most recommender system design schemes use the explicit rating information to ensure more reliable predictions.

	U1	U2	U3	U4	U5
R1	5	1	?	1	3
R2	0	1	?	4	5
R3	?	3	?	?	2
R4	2	4	?	?	3
R5	?	5	?	5	?

Fig. Examples of (Explicit) Rating Matrix

CF models [12] are built on the assumption that if two users rate a few items similarly, they will most probably rate others also in similar fashion. This belief can be exploited in two ways – neighborhood based approach and latent factor model (LFM) based approach. LFMs are the most widely adopted

3.1. Matrix Factorization Framework for Latent Factor Model:

	$\theta_0^{(1)}$	$\theta_0^{(2)}$	$\theta_0^{(3)}$	$\theta_0^{(4)}$	$\theta_0^{(5)}$	$\theta_0^{(6)}$	$\theta_0^{(7)}$		
	$\theta_1^{(1)}$	$\theta_1^{(2)}$	$\theta_1^{(3)}$	$\theta_1^{(4)}$	$\theta_1^{(5)}$	$\theta_1^{(6)}$	$\theta_1^{(7)}$		
Restaurants	**Active user**	**user1**	**user2**	**user3**	**user4**	**user5**	**user6**	**Feature1 (x0)**	**Feature2 (x1)**
Restaurant_ 1	2	1	4	?	1	4	1	$x_0^{(1)}$	$x_1^{(1)}$
Restaurant_ 2	1	2	5	?	2	5	2	$x_0^{(2)}$	$x_1^{(2)}$
Restaurant_ 3	?	3	?	?	3	?	3	$x_0^{(3)}$	$x_1^{(3)}$
Restaurant_ 4	?	4	?	?	4	?	4	$x_0^{(4)}$	$x_1^{(4)}$
Restaurant_ 5	4	5	1	?	5	1	?	$x_0^{(5)}$	$x_1^{(5)}$
Restaurant_ 6	?	?	?	?	3	?	2	$x_0^{(6)}$	$x_1^{(6)}$
Restaurant_ 7	?	1	3	?	1	3	1	$x_0^{(7)}$	$x_1^{(7)}$

Fig. Matrix (M) decomposition into its Latent factors

Matrix Factorization (MF), is one of the most widely adopted means of recovering the latent factor representation of users and items owing to its performance and scalability. MF considers the task of estimation of the latent factor vectors of both users and items as a regularized optimization problem

Objective: $\min\left(J\left(x^{(1)}, x^{(2)}, x^{(3)}, \dots x^{(n_r)}, \theta^{(1)}, \theta^{(2)}, \theta^{(3)}, \dots . \theta^{(n_u)}\right) \right)$

Cost Function: $\boldsymbol{J\left(x^{(1)}, x^{(2)}, x^{(3)}, \dots x^{(n_r)}, \theta^{(1)}, \theta^{(2)}, \theta^{(3)}, \dots . \theta^{(n_u)}\right)}$

$$= \mathbf{1/2}\left(\sum_{(i,j):r(i,j)=1} (\boldsymbol{\theta^{(j)^T} x^{(i)}} - \boldsymbol{y^{(i,j)}})^2 + \frac{\boldsymbol{\lambda}}{\mathbf{2}} \sum_{i=1}^{n_r} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\boldsymbol{\lambda}}{\mathbf{2}} \sum_{j=1}^{n_r} \sum_{k=1}^{n} (\theta_k^{(j)})^2 \right)$$

Where $\begin{bmatrix} x_0{}^{(i)} \\ x_1{}^{(i)} \end{bmatrix} - Feature\ vectors(X)$ $\quad$ $\begin{bmatrix} \theta_0{}^{(j)} \\ \theta_1{}^{(j)} \end{bmatrix} - user\ profile\ (\theta)$

$X \epsilon \mathbb{R}^{(n_r \times n_f)}$ *and* $\theta \epsilon \mathbb{R}^{(n_u \times n_f)}$

$n_u - number\ of\ users\ n_f - number\ of\ features\ , n_r - number\ of\ restaurants$

Once, the latent factor vectors for all users and items are recovered, the filled matrix of interaction values can be computed as $\theta^t * X = M$. Although cost function is bilinear, the two variables (θ and X) are separable and hence, each of them can be efficiently solved. Commonly used approach for solving the MF formulation are Stochastic Gradient Descent (SGD).

Using the Stochastic Gradient Descent Algorithm:

Update the parameters in each iteration in order to minimize the cost. The updated parameters are:

$$\boldsymbol{x_k^{(i)} = x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} (\theta^{(j)^T} x^{(i)} - y^{(i,j)})\ \theta_k{}^{(j)} + \lambda x_k^{(i)} \right)}$$

$$\boldsymbol{\theta_k^{(j)} = \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} (\theta^{(j)^T} x^{(i)} - y^{(i,j)})\ x_k{}^{(i)} + \lambda \theta_k^{(j)} \right)}$$

Where

$$\alpha - learning\ rate \quad and \quad \lambda - regularization\ parameter$$

Business Problem:

Now a days, almost everyone had access to internet. So, I decided to start a website and want to provide location services to users like Foursquare and Google Maps. My revenue is directly proportional to number of users in my website. There are particularly two types of customers to my site. One such type is given below as example

1. Customers looking for Food services : john
2. Customers who are providing food services such as people running restaurants in this case.

John is a foodie and also an active user of my website. He is currently in Noida city. John is always facing problem to decide which nearby restaurant is best suitable for him. So, we have to provide solution to him by recommending k-most likelihood restaurants near him. There are so many food services provided in a particular location which makes john so confused and difficult to choose and to visit a restaurant.

AIM: To increase number of users by providing Recommendations

Targeted USERS:

- Location providers : Restaurants
- People looking for Restaurants

PROBLEMS faced by:

- Restaurants: for a new restaurant, it takes time to get popular among people. Spends lot of investment in advertising to show their existence
- People: Because of so many choices, they get always confused to choose restaurants

4. Analytic Approach:

Traditionally, there are two methods to construct a recommender system:

- Content-based recommendation
- Collaborative Filtering(_we are following_)

The first one analyzes the nature of each restaurant. For instance, recommending restaurants to a user from the hand-picked features of each restaurant. Collaborative Filtering, on the other hand, does not require any information about the restaurants or the users themselves. It recommends restaurants based on users' past behavior.

For instance taking john's problem,

Step-1: From venue's data, we will collect all the users who rated that venue and their respective ra tings as well

Step-2: We will build the data set of users, restaurants and their collective respective ratings

Step-3: After building the dataset, split the data set into training and testing datasets

Step-4: we will apply the machine learning model or Deep Learning model on the training dataset

Step-5: Evaluation is done using testing dataset

Step-6: Repeating the above 2 steps for different Depp Learning and Machine Learning models

5. Data:

Requirements:

1. venues(Restaurants) data - collected from Search Query for a specific venue category https://api.foursquare.com/v2/venues/search?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&ll=LATITUDE,LONGITUDE&v=VERSION&query=QUERY&radius=RADIUS&limit=LIMIT

2. Because of unavailable Users data...Users rating for the venues will generated randomly

Preparation:

Restaurants data is collected Using the Foursquare developer's API. User Location is pre-determined (NOIDA SECTOR 62, IN). Restaurants data in Noida city is collected by exploring the city using explore API call

1. Search for a specific venue category https://api.foursquare.com/v2/venues/search?client_id=CLIENT_ID&client_set=CLIENT_SECRET&ll=LATITUDE,LONGITUDE&v=VERSION&query=QUERY&radius=RADIS&limit=LIMIT

After cleaning data, Final Restaurant dataset is prepared

Note: Due to unavailable rated restaurant user's dataset, I am considering 10 random users and their ratings are also considered random. Objective of my Project is to build recommender system to users using different Machine Learning and Deep Learning models. So, in this case users' data is considered randomly

Now, we have both user and restaurants datasets. Merge them together and make restaurant names as row index

Matrix Factorization**:** It is a representation of a matrix into a product of matrices. There are many different matrix factorization and each used for different class of problems.

Final Datset Preparation

In [58]:
```
user_ratings['venues'] = Restaraunts['name']
df = user_ratings.set_index('venues')

df
```

Out[58]:

	Charles	David	George	James	Johann	John	Paul	Richard	Robert	William
venues										
Dev's Restaurant & Bar	1	0	0	0	0	0	0	1	0	0
shakuntalam sweets namkeen & restaurant	0	5	5	0	0	0	0	3	0	1
The Yellow Chilli Restaurant	0	0	0	5	1	0	0	0	0	0
Khidmat Restaurant	2	0	0	0	0	0	0	0	4	0
Bonanza Restaurant Vaishali sec 4	2	0	0	4	2	0	5	0	0	1
1 The Restaurant	0	0	0	0	0	0	0	0	3	1
FEZ Restaurant	0	2	0	0	4	0	3	1	0	0
Galords Restaurant	0	0	0	0	5	0	0	0	0	2
Green Leaf Restaurant	0	0	0	3	0	0	0	0	0	0
Haveli Restaurant	4	0	0	0	0	0	0	3	0	0
The Flame Restaurant	4	0	0	0	0	0	0	0	0	3
Niwala Restaurant	0	1	0	0	1	0	0	0	4	0
Berco's Garden Restaurant	0	4	0	0	0	0	0	0	0	1
Chauhan Vatika Restaurant	2	2	2	0	0	0	0	0	0	5
Gulati Restaurant	1	0	2	0	0	0	0	0	1	0
mehfill restaurant	0	0	0	0	0	0	5	0	0	4
kavita Restaurant	4	1	4	0	2	0	1	0	3	0

Fig. Dataset

6. Modelling:

There are five steps involved in this model:

1. STEP-1: Defining the cost and gradient functions
2. STEP-2: Setting Initial parameters
3. STEP-3: Apply Gradient Descent Algorithm
4. STEP-4: Reconstructing the Rating Matrix
5. STEP-5: Recommendation of TOP 5 Restaurants

STEP-1: Defining the cost and gradient functions:

```
def cofiCostFunc(params, *args):
    X = np.reshape(params[0:num_rest*num_features], (num_rest, num_features))
    Theta = np.reshape(params[num_rest*num_features:], (num_users, num_features))
    error = 0
    pred_values = np.around(X@Theta.T,2)

    pred_values_rated = np.multiply(pred_values, R)
    Y_rated = np.multiply(Y,R)
    difference = pred_values - Y
    difference_rated = np.multiply(difference,R)
    error = sum(sum(np.multiply(difference_rated,difference_rated)))

    J = (1/2)*(error)
    J = J + (   (lamda/2)*( np.sum(np.square(Theta))  +  np.sum(np.square(X))  )    )

    return J
```

Fig. cost function implementation in python

```
def grad_fun(params, *args):
    X = np.reshape(params[0:num_rest*num_features], (num_rest, num_features))
    Theta = np.reshape(params[num_rest*num_features:], (num_users, num_features))

    X_grad = np.zeros(X.shape);
    Theta_grad = np.zeros(Theta.shape);

    for i in range(num_rest):
        idx = np.where(R[i]==1)[0]
        Thetatemp = Theta[idx]
        Ytemp = Y[i][idx]
        X_grad[i] = ( ((X[i]@np.transpose(Thetatemp))-Ytemp)@Thetatemp  + lamda*X[i]   )

    for i in range(num_users):
        idx = np.where(R[i]==1)[0]
        Xtemp = X[idx]
        Ytemp = Y[idx,[i]]
        Theta_grad[i] = (  ((Theta[i]@np.transpose(Xtemp))-np.transpose(Ytemp))@Xtemp   + lamda*Theta[i]   )


    grad = np.reshape(np.concatenate((X_grad,Theta_grad), axis=0),(1,(num_rest+num_users)*num_features))[0]

    return grad
```

Fig. gradient function implementation in python

STEP-2: Setting Initial parameters:

Setting Intital weights and required parameters

```
# use values
num_users = Y.shape[1]
num_rest = Y.shape[0]
num_features = 5;


# Set Initial Parameters (Theta, X)
from random import *
from scipy import special, optimize
# from optimize import fmin_cg

X = np.zeros((num_rest, num_features))
Theta = np.zeros((num_users, num_features))

for i in range(num_rest):
    for j in range(num_features):
        X[i][j] = round(uniform(-1, 1),2)


for i in range(num_users):
    for j in range(num_features):
        Theta[i][j] = round(uniform(-1, 1),2)
```

STEP-3: Applying Gradient Descent Optimization:

In [65]:
```
# normalising the ratings
Ynorm, Ymean = normalizeRatings(Y, R)
```

Using fmin_cg function for OPTIMIZATION problem

In [66]:
```
initial_parameters = np.reshape(np.concatenate((X,Theta), axis=0),(1,(num_rest+num_users)*num_features))[0]


# Set Regularization
lamda = 10

args = (Y, R, num_users, num_rest, num_features, lamda)
theta = optimize.fmin_cg ( cofiCostFunc, initial_parameters, fprime=grad_fun ,args=args)
```

```
Warning: Desired error not necessarily achieved due to precision loss.
         Current function value: 407.085979
         Iterations: 5
         Function evaluations: 62
         Gradient evaluations: 56
```

STEP-4: Predicting ratings and reconstruction:

STEP - 4 : Predicting User Ratings

```
users_intrst_pred = np.reshape(theta[0:num_rest*num_features], (num_rest, num_features))
movie_features_pred = np.reshape(theta[num_rest*num_features:], (num_users, num_features))

predictions = (users_intrst_pred@movie_features_pred.T)+Ymean

predictions = np.around(predictions,1)
predictions
```

Step5: Recommendation of TOP 5 Restaurants:

STEP - 5: Recommendations

```
print('Top 5 recommended Restaraunts to Jhon with predicted ratings: ',end='\n\n')
for i in range(5):
    venue_name = venue_names[top_5_list[i]]
    user_rating = user_not_rated_move_rating[0][top_5_list[i]]

    print("{0:50} {1}".format(venue_name, user_rating))
```

```
Top 5 recommended Restaraunts to Jhon with predicted ratings:

mehfill restaurant                                 4.5
Galords Restaurant                                 3.5
shakuntalam sweets namkeen & restaurant            3.5
Anand Restaurant                                   3.3
The Yellow Chilli Restaurant                       3.0
```

MATRIX FACTORIZATION: Singular Value Decomposition

Instead of factorizing matrix using random vectors, SVD technique decomposes the matrix with a reduced number of singular values that can closely approximate the original matrix (M).

Decomposition:

$$M = U\Sigma V^T$$

Where

$$U - (n_r \times n_r)\ Unitary\ matrix. (left\ singular\ vector)$$

$$\Sigma - (n_r \times n_u)\ diagonal\ matrix\ with\ non\ negative\ real\ numbers$$

$$V - (n_u \times n_u)\ Unitary\ matrix. (right\ singular\ vector)$$

U is considered as Feature matrix (X)

ΣV^T - combinely considered as User profile matrix (Theta).

After this Optimization algorithms are applied as above.

IMPLEMENTATION in python:

Decomposing Rating Matrix Using SVD

STEP - 2: setting initial parameters

```
P, D, Q = np.linalg.svd(Y, full_matrices=False)

# parameters
num_features = Q.shape[1]
num_users = Y.shape[1]
num_rest = Y.shape[0]
Theta = np.diag(D)@Q
X=P
```

7. CONCLUSION:

Recommendation System is implemented using Collaborative Filtering Technique.

Because for each restaurant, every feature estimation is time consuming and may not be available at all sometimes, So, Collaborative filtering is used here which do not require any prior knowledge on Features.

8. References

[1] Borchers, Al., Herlocker, Jon., Konstan, Joseph & Riedl, John. (April 1998). “Ganging up on Information Overload”. *Computer (IEEE Computer Society Press)* 31 (4): 106108.