CS771: Group 1
Online Recommender System

Akshay Kumar
kakshay@iitk.ac.in
Dept. of CSE

Arpit Jain
arpitj@iitk.ac.in
Dept. of CSE

Dhananjay Sharma
djsharma@iitk.ac.in
Dept. of CSE

Indian Institute of Technology, Kanpur

Project Report
November 12, 2013

Abstract

Online Recommender System are ubiquitous in present world. It is used by a wide array of companies like Netflix, Amazon, etc. to suggest products to their users. In this project, we have tried to build a recommender system which predicts the rating which can be given by a user. We started off with implementing already existing algorithms and then improvised it a bit and analyzed the accracies offered by each of the algorithm.

Description of Problem Statement

As already stated, the main problem is to predict the rating given by a user to a movie. The `MovieLens` database consists of 100000 ratings given by different users. A total of 1682 movies have been rated by 943 movies. Each user has rated at least 20 movies. Additionally, we are also provided with different attributes of movies and users. Specifically speaking, for a movie, we are provided with its title, different genres a movie falls, its release date, video release data and IMDB URL. For a user, we are provided with his/her demographics information such as their age, gender, occupation and zip code. Finally each user has rates some movies (atleast 20) and we are provided the rating and the timestamp.

Approach

The most naive approach is to pose it as a classification problem. Classification can be in any of the classes from 1 to 5 if a movie has been rated by the user. If we have a *.arff* file with different feature vector for each rating, then this problem can be posed as a classification problem. For any rating, their exists a corresponding movie and user. So what we did was to replace movie and users by their respective attribute and classify. The whole data was split into 80% training data and 20% test data. The results are discussed in the subsequent section.

Existing Algorithm

There are primarily two approaches followed in movie recommender systems:

1. Collaborative Filtering
2. Content Based Filtering

Content Based Filtering predicts movie rating by a user based on his previous ratings. It actually doesn't lay any accord on the connections between a user. Hence, it is efficient to predict ratings for a new user. Hence, companies adopting Content Based Ratings generally ask to new users to fill a form to know their tastes. On the the other hand, Collaborative Filtering predicts a rating based on the simillarity between two users. Similar users are goins to rate movies similarly.

Mathematics of the two algorithm

The algorithms implemented by us are the ones described by Andrew Ng in Machine Learning course on Coursera.

Coming to the notations used,

$$u : \text{Set of users} \quad |u| = n_u$$

$$m : \text{Set of movies} \quad |m| = n_m$$

$$r_{(i,j)} = \begin{cases} 1 & \text{if user } j \text{ has rated movie } i \\ 0 & \text{otherwise} \end{cases}$$

$$y_{(i,j)} = \text{rating} \quad \text{if} \quad r_{(i,j)} = 1$$

Consider the following example,

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

Suppose the genre of each movie is also given (a valid assumption as it's already been provided in the movielens database). Let it be as follows:

Movie	x_1(romance)	x_2(action)
Love at last	0.9	0
Romance forever	1.0	0.01
Cute puppies of love	0.99	0
Nonstop car chases	0.1	1.0
Swords vs. karate	0	0.9

We also define the following parameters denoting the attributes of users and movies. Corresponding to each user j, we define a parameter $\theta^{(j)}$. Similarly, define a parameter $x^{(i)}$ for each movie i.

$$\theta^{(j)} = \text{parameter vector for user } j$$

$$x^{(i)} = \text{parameter vector for movie } i$$

For user j, movie i, predicted rating would then be $(\theta^{(j)})^T x^{(i)}$.

The error function for any predicted rating is $\left((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}\right)^2$. Hence the objective funcion to be minimized is:

$$\min_{\theta^{(j)}} \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}\right)^2$$

$$+\frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} \left(\theta_k^{(j)}\right)^2$$

The last term is the penalty term for high value coefficients. Half is added for simplicity. To learn $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)},\theta^{(2)},\cdots,\theta^{(n_u)}} \frac{1}{2}\sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}\right)^2$$

$$+\frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n} \left(\theta_k^{(j)}\right)^2$$

This can be done by using Gradient Descent Algorithm:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}\right)x_k^{(i)} \ (k=0)$$

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

$$(k \neq 0)$$

α is the learning rate. The results are discussed in the results section.

Suppose instead of genre of different movies, we are provided with tastes of different users.

$$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}; \theta^{(2)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}; \theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}; \theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix};$$

Here, given $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$, we need to learn $x^{(i)}$:

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

Summing it overall all $x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}$, the objective function becomes:

$$\min_{x^{(1)},x^{(2)},\cdots,x^{(n_m)}} \frac{1}{2}\sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}\right)^2$$

$$+\frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n} \left(x_k^{(i)}\right)^2$$

We tested Content Based Filtering on `movielens` database the results of which are discussed in the results section. Since users have very few attributes known a proiri, the second objective function was not minimized

Low Rank Matrix Factorization

The discussion is continued from the previous section.

Note that we now have two different problems. Their overall structure is as follows:

1. Given $x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}$, predict $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$
2. Given $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$, predict $x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}$

So, one plausible approach can be to find both θ's and x's simultaneously. Start off with some random θ and update x to miniimze the error. Based on this x, update θ to minimize error and keep doing so.

$$\theta \to x \to \theta \to x \to \theta \to x \to \theta \to x \to \theta \to x \cdots$$

Another approach is to change the objective function to find both θ's and x's. The updated objective function is:

$$J(x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}, \theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}) =$$
$$\frac{1}{2} \sum_{(i,j):r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 +$$
$$\frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(\theta_k^{(j)} \right)^2$$

The last two terms are simple the penalties of the two objective functions. The first term is formed by coalescing the first term of the wo objective function thereby removing the restriction of i or j.

We need to minimize J *i.e*

$$\min_{\substack{x^{(1)},x^{(2)},\cdots,x^{(n_m)} \\ \theta^{(1)},\theta^{(2)},\cdots,\theta^{(n_u)}}} J(x^{(1)}, \cdots, x^{(n_m)}, \theta^{(1)}, \cdots, \theta^{(n_u)})$$

One way to do it is via Gradient Descent Algorithm. Update θ and x simultaneously.

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

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

where α is the learning rate.

This is a novel algorithm yet mysteriously simple algorithm used used in Movie Recommendation Engines. It was first demonstrated in nexflix challenge that matrix factorization models are superior of classical nearest-neighbour technoques. This methods allows incorporation of additional information such as feedback, temporal effects and confidence levels.

We will now try to express the same algorithm in a slightly different way. Given a set of users and a set of movies, the partial information about the ratings can be stored in a matrix whose $(i,j)^{th}$ entry corresponds to the rating of i^{th} movie by j^{th} user. Matrix Factorization exploits the fact that there should be some latent features that determine how a user rates an item.

With a set U of users and D of items, let $R : U, D \to U \times D$ be the rating matrix. Our task is to find two matrices $P(K \times |U|)$ and $Q(K \times |D|)$ such that their product approximates R.

$$R \approx P^T \times Q = \hat{R}$$

Here K is the number of features chosen. Note that this is exactly the same as what we have already analyzed except for the fact that vectors have now been embedded into a matrix.

The rating corresponding to u_i and d_j would be:

$$\hat{r_{ij}} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

The error term defined here is:

$$e_{ij}^2 = (r_{ij} - \hat{r_{ij}})^2 = (r_{ij} - \sum_{k=1}^{k} p_{ik} q_{kj})^2$$

The total error would then be,

$$E = \sum_{(i,j) \in \kappa} e_{ij}^2 = \sum_{(i,j) \in \kappa} (r_{ij} - \hat{r_{ij}})^2$$

$$= \sum_{(i,j) \in \kappa} (r_{ij} - p_i^T q_j)^2 + \lambda(\|p_i\|^2 + \|q_j\|^2)$$

where κ is the set of pairs of movies and users which have been rated. Minimizing the error, the gradient descent update rule will then be:

$$p_i \leftarrow p_i + \alpha(e_{ij} q_j - \lambda p_i)$$

$$q_j \leftarrow q_j + \alpha(e_{ij} p_i - \lambda q_j)$$

The process can be stopped when the total error E becomes less than a certain threshold.

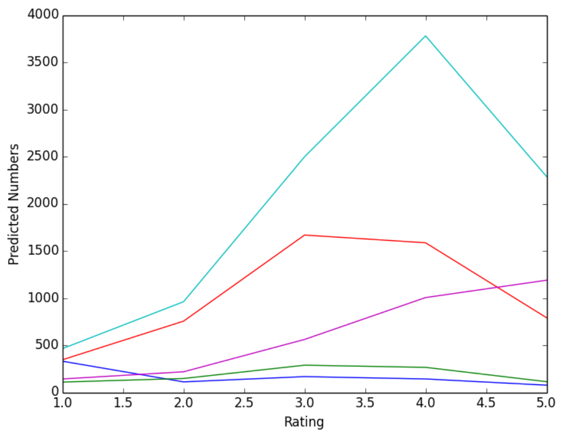

Figure 1: A graph showing th number of predicted classes for different ratings for a simple **classifier**. The lines corresponding to ratings of $1, 2, 3, 4, 5$ are, respectively, dark blue, green, red, light blue and pink. The test dataset consisted of a total of 20000 data points. The number of points in classes $1, 2, 3, 4, 5$ were $1391, 2192, 5182, 6778, 4457$ respectively.

Results

We firstly experimented using a Simple classifier. The graph here shows the results obtained: As evident from the figure 1, the number of predicted ratings peak at the point of actual rating. For example, blue line having an actual rating of 4 peaks at the rating of 4. Similarly, pink line of actual rating 5 peaks at 5 and so on. Actually, most of the data point was concentrated around a region of actual rating plus or minus one.

The number of correctly classified instance for this classifier was very low *viz.* 35.57%. One prime reason was the fact that a classifier assumes it to be a classification problem whereas it actually is a rating prediction problem. Hence, nearly correct predicted ratings were deemed incorrect in this model.

Coming to the Content Based Filtering and Low Rank Matrix Factorization, the results were fairly accurate. The plotted histograms show the number of predicted users for different actual ratings *viz.* $1, 2, 3, 4, 5$. As shown in the histogram, the predicted rating peaks at the actual rating most of the time. The error in this phase was only very low. We reported an error of approximately 18.11% *i.e.* an accuracy of more than 81% in this method.

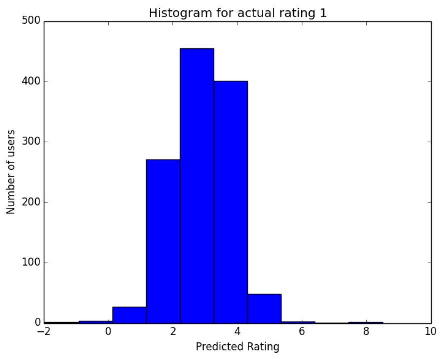

Figure 2: Histogram for actual rating 1

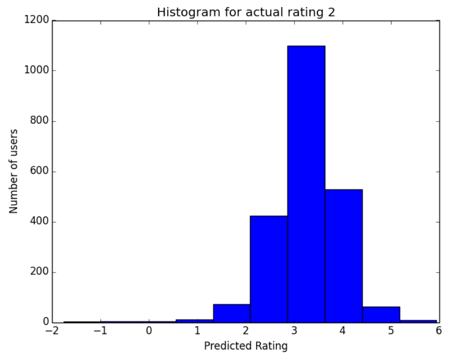

Figure 3: Histogram for actual rating 2

The error was calculated as follows :– the Content Filtering Based Linear Regression model predicts a rating which is a real number, say $\hat{r}$. Suppose the actual rating is r. Then the error defined for this prediction is simpy $|r - \hat{r}|$. This is summed up over all the test instances and then the average value of this error is taken. It turned out to be nearly 18.11% which means on an average, the predicted value deviates by 18.11% from the actual value.

This method lacked one major feature : inability to predict the ratings for new users because it doesn't lay any accord on user-user interaction or the similarity between two users. To mitigate this problem, we moved on to the method of Low Rank Matrix Factorization. Note that by using this method, the accuracy is not improved consideraly but now we are able to predict ratings for a entirely new user.

The results for this method have been omitted due to 2 reasons:

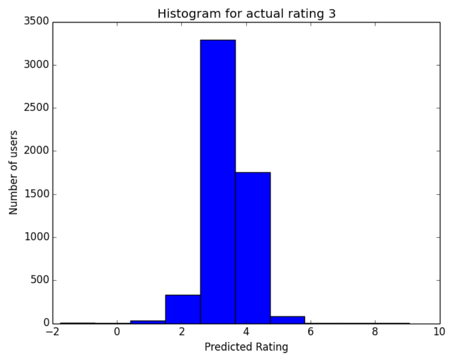

Figure 4: Histogram for actual rating 3

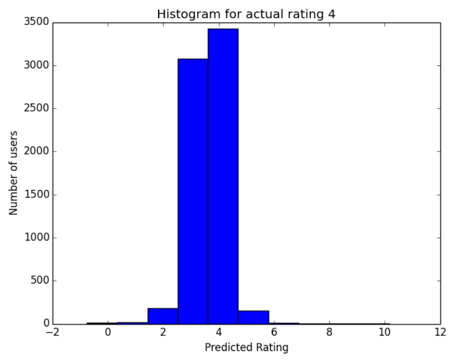

Figure 5: Histogram for actual rating 4

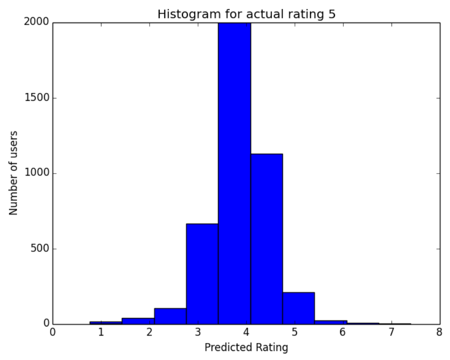

Figure 6: Histogram for actual rating 5

1. This method didn't give considerable edge over accuracy. For exisiting users who have already rated some of the movies, the results obtained were more or less similar to the ones obtained in Content Based Filtering.

2. Training the classifier took a considerably large amount of time. Once trained, we saved this classifier as an object. Hence, we were not able to train to multiple times varying different parameters so as to obtain the best possible result.

Shortcomings and Future Work

Here are a few points that can be looked into:

- Value of K for feature matrix. In the Low Rank Matrix Factorization method, this K is chosen arbitrarily. This has to be fixed somehow so as to reflect different attributes/features of movies and users.

- Prone to extreme ratings. Users with similar tastes can rate movies differently (one may be lenient whereas the other is strict). So our, our method doesn't account for this. Actually, a bias term μ can be introduced to keep a tab on this.

Platform Used

For almost all the Meachine Learning aspect of the project, we used `weka` barring one part where we used python for building a feature matrix. We used python for building all the relavant *.arff* files. For making different graphs and histograms, we again used python.

References

1. Video Lecture on Mideo Recommender Systems in Machine Learning Course on Coursera by Andrew Ng

2. Matrix Factorization Techniques For Recommender Systems, *Yehuda Koren, Robert Bell, Chris Volinsky*

3. Algorithms and Methods in Recommender Systems, *Daniar Asanov*