09.02 - Recommender Systems

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	?

Figure 1: Movie ratings table

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

Figure 2: Movie ratings table with features

1 Problem Formulation

Example: Predicting movie ratings.

Imagine a company where users rate movies using zero to five stars.

Notation:

- n_u = number of users
- n_m = number of movies
- $r(i, j) = 1$ if user j has rated movie i
- $y^{(i,j)}$ = rating given by user j to movie i (defined only if $r(i, j) = 1$)

The problem is: given this dataset, to look through all the missing ratings on the dataset and fill them (fig. 1).

2 Content Based Recommendations

How do we predict the "?" in the movies table (fig. 1)?

Let's guess that we have two additional features in the table: one that measures the amount of romance a movie has and another one that measures the amount of action (fig. 2).

If we then add the interceptor term $x_0 = 1$, we have a feature vector for each movie ($x^{(1)} = [1, 0.9, 0]^T$, for example).

The idea is to use linear regression for each different user and predict the missing parameters with that. More formally: for each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^3$. Predict user j as rating movie i with $(\theta^{(j)})^T x^{(i)}$ stars.

For example, in fig. 2, if we wanted to predict "Cute pupies of love" for "Alice", and we already got θ, we would do: $(\theta^{(1)})^T x^{(3)}$.

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
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	?	?	?

Figure 3: Move ratings with unknown features

2.1 Optimization Objective

We can describe this intention (learn θ for the user j) with the formula:

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

Where $\sum_{i:r(i,j)=1}$ is a for loop with all the rated movies by that user.

To learn all the θ, we can do:

$$\min_{\theta^{(1)},\dots,\theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

Which is very, very similar to the standard linear regression equation (only filtering the unknows votes out).

2.1.1 Gradient descent update

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

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

3 Collaborative Filtering

This algorithm has the characteristic of learning the features it needs by himself.

In the last example (fig. 2) we had some features. Extracting those features is quite time consuming, though.

In this new algorithm, we won't have the features (fig. 3). Let's imagine that the users tells us whether they like romance and action movies (the θ). Then, it's possible to infer the values of x_1 and x_2.

We can exploit that we know the ratings of the users and their preferences to know which value to assign to each movie feature. For example, if a user says that he likes action movies and he rates a movie very high, we can assume that that movie will have action.

3.1 Optimization Algorithm

Given $\theta^{(1)}, \dots, \theta^{(n_u)}$, to learn $x^{(i)}$:

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

Which reads as: choosing the features so that the predicted value will be similar to the actual value that we observe on the rating of the users.

We can then compute this for all the movies. Given $\theta^{(1)}, \dots, \theta^{(n_u)}$, to learn $x^{(1)}, \dots, x^{(n_m)}$:

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

3.1.1 Gradient descend update

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

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

3.2 How to continue

We've seen in content based filtering that given $x^{(1)}, \dots, x^{(n_m)}$ we can estimate $\theta^{(1)}, \dots, \theta^{(n_u)}$.

In collaborative filtering, given $\theta^{(1)}, \dots, \theta^{(n_u)}$, we can estimate $x^{(1)}, \dots, x^{(n_m)}$.

So, this is the egg-chicken problem. What we can do, therefore, is randomly guess θ to learn x (features). We can then use the x to learn θ, etc.

4 Collaborative Filtering Algorithm

We've talked about the ideas of how if you're given features for movies, we can learn parameters θ for users. We've also seen how if we're given parameters for the users we can use that to extract features for the movies. We'll now find an algorithm to solve that problem.

We could to the chicken-egg back and forth optimization algorithm, but there is an algorithm that optimizes both x and θ at the same time, so it's better.

The cost function is:

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

Which is basically merging both cost functions (collaborative and content based filtering) into one, taking advantage that the less squares sumation is the same in both and then adding the respective regularization term.

We also suppress the interceptor term $x_1 = 1$, as the algorithm will learn it by himself if it's really needed.

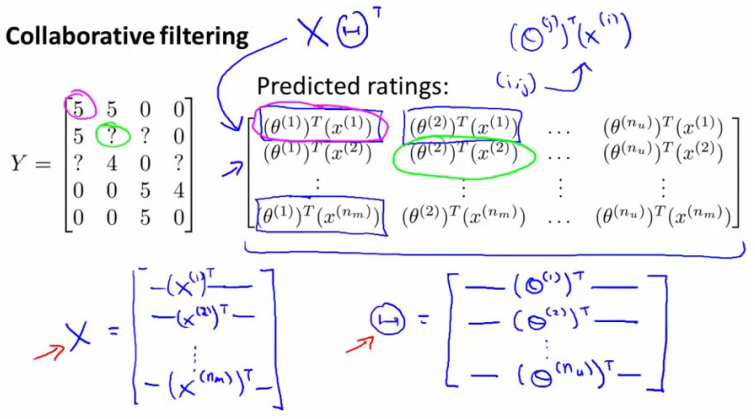

Figure 4: Low rank matrix factorization

4.1 Gradient descent

The same as before for the θ and x respectively.

4.2 Detailed algorithm

We can find a more detailed explanation in algorithm 1.

Algorithm 1 Collaborative filtering algorithm

1: Initialize $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values.
2: Minimize $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$ using gradient descent (or an advanced optimization algorithm).
For every $j = 1, \dots, n_u, i = 1, \dots, n_m$:
$x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$
$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$
3: For a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$.

5 Vectorization: Low Rank Matrix Factorization

We'll see an alternative algorithm for solving the collaborative filtering problem.

Take the dataset we have (fig. 1), and put it directly onto a matrix called Y. Then, we can stack the x vectors and the θ vectors in two matrices, X and Θ. Afterwards, we can compute the prediction of the ratings simply by doing $X\Theta^T$. All this can be seen in fig. 4

5.1 Finding related movies

For each product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$.
How to find 5 movies j most related to movie i?
Find the 5 movies j with the smallest $\left\| x^{(i)} - x^{(j)} \right\|$.