Machine Learning
Week 9
Section 2
Recommender Systems

When you buy a product online, most websites automatically recommend other products that you may like. Recommender systems look at patterns of activities between different users and different products to produce these recommendations. In this module, we introduce recommender algorithms such as the collaborative filtering algorithm and low-rank matrix factorization.

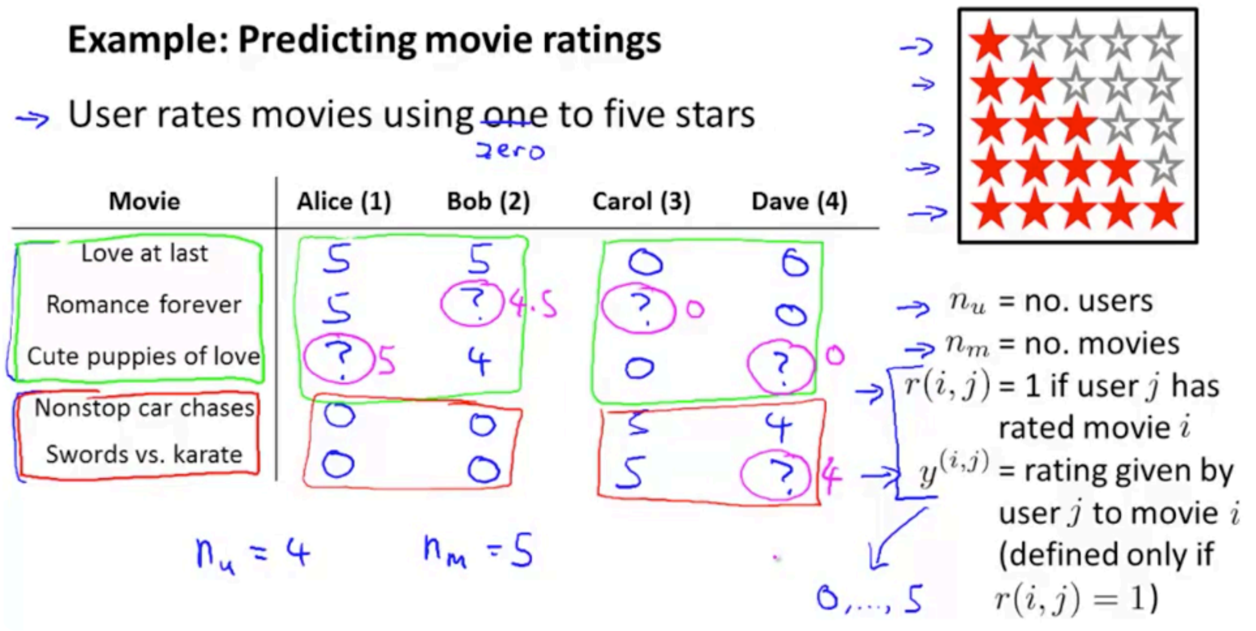
Predicting Movie Ratings

Problem Formulation

Recommender systems are currently one of the most valuable applications of ML currently used. Ex. Netflix, Amazon, Youtube, etc.

Recall, the features we choose to use are very important for ML algorithms. There are some algorithms that are able to learn which features to use. Recommender Systems can do this to some extent.

Problem Formulation: Ex. Predicting movie ratings



Example: Predicting movie ratings

→ User rates movies using ~~one~~ zero to five stars

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	? 4.5	? 0	0
Cute puppies of love	? 5	4	0	? 0
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	? 4

$n_u = 4$ $n_m = 5$

→ n_u = no. users

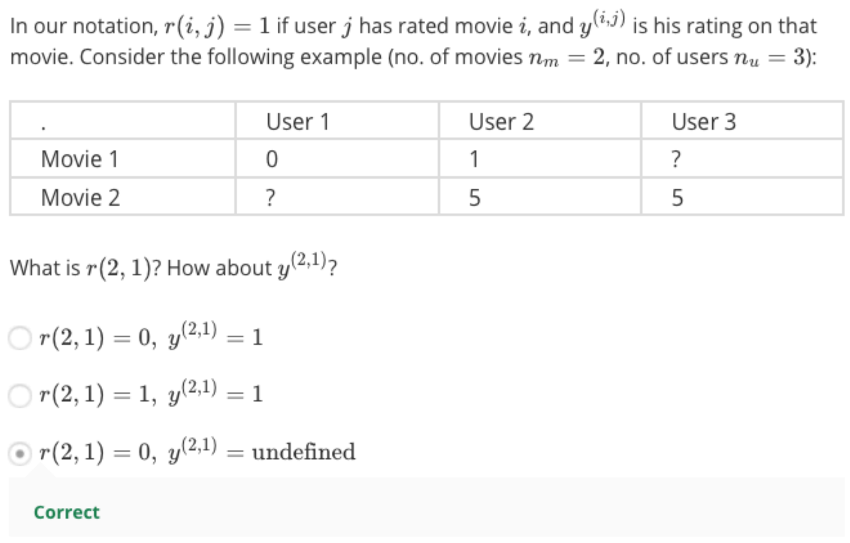
→ n_m = no. movies

→ $r(i,j)$ = 1 if user j has rated movie i

→ $y^{(i,j)}$ = rating given by user j to movie i (defined only if $r(i,j) = 1$)

$0, \dots, 5$

Given this dataset, the r(i, j)s and the y(i, j)s we need to predict what the 'question mark' values are.



In our notation, $r(i,j) = 1$ if user j has rated movie i, and $y^{(i,j)}$ is his rating on that movie. Consider the following example (no. of movies $n_m = 2$, no. of users $n_u = 3$):

.	User 1	User 2	User 3
Movie 1	0	1	?
Movie 2	?	5	5

What is $r(2,1)$? How about $y^{(2,1)}$?

- ○ $r(2,1) = 0,\ y^{(2,1)} = 1$
- ○ $r(2,1) = 1,\ y^{(2,1)} = 1$
- [✓ $r(2,1) = 0,\ y^{(2,1)} = \text{undefined}$]

Correct

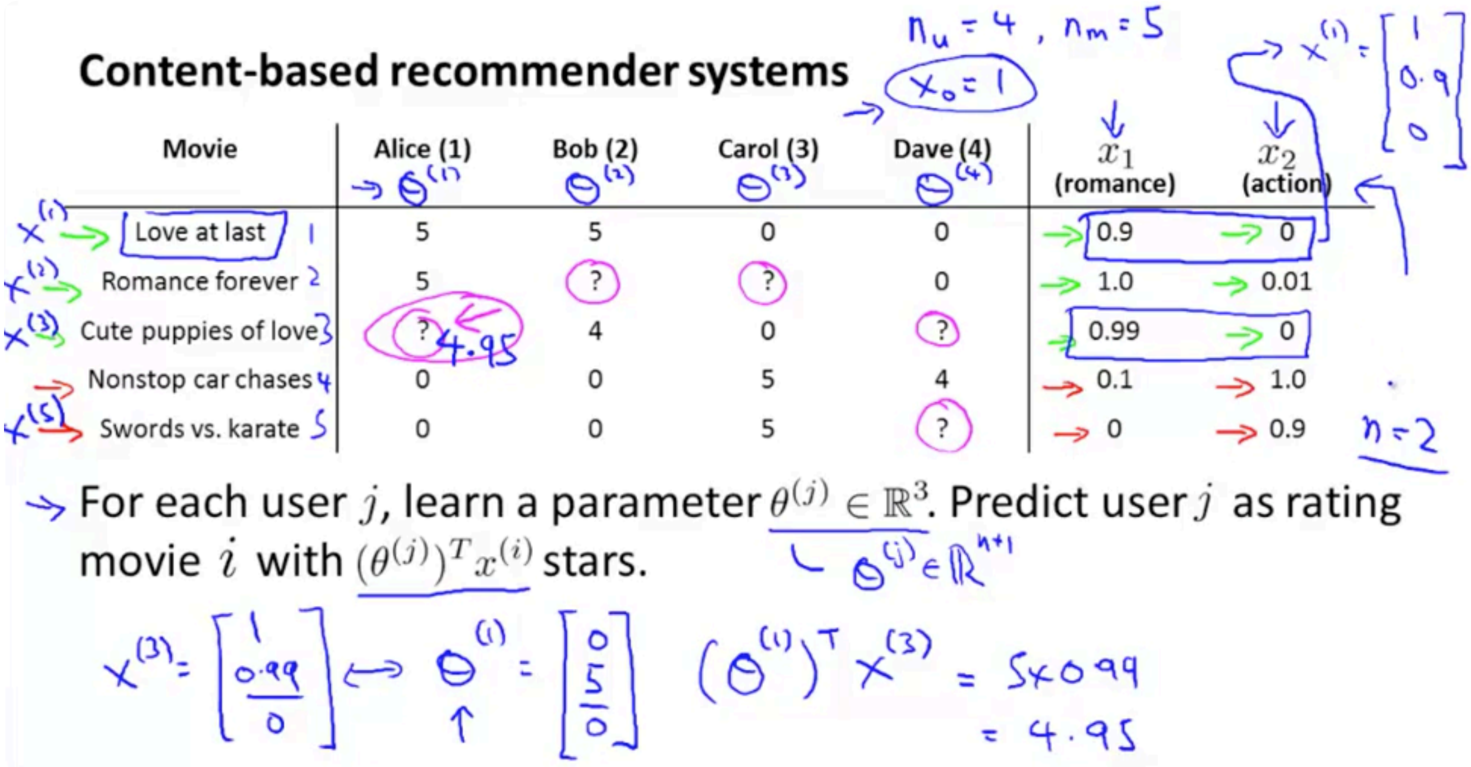
Content-based Recommendations (Algorithm)

We will now look at an approach to building a recommender system called a content-based approach.

How do we predict what the missing values will be?

Let us say we have a set of features for our movies that measure the degree to which a movie is a particular genre of movie. Then, each movie has a a feature vector x^i from 1 to m with # features n (2 in this example).

We are applying a different instance of linear regression for each user.



Content-based recommender systems

$n_u = 4$, $n_m = 5$ $\qquad$ $x_0 = 1$ $\qquad$ $x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$

	Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
$x^{(1)}$	Love at last 1	5	5	0	0	0.9	0
$x^{(2)}$	Romance forever 2	5	?	?	0	1.0	0.01
$x^{(3)}$	Cute puppies of love 3	? 4.95	4	0	?	0.99	0
	Nonstop car chases 4	0	0	5	4	0.1	1.0
$x^{(5)}$	Swords vs. karate 5	0	0	5	?	0	0.9

$n = 2$

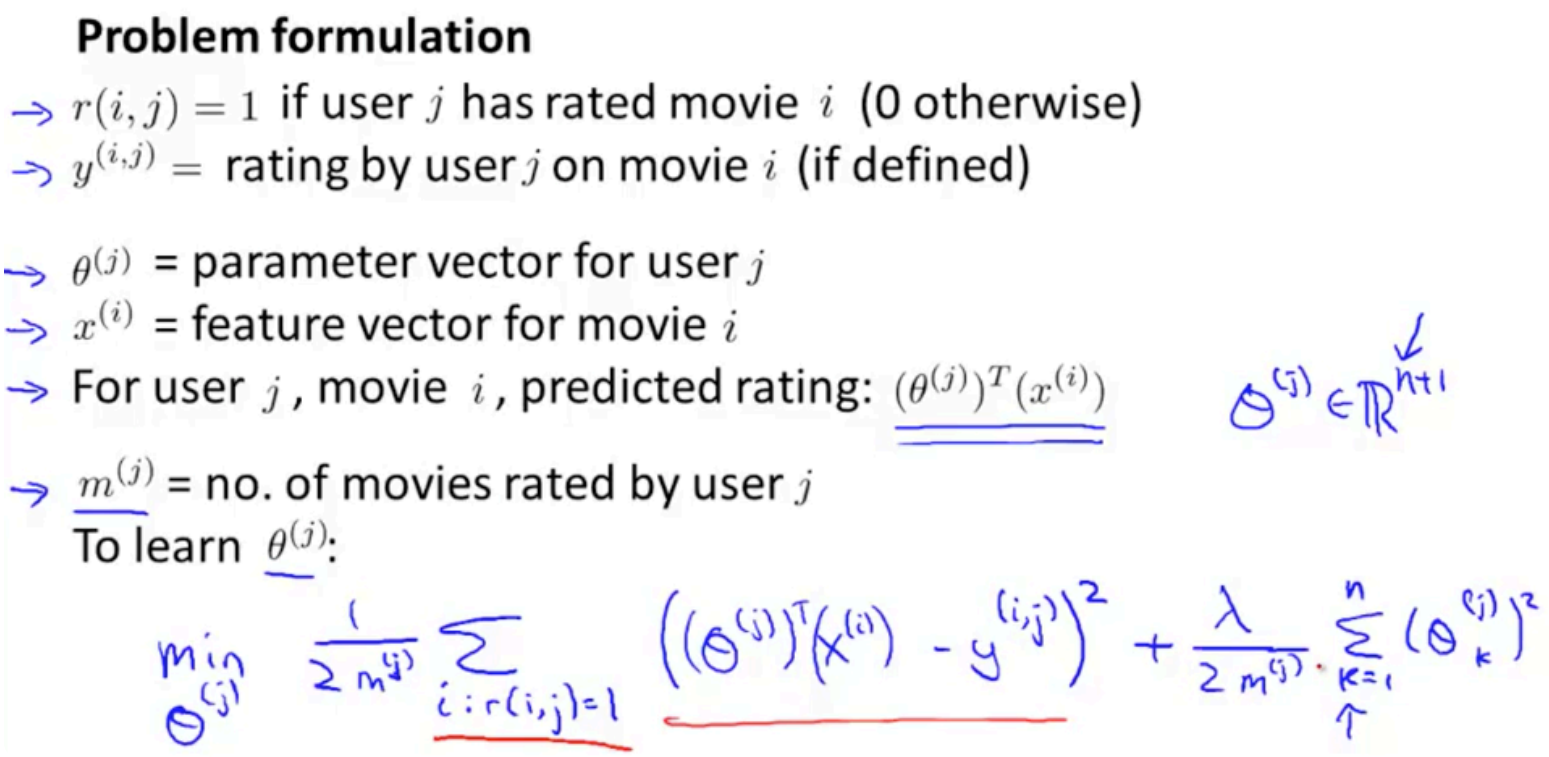
→ For each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^3$. Predict user j as rating movie i with $\underline{(\theta^{(j)})^T x^{(i)}}$ stars.

$\theta^{(j)} \in \mathbb{R}^{n+1}$

$$x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix} \longleftrightarrow \theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix} \qquad (\theta^{(1)})^T x^{(3)} = 5 \times 0.99 = 4.95$$

We are taking the learned parameters (theta1) for a similar user and applying them to a user with unknown variables (user 3 / x3)

More formally, here is how we can write down the problem:



Problem formulation

→ $r(i,j) = 1$ if user j has rated movie i (0 otherwise)

→ $y^{(i,j)} =$ rating by user j on movie i (if defined)

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

→ $x^{(i)}$ = feature vector for movie i

→ For user j, movie i, predicted rating: $\underline{(\theta^{(j)})^T (x^{(i)})}$ $\qquad \theta^{(j)} \in \mathbb{R}^{n+1}$

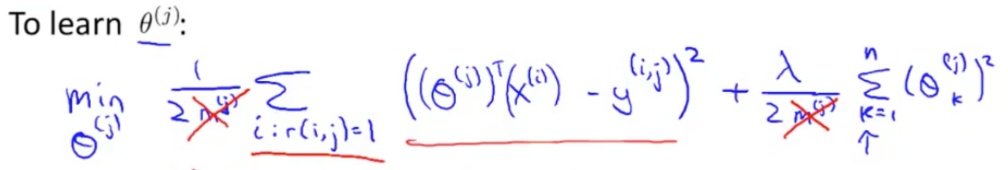
→ $m^{(j)}$ = no. of movies rated by user j

To learn $\theta^{(j)}$:

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

Using this approach we get a pretty good estimate of theta j with which to make predictions for user Js movie ratings.

For recommender systems we need to change some of the notation a bit:

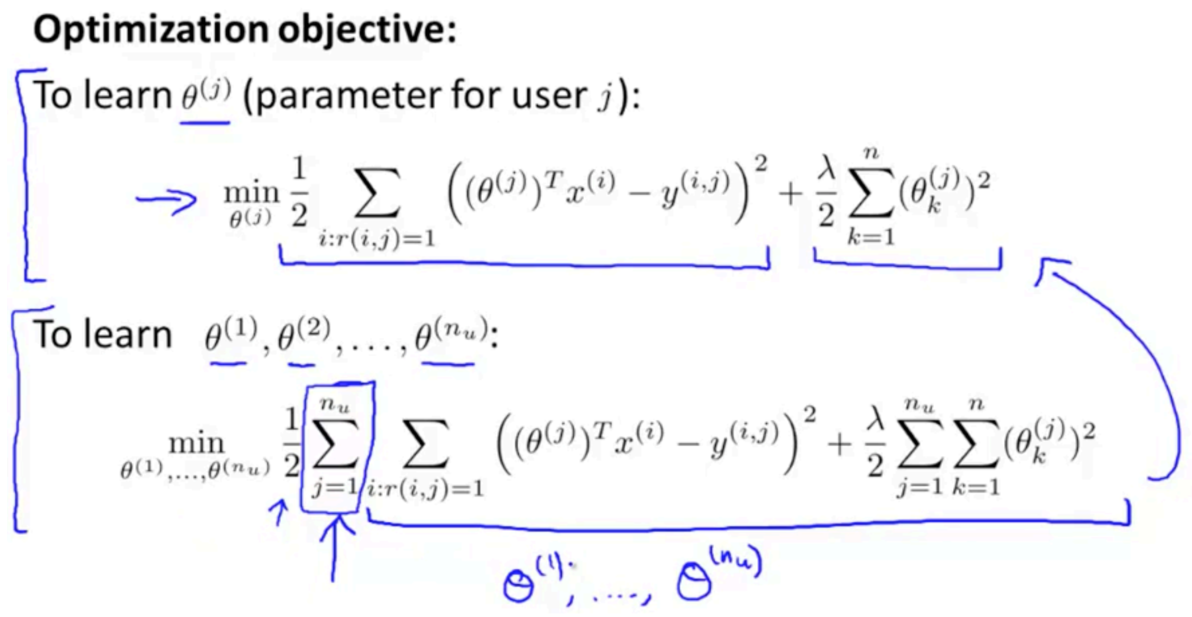


To learn $\underline{\theta^{(j)}}$:

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

We drop the m(j) but with the minimizer should still get the same value of theta j as before.

More clearly to learn all the parameter vectors theta:



Optimization objective:

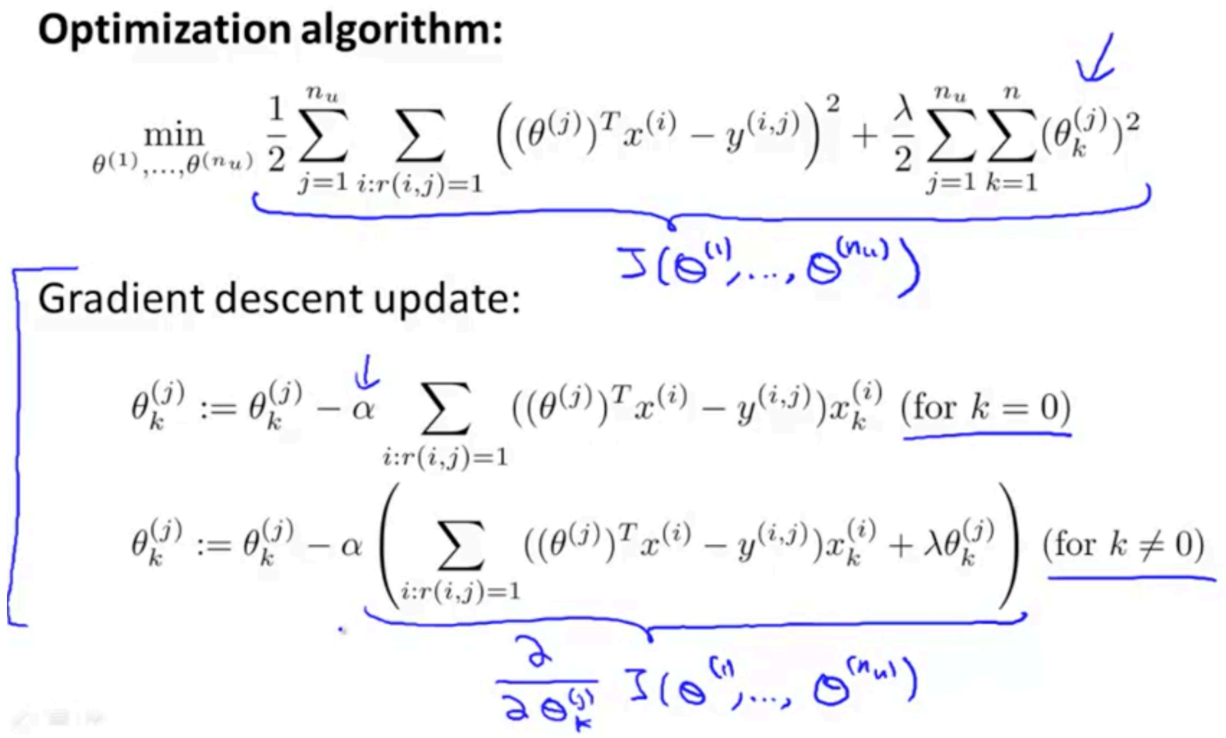
To learn $\underline{\theta^{(j)}}$ (parameter for user j):

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

To learn $\underline{\theta^{(1)}}, \underline{\theta^{(2)}}, \ldots, \underline{\theta^{(n_u)}}$:

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

The whole picture:



Optimization algorithm:

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

Gradient descent update:

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

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \underbrace{\left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right)}_{\frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)}, \ldots, \theta^{(n_u)})} \quad \underline{(\text{for } k \neq 0)}$$

Note this looks identical to the linear regression we performed before, just without the 1/m term in front of the sum.

**Note: This algorithm assumes we have the features of different movies available to us and that the features capture the content of those movies. However in practice we don't typically have such a dataset. Not just for movies but for whatever we are trying to recommend. In the next video we will discuss how to get around this with a different approach.

Collaborative Filtering

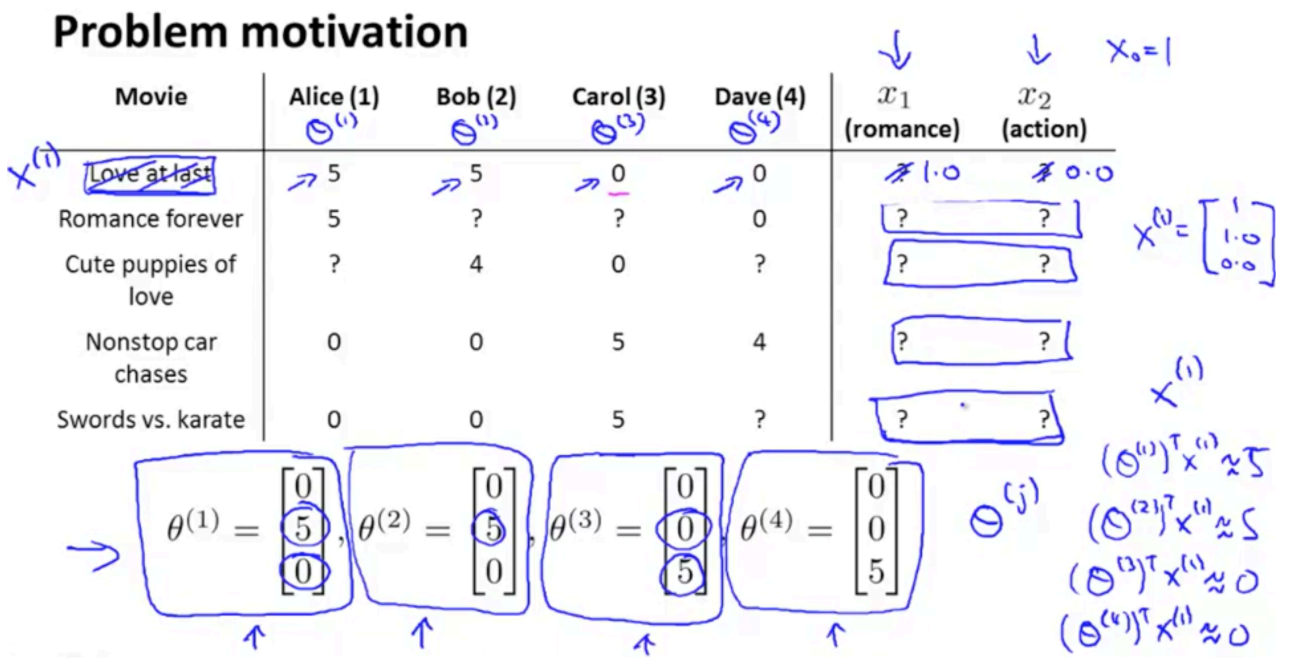
In this video we will discuss another approach to building a recommender system called collaborative filtering. This algorithm has a interesting property called featuring learning in which it can start to learn for itself which features to use.

Here was our previous dataset without the values for the features. Where do we get these features from now?

We only have user ratings from some of our users for some of our movies.

Lets say we can go to (some) of our users and ask them what the value of theta j is for them. (Ask them which movies and which types of movies they like)

With this, we can begin to infer some of the unknowns.



Problem motivation

$x_0 = 1$

Movie	Alice (1) $\Theta^{(1)}$	Bob (2) $\Theta^{(2)}$	Carol (3) $\Theta^{(3)}$	Dave (4) $\Theta^{(4)}$	x_1 (romance)	x_2 (action)
$x^{(1)}$ ~~Love at last~~	5	5	0	0	~~?~~ 1.0	~~?~~ 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	?	?	?

$$x^{(1)} = \begin{bmatrix} 1 \\ 1.0 \\ 0.0 \end{bmatrix}$$

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

$\Theta^{(j)}$

$x^{(1)}$

$(\Theta^{(1)})^T x^{(1)} \approx 5$

$(\Theta^{(2)})^T x^{(1)} \approx 5$

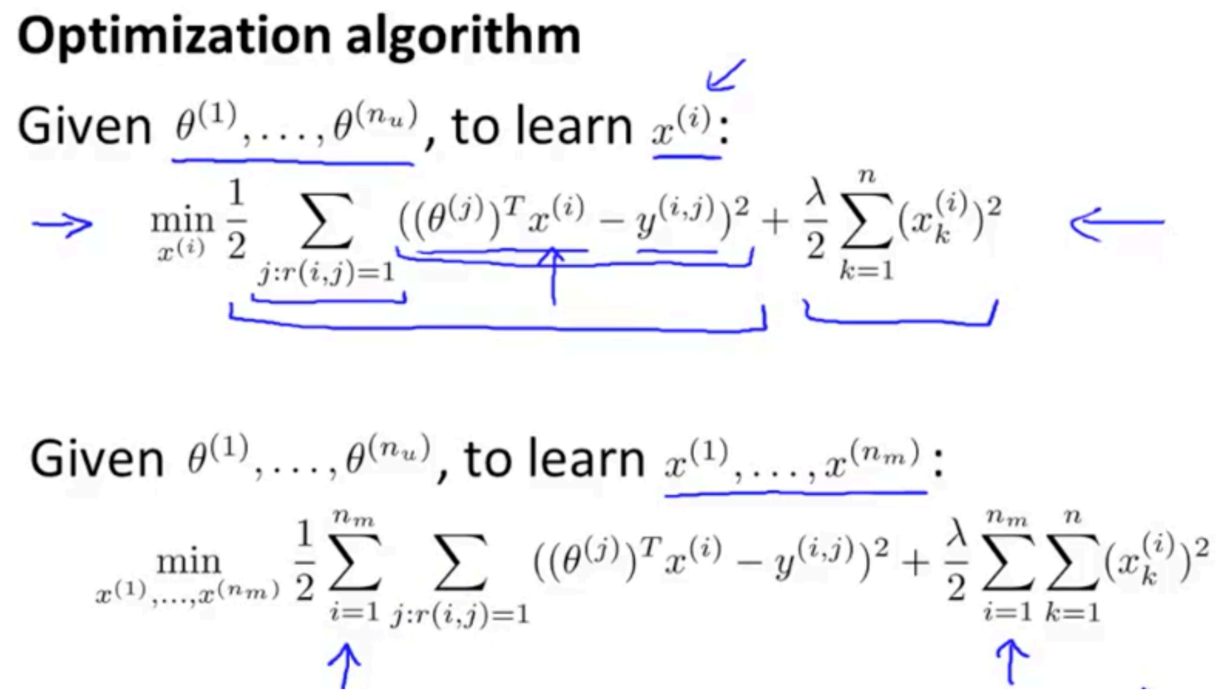
$(\Theta^{(3)})^T x^{(1)} \approx 0$

$(\Theta^{(4)})^T x^{(1)} \approx 0$

Let us formalize this problem:

Given our all thetas for a users that have rated a particular movie, we need to predict x^i.

And then for all our thetas for all movies, we can try to predict X.



Optimization algorithm

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

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

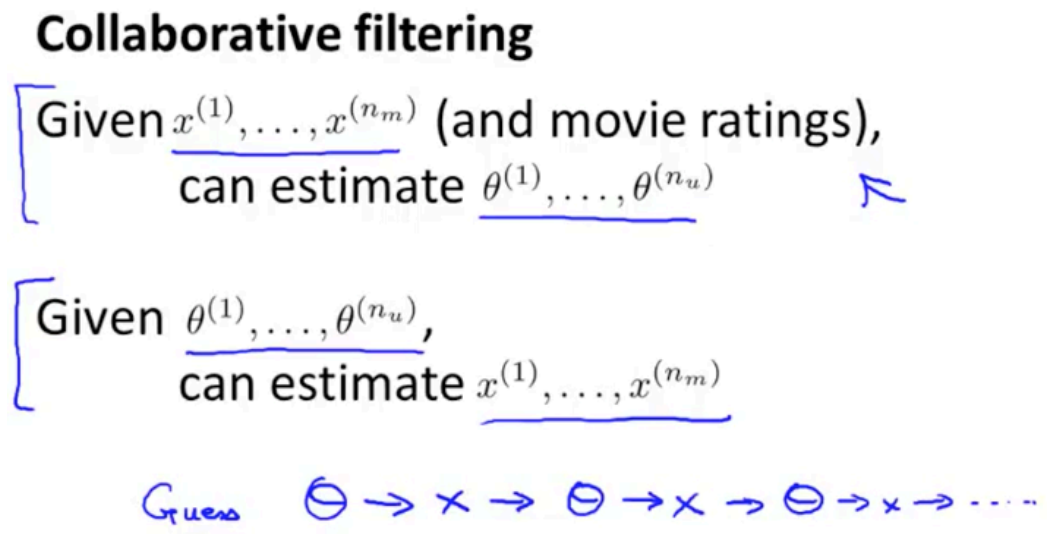
Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(1)}, \ldots, x^{(n_m)}$:

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

Putting this all together.

In the previous video we showed that if we have a set of movie ratings (the r(i,j)s and the y(i,j)s) we can learn our parameters theta and then make predictions for different users.

In this video, we showed that if your users are willing to give you parameters theta, we can then estimate features for the different movies.



Collaborative filtering

Given $\underline{x^{(1)},\ldots,x^{(n_m)}}$ (and movie ratings),
can estimate $\underline{\theta^{(1)},\ldots,\theta^{(n_u)}}$

$r^{(i,j)}$
$y^{(i,j)}$

Given $\underline{\theta^{(1)},\ldots,\theta^{(n_u)}}$,
can estimate $\underline{x^{(1)},\ldots,x^{(n_m)}}$

Guess $\Theta \rightarrow x \rightarrow \Theta \rightarrow x \rightarrow \Theta \rightarrow x \rightarrow \cdots$

Notice this is a chicken & egg situation. With the ratings (features), we can learn the parameters, but with the features, we can learn the parameters.

What we can do is start with a random theta, estimate our features, and then iterate, improving our theta, our features, etc. as the system matures.

With each user contributing ratings on a subset of the data, the users are <u>collaborating</u> to all improve the system and algorithm.

In the next video we will discuss an even better technique for collaborative filtering.

Collaborative Filtering Algorithm

From the ideas we discussed in the previous video of how we can use features to find parameters or parameters to find features, we are going to put those ideas together to come up with a collaborative filtering algorithm.

Collaborative filtering optimization objective

$\rightarrow$ Given $x^{(1)},\ldots,x^{(n_m)}$, estimate $\theta^{(1)},\ldots,\theta^{(n_u)}$:

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

$\rightarrow$ Given $\theta^{(1)},\ldots,\theta^{(n_u)}$, estimate $x^{(1)},\ldots,x^{(n_m)}$:

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

It turns out that instead of going back and forth, there is a more efficient algorithm that does not need to go back and forth and can instead solve for theta and x simultaneously and put them into the same objective:

Minimizing $x^{(1)},\ldots,x^{(n_m)}$ and $\theta^{(1)},\ldots,\theta^{(n_u)}$ simultaneously:

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

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

First, looking at the two equations above, notice that the squared error sum in the middle is the same but using i or j, and then summed over all movies that have ratings. As such we can combine those terms into a combined optimization objective + the respective regularizations.

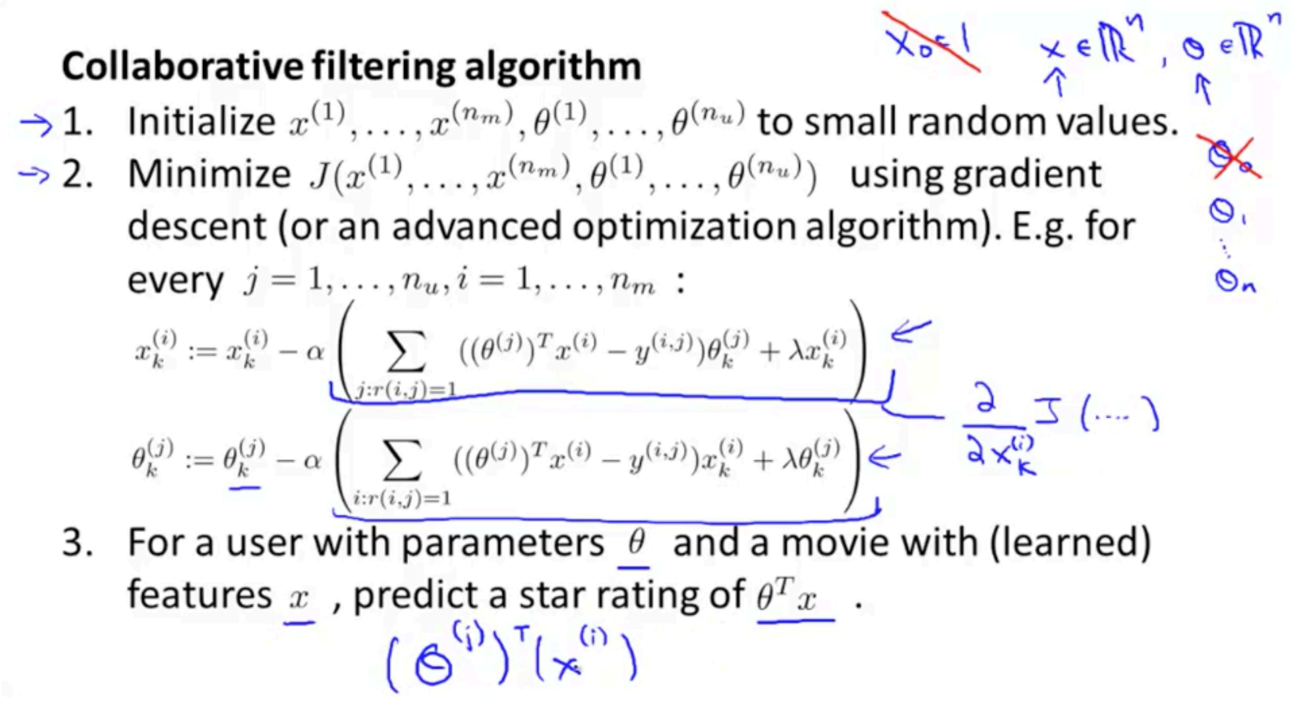
Notice in the combined minimization objective, if you were to hold the x's constant or the theta's constant you would effectively be solving for the above equations.

**Note: Previously we have been using the convention to add a bias term x0 = 1, for this implementation we will not be doing that. As such theta will also be an element of R^n like x. (And we are regularizing all our terms, no 0 term that doesn't get regularized)

$\cancel{x_0 = 1}$ $\quad x \in \mathbb{R}^n \quad$ $\cancel{x \in \mathbb{R}^{n+1}}$

This is because the algorithm now has the flexibility to learn it's own features and for example could set it's own bias term x1 = 1 if it needed to.

Putting everything together:



Collaborative filtering algorithm

~~$x_0 = 1$~~ $x \in \mathbb{R}^n$, $\theta \in \mathbb{R}^n$

~~θ_0~~ $\theta_1, \ldots, \theta_n$

1. Initialize $x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}$ to small random values.
2. Minimize $J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)})$ using gradient descent (or an advanced optimization algorithm). E.g. for every $j = 1, \ldots, n_u, i = 1, \ldots, n_m$:

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

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

3. For a user with parameters $\underline{\theta}$ and a movie with (learned) features $\underline{x}$, predict a star rating of $\underline{\theta^T x}$.

$(\theta^{(j)})^T (x^{(i)})$

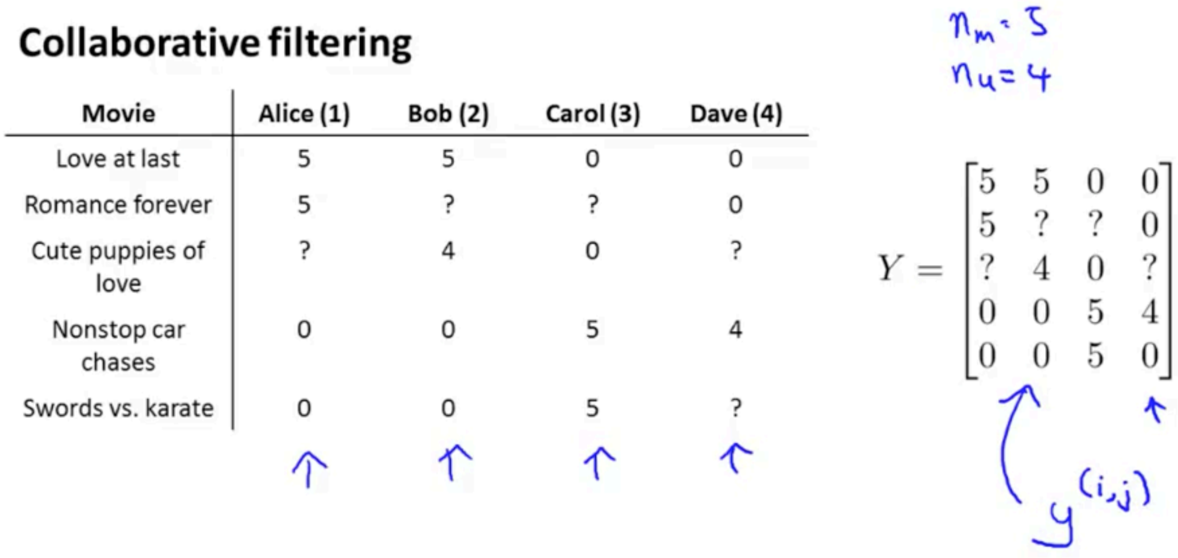
**Notice that we initialize our x's and theta's to small random variables to perform symmetry breaking (similar to the random initialization of a neural network's parameters) and ensures the algorithm learns features that are different from one another.

Low Rank Matrix Factorization

Vectorization: Low Rank Matrix Factorization

In this video we will discuss the vectorization implementation of the collaborative filtering algorithm and some other tips and tricks.

We want to write out an alternative way of writing out the predictions of the collaborative filtering algorithm.



Collaborative filtering

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	?

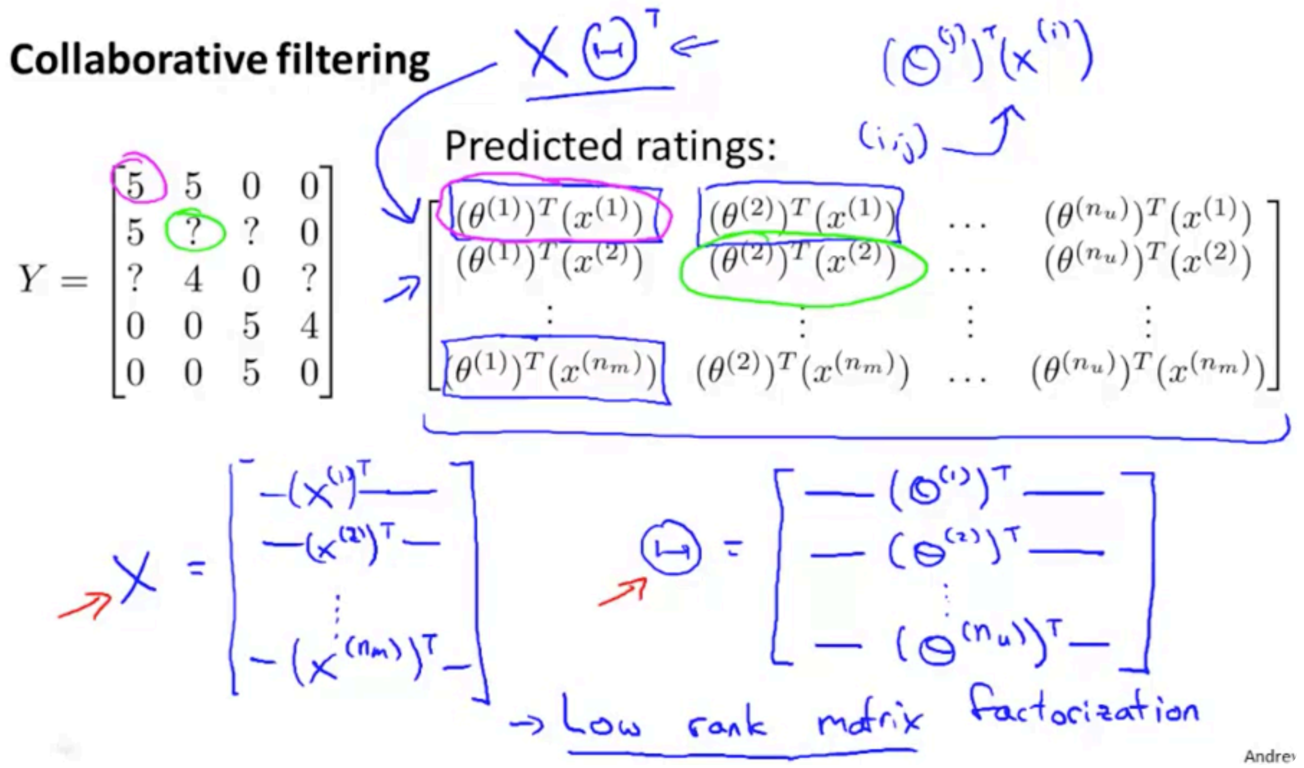
$n_m = 5$

$n_u = 4$

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

$y^{(i,j)}$

The rating user j would give to movie i. is at index (i,j) of the predicted ratings matrix.



Collaborative filtering

$X \Theta^T$

$(\Theta^{(j)})^T(x^{(i)})$

(i,j)

Predicted ratings:

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix} \quad \begin{bmatrix} (\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \dots & (\theta^{(n_u)})^T(x^{(1)}) \\ (\theta^{(1)})^T(x^{(2)}) & (\theta^{(2)})^T(x^{(2)}) & \dots & (\theta^{(n_u)})^T(x^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ (\theta^{(1)})^T(x^{(n_m)}) & (\theta^{(2)})^T(x^{(n_m)}) & \dots & (\theta^{(n_u)})^T(x^{(n_m)}) \end{bmatrix}$$

$$X = \begin{bmatrix} - (x^{(1)})^T - \\ - (x^{(2)})^T - \\ \vdots \\ - (x^{(n_m)})^T - \end{bmatrix} \qquad \Theta = \begin{bmatrix} - (\Theta^{(1)})^T - \\ - (\Theta^{(2)})^T - \\ \vdots \\ - (\Theta^{(n_u)})^T - \end{bmatrix}$$

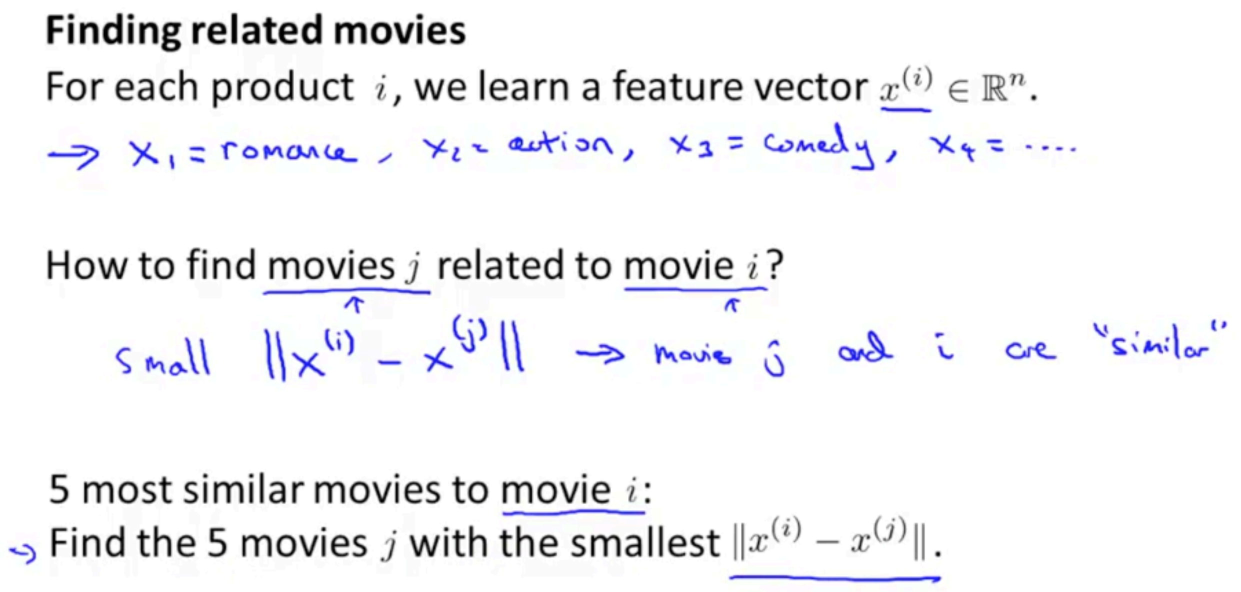
→ Low rank matrix factorization

We can also rename the collaborative filtering algorithm to 'Low Rank Matrix Factorization'

This comes from the property that the X Theta ^ T matrix has from linear algebra which is that it is a low rank matrix.

Once we have run the collaborative filtering algorithm, we can use the learned features in order to find related movies (or related products if these were goods for sale)

Ex. For a user currently watching movie 'j', which movie 'i' should we recommend to them afterwards?



Finding related movies

For each product i, we learn a feature vector $\underline{x^{(i)}} \in \mathbb{R}^n$.

$\rightarrow$ x_1 = romance, x_2 = action, x_3 = comedy, x_4 = ...

How to find $\underline{\text{movies } j}$ related to $\underline{\text{movie } i}$?

small $\|x^{(i)} - x^{(j)}\|$ $\rightarrow$ movie j and i are "similar"

5 most similar movies to $\underline{\text{movie } i}$:

$\rightarrow$ Find the 5 movies j with the smallest $\underline{\|x^{(i)} - x^{(j)}\|}$.

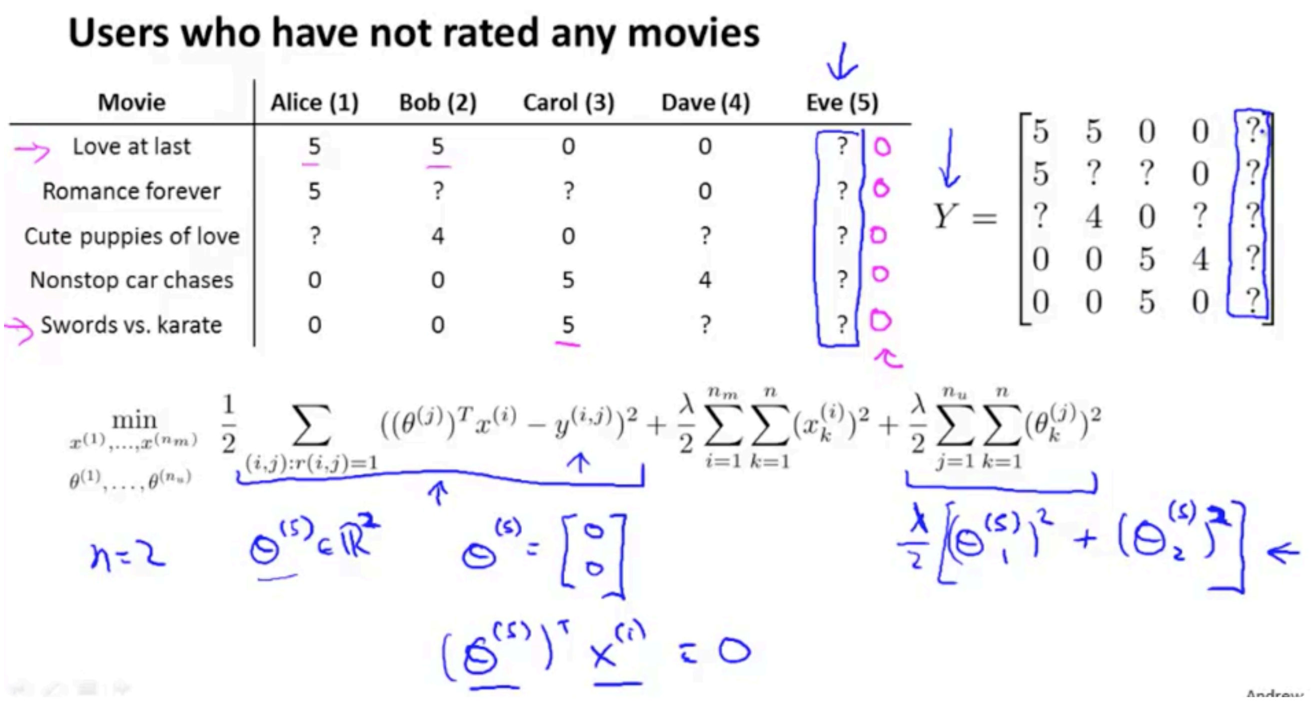
Implementation Detail: Mean Normalization

We have covered all the main pieces of the recommender system algorithm / collaborative filtering algorithm.

In this video we will discuss one last implementation detail - mean normalization - which can sometimes make the algorithm work a bit better.

Let us consider the exam where a user has not rated any movies.

What will our collaborative filtering algorithm do on this user?



Users who have not rated any movies

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

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

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

$n=2$ $\quad \theta^{(5)} \in \mathbb{R}^2 \quad \theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\quad \frac{\lambda}{2}\left[(\theta_1^{(5)})^2 + (\theta_2^{(5)})^2\right]$

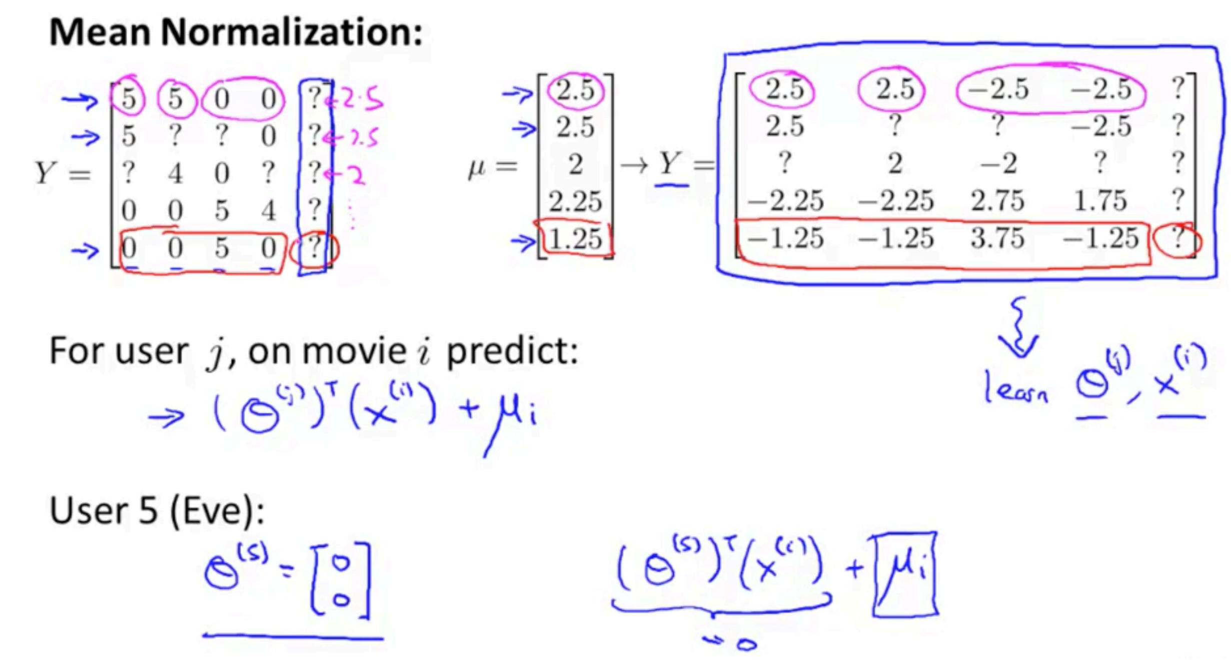
$$(\theta^{(5)})^T x^{(i)} = 0$$

The first (Left Side) term in our optimization algorithm plays no role since theta^5 is undefined. The only term that affects Theta^5 is our regularization term. As such this will regularize our theta^5 term to [0 0] ^ T

Then, when we attempt to predict how user 5 would rate any movie, they are also going to be zero.

Obviously, this doesn't seem very useful to just predict all zeros. We also don't have a way of making any recommendations for user 5.

Instead, we are going to perform mean normalization to solve this problem.



Mean Normalization:

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \begin{matrix} \leftarrow 2.5 \\ \leftarrow 2.5 \\ \leftarrow 2 \\ \\ \\ \end{matrix} \qquad \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \rightarrow Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

learn $\theta^{(j)}, x^{(i)}$

For user j, on movie i predict:

$$\rightarrow (\theta^{(j)})^T(x^{(i)}) + \mu_i$$

User 5 (Eve):

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \underbrace{(\theta^{(5)})^T(x^{(i)})}_{=0} + \boxed{\mu_i}$$

We are going to compute the average rating of each movie and store it in a new vector. Then subtract the mean rating off of each movie. Thus, the average rating of each movie is zero.

We will then use this new set of ratings for our collaborative filtering algorithm to learn Theta^j and X^i.

Lastly then when making a predict we need to add the mean back in. Then, for new users without any movie ratings, we predict the mean of the movie rating.

Finally as an aside, in case we have a movie with no ratings, we can play with version of the algorithm to normalize the feature columns, although this is probably less trivial. In practice, if you have a movie or product without any ratings perhaps it is best not to recommend it until it has some ratings.

Also, since we have used mean normalization here, unlike other applications of feature scaling we have performed, we did not scale the movie ratings by dividing by the range (max-min) because all the movie ratings here have the same scale. (Ex. 0-5)