Recommender Systems:
The Problem:

- To recommend new products to a user based on prior data about the user

Eg:
Predicting product ratings
Notation:

$$n_u = no\ of\ users$$
$$n_m = number\ of\ products$$
$$r(i,j) = \{0,1\}1\ if\ user\ has\ rated\ product\ i$$
$$y^{(i,j)} = rating\ given\ by\ user\ j\ to\ movie\ i\ (defined\ only\ if\ r(i,j) = 1$$

Content based recommendation:
Given a dataset of x^mratings:
For each user j, learn a param $\theta^{(j)} \in \mathbb{R}^3$predicts user j as rating new movie i with $\theta^{(j)T}x^{(i)}$stars.
For each θ:

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

Hence we learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$. Much like a Linear regression model

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

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

However, we do not always have such data on hand.

Alternatively:
Using User preferences to predict theta values:
Given $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$, learn $x^{(i)}$:
For each $x^{(i)}$

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

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

A key observation to make is that the first term for both is the same!
Also, this algorithm along with the previous one can already cycle to convergence given random initialization of theta values.

However, there is a way to combine the 2 approaches:

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

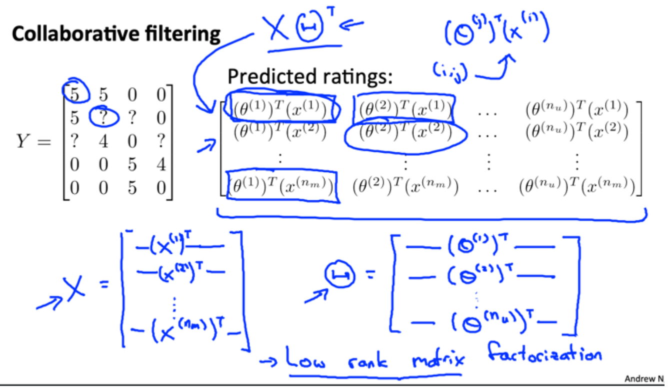
Use as such to compute gradient for both $\theta^{(i)}$ *and* $x^{(i)}$

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

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

Use Gradient descent or an advanced optimization algorithm to reach convergence of x and θ

Vectorisation Approach:



Collaborative filtering

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

Predicted ratings:

$$\begin{bmatrix} (\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \dots & (\theta^{(n_u)})^T(x^{(1)})\\ (\theta^{(1)})^T(x^{(2)}) & (\theta^{(2)})^T(x^{(2)}) & \dots & (\theta^{(n_u)})^T(x^{(2)})\\ \vdots & \vdots & \vdots & \vdots\\ (\theta^{(1)})^T(x^{(n_m)}) & (\theta^{(2)})^T(x^{(n_m)}) & \dots & (\theta^{(n_u)})^T(x^{(n_m)}) \end{bmatrix}$$

$X\Theta^T$
$(\Theta^{(j)})^T(x^{(i)})$
(i,j)
$X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ \vdots \\ -(x^{(n_m)})^T- \end{bmatrix}$
$\Theta = \begin{bmatrix} -(\Theta^{(1)})^T- \\ -(\Theta^{(2)})^T- \\ \vdots \\ -(\Theta^{(n_u)})^T- \end{bmatrix}$
→ Low rank matrix factorization
Andrew N

How the algorithm learns each feature is for each $x^{(i)} \in \mathbb{R}^n$

The algorithm learns some meaningful feature.

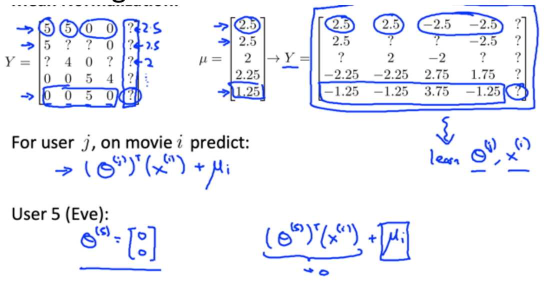
However, this feature may be hard for us to convert into a intuition for humans to understand.

How to find movies j related to movies i?

- ⇨ Small $\left\|x^{(i)} - x^{(j)}\right\|$ meaning movies j and i are "similar"

Mean Normalisation:

Initialising a Theta and x value for a new user who has not rated anything:



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

For user j, on movie i predict:

User 5 (Eve):

2.5
2.5
2
learn $\Theta^{(j)}, x^{(i)}$
$\rightarrow (\Theta^{(j)})^T(x^{(i)}) + \mu_i$
$\Theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
$\underbrace{(\Theta^{(5)})^T(x^{(i)})}_{\to 0} + \boxed{\mu_i}$