Problem Formulation

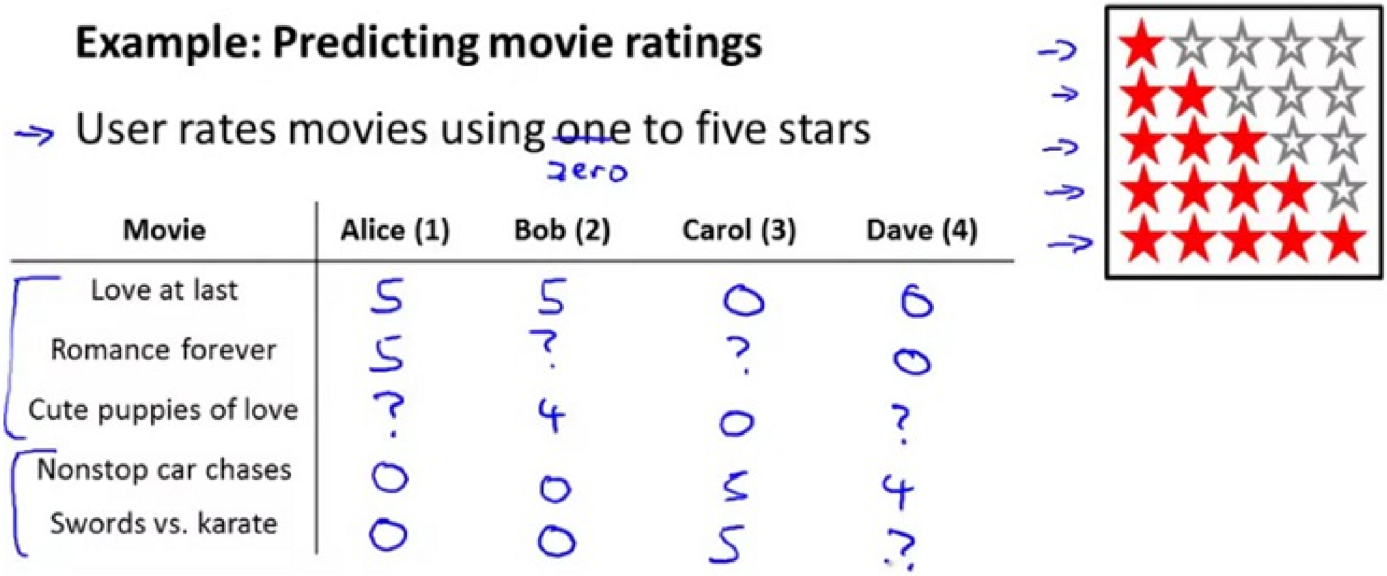
Example: Predicting movie ratings

Recommendation is currently a very popular application of machine learning.

Say we are trying to recommend movies to customers.

We're a company who sells movies

- we let users rate movies using a 1-5 star rating
- To make the example nicer, allow 0-5 (makes math easier)



We can use the following definitions:

- 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)

Summary of scoring:

- Alice and Bob gave good ratings to romantic coms, but low scores to action films
- Carol and Dave game good ratings for action films but low ratings for romantic coms

We have the data given above:

- The problem is as follows:
  - Given $r(i, j)$ and $y(i, j)$ - go through and try and predict missing values (?)
  - Come up with a learning algorithm that can fill in these missing values

Video Question: 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 $y^{(2,1)}$?

- $r(2,1) = 0, \; y^{(2,1)} = 1$
- $r(2,1) = 1, \; y^{(2,1)} = 1$

> $r(2,1) = 0, \; y^{(2,1)} = \text{undefined}$

- $r(2,1) = 1, \; y^{(2,1)} = \text{undefined}$

Content Based Recommendations

Using our example above, how do we predict?

We can introduce two features, x_1 and x_2 which represents how much romance or how much action a movie may have (on a scale of 0−1).

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

If we have features like these, each film can be recommended by a feature vector:

- Add an extra feature (intercept term) which is x_0 = 1 for each film
- So for each film we have a [3 x 1] vector, which for film number 1 ("Love at Last") would be:

$$x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$$

  i.e. for our dataset we have: $\{x_1, x_2, x_3, x_4, x_5\}$
- To be consistent with our notation, n is going to be the number of features not counting the x_0 term, so n = 2

One approach is that we could do linear regression for every single user. 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. So we determine a future rating based on their interest in romance and action based on previous films

- $\theta^{(j)}$ = parameter vector for user j
- $x^{(i)}$ = feature vector for movie i

Video Question: Consider the following set of movie ratings:

Movie	Alice (1)	Bob (2)	Carol (3)	David (4)	(romance)	(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

Which of the following is a reasonable value for $\theta^{(3)}$? Recall that x_0 = 1.

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

$\theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$

Problem Formulation

- 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)

- $\theta^{(j)}$ = parameter vector for user j
- $x^{(i)}$ = feature vector for movie i

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

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

To learn $\theta^{(j)}$, we do the following

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

This is our familiar linear regression. The base of the first summation is choosing all i such that $r(i, j)$ = 1.

- Sum over all values of i (all movies the user has used) when $r(i, j)$ = 1 (i.e. all the films that the user has rated)

- This is just like linear regression with least-squared error

But for our recommender system we want to learn parameters for all users, so we add an extra summation term to this which means we determine the minimum $\theta^{(j)}$ value for every user

- To get the parameters (to learn $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n_u)}$) for all our users, we do the following:

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

We can apply our linear regression gradient descent update using the above cost function.

The only real difference is that we **eliminate the constant** $\frac{1}{m}$.

In order to do the minimization we have the following gradient descent

Gradient descent update:

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

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

Slightly different to our previous gradient descent implementations

- k = 0 and k != 0 versions
- We can define the middle term above as:

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

$$\cancel{\frac{1}{m^{(j)}}}$$

- Difference from linear regression:
  - No 1/m terms (got rid of the 1/m term)
  - Otherwise very similar

This approach is called content-based approach because we assume we have features regarding the content which will help us identify things that make them appealing to a user. However, often such features are not available