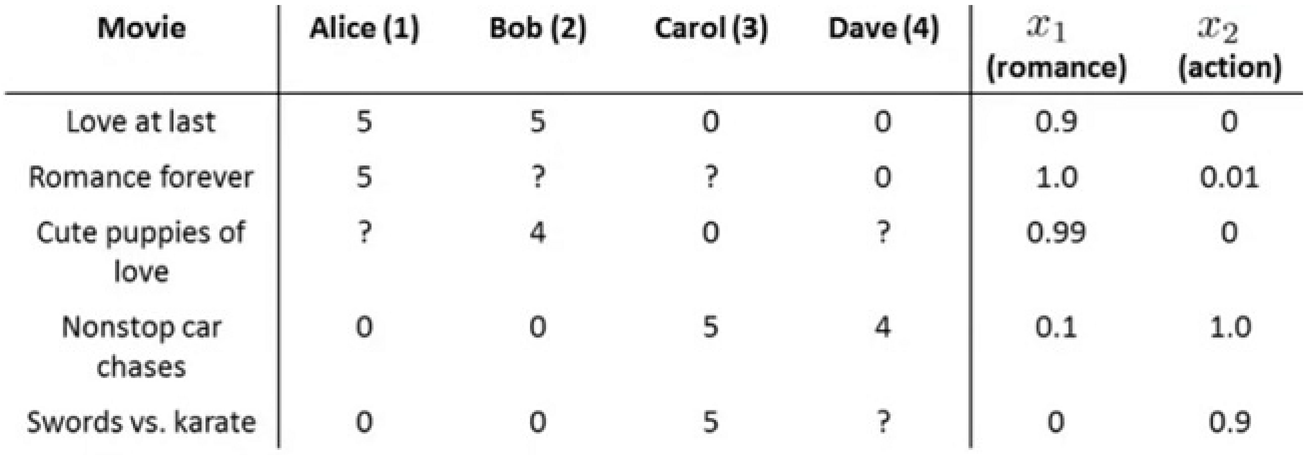
Collaborative Filtering ¶

The collaborative filtering algorithm has a very interesting property what is called **feature learning**:

- An algorithm that can start to learn for itself what features to use
  - i.e. it can learn for itself what features it needs to learn

Recall our original data set above for our five films and four raters

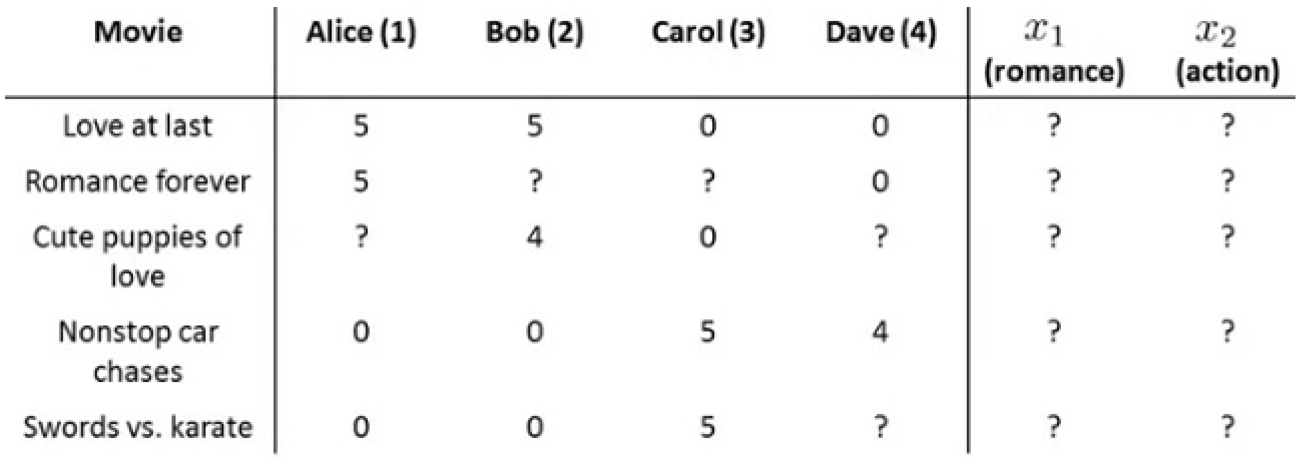
- Here we assume someone had calculated the "romance" and "action" amounts of the films



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

  It can be very difficult to find features such as "amount of romance" or "amount of action" in a movie. To figure this out, we can use **feature finders**.
- Often want more features than just two

So - let's change the problem and pretend we have a data set where we don't know any of the features associated with the films



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	?	?	?

Now let's make a different assumption:

- We've polled each user and found out how much each user likes
  - Romantic films
  - Action films
- Which has generated the following parameter set:

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

- Alice and Bob like romance but hate action
- Carol and Dave like action but hate romance

So let's assume that somehow we can go to users and each user j just tells us what is the value of $\theta^{(j)}$ for them. And so basically specifies to us of how much they like different types of movies, and so basically specifies to us of how much they like different types of movies.

- We can let the users tell us how much they like the different genres, providing their parameter vector immediately for us.
- If we can get these parameters θ from our users then it turns out that it becomes possible to try to infer what are the values of x_1 and x_2 for each movie.

We know from the feature vectors Alice and Bob love romantic films, while Carol and Dave hate them:

- Based on the factor Alice and Bob liked "Love at Last" and Carol and Dave hated it we may be able to (correctly) conclude that "Love at Last" is a romantic film

What we're really asking is: **"What feature vector should x_1 be?"** so that:

$$(\theta^{(1)})^T x^{(1)} \approx 5$$
$$(\theta^{(2)})^T x^{(1)} \approx 5$$
$$(\theta^{(3)})^T x^{(1)} \approx 0$$
$$(\theta^{(4)})^T x^{(1)} \approx 0$$

From this we can guess that $x^{(1)}$ may be:

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

Using that same approach we should then be able to determine the remaining feature vectors for the other films

Video Question: Consider the following movie ratings:

.	User 1	User 2	User 3	(romance)
Movie 1	0	1.5	2.5	?

Note that there is only one feature x_1. Suppose that:

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

What would be a reasonable value for $x_1^{(1)}$ (the value denoted "?" in the table above)?

```
0.5
```

- 1
- 2
- Any of these values would be equally reasonable.

Optimization algorithm

Let's formalize this problem of learning the features $x^{(i)}$.

- Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$ (given the parameter vectors for each users' preferences), to learn $x^{(i)}$

We must minimize an optimization function which tries to identify the best parameter vector associated with a film

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

- So we're summing over all the indices j for where we have data for movie i
- Like before, the above equation gives us a way to learn the features for one film
  - We want to learn all the features for all the films - so we need an additional summation term

To infer the features from given parameters, we use the squared error function with regularization over all the users: **Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$ (given the parameter vectors for each users' preferences), to learn** $x^{(i)}, \ldots, x^{(n_m)}$

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

We can also **randomly guess** the values for theta to guess the features repeatedly. We will actually converge to a good set of features.

Video Question: Suppose you use gradient descent to minimize:

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

Which of the following is a correct gradient descent update rule for $i \neq 0$?

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

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

Collaborative Filtering Algorithm

Here we combine the ideas from before to build a collaborative filtering algorithm

Our starting point is as follows:

- So one of the things we worked out earlier is that if we have features for the movies then we can solve the following minimization problem to find the parameters theta for our users.

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

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

- We also worked that out, if we are given the parameters theta, we can also use that to estimate the features x, and we can do that by solving the following minimization problem.

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

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

One thing we could do is:

- Randomly initialize parameter
- Solve for x, solve for θ and go back and forward

But there's a more efficient algorithm which can solve θ and x simultaneously.

- Define a new optimization objective which is a function of x and θ

To speed things up, we can simultaneously minimize our features and our parameters:

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

It looks very complicated, but we've only combined the cost function for θ and the cost function for x.

Understanding this optimization objective:

- The squared error term is the same as the squared error term in the two individual objectives above

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

- So it's summing over every movie rated by every users
  - Note the ":" means, "for which"
  - Sum over all pairs (i, j)
- The regularization terms are simply added to the end from the original two optimization functions
  - This newly defined function has the property that
    - If we held x constant and only solved θ then we solve the, "Given x, solve θ" objective above
    - Similarly, if we held θ constant we could solve x.
- In order to come up with just one optimization function we treat this function as a function of both film features x and user parameters θ
  - Only difference between this in the back-and-forward approach is that we minimize with respect to both x and θ simultaneously

Because the algorithm can learn them itself, the bias units where x_0 = 1 have been removed, therefore $x \in \mathbb{R}^n$ and $\theta \in \mathbb{R}^n$.

- We do this because we are now learning all the features so if the system needs a feature always = 1 then the algorithm can learn one

Collaborative Filtering Algorithm (steps in the algorithm)

These are the steps in the algorithm:

1. Initialize $x^{(i)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}$ to small random values. This serves to break symmetry and ensures that the algorithm learns features $x^{(i)}, \ldots, x^{(n_m)}$that are different from each other.

- A bit like neural networks - initialize all parameters to small random numbers

2. Minimize $J(x^{(i)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$ using gradient descent (or an advanced optimization algorithm).

E.g. for every j = 1, … ,n_u, i = 1, …, n_m :

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

- Where the top term is the partial derivative of the cost function with respect to x_k^i while the bottom is the partial derivative of the cost function with respect to θ_k^i.
- So here we regularize every parameters (no longer x_0 parameter) so no special case update rule.

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

Video Question: In the algorithm we described, we initialized $x^{(1)}, \dots, x^{(n_m)}$ and $\theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values. Why is this?

- This step is optional. Initializing to all 0's would work just as well.
- Random initialization is always necessary when using gradient descent on any problem.
- This ensures that $x^{(i)} \neq \theta^{(j)}$ for any i, j$.

> This serves as symmetry breaking (similar to the random initialization of a neural network's parameters) and ensures the algorithm learns features $x^{(1)}, \dots, x^{(n_m)}$ that are different from each other.