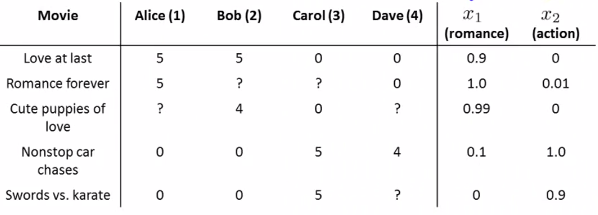
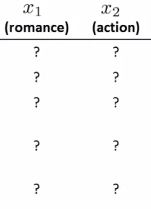
***Collaborative Filtering***

**I. COLLABORATIVE FILTERING**

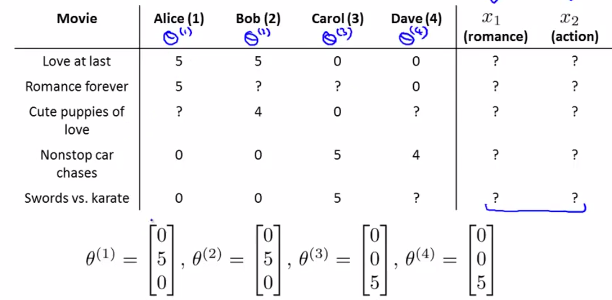
* Another approach to building a recommender system is called **collaborative filtering** which has a very interesting property called **feature learning 🡪** algorithm that can start to learn for itself what features to use.
* Here was the data set that we had + we assumed that for each movie, someone told us how romantic a movie was or how much action there was in a movie.



* It can be very difficult, time-consuming, + expensive to get someone to watch each movie + tell you these things, + often you'll want even more features than just these 2.
* So where do you get these features from?
* Let's change the problem a bit + suppose we have a data set where we don’t know the values of these features (given the movies + how users rated, but that’s it)



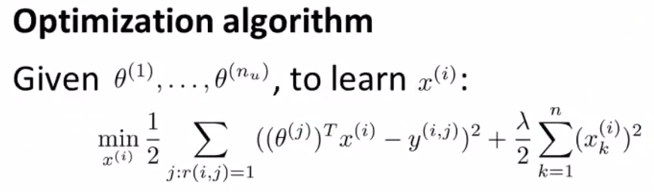
* Now let's make a slightly different assumption: say we've gone to each user (each associated w/ a Ө) + they told us how much they like romantic movies + how much they like action packed movies.
* Say Alice really likes romantic movies (5x multiplier associated w/ x1) + really doesn't like action movies (0x multiplier for x2)
* Bob tells us something similar, Carol really likes action
* Remember there's X0 = 1 for all
* So assume somehow we can go to users + each user j tells us the value of ӨJ for them.

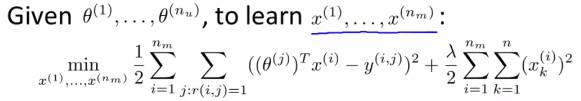
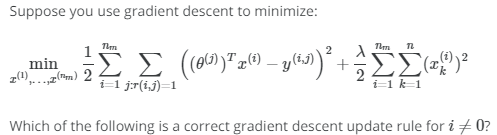


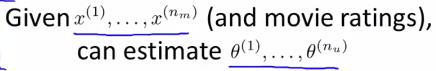
* This basically specifies to us how much they like different types of movies.
* If we can get these parameters Ө from users, it becomes possible to try to infer values of x1 + x2 for each movie.
* Ex: movie 1 has associated w/ it a feature vector x1. All we know is that Alice + Bob loved this movie, + Carol + Dave hated this movie.
* We know from the feature vectors Ө that Alice + Bob love romantic movies b/c they told us that
* Whereas Carol + Dave hate romantic movies + love action movies.
* Based on the fact movie 1 is loved by Alice + Bob + hated by Carol + Dave, we reasonably conclude this is probably a romantic movie + probably not much of an action movie.
* This example is a mathematically simplified, but what we're really asking is what feature vector should X1 be so Ө1(t)\*x1 is approximately = 5 (Alice's rating) + Ө2(t)\*x1 is also approximately = 5, Ө3(t)\*x1 is approximately = 0, + Ө4(t)\*X1 is approximately = 0.
* From this it looks like, you know, X1 equals this:



* Where 1 is the intercept term, + then 1.0 + 0.0 make sense given what we know of Alice, Bob, Carol, + Dave's preferences + the way they rated this movie
* More generally, we can go down the list of movies + try to figure out what might be reasonable feature values for other movies as well.
* To formalize this problem of learning the features X(i) + say our users have given us their preferences/values for Ө1 – Өn(u) + we want to learn the feature vector x(i) for movie number (i).



* We can pose the above optimization problem 🡪 sum, over all indices j for which we *actually* have a rating for movie (i) [ **r(i, j)**, b/c trying to learn features for movie i 🡪 feature vector x(i) ]
* Want to do is minimize the squared error 🡪 choose features x(i) such that the predicted value of user j rating for movie I will be similar to the *actual* value Y(i, j) actually observed
* To summarize, this term tries to choose features x(i) so that for all users j that HAVE rated that movie i, the algorithm predicts a value for that user j’s rating for movie i is not too far (in the squared error sense) from the actual value the user j rated that movie i.
* As usual, can also a regularization term to prevent features from becoming too big.
* This is how we learn the features for 1 specific movie, but we want to learn ALL features for ALL movies
* 
* Add an extra summation over all i = 1-n(m) + minimize our original objective over all movies
* If you minimize this, you hopefully have a reasonable set of features for all movies.
* 
* 
* Before, if we HAVE the set of movie ratings (the r(i, j)'s + the Y(i, j)'s, then given the features for different movies, we can learn the parameters Ө.
* So, if you *knew* the features, you can learn the parameters Ө for different users.



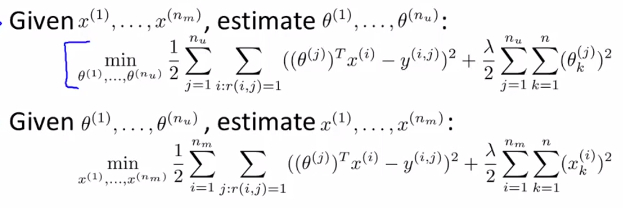
* If users are willing to *give you* the parameters, you can estimate features for different movies.



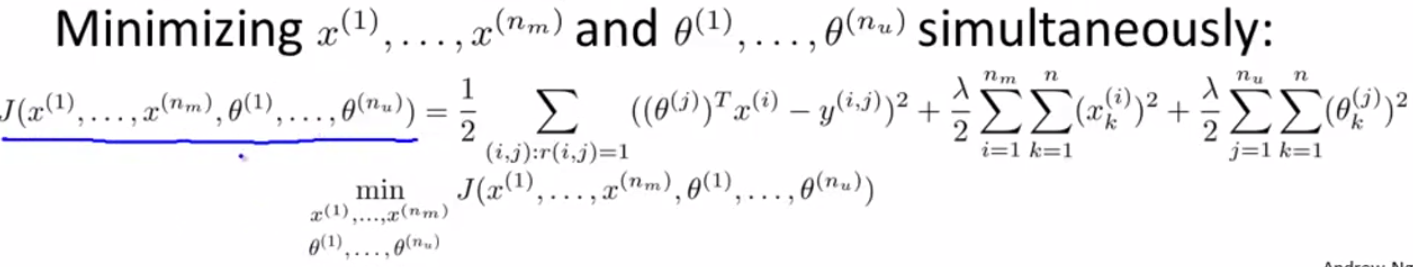
* This is kind of a chicken + egg problem. Which comes first?
* If we can get the Ө’s, we can learn the X’s. If we have the X’s, we can learn the Ө’s
* What you can do is in fact *randomly guess* some value of the Ө’s + based on this initial random guess, use the 2nd procedure to learn features for different movies.
* Now, given this initial set of features for movies from our guessing, you can use the 1st method to try to get an even better estimate for parameters Ө.
* And then, since you now have a better setting of the parameters Ө for users, use *that* to get an even *better* set of features, + so on.
* Can keep iterating, + optimizing Ө, x, Ө, x, Ө, + this will actually cause the algorithm to converge to a reasonable set of features for movies + a reasonable set of parameters for users.
* This was a basic collaborative filtering algorithm + isn't actually the *final* algorithm

**II. COLLABORATIVE FILTERING ALGORITHM**

* Now we know if given features for movies you can solve the minimization problem to find the parameters Ө for users + if given the parameters Ө, you can *that* to estimate the features x by solving a different minimization problem.



* 1 thing you could do is go back + forth, randomly initialize parameters + then solve for Ө, solve for x, solve for Ө, solve for x, etc.
* There is a more efficient algorithm that doesn't need to go back + forth between x's + Ө’s, but can solve for Ө + x *simultaneously*
* We basically take both of these optimization objectives, + put them into the same objective.



* Define the new optimization objective J, a cost function of both my features x + of my parameters Ө
* Notice the squared error term is the very, very similar in the 2 separate optimizations
* The 1st is the sum over all users j + over all movies *actually rated* by each user 🡪 summing over all pairs r(i, j) = a movie that was rated by a user.
* “For every user, sum over all movies rated by that user”
* This 2nd summation does things in the opposite order 🡪 for every movie i, sum over all users j that actually rated that movie
* Both of these are just summations over all pairs r(i, j) for which r(i,j) = 1 🡪 over all user-movie pairs for which we have a rating
* We’ve defined a *combined* optimization objective we want to minimize in order to solve for x + Ө simultaneously
* The other terms in the optimization objective are regularization in terms of Ө + in terms of the x's
* This new optimization objective J has an interesting property 🡪 if you were to hold the x's constant + just minimize w/ respect to the Ө’s, you'd be solving the 1st separate optimization problem
* Whereas if you were to do the opposite + hold the Ө’s constant + minimize J only w/ respect to the x's, it becomes equivalent to the 2nd separate optimization function.
* In this new version, instead of sequentially going between the 2 sets of parameters x + Ө, just minimize w/ respect to *both* sets of parameters simultaneously.
* 1 last detail 🡪 when learning features previously, we used the convention of a feature x0 = 1 that corresponds to an intercept term.
* When using this sort of formalism, we're are actually *learning* the features + are going to do away w/ this convention
* So the features we learn, x, will be in R(n), whereas previously we had features x in R(n + 1) b/c we’re including the intercept term.
* By getting rid of x0, we now have x in R(n) + b/c the parameters Ө are in the same dimension, we now also have Ө in R(n) 🡪 there's no x0, there's no need for a parameter Ө0 either
* The reason we do away w/ this convention is b/c we're now learning *all* the features so there is no need to hard code the feature that’s always = 1
* B/c if the algorithm *really* wants a feature always = 1, it can choose to learn one for itself (can set feature X1 = 1)
* The algorithm now has the flexibility to just learn it by itself.
* So, putting everything together for our collaborative filtering algorithm:
* 1) Initialize x + Ө to small random values (a bit like NN training where we also initialize all parameters of a NN to small random values)
* 2) Minimize the cost function using gradient descent or 1 of the advance optimization algorithms
* So, if you take derivatives, you find gradient descent updates = the partial derivative of the cost function w/ respect to the feature value x(i)(k) + a partial derivative value of the cost function w/ respect to the parameter Ө
* Just as a reminder, in this formula, we no longer have X0 = 1 + so x + Ө are in R(n).
* In this new formalism, we're regularizing *every one* of our parameters Ө + x + there's no longer the special case Ө0 (which was regularized different/was not regularized compared to the parameters Ө1-Ө(n))
* That’s why we do not break out a special case for k = 0.
* 3) Finally, given a user, if a user has some parameters, Ө + if there's a movie w/ some sort of learned features x, we predict movie’s rating by that user of Өj(t)\*x
* If user j has not yet rated movie i, we predict user j is going to rate movie i according to Өj(t)\*X(i)
* That's the collaborative filtering algorithm
* If you implement this, you get a pretty decent algorithm that will simultaneously learn good features for all movies + learn parameters for all users + hopefully give pretty good predictions for how different users will rate different movies that they have not yet rated
* 
* 