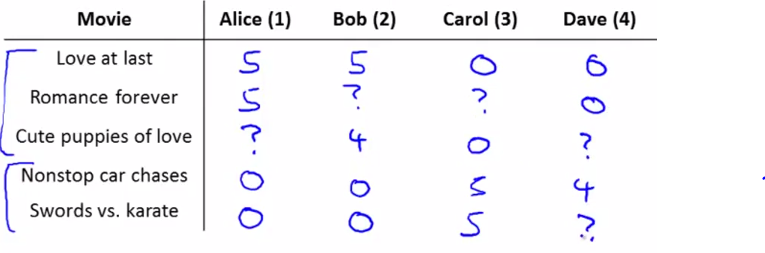
***Predicting Movie Ratings***

**I. PROBLEM FORMULATION**

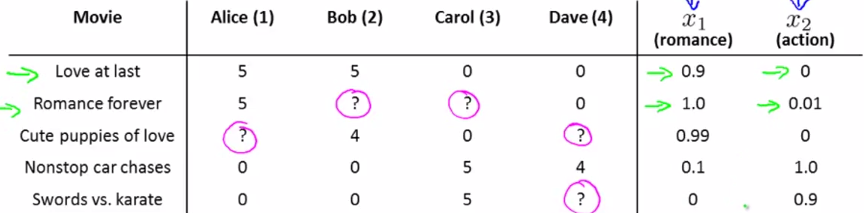
* **Recommender systems** are an important application of ML
* Many groups out in Silicon Valley are trying to build better recommender systems: Amazon, Netflix eBay, iTunes Genius
* These sorts of recommender systems look at what books you may have purchased in the past, or what movies you have rated in the past
* The systems are responsible for a substantial fraction of Amazon's revenue + for movies watched by Netflix users.
* An improvement in performance of a recommender system can have a substantial + immediate impact on the bottom line of many of these companies.
* Recommender systems are kind of a funny problem
* W/in academic ML, the problem of recommender systems actually receives relatively little attention, or at least a smaller fraction of what goes on w/in academia.
* But for many tech companies, the ability to build these systems seems to be a high priority
* There's this big idea in ML that, for some problems, there are algorithms that can try to *automatically learn a good set of features for you*.
* So rather than trying to hand design/code features, there are a few settings where you might be able to have an algorithm just learn what feature to use + a recommender system is just 1 example of that sort of setting.
* Modern problem: predicting movie ratings 🡺 4 users rate different 5 movies using a 0-5 star rating



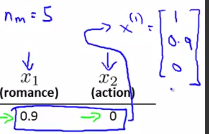
* Use n(u) = 4 to denote # of users + n(m) = 5 to denote the # of movies.
* It looks like Alice + Bob give high ratings to romcoms or movies about love + very low ratings to action movies
* For Carol and Dave, it's the opposite 🡪 really like the action movies + give them high ratings, but don't like romance + love-type movies as much.
* Our data comprises the following: we have these values r(i, j), which = 1 if user j has rated movie i.
* We also have a number y(i, j), which is the *actual rating* given by user j to movie I 🡪 a # from 0-5
* This is only defined when r(i, j) = 1
* The recommender system problem is given these r(i, j)'s + y(i, j)'s to look through the data + all the missing movie ratings to try to predict what these missing ratings should be.
* In the particular example, I have a very small number of movies + very small number of users, so most users have rated most movies
* In realistic settings, users may have rated only a minuscule fraction of movies
* Our job in developing a recommender system is to come up w/ a learning algorithm that can automatically fill in these missing values for us so that we can look at movies a user has not yet watched + recommend new movies to that user to watch.

**II. CONTENT-BASED RECOMMENDATIONS**

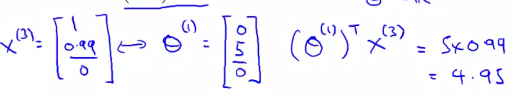
* Our 1st approach to building a recommender system is called **content based recommendations**.
* Suppose that for each movie, we have a set of 2 features, x1 🡪 measures the degree to which a movie is a romantic movie + x2 🡪 measures the degree to which a movie is an action movie



* If we have features like these, each movie can be represented w/ a feature vector w/ an intercept term x0

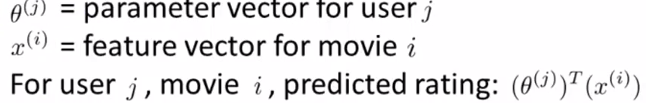


* In these, n =2 b/c we have 2 features x1 + x2 capturing degree of romance + of action in each movie
* In order to make predictions, 1 thing we do is treat predicting ratings of each user as a *separate linear regression* problem (1 for each user)
* Specifically, say that for each user j, we're going to learn the parameter vector Өj in R3
* Ө(j) would be in R (n+1), where n = # of features not counting intercept term
* We're going to predict user j as rating movie i w/ just the inner product between parameters vector Ө + the features x(i).
* Associated w/ user 1 Alice would be some parameter vector Ө1 + our 2nd user, Bob, will be associated w/ a different parameter vector Ө2, etc.
* You want to make a prediction for what Alice will think of the movie “Cute puppies of love”.
* That movie is going to have some parameter vector x3 = [1, 0.99, 0]
* Say we've somehow already gotten a parameter vector Ө1 for Alice (see later) via some unspecified learning algorithm learning it 🡪 Ө1 = [0, 5, 0]
* So our prediction for this entry is going to be Ө1(t)\*x3, inner product between these2 vectors



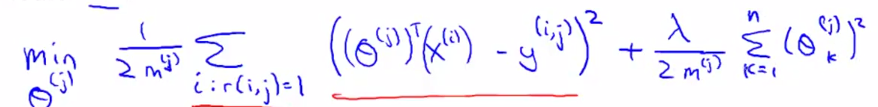
* Maybe that seems like a reasonable value if indeed this is my parameter vector Ө1.
* All we're doing is applying a different copy of linear regression for each user + Alice has some parameter vector Ө1 we use to predict her ratings as a function of how romantic + how action packed a movie is
* Bob + Carol + Dave each of have a different linear function of the romantic-ness + action-ness + that that's how we predict their ratings.
* A reasonable value for Ө3 would be [0, 0, 5] (see ratings above)
* More formally, here's how we can write down the problem.



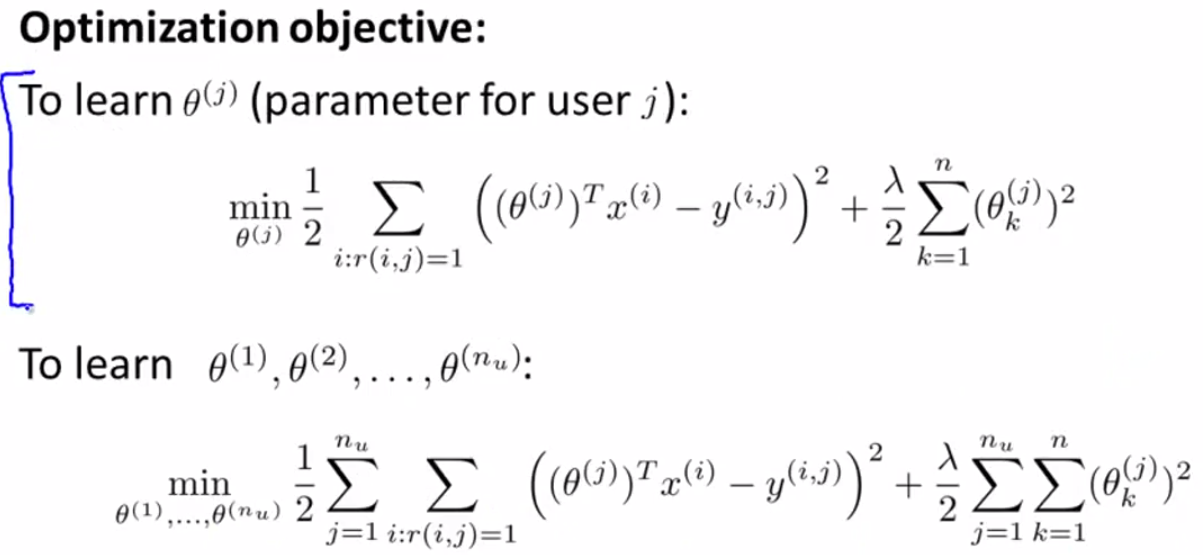




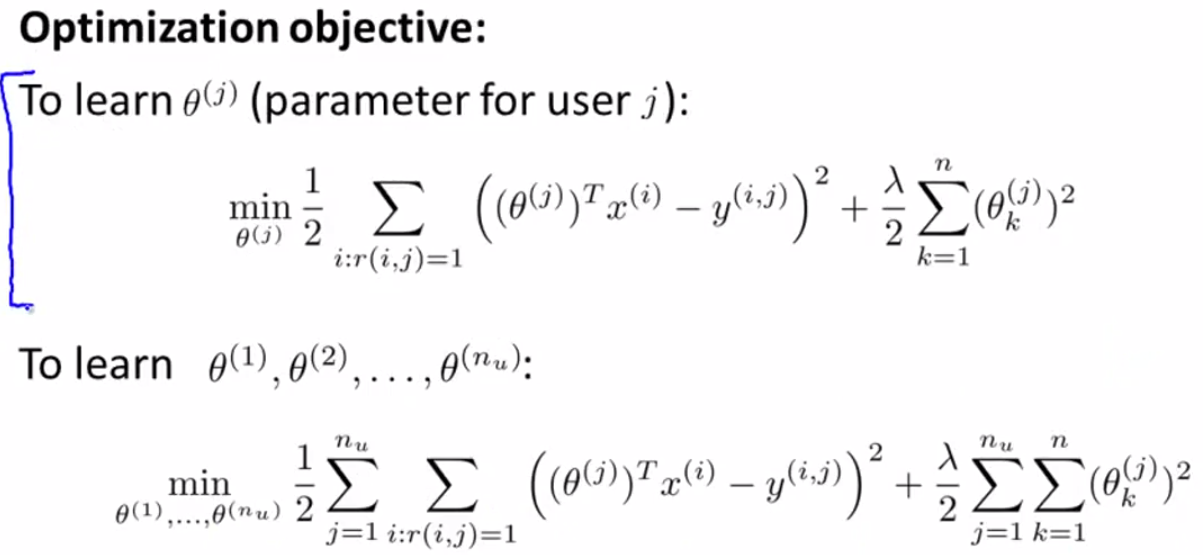
* In order to learn the parameter vector Өj, how do we do so? 🡪 This is basically a linear regression problem.
* We can choose a parameter vector Өj so that the predicted values Өj(t)\*x(i) are as close as possible to the values observed in our training sets/our data.
* So, in order to learn the parameter vector Өj, minimize (over the parameter vector Өj) the sum over all movies user j (AKA over all values of i conditioned on r(i, j) = 1) of the computed Өj(t)\*x(i) (the prediction of user j's rating on movie i) minus - y(i, j)^2, the actual observed rating squared
* Then take this result + divide by the # of movies user j has actually rated 🡪 2m(j)



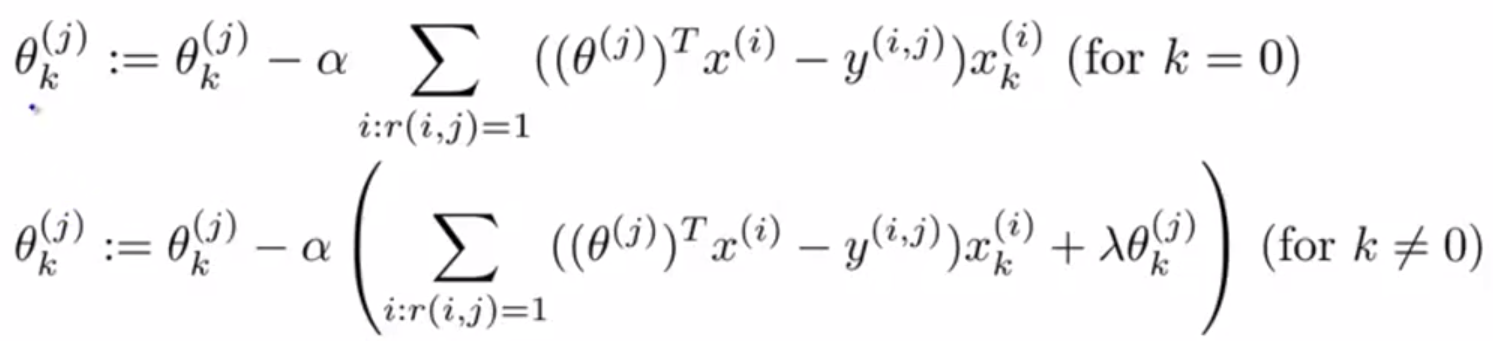
* This is just like the **least squares** **linear regression** where we want to choose the parameter vector Өj to minimize a squared error term
* If you want, you can add that **regularization term 🡪** lambda over 2\*the m(j) examples (the data points w/ which to fit the parameters of Өj)
* Өj is going to be an (n + 1)-dimensional vector, where n = the # of features we have per movie + we don't regularize over Ө0/over the **bias terms** so the sum is from k = 1 through n.
* If you minimize this as a function of Өj, you get a pretty good estimate of a parameter vector Өj w/ which to make predictions for user j's movie ratings.
* For recommender systems, we change this notation a bit to simplify the subsequent math, get rid of the m(j)
* It’s a constant so we can delete it w/out changing the value of Өj we get out of this optimization.
* Think of it as taking this whole expression + multiplying it by m(j) get rid of that constant
* When I minimize this, I should still get the same value of Өj as before.



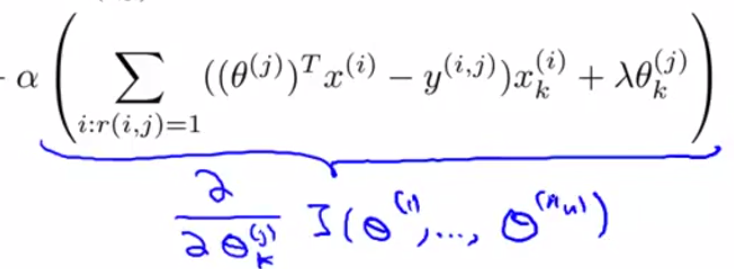
* In order to learn Өj, the parameter for user j, minimize, over Өj, the optimization objectives: our usual squared error term + our regularizations term.
* Now of course in building a recommender system, we don't just want to learn parameters for a *single* user, but for all users 🡪 n(u) users



* We take the optimization objective + add an extra summation for j = 1 to n(u) for both terms
* This sums the objective over all users + then minimize this *overall* optimization objective/cost function
* When I minimize this as a function of Ө1, Ө2, up to Өn(u), I get a separate parameter vector for each user + I can then use that to make predictions for all n(u) users.
* Putting everything together, our optimization objective = **J(Ө1, ..., Өn(u)).**
* Next, in order to actually DO the minimization, derive the gradient descent update:



* There's slightly different cases when k = 0 + when k != 0, b/c our regularization term regularizes only values of Өj(k) for k != 0 (don't regularize Ө0)
* Remember the term alpha/learning rate is multiplied by is just the partial derivative w/ respect to your parameter of your optimization objective.



* These gradient descent updates are essentially the same as in linear regression, but for linear regression we have these 1/m(j), but here, we got rid of it
* Now we just have the sum of squared errors of the training examples multiplied by x(k) + the regularization term that contributes to the derivative
* So, if using gradient descent, this is how you can minimize the cost function J to learn all the parameters
* Using these formulas for the derivative, if you want, you can also plug them into a more advanced optimization algorithm, like conjugate gradient, LBFGS, etc. to try to minimize cost J
* So, you can apply essentially a deviation on linear regression in order to predict different movie ratings by different users.
* This particular algorithm is called a **content based recommendations**, or a **content based approach**, b/c we assume we have features that capture the content for different movies available to us we use to make our predictions.
* But for many movies, we don't actually have such features, or it may be very difficult to get such features for all our movies, for all items we're trying to sell, etc.
* There is an approach to recommender systems that *isn't* content based + does NOT assume we have someone else giving us all of these features for all of the movies in our data set.