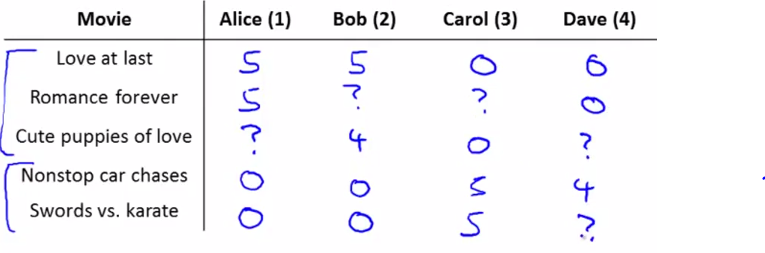
***Predicting Movie Ratings***

**I. PROBLEM FORMULATION**

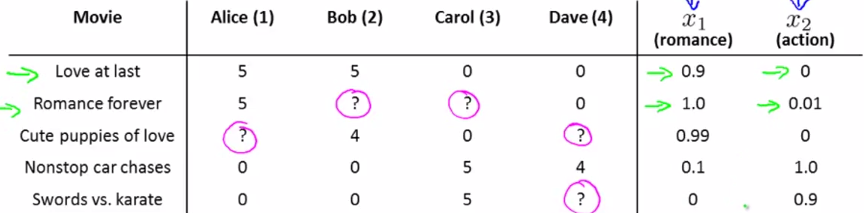
* **Recommender systems** are an important application of ML
* Many groups out in Silicon Valley are trying to build better recommender systems: Amazon, Netflix eBay, iTunes Genius
* These sorts of recommender systems look at what books you may have purchased in the past, or what movies you have rated 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



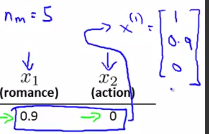
* Use n(u) = 4 to denote # of users + n(m) = 5 to denote the # of movies.
* It looks like Alice + Bob give high ratings to romcoms or movies about love + very low ratings to action movies
* For Carol and Dave, it's the opposite 🡪 really like the action movies + give them high ratings, but don't like 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**

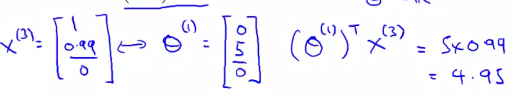
* Our 1st approach to building a recommender system is called **content based recommendations**.
* Suppose that for each movie, we have a set of 2 features, x1 🡪 measures the degree to which a movie is a romantic movie + x2 🡪 measures the degree to which a movie is an action movie



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

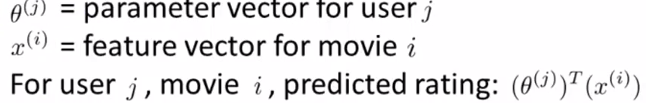


* In these, n =2 b/c we have 2 features x1 + x2 capturing degree of romance + of action in each movie
* In order to make predictions, 1 thing we do is treat predicting ratings of each user as a *separate linear regression* problem (1 for each user)
* Specifically, say that for each user j, we're going 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



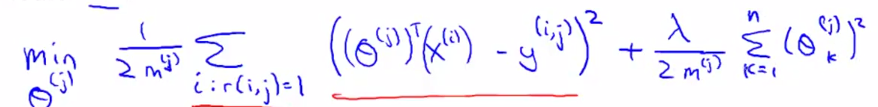
* Maybe that seems like a reasonable value if indeed this is my parameter vector Ө1.
* All we're doing is applying a different copy of linear regression for each user + Alice has some parameter vector Ө1 we use to predict her ratings as a function of how romantic + how action packed a movie is
* Bob + 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.
* So 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 little bit. So to simplify the subsequent math,
* I w/ to get rid of this term mj. So that's just a constant, right? So I can delete it w/out changing
* the value of Ө j that I get out of this optimization. So if you imagine taking this whole
* equation, taking this whole expression + multiplying it by mj,
* get rid of that constant. + when I minimize this, I should still
* get the same value of Ө j as before. So just to repeat what we
* wrote on the previous slide, here's our optimization objective. In order to learn Ө j
* which is the parameter for user j, we're going to minimize over
* Ө j of this optimization objectives. So this is our usual squared error term
* + then this is our regularizations term. Now of course in building
* a recommender system, we don't just want to learn parameters for
* a single user. We want to learn parameters for
* all of our users. I have n subscript u users, so
* I want to learn all of these parameters. + so, what I'm going to do is take
* this optimization objective + just add the mixture summation there. So this expression here w/ the one
* half on top of this is exactly the same as what we had on top. Except that now instead of just doing
* this for a specific user Ө j, I'm going to sum my objective
* over all of my users + then minimize this overall optimization
* objective, minimize this overall cost on. + when I minimize this as
* a function of Ө 1, Ө 2, up to Ө nu, I will get a separate
* parameter vector for each user. + I can then use that to make
* predictions for all of my users, for all of my n subscript users. So putting everything together, this
* was our optimization objective on top. + to give this thing a name, I'll
* just call this J(Ө1, ..., Ө nu). So j as usual is my optimization
* objective, which I'm trying to minimize. Next, in order to actually do
* the minimization, if you were to derive the gradient descent update, these
* are the equations that you would get. So you take Ө j, k, +
* subtract from an alpha, which is the learning rate,
* times these terms over here on the right. So there's slightly different cases when
* k equals 0 + when k does not equal 0. B/c our regularization term here
* regularizes only the values of Ө jk for k not equal to 0, so
* we don't regularize Ө 0, so w/ slightly different updates when
* k equals 0 + k is not equal to 0. + this term over here, for example, is just the partial derivative
* w/ respect to your parameter, that of your optimization objective. Right + so
* this is just gradient descent + I've already computed the derivatives +
* plugged them into here. + if this gradient descent update
* look a lot like what we have here for linear regression. That's b/c these are essentially
* the same as linear regression. The only minor difference is that for
* linear regression we have these 1 over m terms,
* this really would've been 1 over mj. But b/c earlier when we are deriving
* the optimization objective, we got rid of this, that's why we
* don't have this 1 over m term. But otherwise, it's really some of
* my training examples of the ever times xk plus that regularization term, plus that term of regularization
* contributes to the derivative. + so if you're using gradient
* descent here's 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 or
* LBFGS or what have you. + use that to try to minimize
* the cost function j as well. So hopefully you now know how 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 that we have available
* to us features for the different movies. + so where features that capture
* what is the content of these movies, of how romantic is this movie,
* how much action is in this movie. + we're really using features
* of a content of the movies to make our predictions. But for many movies,
* we don't actually have such features. Or maybe very difficult
* to get such features for all of our movies, for all of
* whatever items we're trying to sell. + so, in the next video, we'll start
* to talk about an approach to recommender systems that isn't content based +
* does not assume that we have someone else giving us all of these features for

all of the movies in our data set.