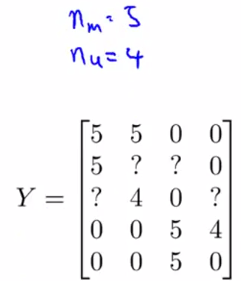
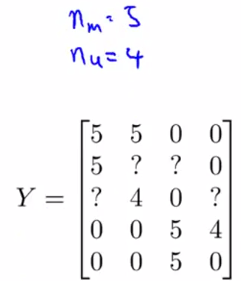
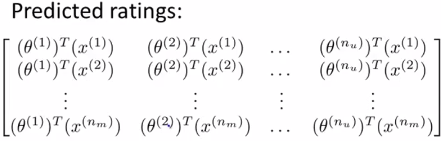
***Low Rank Matrix Factorization***

**I. VECTORIZATION: LOW RANK MATRIX FACTORIZATION**

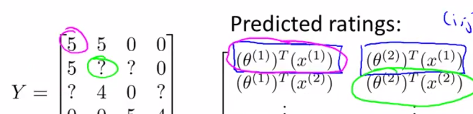
* There is a vectorization implementation of the collaborative filtering algorithm
* There are also other things you can do w/ this algorithm such as, given a product, find other products related to it
* Ex: User has recently been looking at a product, are there other related products you could recommend to this user?
* We will look at an alternative way of writing out predictions of the collaborative filtering algorithm
* From our dataset, take all the movie ratings (5) by all the users (4) + group them into a 5x4 matrix.



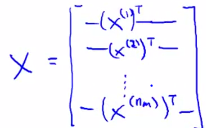
* Given this matrix Y of all the ratings we have, there's an alternative way of writing the predicted ratings of the algorithm



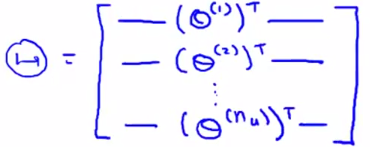
* In particular, if you look at what a certain user j predicts on a certain movie i, it’s given by this formula
* The (i, j)th entry corresponds to the rating we predict user j will give to movie i and this is exactly equal to that Өj(t)\*x(i)

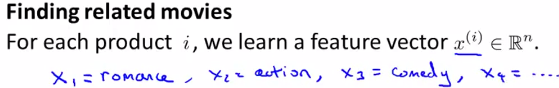


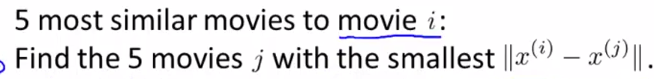
* Given this matrix of predictive ratings, there is then a simpler vectorized way of writing these out.
* Define a matrix X of x1(t) to x(n)(m)(t) 🡪 take all feature values for the movies + stack them in rows



* Then find a matrix Ө + take each user parameter vector + stack them in rows 🡪 Ө1 = parameter vector for 1st user, Ө2 = for 2nd user + end up w/ an n(u) by 1 parameter vector

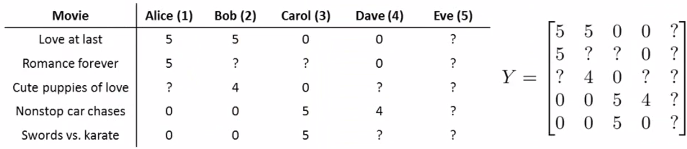


* Given these matrix definitions, to get a vectorized way of computing the predicted ratings matrix, just compute X\*Ө(t)
* This algorithm = **low-rank matrix factorization 🡪** X\*Ө(t) is a low-rank matrix
* Having run the collaborative filtering algorithm, something else we can do is *use learned features* in order to find *related* movies.
* For each movie/product i, we learned a feature vector x(i) w/out knowing in advance what the different features are going to be
* If you run the algorithm correctly, the features will tend to capture the important aspects of different movies/products that cause some users to like some movies + cause some users to like different movies.
* Ex: x1 = romance, x2 = action, x3 = degree to which it is a comedy, etc.
* 
* We get n features all together
* After you have learned features, it's often pretty difficult to go in to the learned features + come up w/ a human-understandable interpretation or visualization of what these features really are.
* But usually, the algorithm will still learn features that are very meaningful for capturing the most important/salient properties of a movie that causes one to like or dislike it.
* Say you have some specific movie i + you want to find other movies j related to that movie.
* Maybe you have a user browsing movies, + they're currently watching movie j, so what's a reasonable movie to recommend to them to watch after they're done w/ movie j?
* Now that we’ve learned these feature vectors, we have a very convenient way to measure how similar 2 movies are.
* In particular, movie i has a feature vector x(i), so if you can find a different movie j such that the distance between x(i) + x(j) is small, it’s a pretty strong indication that movies j + i are somehow similar (at least in the sense that those who like movie i are more likely to like movie j as well)

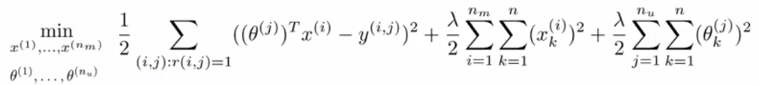


**II. IMPLEMENTATIONAL DETAIL: MEAN NORMALIZATION**

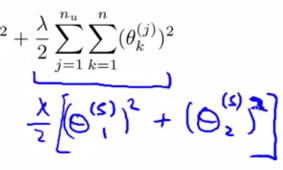
* Now you've seen all the main pieces of the recommender system/collaborative filtering algorithm.
* There is 1 last implementational detail, **mean normalization**, which can sometimes make the algorithm work a little bit better.
* To motivate the idea of mean normalization, consider an example of a user that hasn’t rated *any* movies 🡪 a 5th user, Eve



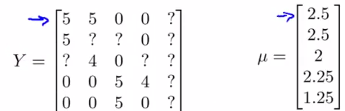
* Let's say n = 2 (i.e. we're going to learn 2 features + have to learn a parameter vector Ө5, which is going to be in R(2) for user 5, Eve)

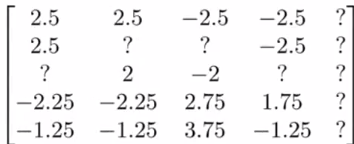


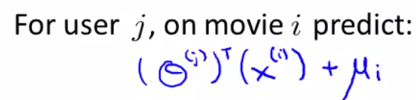
* In the 1st term in this optimization objective, user 5 hasn't rated any movies, so there are no movies for which r(i, j) = 1 🡪 1st term (sum of squared errors) + its regularization terms (2nd term) play *no role at all in determining Ө5*
* So, we want to choose a vector Ө5 such that the last regularization term (3rd overall) is as small as possible 🡪 want to minimize this component/regularization term that corresponds to user 5



* If your goal is to minimize this term, you're going to end up w/ just Ө5 = 0.0
* This is b/c a regularization term is encouraging us to set parameters close to 0, + if there’s no data there to try to pull the parameters away from 0 b/c the 1st optimization term doesn't effect Ө5), we just end up w/ Ө5 = a vector of all 0’s
* So, when we go to predict how user 5 would rate any movie, we have that Ө5(t)\*x(i) = 0 for any i.
* Therefore, what we're going to predict is user 5 is going to rate every single movie = 0
* This doesn't seem very useful to just predict Eve is going to rate everything 0 stars.
* If we're predict she going to rate everything 0, we also don't have any good way of recommending any movies to her b/c all movies are getting the same predicted rating
* The idea of **mean normalization** will let us fix this problem.
* As before, group all movie ratings into a matrix Y w/ a final column all question marks for user 5.
* To perform mean normalization, compute the average rating each movie obtained + store that in a vector that we'll call µ



* Then look at all movie ratings + subtract off the mean rating.
* 
* This is normalizing each movie to have an average rating = 0
* Now each movie in this new Y matrix has an average rating = 0.
* Now *take this new set of ratings + use it w/ the collaborative filtering algorithm.*
* i.e. pretend the new Y matrix was the data gotten from the users + use it w/ which to learn parameters Өj + features X(i)
* To make predictions of movie ratings, do the following:
* For user j on movie i, predict ӨJ(t)\*X(I), where X + Ө are the parameters *learned from this mean normalized data set*.
* But, b/c in the data set we subtracted off the means in order to actually make a prediction on movie i, we need to add back in the mean µ(i).



* For user 5, the same argument as before in the sense that they had not rated any movies so the learned parameter for user 5 is still going to be equal = [0, 0].
* So for a particular movie i + for user 5, we predict Ө5(t)\*x(i) + µ(i)
* This 1st component w/ Ө5(t) is going to = 0, so on movie i, *we predict µ(i)*.
* This makes sense b/c it says if Eve hasn't rated any movies + we don't know anything about this new user, we just predict the average rating those movies got for each of the movies
* In case you have some movies w/ no ratings (analogous to a user who hasn't rated anything), you can w/ versions of the algorithm where you normalize the different *columns* to have means = 0 instead of normalizing *rows* to have mean = 0
* Although that's maybe less important, b/c if you really have a movie w/ no rating, maybe you shouldn't recommend that movie to anyone (unless it’s new)
* Taking care of the case of a user who hasn't rated anything might be more important than taking care of the case of a movie that hasn't gotten a single rating.
* That's how you do mean normalization as a sort of pre-processing step for collaborative filtering
* Depending on your data set, this might sometimes make your implementation work just a little bit better.