ColumbiaX: Machine Learning

Lecture 17

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

Another set of unsupervised models called matrix factorization models.

COLLABORATIVE FILTERING

OBJECT RECOMMENDATION

Motivation: object recommendation problem

Matching consumers to products is an important practical problem.

We can often make these connections using user feedback about subsets of products. To give some prominent examples:

- Netflix lets users to rate movies
- Amazon lets users to rate products and write reviews about them
- Yelp lets users to rate businesses, write reviews, upload pictures
- YouTube lets users like/dislike a videos and write comments

Recommendation systems use this information to help recommend new things to customers that they may like.

Specifically we wanna say, a user has not given me any feedback on this product. But I'm gonna assume that's because that user doesn't know anything about the product, so do I wanna match this user together with this product? What do I think this user would think about this product? Or what do I think this person would rate this movie?

Content filtering

One strategy for object recommendation is:

(Not the focus today)

Content filtering: Use known information about the products and users to make recommendations. Create profiles based on

- Products: movie information, price information, product descriptions
- Users: demographic information, questionnaire information

And then based on the profile of the user and the profile of the song, they're matched together to see how similar is the profile of each to each other.

Example: A fairly well known example is the online radio Pandora, which uses the "Music Genome Project."

- An expert scores a song based on hundreds of characteristics
- A user also provides information about his/her music preferences
- Recommendations are made based on pairing these two sources

Drawback:

But notice that this approach is not using any behavioral information. It's not using any of my listening behavior. It's not letting me rate particular songs or particular products or looking at my viewing behavior. It's not taking into consideration the actual patterns of how people are using these products etc etc. So that leads to the collaborative filtering approach.

COLLABORATIVE FILTERING

Content filtering requires a lot of information that can be difficult and expensive to collect. Another strategy for object recommendation is:

based on actual data, it's given by the user.

Collaborative filtering (CF): Use previous users' input/behavior to make future recommendations. Ignore any *a priori* user or object information.*

- CF uses the ratings of similar users to predict my rating.
- CF is a domain-free approach. It doesn't need to know what is being rated, just who rated what, and what the rating was.

One CF method uses a *neighborhood-based* approach. For example,

1. define a similarity score between me and other users based on how much our overlapping ratings agree, then
2. based on these scores, let others "vote" on what I would like.**

These filtering approaches are not mutually exclusive. Content information can be built into a collaborative filtering system to improve performance.

And it's going to ignore any sort of a priori object or
user specific information.
So it's not gonna care whether a user is young or old.
It's not gonna care whether a song is a rock song or classical song.
It's only going to purely base its recommendations on what
users give as feedback to a particular product.

① So notice the thing that's going on here is that
there is no background information.
The objects, the same exact approach can be used for any sort of a recommendation

② All that matters is the feedback that people have given to products.
And then we're trying to pair up people with other people in
order to recommend objects to each other.

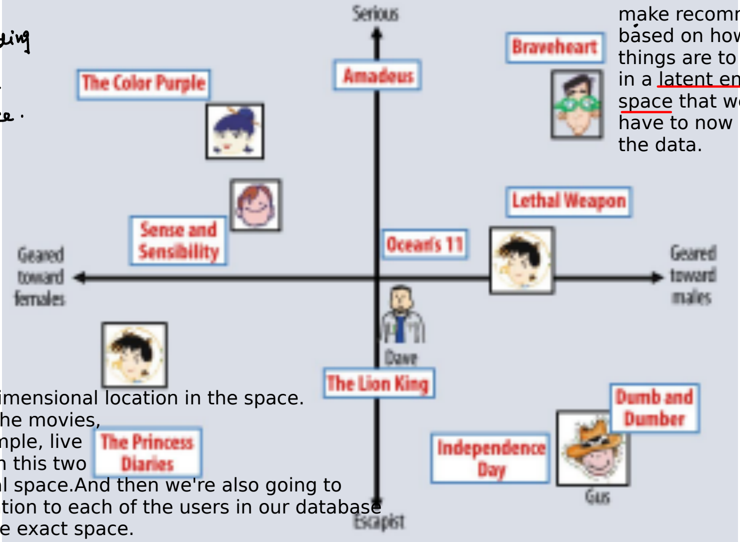
Location-based CF methods (intuition)

Location-based approaches embed users and objects into points in $\mathbb{R}^d$.[1]

A method for embedding objects in a latent space.

Recommendation:

make recommendations based on how close things are to each other in a latent embedding space that we have to now learn from the data. ?



Space:

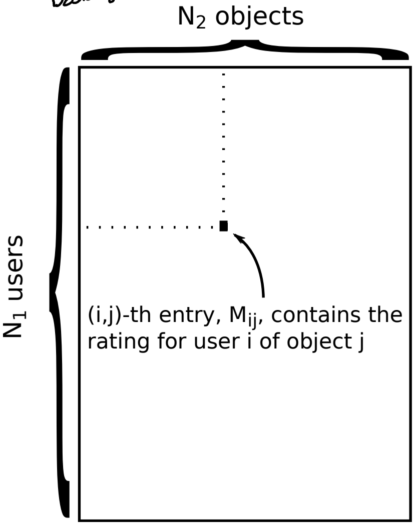
It's a two dimensional location in the space. And so all the movies, in this example, live at a point in this two dimensional space.And then we're also going to give a location to each of the users in our database in the same exact space.

[1] Koren, Y., Robert B., and Volinsky, C.. "Matrix factorization techniques for recommender systems." Computer 42.8 (2009): 30-37.

Matrix factorization

MATRIX FACTORIZATION

Data structure:



Matrix factorization (MF) gives a way to learn user and object locations.

First, form the rating matrix M:

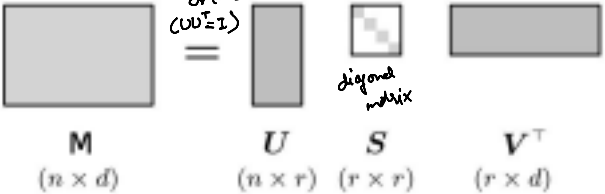
- Contains every user/object pair.
- Will have many missing values.
  Because each user can rate a tiny fraction of the products
- The goal is to fill in these missing values.
  tries to learn a low-rank factorization of this matrix using only the data we've observed and ignoring the data we don't have. And then imputing values in the matrix.

MF and recommendation systems:

- We have prediction of every missing rating for user i.
- Recommend the highly rated objects among the predictions.

Our goal is to factorize the matrix M. We've discussed one method already.

orthonormal columns
($UU^T = I$)



diagonal matrix

Singular value decomposition: Every matrix M can be written as $M = USV^T$, where $U^T U = I$, $V^T V = I$ and S is diagonal with $S_{ii} \geq 0$.

(A low rank matrix factorization can restrict the degrees of freedom of how many different types of things

$r = \text{rank}(M)$. When it's small, M has fewer "degrees of freedom." we can model in the matrix M.

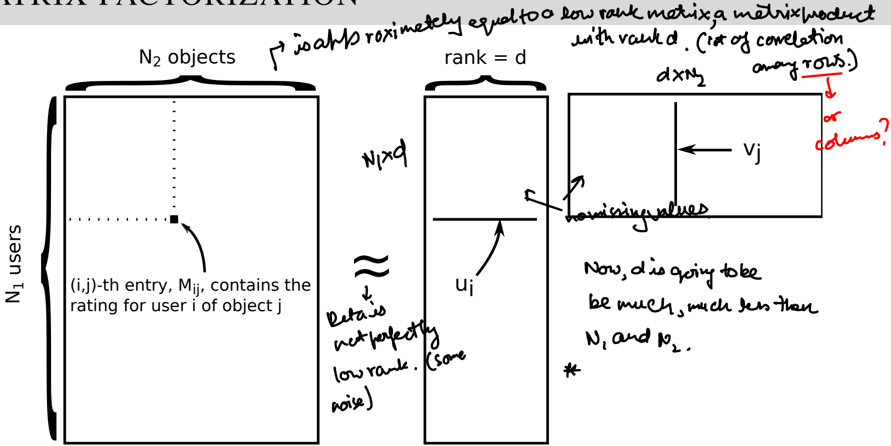
↑ why?

low rank → high correlation

Collaborative filtering with matrix factorization is intuitively similar.

So, this is going to be what we exploit, this idea of a low-rank matrix vectorization in the collaborative filtering problem.

MATRIX FACTORIZATION



We will define a model for learning a low-rank factorization of M. It should:

1. Account for the fact that most values in M are missing
2. Be low-rank, where $d \ll \min\{N_1, N_2\}$ (e.g., $d \approx 10$)
3. Learn a location $u_i \in \mathbb{R}^d$ for user i and $v_j \in \mathbb{R}^d$ for object j (dot product ith with jth)

* Example:

So if you imagine that we have hundreds of thousands of users, and
tens of thousands of objects, this will be a huge matrix.
But d is going to be something much smaller, so
it'll be something on the order of 10.
That'll say how many degrees of freedom we really have in this matrix.

Intuition:

Intuitively, you could almost think of it as saying how many
things are really taken into consideration when giving a movie a rating.

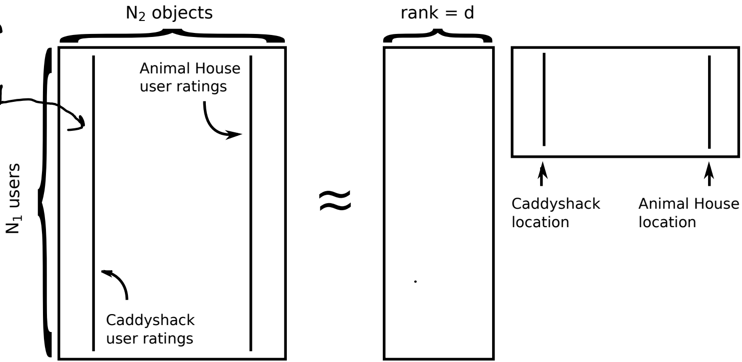
If d is equal to 10, then that's like saying there are 10 underlying
things that factor into a particular person's rating of a particular object.
But we aren't gonna define the meaning of what those are,
it's a latent sort of thing.

ij^{th} entry of M

Notice that the ijth entry in this matrix
can be viewed as the ith row of the left matrix
times the jth column of the right matrix.
So this is the rules that you have with matrix products,
that if you look at the entry ij here, that's approximately going to be
equal to the ith row of this matrix times the jth column of this right matrix.

LOW-RANK MATRIX FACTORIZATION

column has all of the ratings any user has given to Caddy Shack.



Why learn a low-rank matrix?

- We think that many columns should look similar. For example, movies like *Caddyshack* and *Animal House* should have **correlated** ratings. *
- Low-rank means that the N_1-dimensional columns don't "fill up" $\mathbb{R}^{N_1}$.
- Since $> 95\%$ of values may be missing, a low-rank restriction gives hope for filling in missing data because it models correlations.

*overlap (structure)

And their ratings also fall along the corresponding rows of this column.
And we can imagine that some of those rows are going to overlap,
the same user will have watched and rated both movies.
But also, some users watched one movie and
rated it while not watching the other, and vice versa.

Correlation:

Now if you know about these movies, you know that they're very similar.
They're from the same period of time,
they're both the same type of comedy movie.
And you can imagine, although it's not necessarily always the case,
you can imagine that if somebody really likes the movie Caddyshack,
they're also going to really like the movie Animal House.
And therefore, if a person rates Caddyshack highly,
they're also most likely going to rate Animal House highly.
On the flip side, if a person hates one of the movies,
they're also very likely going to hate the other movie.
And so we imagine that the ratings along these two columns
are very highly correlated, they look very similar to each other.

Low-rank factorizations do is learn, they strictly enforce this type of correlation.

So if we look at the column for Caddyshack and Animal House and
we have a rank d factorization, what we're saying is that this column
is equal to this d-dimensional vector times this matrix.
Whereas this column is approximately equal to this d-dimensional vector
times this matrix.
And so in that sense, because we are restricting all of the objects and
users to live in this d-dimensional space,
we're restricting what types of matrices that we can learn.
And we're forcing there to be correlations. ← how?

And so because we're only going to have a very tiny fraction of the ratings in this matrix, by enforcing this type of a low-rank, high-correlation factorization, we're going to be able to borrow information across movies in order to learn these locations. ← how?

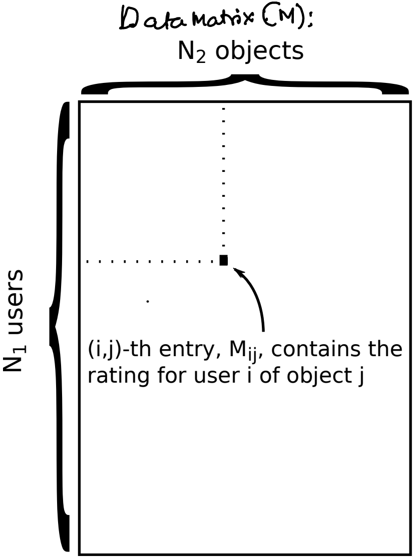
So we're gonna be able to borrow information across the users that rated Caddyshack in order to help predict what those users would rate for Animal House, and vice versa.

And we're actually going to take into consideration all of the correlations among all of movies in making these predictions

PROBABILISTIC MATRIX FACTORIZATION

(one particular model for learning a low-rank factorization in the missing data problem.)

Some notation



- Let the set Ω contain the pairs (i, j) that are observed. In other words,

$$\Omega = \{(i, j) : M_{ij} \text{ is measured}\}.$$

So $(i, j) \in \Omega$ if user i rated object j.

Just for notation:

- Let Ω_{u_i} be the index set of objects rated by user i.

- Let Ω_{v_j} be the index set of users who rated object j.

PROBABILISTIC MATRIX FACTORIZATION

does is, it assumes a generative model which why it is called probabilistic matrix factorization.

Generative model

For N_1 users and N_2 objects, generate

For each of the N_1 users, we assume their location is going to be generated from a zero mean Gaussian with some covariance.

Prior on all of the model variables

User locations: $u_i \sim N(0, \lambda^{-1} I), \quad i = 1, \ldots, N_1$

Object locations: $v_j \sim N(0, \lambda^{-1} I), \quad j = 1, \ldots, N_2$

And for each of N_2 objects we're also going to assume that their location is generated iid from the same Gaussian.

?

Given these locations the distribution on the data is

(Just a simple case of what types of priors we could use.)

data level distribution

some variance.

$$M_{ij} \sim N(u_i^T v_j, \sigma^2), \quad \text{for each } (i, j) \in \Omega .$$

dot product of (user location, object location) pair

?

Comments:

- Since M_{ij} is a rating, the Gaussian assumption is clearly wrong.* *
- However, the Gaussian is a convenient assumption. The algorithm will be easy to implement, and the model works well.***

* So these are the priors that we define on the respective locations of all the users and all of the objects in Rd.

Bad model assumption:

** → M_{ij} is going to be assumed to be a rating matrix. In cases, where it is a rating matrix, it's clear that the Gaussian assumption is wrong, because the ratings are discrete. They take values 1, 2, 3, 4, 5.

→ Whereas the Gaussian is a distribution on a continuous valued random variable that can take any value.

→ So, this is a bad model definition in the sense of being defined on the support of our data.

**

Because even though the data, it can only take one of say, five or ten values, they're still ordinal.
Meaning that order matters, the relationship of the rating 1 to 2 and 2 to 3, in a sense, means the same.
1 and 2 are the same distances, 2 to 3, and 1 and 3 is twice the distance, so intuitively that makes sense.
And so this Gaussian is going to be able to still model that relationship correctly.

Model inference (inverse problem of infering the model parameters from data.)

Q: There are many missing values in the matrix M. Do we need some sort of EM algorithm to learn all the u's and v's? (like in 2 lectures before.)

- Let M_o be the part of M that is observed and M_m the missing part. Then

maximise the likelihood of observations we've seen given u, v.

$$p(M_o|U,V) = \int p(M_o, M_m|U,V)dM_m.$$

integral over missing values of the complete matrix.

- Recall that EM is a ***tool*** for maximizing $p(M_o|U,V)$ over U and V. (marginal likelihood where there's no missing information.)
- Therefore, it is only needed when
  1. $p(M_o|U,V)$ is hard to maximize,
  2. $p(M_o, M_m|U,V)$ is easy to work with, and
  3. the posterior $p(M_m|M_o,U,V)$ is known.

A: If $p(M_o|U,V)$ doesn't present any problems for inference, then no. ?

(Similar conclusion in our MAP scenario, maximizing $p(M_o, U, V)$.)

for ex: if we can closed-form updates for MLE. Or in our case, it is going to be maximum a posteri over u and v of this likelihood. Then EM doesn't buy us anything

MODEL INFERENCE

To test how hard it is to maximize $p(M_o, U, V)$ over U and V, we have to

And then if we can do that without having to introduce EM.

1. Write out the joint likelihood
2. Take its natural logarithm
3. Take derivatives with respect to u_i and v_j and see if we can solve

(all things we want to update.)

user location for user i

object location for object j.

[over all users & objects]

The joint likelihood of $p(M_o, U, V)$ can be factorized as follows:

(Probabilistic matrix factorization model.)

$$p(M_o, U, V) = \underbrace{\left[\prod_{(i,j)\in\Omega} p(M_{ij}|u_i, v_j)\right]}_{\text{conditionally independent likelihood}} \times \underbrace{\left[\prod_{i=1}^{N_1} p(u_i)\right]\left[\prod_{j=1}^{N_2} p(v_j)\right]}_{\text{independent priors}}.$$

product over all the observed ratings

By definition of the model, we can write out each of these distributions.**

* Rating user i gives to object j given the user location and object location for that rating, times the priors on these locations.

** So remember the user locations were all iid from a distribution as were the object locations. So the prior factorizes into a product over the individual priors. ← ?

And by def. we can write each of these. *, ** is Gaussian

We take this likelihood, we take its log, and we get the objective we're trying to maximize.

Log joint likelihood and MAP

why are these priors?

The MAP solution for U and V is the maximum of the log joint likelihood

$$U_{\text{MAP}}, V_{\text{MAP}} = \arg\max_{U,V} \underbrace{\sum_{(i,j)\in\Omega} \ln p(M_{ij}|u_i, v_j)}_{\text{log of each individual measurement given the locations for the user and object}} + \underbrace{\sum_{i=1}^{N_1} \ln p(u_i) + \sum_{j=1}^{N_2} \ln p(v_j)}_{\text{sum over logs of priors}}$$

Plug in what these distributions are:

Calling the MAP objective function $\mathcal{L}$, we want to maximize

$$\mathcal{L} = \underbrace{- \sum_{(i,j)\in\Omega} \frac{1}{2\sigma^2} \|M_{ij} - u_i^T v_j\|^2}_{\text{negative sum of squared error term}} \underbrace{- \sum_{i=1}^{N_1} \frac{\lambda}{2} \|u_i\|^2 - \sum_{j=1}^{N_2} \frac{\lambda}{2} \|v_j\|^2}_{\text{additional regularizations.}} + \text{constant}$$

log of joint likelihood

scalar

The squared terms appear because all distributions are Gaussian.

Don't know either u_i & v_j unlike regression.

independent Gaussian priors with 0 mean. And so that's where these terms come from.

MAXIMUM A POSTERIORI

To update each u_i and v_j, we take the derivative of $\mathcal{L}$ and set to zero.

sum over all the objects user i has rated.

additional prior on u_i

$$\nabla_{u_i}\mathcal{L} = \sum_{j\in\Omega_{u_i}} \frac{1}{\sigma^2}(M_{ij} - u_i^T v_j)v_j - \lambda u_i = 0$$

derivative with respect to prior.

$$\nabla_{v_j}\mathcal{L} = \sum_{i\in\Omega_{v_j}} \frac{1}{\sigma^2}(M_{ij} - v_j^T u_i)u_i - \lambda v_i = 0$$

[because of the specific model structure we assumed.]

We can solve for each u_i and v_j individually (therefore EM isn't required),

User i location can be updated using this equation

$$u_i = \left(\lambda\sigma^2 I + \textstyle\sum_{j\in\Omega_{u_i}} v_j v_j^T\right)^{-1}\left(\textstyle\sum_{j\in\Omega_{u_i}} M_{ij}v_j\right)$$

what structure? different from before?

Object j's location can be updated using this equation

$$v_j = \left(\lambda\sigma^2 I + \textstyle\sum_{i\in\Omega_{v_j}} u_i u_i^T\right)^{-1}\left(\textstyle\sum_{i\in\Omega_{v_j}} M_{ij}u_i\right)$$

→ interpretation.

However, we can't solve for all u_i and v_j at once to find the MAP solution. Thus, as with K-means and the GMM, we use a coordinate ascent algorithm.

Update of u_i that maximise that objective:

* sum of the outer product of all of the locations of the object that user i has rated. (Invert this matrix)

** vector we get by taking the location of each object that user has rated. And multiplying it by the rating and summing those up.

However, notice that if we wanted to solve for all ui's and
all vj's simultaneously, we would not be able to do that.
Because the update for ui involves v and
the update for vj involves u.
So we can't say that this value for ui or this value for vj is the optimal one.
Because it depends on the other parameter settings that we have.
So what we're gonna end up having is a coordinate ascent
algorithm like we've been discussing for the previous few lectures.

that we want to learn. So we input the dimensionality of this latent space that we want to embed all these users and objects into.

MAP inference coordinate ascent algorithm

locations

Input: An incomplete ratings matrix M, as indexed by the set Ω. Rank d.

Output: N_1 user locations, $u_i \in \mathbb{R}^d$, and N_2 object locations, $v_j \in \mathbb{R}^d$.

Initialize each v_j. For example, generate $v_j \sim N(0, \lambda^{-1}I)$.

of object locations. (we could have also initialized the user locations 1st & then flipped the algo.)

for each iteration **do**

2 coordinate ascent steps.

- **for** $i = 1, \ldots, N_1$ **update user location** by maximizing the objective over each user location given the current values of all of the locations of all the objects.

$$u_i = \left(\lambda\sigma^2 I + \textstyle\sum_{j\in\Omega_{u_i}} v_j v_j^T\right)^{-1} \left(\textstyle\sum_{j\in\Omega_{u_i}} M_{ij} v_j\right)$$

- **for** $j = 1, \ldots, N_2$ **update object location**

relevant locations of the users. We use all of the users who have rated that object and all of the ratings here.

$$v_j = \left(\lambda\sigma^2 I + \textstyle\sum_{i\in\Omega_{v_j}} u_i u_i^T\right)^{-1} \left(\textstyle\sum_{i\in\Omega_{v_j}} M_{ij} u_i\right)$$

*

Predict that user i rates object j as $u_i^T v_j$ rounded to closest rating option

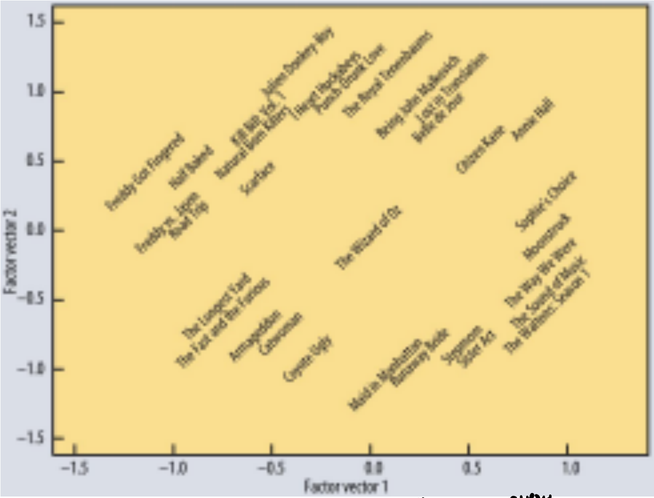
that we don't have in the matrix M

*Convergence :

So we iterate back and forth between these two steps and
eventually be algorithm will converge.
We can assess this convergence by calculating the log of the joint
likelihood that's that's the function we're trying to maximize so
we can evaluate that function after each iteration to see if it's converged or not.

ALGORITHM OUTPUT FOR MOVIES

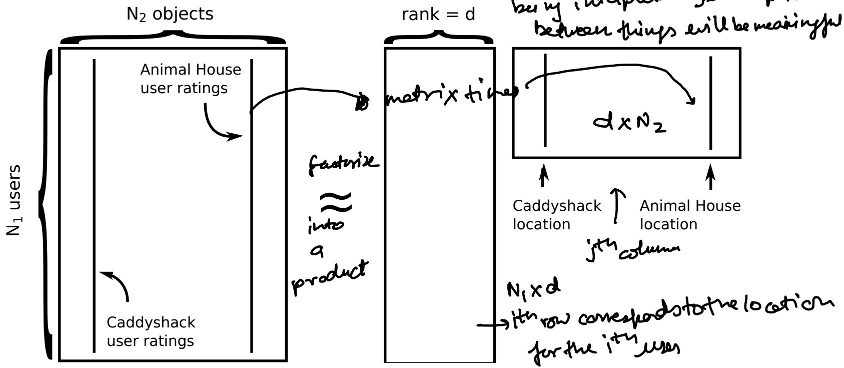
Data living in $\mathbb{R}^2$



Hard to show in $\mathbb{R}^2$, but we get locations for every movies and every users. Their relative locations captures relationships (that can be hard to explicitly decipher).

[1] Koren, Y., Robert B., and Volinsky, C.. “Matrix factorization techniques for recommender systems.” Computer 42.8 (2009): 30-37.

If we return to the original example that I showed, it's easy to understand why this latent space embedding of users and objects should end up being interpretable, where proximities between things will be meaningful.



Returning to *Animal House* (j) and *Caddyshack* (j'), it's easy to understand the relationship between their locations v_j and $v_{j'}$:

- For these two movies to have similar rating patterns, their respective v's must be similar (i.e., close to each other in $\mathbb{R}^d$). *
- The same holds for users who have similar tastes across movies. **

* Similarity for movies

So if these two columns look alike, if they are very close to each other,
then that means that these two columns also have to be very close to each other.
Because if the values in these two columns differ significantly,
then the product of this vector with this matrix
will be very different from the product of this vector with this matrix.

And so if these two columns are highly correlated,
in the matrix factorization language, that means that these two
much smaller vectors are very close should be very close to each other.

Error:

?

And whatever error there is is either due to the Gaussian likelihood which absorbs
a lot of the error or
corresponds to the slight variation in the values in these two vectors.

** Similarity for users:

If I pick two users and they have very similar ratings.
Then then their corresponding rows of ratings will be equal to the corresponding
rows of these vectors, times the entire matrix you're on the right.
And again,
those two users should be located very close to each other in that space.

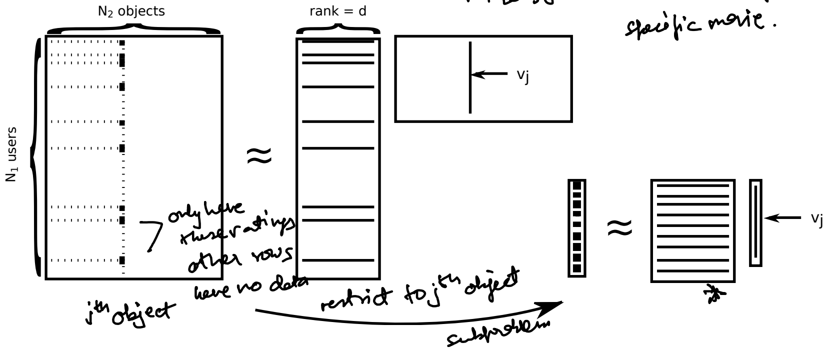
Conclusion:

So that's where this idea of collaborative filtering comes in,
that users can kind of influence each other's ratings and give information about
what other people will rate based on what they like.

MATRIX FACTORIZATION AND RIDGE REGRESSION

the difference, the boundary between unsupervised and
supervised learning algorithms is a little bit vague.
And in this sense we can almost view the collaborative filtering
problem with using matrix factorization as kind of like
a supervised learning algorithm, although not exactly.

To see the relationship: let's focus on the update for one specific movie.



There is a close relationship between this algorithm and ridge regression.

- Think from the perspective of object location v_j.
- Minimize the sum squared error $\frac{1}{\sigma^2}(M_{ij} - u_i^T v_j)^2$ with penalty $\lambda\|v_j\|^2$. **
- This is ridge regression for v_j, as the update also shows:

$$v_j = \left(\lambda\sigma^2 I + \sum_{i\in\Omega_{v_j}} u_i u_i^T\right)^{-1}\left(\sum_{i\in\Omega_{v_j}} M_{ij}u_i\right) \quad ***$$

- So this model is a set of $N_1 + N_2$ coupled ridge regression problems.

*

And now before I update the location for object J,
what I wanna do is actually solve a miniature problem that looks like this.

where did the missing values go.

** Ridge regression

What we did there was we minimized the sum of the squared errors
of this approximation to this data vector. [:]
In that sense, we could also think of it like minimizing RHS
the sum of squares of the corresponding value, MIJ, in here,
minus the corresponding vecotr UI.
Which is the row here times the vector VJ.
We sum to those up and
then we added an L2 to penalty in the ridge regression problem to the magnitude of V_j

*** Terms correspondence :

Where for any particular user i that rated object j
we're using the users location as the covariant vector for that rating.
And then MIJ corresponds to the output that we want to now predict for
that particular rating.

RR: Mij

So in ridge regression, remember we actually had these outputs and
we had the inputs as well. ui
We wanted to predict an output based on the input covariant vector.
In this case, the those corresponding covariance are themselves unknown and
we want to learn them. matrix :

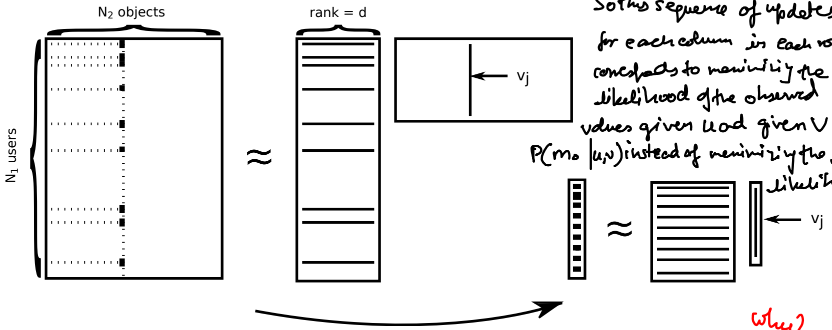
In this case, the those corresponding covariance are themselves unknown and
we want to learn them

Conclusion (over all problem relation):

So we can view the matrix factorization as a sequence of N1 plus N2
different retrogression problems that we solve iteratively.

why?

So this sequence of updates for each column in each row corresponds to maximizing the likelihood of the observed values given u and given V P(mo | u,v) instead of maximizing the joint likelihood.



why?

We can also connect it to least squares.

- if we Remove the Gaussian priors on u_i and v_j. The update for, e.g., v_j is then

removed the additional regularization term

$$v_j = \Big(\sum_{i\in\Omega_{v_j}} u_i u_i^T\Big)^{-1}\Big(\sum_{i\in\Omega_{v_j}} M_{ij}u_i\Big)$$

, that would to doing maximum likelihood for this problem

- This is the least squares solution. It requires that every user has rated at least d objects and every object is rated by at least d users. (for inverting the matrix)
- This probably isn't the case, so we see why a prior is *necessary* here. *

* So for this problem, it's mostly likely that that's not the case that we have that every user has rated more than d objects in every object has been rated more than D times. Especially if it user is new or an object is new.

Why Bayesian approach is required?

$\lambda \sigma^2 I$

And so we can kinda see here where the Bayesian approach is actually necessary for this model.
Previously, it wasn't required in order to make progress.
For this type of a model, we need the additional regularization represented by this additional matrix here which corresponds to a Gaussian prior.
In order to make sure that we can actually invert this matrix for every single user, and therefore that the algorithm won't crash because we have a non invertible matrix.

$$\left(\lambda \sigma^2 I + \sum_{i \in \Omega_{v_j}} u_i u_i^T\right)$$ ← for slide 31