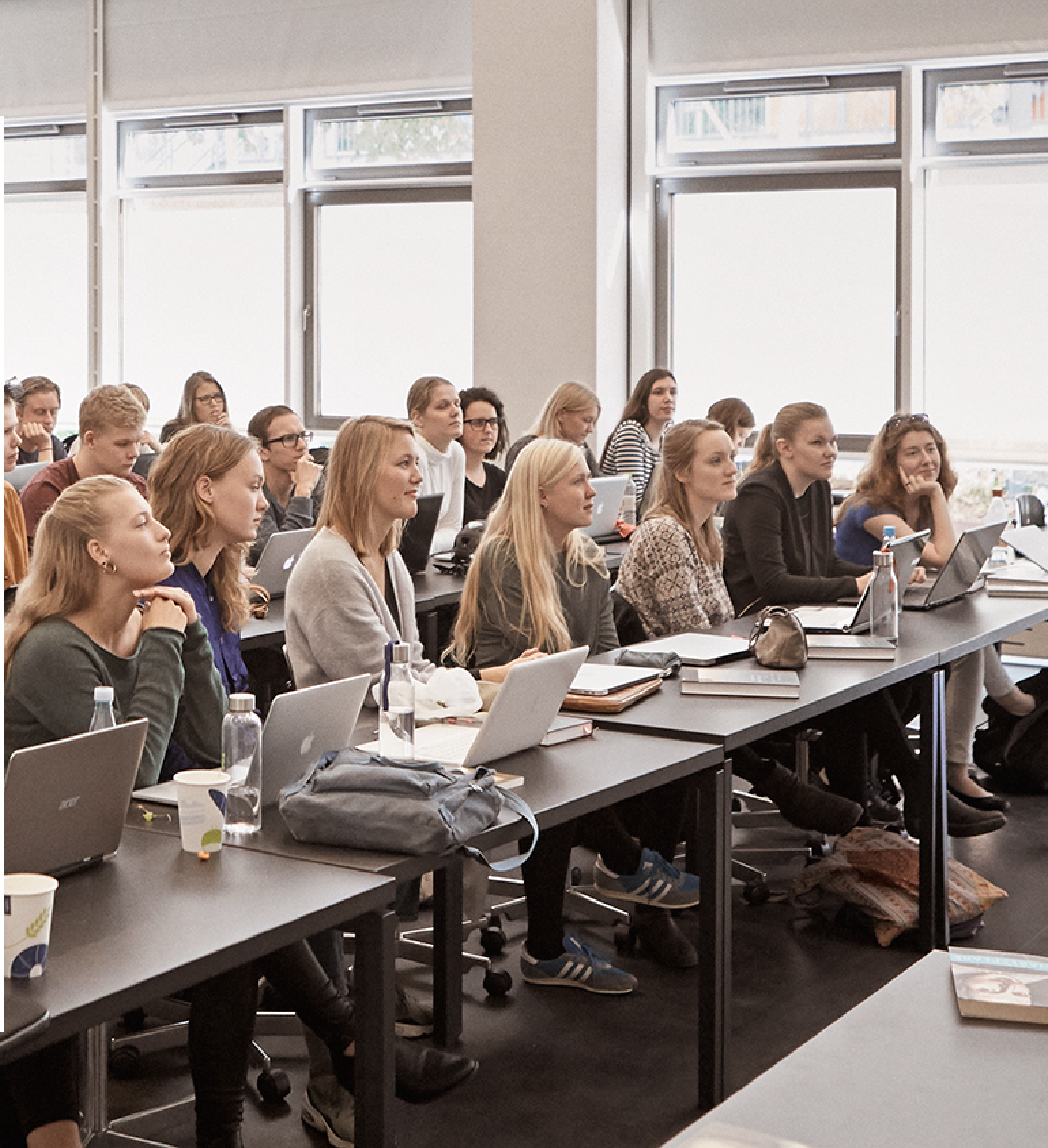
Recommender Systems: Part 2

Web Science Lecture

22 March 2020

Sagnik Ray Choudhury
src@di.ku.dk
http://sagnik.in



UNIVERSITY OF COPENHAGEN

Organization

- Part 1: Collaborative Filtering: Better Representation Through Dimensionality Reduction: PCA and SVD
- Part2: Latent factor models: more discussions.
- Part3: Intro to DNNs for RecSys
- Acknowledgements:
  - Charu C Agarwal (Recommender Systems, The Textbook)
  - Jon Shlens (PCA).
  - Jeremy Kun (SVD)

Announcement

- On **March 15** you will receive a link for the course evaluation in your KU e-mailbox.
- The deadline to complete the course evaluation is **March 28**.
- Please remember to fill out the course evaluation, even if you do not have anything wrong to report!
- Your feedback is very important to us, so we can understand how to improve this course for the next years.

Recommender Systems: Recap

- Problem definition: Given a utility matrix (UxN, U = |Users|, N = |Items|) with some values unspecified, 1. predict the unspecified values or 2. Rank the items according to the relevance of the users.
  - 2 comes naturally from 1.
- Evaluation methods for recommender systems
  - RMSE (evaluate 1)
  - Precision, Recall, F1@k. (evaluate 2)
  - Average precision @K, MAP@K.
  - Cumulative gain, DCG and NDCG.
- Recommender system algorithms
  - Content based
  - Collaborative filtering: User-User and Item-Item.
  - Latent Factor Models

Recommender Systems: Recap

- Content-based methods:
  - Start by representing an item in a feature space.
  - Represent user by combining the item she rates -> user is represented in the item feature space.
  - For a new item, the score is calculated by a distance metric between the user and the items.
  - Feature space is manually engineered: hard to create.
- Collaborative filtering:

1. For a target entry (u, i) determine the most similar rows/columns of the ratings matrix with the use of the cosine coefficient between rows/columns. For user-based methods rows are used, whereas for item-based methods, columns are used.
2. Predict the target entry (u, i) using a weighted combination of the ratings in the most similar rows/columns determined in the first step.

  - Item (user) feature space is the users (items): this is sparse.

Latent Factor Models: Sparsity to Compactness

- Consider a utility matrix U x I (number of users X number of items).
- Collaborative Filtering problems:
  - Feature space for item/user is sparse: lots of redundancy.
  - But complexity is $\mathcal{O}(N^2U)$ (item-item) . Reducing N to d (d << N) would be a huge improvement.
  - Side note: This happens in word vectors as well.
- Possible solution (for user-user, works the same way for items as well):
  - Reduce the dimensionality to d where d <U and I.
  - This will produce a new representation of the data.
  - Use this to represent each user and then repeat the usual steps.
- What dimensionality reduction technique to use?
  - PCA.
  - Singular value decomposition (SVD).

Dimensionality Reduction: Refresher

- Users are represented in item space and items are represented in user space.
- It's a vector space: it has a set of *basis vectors*. The basis vectors *span* that space.
- Basis vectors are linearly independent. A span of basis vectors is a set of vectors that can be expressed as a linear combination of these basis vectors.

1. Ratings between -1 and 1.
2. No missing values

NERO	Julius Caesar	Patriot	Sleepless Seat	Pretty Woman	Casablanca
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	1	1	1
-1	-1	-1	1	1	1
-1	-1	-1	1	1	1
-1	-1	-1	1	1	1

- There's a correlation in the data.
- There is one vector that represents user 1-3, 1 vector that can represent user 5-7, and one vector that can represent user 4. So obviously, we can represent each user by at least them.

1	1	1	0	0	0
1	1	1	1	1	1
-1	-1	-1	1	1	1

1	0	0
1	0	0
1	0	0
0	1	0
0	0	1
0	0	1
0	0	1

New (not quite) basis vectors

- What is the benefit of that?
- Dimensionality is reduced to 3.
- Can we do a little better?

Is This the Best We Can Do?

- (Row/column) Rank of a matrix is the number of linearly independent vectors (they are the same, why?).
- A data matrix **X** [m x n] can be represented in a k dimensional subspace where rank(X) = k < (m, n)
- **X** = **U** x $\mathbf{V}^T$ (U = new data matrix, V new basis vectors, with **U**=[m x k] and **V** = [n x k])
- If rank(**X**) = k, that decomposition always exists.

NERO	Julius Caesar	Patriot	Sleepless Seattle	Pretty Woman	Casablanca
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	1	1	1
-1	-1	-1	1	1	1
-1	-1	-1	1	1	1
-1	-1	-1	1	1	1

1. Ratings between -1 and 1.
2. No missing values

1	0
1	0
1	0
1	1
-1	1
-1	1
-1	1

This matrix has rank 2. Verify.

Here, the basis vectors are changed to [1, 1, 1, 0, 0, 0] and [0, 0, 0, 1, 1, 1]. That is a reduction from 6 to 2. Now, we need exactly 2 dimensions to represent the data.

1	1	1	0	0	0
0	0	0	1	1	1

Is This the Best We Can Do?: Contd.

1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	1	1	1
-1	-1	**1**	1	1	1
-1	-1	-1	1	1	1
-1	-1	-1	1	1	1

Previously, we had rank 2. This means, we could represent the data with 2 basis vectors. Now, we have a matrix of rank 3, so we will need at least 3 such vectors. Equivalently, We can not completely represent this with 2 dimensions, so there will be some error.

U

1	0
1	0
1	0
1	1
-1	1
-1	1
-1	1

$\mathbf{V^T}$

1	1	1	0	0	0
0	0	1	1	1	1

$\mathbf{U} \times \mathbf{V^T}$

1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	**2**	1	1	1
-1	-1	**0**	1	1	1
-1	-1	**0**	1	1	1
-1	-1	**0**	1	1	1

- Let's assume we are okay with losing some information.
- But how do we find out these matrices U and V?
- If the data matrix had rank k, not hard to reduce the ranks. But what happens when we want to *approximate* k?

PCA: we will find out a projection matrix **P** such that **XP** = **Y**.
Original formulation was $\mathbf{X} = \mathbf{UV^T}$.
Not hard to see that $(\mathbf{V^T})^{-1}$ would be **P.**

PCA

- We are projecting the data to a new space with a *linear* transformation.
- This is the same as changing the basis vectors (these new basis vectors spans the new space, and we are okay with some error).
- What are the properties we would want in such a transformation?
  - We want to minimize the reconstruction error.
  - We want to project the data in the directions of highest variance.
    - User 1-3 (5-7) are similar to each other. But User 3,4,5 are dissimilar.
    - If there's one dimension, you would want to have 3 numbers each significantly different.
    - As you can see, that is not possible in this data. You need at least 2 dimensions to express that.

1	0
1	0
1	0
1	1
-1	1
-1	1
-1	1

- Turns out, we can choose a projection matrix that satisfies both this criteria.
  - We showed how in the last class, but we will do that here again.

Covariance

- Variance: how do the values of a variable vary across itself [X takes values ($x_1..x_n$), bars indicates mean]

$$S^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

- Two variables, X ($x_1...x_n$) and Y ($y_1...y_n$), how do the values vary across themselves?

$$cov_{x,y} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

$$Cov(X_i, X_j) = \frac{\sum_{k=1}^{m}(X_{ik} - \bar{X}_i)(X_{jk} - \bar{X}_j)}{m-1}$$

Data matrix

$$\mathbf{M} = \begin{bmatrix} \boxed{\begin{matrix} X_{11} \\ X_{21} \\ \cdot \\ \cdot \\ \cdot \\ X_{m1} \end{matrix}} & \begin{matrix} X_{12} \\ X_{22} \\ \cdot \\ \cdot \\ \cdot \\ X_{m2} \end{matrix} & \begin{matrix} \dots \\ \dots \\ \cdot \\ \cdot \\ \cdot \\ \dots \end{matrix} & \begin{matrix} X_{1n} \\ X_{2n} \\ \cdot \\ \cdot \\ \cdot \\ X_{mn} \end{matrix} \end{bmatrix}$$

Covariance Matrix

$$\begin{bmatrix} \mathrm{Var}(X_1) & \mathrm{Cov}(X_1, X_2) & \dots & \mathrm{Cov}(X_1, X_n) \\ \mathrm{Cov}(X_2, X_1) & \mathrm{Var}(X_2) & \dots & \mathrm{Cov}(X_2, X_n) \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{Cov}(X_n, X_1) & \mathrm{Cov}(X_n X_2) & \dots & \mathrm{Var}(X_n) \end{bmatrix}$$

- Each feature is a random variable that is represented in a column (called X_i).
- Each X_i takes values X_{i1} to X_{im} .
- There are n such columns (random variables), X_1 to X_n.

- $Cov(X_i, X_j) = Cov(X_j, X_i)$ (verify) => The matrix is symmetric.
- If you subtract the column means from the column values, Cov(**X**) = ($\mathbf{X}^T\mathbf{X}$)/m-1 (Under the assumption that the rows are data points and columns are features.).
- We will just take the numerator from now on because the denominator does not depend on **X** values.

What is the Relationship with PCA?

- If two vectors (X_i, X_j) are highly correlated (think one as the multiple of the other), the covariance is high.
  - Also, if we had one of them as the basis vector, would we need the other?
- If the correlation is low (think two perpendicular vectors) the covariance is low (0 when the vectors are perpendicular to each other).
- PCA definition: $\mathbf{XP} = \mathbf{Y}$
- In the covariance matrix of **Y** ($\mathbf{C_Y}$), we would want the diagonal entries (variances) to be high, and the non diagonal entries (covariances) to be low.
- In other words, we would want $\mathbf{C_Y}$ to be a *diagonal matrix*.
- What is the projection matrix **P** that makes this happen?

Finding the Proper P

- Eigenvector of a matrix **A** is a vector X such that: **A**X = cX (c is a constant).
  - What does that mean, intuitively?
  - A matrix is a rotation + scaling transformation on a vector. Eigenvectors are those vectors which only gets scaled by this operation.
  - We only consider the *normalized* ones, because if E is an eigenvector, so is dE (d=constant)

- A symmetric matrix is diagonalized by an orthonormal matrix of its eigenvectors.

  Let's assume this is true for now, I will upload the proof separately

  - If there is a symmetric matrix **A** with associated eigenvectors $\{e_1, e_2, \cdots, e_n\}$, and **E** is a matrix such that the i^{th} column of E is e_i, then there exists a diagonal matrix **D** such that **A** = $\mathbf{EDE}^T$.

- We are looking for a **P** that makes $\mathbf{C_Y}$ (Covariance matrix of **Y**) diagonal.

- The trick: Let us select **P** to be the matrix where each column is the eigen vector of $\mathbf{C_X}$.

Finding the Proper P, Contd.

Unit vectors, product of each other 0

- If there is a symmetric matrix **A** with associated *orthonormal* eigenvectors $\{e_1, e_2, \cdots, e_n\}$ (they are always orthogonal if A is symmetric, *why*?), and **E** is a matrix such that the i^{th} column of E is e_i, then there exists a diagonal matrix **D** such that $\mathbf{A} = \mathbf{EDE^T}$.

- We are looking for a **P** that makes $\mathbf{C_Y}$ (Covariance matrix of **Y**) diagonal.

- The trick: Let us select **P** to be the matrix where each column is the orthonormal eigen vectors of $\mathbf{C_X}$. Then we can write $\mathbf{C_X}$ as $\mathbf{PDP^T}$.

Not considering denominators, hence approximate

$\mathbf{C_Y} = \mathrm{Cov}(\mathbf{Y}) = \mathrm{Cov}(\mathbf{XP}) \sim (\mathbf{XP})^T(\mathbf{XP}) = \mathbf{P^T X^T X P} = \mathbf{P^T C_X P}$

If I make P according to the last condition, we can write $\mathbf{C_X} = \mathbf{PDP^T}$. Replace $\mathbf{C_X}$ with this.

$\mathbf{C_Y} = \mathbf{P^T (PDP^T) P}$. Now, **P** is orthonormal, therefore, $\mathbf{P^T P} = \mathbf{I}$. Therefore, $\mathbf{C_Y} = \mathbf{D}$ (which we wanted to do).

PCA, Finally.

- We would call the columns of **P** (the projection vectors) the principal components.
- This is obtained by diagonalizing $\mathbf{C_Y}$: this could be done in multiple ways.
- PCA is one such diagonalization where each projection vector is orthonormal.
- We also have a way to judge the importance of these dimensions (by the variance) .
- Dimensions
  - Data (**X**) dimensions: m x n
  - Cov matrix $\mathbf{C_X}$: n x n
  - Rank($\mathbf{C_X}$) = r => r orthonormal eigenvectors (which will come to be the principal components). You will choose the top k (The ones with the highest variance). This will make the projection matrix **P**.

- We have found a way of dimensionality reduction that not only removes redundancy but also projects the data in an interesting subspace (variance maximization and covariance minimization).
- The principal components are orthonormal: which helps to cast it as the problem of finding eigenvectors of the covariance matrix.
- Still, computing that covariance matrix is time consuming: matrix multiplication (what is the complexity?) and over/underflow.

Singular Value Decomposition: Concepts

- **X** = *m* x *n*.
- $v_1, v_2, \cdots v_r$ is the set of orthonormal eigenvectors of $\mathbf{X}^T\mathbf{X}$ with eigenvalues ($e_1, e_2, \cdots, e_r$).
  - $(\mathbf{X}^T\mathbf{X})v_i = e_i v_i$, v_i has a dimension of *n* x *1*;
  - As before, $\mathbf{X}^T\mathbf{X}$ is symmetric with rank r, therefore has r real eigenvalues and associated eigenvectors.
  - Let's define $s_i = \mathrm{sqrt}(e_i)$. We will call them singular values.
- Can you see that $\mathbf{X}v_i$ (dimension *m* x *1*) and $\mathbf{X}v_j$ are orthogonal (for any i,j)?
  - It turns out we can get a set $\{u_1,\cdots,u_r\}$ of *orthonormal* vectors such that $\mathbf{X}v_i = s_i u_i$
- This is interesting.
  - $\{v_i\}$ and $\{u_i\}$ are orthonormal sets of vectors that span the column space **and** the row space.
  - For PCA, we got a set of orthonormal vectors that spanned *one* of row/ column space (depending on how you represent the data).

SVD, Construction.

- $\{v_i\}$ and $\{u_i\}$ are orthonormal sets of vectors that span the column space **and** the row space.
- Both sets have cardinality r where rank($\mathbf{X}^T\mathbf{X}$) = r
- Let's build the matrices: $\mathbf{X}v_i = s_i u_i$ or, $\mathbf{X}v_i = u_i s_i$.
  - $s_1 = 1, s_2 = 1$

X (4 x 3)

1	2	3
4	5	6
7	8	9
10	11	12

v_1	v_2
1	2
3	4
5	6

u_1	u_2
22	28
49	64
76	100
103	136

V (3x2)

1	2
3	4
5	6

U (4x2)

22	28
49	64
76	100
103	136

These are not orthonormal vectors, here just to explain the construction!

By column stacking, we can write these equations more concisely!, or XV = U

- What happens when $s_1 = 5, s_2 = 3$?

5	0
0	3

- We add a diagonal matrix $\boldsymbol{\Sigma}$
- In the diagonal matrix, the values will be in decreasing order (this would require u and v columns to be rearranged appropriately), and finally, $\mathbf{XV} = \mathbf{U}\,\boldsymbol{\Sigma}$.
- Multiplying both side by $\mathbf{V}^{-1}$, $\mathbf{X} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^{-1}$ => $\boxed{\mathbf{X} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^T}$. ($\mathbf{V}^T = \mathbf{V}^{-1}$ because V is orthonormal).

This is the SVD formulation.

SVD: Implications

- We know that a data matrix **X** (*m* x *n*) can be written as $\mathbf{X} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^{\mathbf{T}}$, where:
  - U is a set of orthonormal basis vectors (**U** => *m* x *r*).
  - V is another set of orthonormal basis vectors (**V** => *n* x r).
  - $\boldsymbol{\Sigma}$ is a diagonal matrix, of some sort.
- What is really happening?

$$
\begin{array}{r c}
 & \begin{array}{ccc} \text{Aisha} & \text{Bob} & \text{Chandrika} \end{array} \\
\begin{array}{r} \text{Up} \\ \text{Skyfall} \\ \text{Thor} \\ \text{Amelie} \\ \text{Snatch} \\ \text{Casablanca} \\ \text{Bridesmaids} \\ \text{Grease} \end{array} &
\begin{pmatrix} 2 & 5 & 3 \\ 1 & 2 & 1 \\ 4 & 1 & 1 \\ 3 & 5 & 2 \\ 5 & 3 & 1 \\ 4 & 5 & 5 \\ 2 & 4 & 2 \\ 2 & 2 & 5 \end{pmatrix}
\end{array}
$$

SVD is a method to do this so that these two discoveries happen simultaneously.
Moreover, you can choose a rank k approximation of the data, by choosing r=k

- Some person rates a movie. It's a complex process. We have no idea about the process, but we have data that says 3 people rated 8 movies.
- With this data we will represent movies in a 3d space, and people in an 8d space.
- New movie comes in, we would want to represent it as a linear combination of existing movies and vice versa.
- We want to discover a special list of vectors v1···v8 to do just that (we did something similar in PCA). The Same goes for the people p1, p2··· p3.

SVD and PCA

- We don't have time to discuss how SVD is implemented.
  - In the QA session, we will go through a very basic solution.
- For now, let's assume very efficient algorithms exist for SVD.
- Can we use SVD to compute PCA?
- We know the principal components of $\mathbf{X}$ are the eigen vectors of $\mathbf{C_X}$.
- Since $\mathbf{C_X}$ is symmetric, we can also write $\mathbf{C_X}=\mathbf{EDE^T}$, where $\mathbf{E}$ is the eigenvectors of $\mathbf{C_X}$.
- $\mathbf{C_X} = \mathbf{X^T X}/n\text{-}1$. According to SVD, $\mathbf{X}=\mathbf{USV^T}$.
- $\mathbf{C_X} = \mathbf{(USV^T)^T(USV^T)}/n\text{-}1 = \mathbf{VS^TU^TUSV^T}/n\text{-}1 = \mathbf{V}(\mathbf{S}^2/n\text{-}1)\mathbf{V^T}$? (Why does $\mathbf{U^TU}$ cancel out)?
- Not difficult to see $\mathbf{V}$ constitutes the principal components of $\mathbf{X}$.

Summary

- Utility matrices are a special kind of data matrices
  - Typically, you would want to change the representation of the data points among the feature dimension, but here both dimensions are important.
- We have seen two linear methods of dimensionality reduction commonly used in the recommender systems: PCA and SVD.
  - The crux of linear dimensionality reduction is finding new basis vectors.
  - PCA can be thought of changing the basis vector for the row/column space (depending on how you put the data) but SVD does that simultaneously.
- But they are on fully specified matrices: that's not the case for recommender systems:
  - Subtract row mean to convert a sparse utility matrix to a fully specified data matrix.

What is The Problem In Such Conversions?

- We have a data matrix X (m x n), we want to reduce it to m x k.
- If m is users and n is movies, it will give us a *compact* representation of users in movie space.
- We can use PCA.
- Also, to get the *compact* movie representation in terms of users, we can just transpose the matrix and run PCA again.
- SVD will do this jointly.

User Index	Godfather	Gladiator	Nero
1	1	1	1
2	7	7	7
3	3	1	1
4	5	7	7
5	3	1	?
6	5	7	?
7	3	1	?
8	5	7	?
9	3	1	?
10	5	7	?
11	3	1	?
12	5	7	?

- Correlation between Gladiator is Nero is extremely high.
- Correlation between Gladiator is Godfather is low.
- It's not possible to do PCA on this matrix because it is not fully specified.
- Let's make it fully specified by using the **average score** (column mean) for Nero which is 4.
- What happens to the covariance matrix?

	Godfather	Gladiator	Nero
Godfather	2.55	4.36	2.18
Gladiator	4.36	9.82	3.27
Nero	2.18	3.27	3.27

- Gladiator and Nero has low correlation, and Gladiator and Godfather has high correlation!

Latent Factor Models to the Rescue

PCA: we will find out a projection matrix **P** such that **XP** = **Y**.
Original formulation was **X** = **UV**^{T} .
Not hard to see that $(\mathbf{V}^T)^{-1}$ would be **P.**

- PCA could be thought as a matrix factorization:
- In general, we can factorize a data matrix.
  - You can factorize a matrix **X** (m x n) in **UV**^{T}.
  - **U** has dimensions mxk and **V** has dimensions nxk (if k is the rank of the matrix, such factorization will always exist).
  - U is a lower dimensional representation of X.
  - PCA can be thought of a factorization with some constraints: in U, covariances are minimized and variances are maximized.
  - SVD was $\mathbf{X} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^T$. You can see by putting the values $\boldsymbol{\Sigma}$ inside **U** or **V**, this is again the same formulation. The constraints are **U** and **V** are orthonormal. Then we will take those **U** and **V** columns that corresponds to the highest k values of $\boldsymbol{\Sigma}$ (to get a k rank approximation of the original matrix)
- We will assume the factorization will discover the latent properties in the data (for movies-users, this can be the genre).
- How to find U and V? How to handle missing values?

Matrix Factorization

- A matrix **X** (*m* x *n*) with rank k (k < min(m, n)) can always be factorized as **X** = **UV**^{T} where **U** is mxk and **V** is nxk.
- If we want a lower rank approximation (<k), we can still do the same, now **X** ≈ **UV**^{T}
- How to find U and V?
- How to handle missing values?
- Let's try an iterative process:
  - Start with random matrices **U** and **V** (**UV**^{T} will obviously not equal **X**).
  - At each step, calculate the approximation error: J = 0.5$\boxed{||\mathbf{X}-\mathbf{UV}^T||^2}$ Frobenius norm
  - Depending on the error: *adjust* **U** and **V**.
  - Continue until E is less than a certain value.

What is the Error?

- $E = 0.5||\mathbf{X}-\mathbf{U}\mathbf{V}^T||^2$
- Let's call $\mathbf{UV^T}$ as **D**. What is the dimension of **D**? m x n.
- What is an element of **D**? $r'_{ij} = \sum_{s=1}^{k} u_{is} \cdot v_{js}$
- What is J then? $\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}(r_{ij} - \sum_{s=1}^{k} u_{is}v_{js})^2$
- But all values are not observed! in other words, you can not take that external sums.
  - Let's assume the set of observed values in **X** is S.
  - Then the sum becomes: $\frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}\right)^2$
  - If we can find u_{is} and v_{js} from this, we have solved the unseen ratings problem!

How to Adjust U and V?

- Gradient descent: $\theta_{new} = \theta_{old} + \alpha \nabla E$
- Suppose I have a parameter θ . (with a value)
- I will find the gradient of the error with respect to θ.
- My new value for the parameter will be in the direction of the gradient, multiplied with some constant.
- This will give me the new value for the parameters.
- What are my parameters here? U and V themselves, more specifically, the values u_{is} and v_{js}.
- remember, $J = \frac{1}{2} \sum_{(i,j)\in S} e_{ij}^2 = \frac{1}{2} \sum_{(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right)^2$
- We will find out $\frac{\partial J}{\partial u_{iq}}$ and $\frac{\partial J}{\partial v_{jq}}$

Formal Algorithm for Matrix Factorization

- Let's look at the informal version first:

Start with random matrices **U** and **V** (**UV**^{T} will obviously not equal **X**).
At each step, calculate the approximation error: J = 0.5||**X**-**UV**^{T}||^2
Depending on the error: *adjust* **U** and **V**.
Continue until E is less than a certain value.

$$J = \frac{1}{2}\sum_{(i,j)\in S} e_{ij}^2 = \frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2$$

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-v_{jq}) \quad \forall i \in \{1\ldots m\}, q\in\{1\ldots k\}$$
$$= \sum_{j:(i,j)\in S}(e_{ij})(-v_{jq}) \quad \forall i \in \{1\ldots m\}, q\in\{1\ldots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-u_{iq}) \quad \forall j \in \{1\ldots n\}, q\in\{1\ldots k\}$$
$$= \sum_{i:(i,j)\in S}(e_{ij})(-u_{iq}) \quad \forall j \in \{1\ldots n\}, q\in\{1\ldots k\}$$

Algorithm *GD*(Ratings Matrix: R, Learning Rate: α)
begin
  Randomly initialize matrices U and V;
  $S = \{(i,j) : r_{ij} \text{ is observed}\}$;
  while not(convergence) **do**
  begin
    Compute each error $e_{ij} \in S$ as the observed entries of $R - UV^T$;
    for each user-component pair (i,q) **do** $u_{iq}^{+} \Leftarrow u_{iq} + \alpha \cdot \sum_{j:(i,j)\in S} e_{ij}\cdot v_{jq}$;
    for each item-component pair (j,q) **do** $v_{jq}^{+} \Leftarrow v_{jq} + \alpha \cdot \sum_{i:(i,j)\in S} e_{ij}\cdot u_{iq}$;
    for each user-component pair (i,q) **do** $u_{iq} \Leftarrow u_{iq}^{+}$;
    for each item-component pair (j,q) **do** $v_{jq} \Leftarrow v_{jq}^{+}$;
    Check convergence condition;
  end
end

What are the Possible Improvements?

- **Regularization**
  - Raters might have bias: some raters will always provide less score than the others.
  - Movies might have bias: some movies will be popular across the board.
  - These biases are unknown: you need to discover them from the data!
  - Can we change the error function itself to capture this?
  - Previously, r'_{ij} = $\sum_{s=1}^{k} u_{is} \cdot v_{js}$
  - Now, , r'_{ij} = $o_i + p_j + \sum_{s=1}^{k} u_{is} \cdot v_{js}$ o_i is the rater bias, p_j is the popularity bias.
  - With that, we want to minimize: $\frac{1}{2} \sum_{i,j \in S} (r_{ij} - o_i - p_j - \sum_{s=1}^{k} u_{is} v_{js})^2$

Note, o_i and p_j won't cancel out!

- **Constraints**
  - If you cast SVD as a factorization problem, then you would want to have the constraints that **U** and **V** are orthonormal sets.

Summary

- Latent factor models work on a basic assumption: if you factorize a matrix and provide a low rank approximation, these low dimensional vectors would capture the latent reason for the data generation.

NERO	Julius Caesar	Patriot	Sleepless Seattle	Pretty Woman	Casablanca
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	1	1	1
-1	-1	-1	1	1	1
-1	-1	-1	1	1	1
-1	-1	-1	1	1	1

User to genre mapping

history	romance
1	0
1	0
1	0
1	1
-1	1
-1	1
-1	1

Movie to genre mapping

history	1	1	1	0	0	0
romance	0	0	0	1	1	1

Discovery of latent factors is equivalent of finding new basis vectors.

This is what brings everything together,.

Summary

- A basic latent factor model is equivalent of linear dimensionality reduction techniques with some caveats.
  - Does not hold the provable guarantees of PCA.
  - The basis vectors are not orthonormal, hard to come to a nice solution mathematically (remember for PCA we had to compute the eigen vectors of a matrix?).
  - Therefore, needs to be computed through iterative method.
- However, they are more general.
  - With addition of constraints, most linear dimensionality reduction techniques can be cast as a latent factor model.
  - Also, solves the problem of missing values nicely!

DNNs for Recommender Systems: A Very Basic Intro

- What is a DNN good for?
  - Finding representation in an unsupervised way.
  - Fusing different modalities (text, image, ratings).
- In content-based recommender systems, you can use a DNN to find a k dimensional representation.
  - This is same as finding the representation for a sentence from text.
- You can also fuse multiple modalities:
  - Because DNNs represent each image/text as a vector, it's not difficult to combine them (typically with a concatenation but other functions can be used).
- Collaborative filtering can also be improved using DNNs.

Lecture/Course Summary

- This lecture
  - There is a unified view of matrix factorization, linear dimensionality reduction.
  - SVD and PCA are powerful tools, but understanding the underlying mathematics is also important.
    - This gives us the intuition to create new methods.
- Course:
  - **Sentiment Analysis**: What is the most complex part of the problem? Handling the context. “You are so bad, man!” . Also, going from word to sentence representation using ConvNet.
  - **Dimensionality reduction and representation**: Representation is challenging; nonlinear methods (Word2Vec) are often better than the linear ones but hard to interpret and comes with no guarantee. Linear methods (PCA) has a nice guarantee of residual error minimization and variance maximization: but does not perform so well on words.
  - **Recommender Systems**: Why they are important, SVD (joint optimization of basis vectors for row and column space), gradient descent (sort of).